

UNIVERSIDADE FEDERAL DE MINAS GERAIS
Instituto de Ciências Exatas
Programa de Pós-Graduação em Ciência da Computação

Flávio Roberto dos Santos Coutinho

**Geração Procedural de Poses de Personagens em Pixel Art com Redes
Neurais Geradoras Adversárias**

Belo Horizonte
2025

Flávio Roberto dos Santos Coutinho

**Procedural Generation of Pixel Art Character Poses with Generative
Adversarial Networks**

Dissertation presented to the Graduate Program in Computer Science of the Federal University of Minas Gerais in partial fulfillment of the requirements for the degree of Doctor in Computer Science.

Advisor: Luiz Chaimowicz

Belo Horizonte
2025

2025, Flávio Roberto dos Santos Coutinho.
Todos os direitos reservados.

Coutinho, Flávio Roberto dos Santos.

C871g Geração procedural de poses de personagens em Pixel Art
com redes neurais geradoras adversárias [recurso eletrônico] /
Flávio Roberto dos Santos Coutinho. – 2025.

1 recurso online (140 f. il.) : pdf.

Orientador: Luiz Chaimowicz.

Tese (doutorado) - Universidade Federal de Minas
Gerais, Instituto de Ciências Exatas, Departamento de Ciência
da Computação.

Referências: f. 131-140.

1. Computação - Teses. 2. Computação gráfica - Teses. 3.
Geração de conteúdo procedural - Teses. I. Chaimowicz, Luiz.
II. Universidade Federal de Minas Gerais, Instituto de Ciências
Exatas, Departamento de Ciência da Computação. III. Título.

CDU 519.6*83 (043)

Ficha catalográfica elaborada por Célio Resende Diniz, bibliotecário CRB
6/2403 - Universidade Federal de Minas Gerais – ICEx.



UNIVERSIDADE FEDERAL DE MINAS GERAIS

GERAÇÃO PROCEDURAL DE POSES DE PERSONAGENS EM PIXEL ART COM REDES NEURAIS
GERADORAS ADVERSÁRIAS

FLÁVIO ROBERTO DOS SANTOS COUTINHO

Tese defendida e aprovada pela banca examinadora constituída pelos Senhores(a):

Prof. Luiz Chaimowicz - Orientador

Departamento de Ciência da Computação - UFMG

Profa. Maria Andréia Formico Rodrigues

Programa de Pós-Graduação em Informática Aplicada - UNIFOR

Prof. Esteban Walter Gonzalez Clua

Instituto de Computação - UFF

Prof. Flávio Vinícius Diniz de Figueiredo

Departamento de Ciência da Computação - UFMG

Prof. Erickson Rangel do Nascimento

Departamento de Ciência da Computação - UFMG

Belo Horizonte, 17 de novembro de 2025.

Documento assinado eletronicamente por **Luiz Chaimowicz, Professor do Magistério**



Superior, em 24/11/2025, às 17:24, conforme horário oficial de Brasília, com fundamento no art. 5º do [Decreto nº 10.543, de 13 de novembro de 2020](#).



Documento assinado eletronicamente por **Erickson Rangel do Nascimento, Professor do Magistério Superior**, em 25/11/2025, às 09:53, conforme horário oficial de Brasília, com fundamento no art. 5º do [Decreto nº 10.543, de 13 de novembro de 2020](#).



Documento assinado eletronicamente por **Flavio Vinicius Diniz de Figueiredo, Professor do Magistério Superior**, em 25/11/2025, às 10:03, conforme horário oficial de Brasília, com fundamento no art. 5º do [Decreto nº 10.543, de 13 de novembro de 2020](#).



Documento assinado eletronicamente por **Esteban Walter Gonzalez Clua, Usuário Externo**, em 25/11/2025, às 18:04, conforme horário oficial de Brasília, com fundamento no art. 5º do [Decreto nº 10.543, de 13 de novembro de 2020](#).



Documento assinado eletronicamente por **Maria Andreia Formico Rodrigues, Usuária Externa**, em 26/11/2025, às 12:01, conforme horário oficial de Brasília, com fundamento no art. 5º do [Decreto nº 10.543, de 13 de novembro de 2020](#).



A autenticidade deste documento pode ser conferida no site https://sei.ufmg.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador 4743050 e o código CRC 61C748C0.

DesabafoAcknowledgments

Não é bem assim que se começa um elemento pré-textual de agradecimento em uma tese, mas aqui vai precisar ser um pouco diferente. Afinal, foram seis anos em que de tudo aconteceu. Pandemia? Temos. Ascensão fascista? Também. Contudo, é o período da maior felicidade da minha vida. Chegaremos lá.

O agito teve início em 2019, quando eu já vinha estudando um tal de Aprendizado de Máquina misturado com Geração Procedural de Conteúdo para propor um projeto de pesquisa e entrar no doutorado. Foi o contato com um artigo que foi escolhido o melhor da trilha de Computação do SBGames 2019 [Serpa and Rodrigues, 2019] que me encantou com a possibilidade de propor algo que pudesse auxiliar artistas e desenvolvedores de jogos, esse ótimo tipo de pessoa, em algumas etapas que fossem mais repetitivas de seu trabalho. Portanto, obrigado Ygor e profa. Andreia Formico.

Propus um projeto que me dava orgulho. Afinal, eu ia resolver aproximadamente 100% (talvez mais) dos problemas da indústria de jogos: começaria apoiando artistas com a criação de imagens simples, de baixa resolução e complexidade, depois migraria para vários outros tipos de conteúdos cosméticos, como modelos tridimensionais e efeitos sonoros. Como é belo e puro o coração de um ingressante no doutorado, não é mesmo? Mas rapidamente a realidade se impôs e a pesquisa foi se lapidando que nem um foguete que vai deixando seus estágios anteriores para conseguir atingir as camadas mais distantes da atmosfera. Essa separação não foi fácil, mas meu orientador, prof. Luiz Chaimowicz, sempre estava lá para me ajudar a olhar para cima, que é para onde eu deveria ir. Obrigado, Chaimo, por toda a parceria, que já é antiga e vai continuar. Seu apoio, revisões, ideias e compreensão foram necessários e muito importantes para mim.

Da pandemia, veio o ensino remoto emergencial no CEFET-MG, em Belo Horizonte, onde trabalho. Nesse período, aproveitei para voltar a morar com minha esposa Marcelle Silva, passando a viver em Itabirito com ela, que já estava nesta cidade, sozinha, desde 2016. Dessa re-união, depois de nove meses veio o resultado: Amora, nossa filhota, que provavelmente está no topo do ranking das pessoas mais amadas do universo, a partir de 01/03/2023. Foi uma gravidez um pouco atribulada, mas deu tudo certo e o prêmio é o melhor do mundo. Agradeço do fundo do meu coração à Marcellinha por ser uma mãe incrível, muito carinhosa, estudiosa e cuidadosa com nossa neném, além de esposa parceira e amiga. Obrigado por tudo o que fez para que eu pudesse ficar quietinho trabalhando e concluindo esta tese.

Falando na pesquisa, como é difícil lidar com o *loop* do método científico (ideia,

desenvolvimento, avaliação) trabalhando com programas que precisam torrar horas e horas processando exemplos para aprender. Quando informo que uma única unidade de modelo levou, sei lá, 20h para treinar, podemos multiplicar isso por 30, 50, ou até 100x, para acomodar todos os outros treinamentos com valores de hiperparâmetros ou topologias de rede que tiveram resultados piores do que o modelo selecionado. Agora pense que trabalhei com 5 modelos nesta tese. Aliás, foram 5 que mereceram estar neste texto, mas houve muitos outros que não obtiveram a mesma glória. Minha nossa, cansa só de explicar. Se eu pegar algum desses caras que criaram isso na rua, não sei não... Hinton, Bengio e LeCun, cuidado comigo.

Esse elevadíssimo tempo entre ter uma ideia e saber se ela resultou em bonequinhos melhor desenhados ou não andou de mãos dadas com minha auto-cobrança hiperdimensionada, recém descoberta pela minha querida terapeuta Aparecida Pimenta. Fiquei preso por muito tempo no seguinte ciclo: se eu ajustar esse *learning rate* aqui em 10%, ou então multiplicar esse λ_{rec} por 5, vou conseguir uma melhora revolucionária; corta para daí a 20 horas, resultados levemente diferentes, não piores nem melhores, e uma nova ideia: agora preciso ajustar é o número de filtros da convolução inicial, aí vai melhorar de vez... e isso se repetia, até o próximo *deadline*, que se impunha e só assim a pesquisa seguia adiante, em novas direções. Portanto, agradeço aos ~~deadlines~~ à Aparecida, Marcelle e Chaimo, que muito me ajudaram a aceitar seguir adiante. E também agradeço à Amora, trazendo sorrisos, frutinhas e causos da escolinha, enquanto eu trabalhava. Também tenho um grande salve para minha amiga Glívia Barbosa, colega de mestrado, de CEFET-MG, e de doutorado. Minha amiga de alegrias e sofrências acadêmicas, com quem sempre pude contar. Obrigado também ao Lucas Chaves, coorientando de graduação, que corajosamente dividiu partes da pesquisa comigo e conseguiu avançá-las, segurando o boi pelo chifre.

E dedico também um agradecimento ao DCC/UFMG, pela oportunidade de estudo e crescimento; dos programas de apoio à divulgação de pesquisa, financiados pelos órgãos de fomento como CAPES, CNPq e Fapemig; e pelo CEFET-MG, que me concedeu um importantíssimo período de afastamento para que eu pudesse trabalhar exclusivamente no doutorado. Isso me lembra de ressaltar a necessidade de fortalecimento das estruturas de ensino e pesquisa, mas também de todas as instituições do país.

Agradeço a Deus, pelas oportunidades que tenho na vida. Uma delas é a de poder ajudar a cuidar da saúde de minha mãe e meu pai, já meio velhinhos, ambos com doenças neurodegenerativas, e apaixonados pela Amora.

Por fim, faz-se bastante necessário agradecer também a todos aqueles indivíduos que, com seu labor, se dedicam a planejar, criar e desenvolver experiências únicas e extremamente marcantes nas vidas das pessoas, sob a forma de jogos eletrônicos. Meu agradecimento se estende a todos, mas cito alguns grandes gênios japoneses que moram no meu coração: Shigeru Miyamoto, Eiji Aonuma, Tetsuya Takahashi, Soraya Saga e Yasunori Mitsuda. Vocês são demais.

Resumo

O desenvolvimento de jogos envolve a criação de diferentes tipos de conteúdo, um processo iterativo e geralmente demorado. Em jogos bidimensionais, os personagens são tipicamente representados em muitos *sprites*, pois podem executar diversas ações, exigindo vários quadros por clipe de animação e desenhos em diferentes direções (e.g., frente, trás, esquerda, direita). Embora a maioria das tarefas envolvidas seja criativa e, portanto, bem apropriada aos artistas, algumas delas podem se tornar repetitivas e, portanto, se beneficiar de automação. A área de Geração Procedural de Conteúdo apresenta técnicas que podem auxiliar na criação de recursos em jogos, mas são poucas as que almejam a criação de conteúdo cosmético, como imagens e sons. No entanto, os pesquisadores têm abordado novos problemas da área ao se apropriar dos avanços recentes no campo de Aprendizado de Máquina relacionados aos modelos geradores.

Esta tese aborda o desafio de gerar personagens em pixel art em diferentes lados (e.g., direito) a partir de imagens deles voltados para outros (e.g., frente). Enquadramos essa tarefa tanto como um problema de tradução de imagem para imagem, quanto como de imputação de dados ausentes, e propomos cinco modelos baseados em redes neurais geradoras adversárias para produzir imagens the personagens em diferentes poses.

As imagens geradas variam em qualidade, de perceptualmente idênticas ao objetivo em um conjunto de dados feito por personagens modulares até pouco reconhecíveis quando usados em personagens muito diferentes da partição de treinamento.

Também investigamos como o conhecimento específico de domínio do estilo pixel art pode ser integrado aos modelos, representando imagens como índices em uma paleta, adicionando perda por histograma e quantizando as imagens geradas, tanto em pós-processamento quanto de forma integrada. Este último método utiliza uma nova abordagem diferenciável que permite o treinamento de modelos geradores profundos de ponta a ponta para produzir imagens que aderem estritamente a uma paleta pretendida.

Nossas principais contribuições incluem a proposição dos modelos geradores capazes de produzir personagens em diferentes poses, as modificações propostas para as arquiteturas base de redes neurais geradoras adversárias que produziram melhores resultados do que suas implementações originais e o conhecimento obtido pela investigação de diferentes hipóteses para melhorar a qualidade de geração de imagens em pixel art.

Palavras-chave: geração procedural de conteúdo; redes geradoras adversárias; tradução de imagem para imagem; imputação de dados ausentes; pixel art.

Abstract

Game development involves creating different types of assets, an iterative and frequently time-consuming process. In two-dimensional games, characters are typically represented in many sprites, as they might perform many actions, requiring several frames for each animation clip while facing different directions (e.g., front, back, left, and right). While most of the involved tasks are creative and, hence, best suited to artists, some of them might become repetitive and may benefit from automation. The Procedural Content Generation field presents many techniques that can assist in creating game assets, but only a few focus on cosmetic content, such as visuals and sounds. However, researchers have been approaching new problems by leveraging recent advancements in the Machine Learning field related to generative models.

In this dissertation, we tackle the challenge of generating pixel art characters in different sides (e.g., right) given one or more images of them facing source ones (e.g., front). We approach this task as both an image-to-image translation and a missing data imputation problem and introduce five generative adversarial network models following different architectures to generate images of characters in different poses.

The model-generated images have varying quality, ranging from near zero distance to the ground truth in a dataset composed of modularly assembled characters to barely recognizable images for characters that are too different to the ones seen during training on other datasets.

We also investigate how domain-specific knowledge of the pixel art style can be integrated with the models. We evaluate representing images as indices in a palette, adding histogram loss term to the generator and quantizing the generated images to their palette both as a post-process step and inside the model. The latter uses a novel differentiable approach that allows training deep generative models end-to-end to produce images that strictly conform to an intended palette.

Our main contributions include the proposition of the generative models, capable of producing characters in different poses, the modifications to base generative adversarial network architectures that yielded better results than the original implementations, and the insights obtained by investigating different hypotheses to improve the quality of the generated pixel art.

Keywords: procedural content generation; generative adversarial networks; image-to-image translation; missing-data imputation; pixel art.

List of Figures

1.1	Comparison of tree bark textures from a game from 1998 and 2017.	18
1.2	A pixel art character showing a geometric inconsistency in his hat.	21
2.1	Diagram of a GAN during training and at inference time	28
2.2	How the value of a single pixel changes the character’s facial expression	35
2.3	Comparison of a forest landscape depicted as pixel art and in a photo	36
2.4	Shading schemes common in pixel art, and the presence of crisp edges	36
3.1	Sample results from MDGAN.	42
4.1	Architecture of the proposed Pix2Pix-based model	49
4.2	Outputs of Pix2Pix-based models with varying patch sizes for the discriminator	51
4.3	Architecture of the StarGAN-based model	54
4.4	A residual block from the StarGAN generator	55
4.5	Encoding domain information for the StarGAN generator	56
4.6	Pose translation using StarGAN generators with different input configurations	56
4.7	Pose translation using different mini-batch sampling strategies in StarGAN . .	58
4.8	Architecture of the CollaGAN-based model	60
4.9	Comparison of input dropout strategies during training in the CollaGAN-based model	64
4.10	Comparison of image replacement procedures for the CollaGAN-based model .	65
4.11	Architecture of the ReMIC-based model	66
4.12	Images generated by ReMIC-based models trained with different λ_{rec}	71
4.13	Architecture of SpriteGAN	73
5.1	Sample images from the datasets	82
5.2	Comparison of model outputs when training with RGB vs. RGBA channels . .	86
5.3	Output images generated with models trained on individual datasets	87
5.4	Sample images generated with individual datasets vs the larger one	89
5.5	Example images generated after training with data augmentation techniques .	91
5.6	Images generated by the proposed models using a single input image	94
5.7	Images generated by the CollaGAN-based model with different source domain combinations	96
5.8	Images generated by the ReMIC-based model with different source domain combinations	97

5.9	Comparison of the CollaGAN-based model using 2 and 3 images as input . . .	100
5.10	Comparison of the best source combinations for a target pose from ReMIC . .	100
5.11	Comparison of the CollaGAN and ReMIC-based models in easier tasks . . .	101
5.12	Comparison of the CollaGAN and ReMIC-based models in harder tasks . . .	102
5.13	Outputs from SpriteGAN showing the imputation of characters	104
5.14	Outputs from SpriteGAN showing different outputs for the same input . . .	105
5.15	Outputs from SpriteGAN showing character inpainting	107
6.1	Early-training output of the generators on validation data from palette-indexed colors varying order in which colors appear in the palette	111
6.2	Example character generated by the model trained with a histogram loss . . .	113
6.3	Images generated by the model trained with the histogram loss, varying the scaling factor	114
6.4	Images generated by the model with the histogram loss	115
6.5	Comparison of different strategies of the palette quantization post-processing step	116
6.6	Temperature parameter (τ) effects on generated pixel art using differentiable palette quantization.	120
6.7	Comparison of images generated using different palette quantization mechanisms.	122
6.8	Comparison of the images generated following each of the investigated hypotheses	125

List of Tables

2.1	Sizes and augmentation techniques of datasets used in Computer Vision tasks	37
3.1	Summary of the characteristics of the related work that generated game art . .	44
4.1	Number of parameters of the networks used by ReMIC.	71
4.2	Number of parameters of the networks used by SpriteGAN.	79
4.3	Space and time statistics of the proposed models	79
4.4	Comparison of the desired capabilities of the proposed models	80
5.1	Description of the datasets used in the experiments	83
5.2	Hyperparameters used for training each model	85
5.3	Training on the LARGER dataset but validating on the individual ones	88
5.4	FID and L_1 values of the generated images for each dataset, per augmentation technique	90
5.5	FID and L_1 values for Pix2Pix, StarGAN, CollaGAN-1, and ReMIC-1 models	92
5.6	FID and L_1 distances for all input images and target domain combinations for the CollaGAN-based architecture	98
5.7	FID and L_1 distances for all input images and target domain combinations for the ReMIC-based architecture	99
5.8	FID and L_1 distance of different input dropout strategies in different number of inputs scenarios	103
5.9	A summary of FID and L_1 averaged over all possible source and target domain combinations with a single input	108
5.10	A summary of FID and L_1 averaged over all possible source and target domain combinations with multiple inputs	108
6.1	FID and L_1 metrics comparing the baseline and the histogram loss hypothesis	113
6.2	FID and L_1 for images after post-processing using different color spaces	116
6.3	FID and L_1 for the models that implement the proposed hypotheses	123

List of Acronyms

ACGAN	Auxiliary Classifier Generative Adversarial Networks
AdaIN	Adaptive Instance Normalization
Adam	Adaptive Moment Estimation
cGAN	Conditional Generative Adversarial Networks
CVAE	Convolutional Variational Autoencoder
DCGAN	Deep Convolutional Generative Adversarial Networks
FID	Fréchet Inception Distance
FiLM	Feature-wise Linear Modulation
GAN	Generative Adversarial Networks
GPU	Graphics Processing Unit
LSGAN	Least Squares Generative Adversarial Networks
LoRA	Low-Rank Adaptation
MDGAN	Multiple Discriminators Generative Adversarial Networks
ML	Machine Learning
MLP	Multilayer Perceptron
PCG	Procedural Content Generation
PCGML	Procedural Content Generation with Machine Learning
ReLU	Rectified Linear Unit
ReMIC	Representational Disentanglement Scheme for Multi-Domain Image Completion
ROM	Read-only Memory
SSIM	Structural Similarity Index Measure
VAE	Variational Autoencoder

VQ-VAE	Vector-Quantized Variational Autoencoder
VRAM	Video Random Access Memory
WGAN	Wasserstein Generative Adversarial Networks
WGAN-GP	Wasserstein Generative Adversarial Networks with Gradient Penalty

Contents

1	Introduction	17
1.1	Motivation	20
1.2	Objectives	22
1.3	Methodology Overview	22
1.4	Main Results	23
1.5	Contributions	25
1.6	Dissertation Outline	26
2	Background	27
2.1	Generative Adversarial Networks	27
2.2	Image-to-Image Translation and Missing-Data Imputation	32
2.3	Pixel Art Characterization	35
3	Related Work	38
3.1	Generation of Game Art through Deep Learning	38
4	Model Proposition	45
4.1	Problem Statement	45
4.1.1	Generative Adversarial Networks Approach	46
4.2	Model Architectures	47
4.2.1	Based on Pix2Pix	48
4.2.1.1	Objective Function	49
4.2.1.2	Generator	50
4.2.1.3	Discriminator	51
4.2.1.4	Training Procedure	52
4.2.2	Based on StarGAN	53
4.2.2.1	Objective Function	53
4.2.2.2	Generator	55
4.2.2.3	Discriminator	57
4.2.2.4	Training Procedure	57
4.2.3	Based on CollaGAN	58
4.2.3.1	Objective Function	59
4.2.3.2	Generator	61
4.2.3.3	Discriminator	62

4.2.3.4	Training Procedure	63
4.2.4	Based on ReMIC	65
4.2.4.1	Objective Function	67
4.2.4.2	Generator and Encoders	69
4.2.4.3	Discriminators	69
4.2.4.4	Training Procedure	70
4.2.5	SpriteGAN	72
4.2.5.1	Objective Function	74
4.2.5.2	Generator and Encoder	75
4.2.5.3	Discriminators	77
4.2.5.4	Training Procedure	77
4.3	Architecture Comparison	79
5	Experiments	81
5.1	Methodology	81
5.1.1	Datasets	82
5.1.2	Evaluation Metrics	84
5.1.3	Training Procedure	84
5.2	General Architecture Decisions	85
5.2.1	RGB vs RGBA	86
5.2.2	Different Datasets	86
5.2.3	Data Augmentation	89
5.3	Translation using a Single Input	91
5.4	Translation using Multiple Inputs	95
5.4.1	Input Dropout Strategies	101
5.5	Results from SpriteGAN	103
5.5.1	Imputation from Different Input Combinations	103
5.5.2	Multi-modality for Images in Absent Domains	105
5.5.3	Inpainting with Different Masks	106
5.6	Discussion	106
6	Hypotheses on Pixel Art Generation	110
6.1	Palette Representation	110
6.2	Histogram Loss	112
6.3	Palette quantization Post-Process	114
6.4	Differentiable Palette Quantization	117
6.4.1	Softmin with Temperature	118
6.4.2	Weighted-Average Replacement	118
6.4.3	Palette Coverage Loss	119
6.5	Discussion	121

7 Conclusion	126
7.1 Future Work	128
7.2 Ethical Considerations	130
References	131

Chapter 1

Introduction

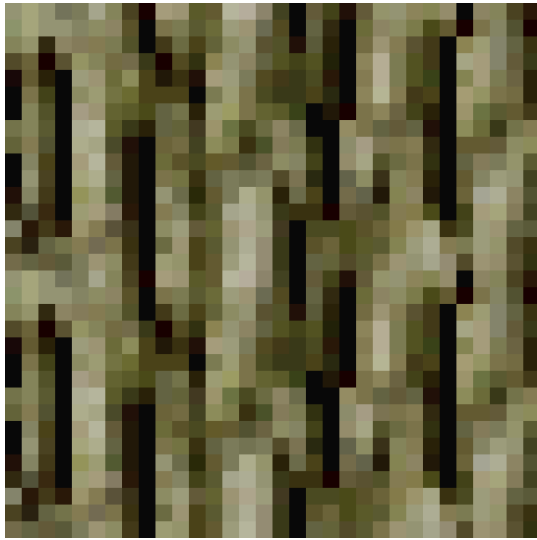
Nintendo, Sony, and Microsoft, the biggest players in console manufacturing, typically introduce a new console generation in the market every 5-7 years, increasing the capabilities of the hardware targeted at game playing [Schreier, 2017]. General-purpose computers are also widely popular as gaming devices, and their evolution, more continuous than that of consoles, is mostly driven by the introduction of more powerful video cards [Novak, 2022].

Such improvements in processing capacity allow the creation of much more complex games with better artificial intelligence, more accurate physics, and more realistic graphics. However, at the same time, it adds pressure to the game development industry as customers expect games to use all of the available capacity most efficiently. From the artistic point of view, the workload has increased a lot. As an anecdotal illustration, a tree bark texture from a Nintendo 64 game (1998) typically had a resolution of 32x32 texels, whereas one from a current-generation game might have at least 1024x1024 texels, which includes much more detail and, hence, can take longer to be created by artists. Such an example can be seen in Figure 1.1. In addition to the texture that defines the color of objects (i.e., the albedo map), recent games that strive for high realism require even more textures per object, such as a normal map, an ambient occlusion map, and a metalness/roughness map [Akenine-Möller et al., 2018].

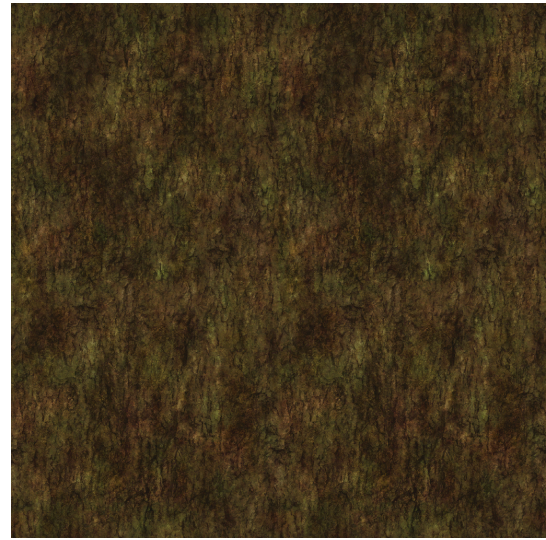
The increased effort involved is not restricted to three-dimensional games. Retro inspired 2D games created nowadays may also demand more assets than the games which inspired them. The integration of some computer graphics techniques in those games, such as dynamic lighting and shadow casting, requires every object to be depicted with additional sprites (e.g., the regular sprite encoding color, a normal map, and an alpha map). In the particular case of pixel art games, in which artists carefully pick the color of each pixel from a palette, the endeavor of producing such assets can be even greater [Silber, 2015].

In fact, the effort, time and hence the cost of game development has been steadily increasing since the dawn of the industry, and it is already very high. While in the early days games were simple and could be created by a group of a few people, nowadays, they can take teams of tens to hundreds of full-time developers along a few years to be

Figure 1.1: Comparison of tree bark textures from a game from 1998 and 2017, showing the increase in texture detail, $\approx 1,000\times$ more pixels to color.



(a) 32x32 texture extracted from the game
The Legend of Zelda: Ocarina of Time
[Nintendo, 1998]



(b) 1024x1024 texture extracted from
The Legend of Zelda: Breath of the Wild
[Nintendo, 2017]

completed. A business survey from 2017¹ revealed the cost growth to be exponential as it is multiplied by a factor of 10 every 10 years. The author observed that AAA games² developed between 2012 and 2017 had budgets averaging \$50 million and foresaw that the average cost might reach \$200 million for games with development starting in the following decade. Despite the high cost of game production due to the team sizes and the project duration, especially the AAA quality ones [Novak, 2022], their selling prices seem to have reached a market peak and saturated at \$60.00 [Schreier, 2017]. Such high production costs, along with the stagnated selling price, can render the gaming industry a non-practical business in the medium-term future. Therefore, reducing production effort, time and costs has become a paramount challenge for game development. We believe that the production pipeline can be optimized by sparing the developers from repetitive chores.

Among the tasks involved in the development process is the creation of systems, usually in the form of code, and content, including rules, levels, visuals and sound assets [Novak, 2022]. In particular, the creation of visual assets such as 3D models, textures, and character sprites might involve a lot of back and forth until their final version is reached. Although having many iterations is a typical and expected part of the creative process, some tasks can be repetitive and demand a lot of time from designers. As an illustration, changing the eye color of a character that is depicted in many sprites (e.g., 200, due to having many animation frames and facing different directions) might require updating all

¹The cost of games: <https://www.gamedeveloper.com/business/the-cost-of-games> (accessed 08/05/2024). Business article written by Raph Koster on 2017.

²AAA games are those produced with large budgets by big teams and with high technical complexity.

such files. Such task is much more repetitive and laborious than creative, and automating it could let the artists focus on more important activities.

A research area that can potentially reduce the effort in creating games is the Procedural Content Generation (PCG) field. It can be described as the “algorithmic creation of game content with limited or indirect user input” [Togelius et al., 2011, (p. 6)]. At its origins, PCG has been used to overcome strict storage game content limitations. For example, Atari 2600 game cartridges had only 4kb of Read-only Memory (ROM) space and the developers of River Raid (1982) [Activision, 1982], a vertical scrolling shooting game, decided to generate the levels at run-time instead of storing their information [Aycock, 2016] so the game code and assets could fit in the available space. In fact, if a game can generate content faster than players consume it, the same technique can also make games virtually infinite. Such characteristic could foster the games’ replay value, as each playthrough would provide a different user experience. However, a common criticism targeted at procedurally generated systems is that they usually create repetitive content, making it less memorable to players [Karavolos et al., 2015]. To some extent, some uses of PCG try to replace human designers in their activity by developing programs that produce content, but they fail to capture the much broader set of possibilities that can be envisioned by artists.

In addition to techniques that allow creating content online, developers can also use PCG *offline* to help producing game assets during the development process. When such methods are embedded in game design tools, for example, PCG can “augment the creativity of individual human creators” [Togelius et al., 2016, (p. 3)], instead of trying to replace them. In fact, there are PCG systems which let human creators and an algorithm work together to produce game content. The automation of only part of the content creation pipeline is usually designated as mixed-initiative PCG, when both the algorithm and the human creator can interact with the content being generated in a co-creation process [Liapis et al., 2016; Smith et al., 2011; Baldwin et al., 2017].

Different PCG systems implemented mixed-initiative approaches, such as Tanagra [Smith et al., 2011], a 2D platform level design tool which can autonomously generate levels and allows fine-tuning of the generated results while ensuring all levels are playable; and the Evolutionary Dungeon Designer [Baldwin et al., 2017], a level editor that uses a search-based strategy to allow its users to configure game design patterns which guide an evolutionary algorithm in the task of suggesting new maps. In accordance with those examples, we cohere with the view that PCG systems should have both algorithm and human in the loop. So, we lean our investigation towards systems that create content offline and that can be used as support for human creators. However, instead of creating levels, we target generating images.

Traditionally, the PCG field spanned methods that could be categorized as either constructive (such as cellular automata, fractals, grammars, and various noise functions)

or generate-and-test algorithms (e.g., evolutionary algorithms, and searching in planning space) [Togelius et al., 2016]. While constructive methods produce content in a single pass, the generate-and-test ones require a solution generation and a solution test phases, sometimes executed for many iterations. Except for textures created from combinations of noise functions, neither type of method had much success creating images or sound effects. However, a new research area that integrates Machine Learning (ML) techniques with PCG (called PCGML) has been gaining traction and among its possibilities is the use of generative models to produce artistic content, such as images and sounds [Summerville et al., 2018].

Of particular interest to this work are the deep generative models that can be trained using a dataset of image examples to learn a statistical distribution that allows them to produce new examples following it, such as Variational Autoencoders (VAEs), Diffusion Models, and Generative Adversarial Networks (GANs) [Rombach et al., 2022]. We chose to focus on GANs as, after training, they can generate images fast enough to allow real-time interaction (like VAEs) and the quality of the images usually does not suffer from blurring artifacts (which frequently happens with vanilla gaussian VAEs [Theis et al., 2015; Isola et al., 2017; Oord et al., 2017], but not with Diffusion Models and GANs).

Generative Adversarial Networks require training two players, a generator and a discriminator (each represented as a neural network), in a minimax game whose objective is to have the former learn the distribution of the images from the training data and the latter to distinguish between real and generated (fake) images [Goodfellow et al., 2014]. That process can be unconditional, in which the network produces new images from noise, or conditional, in which it uses some information prior to guide the generation towards producing a particular type of content. When the generator is conditioned on an image as input and produces another transformed image, the task is called image-to-image translation [Pang et al., 2022]. Similarly, when more than one image is available and can be used by a generative model, we can also frame the problem as a missing-data imputation task. In this work, we investigate different image-to-image translation and missing-data imputation GAN architectures to generate images of characters, as described in the following sections.

1.1 Motivation

In two-dimensional games with a top-down perspective, characters typically face four directions: left, right, top, or bottom. Suppose they are animated using cell anima-

Figure 1.2: A pixel art character showing a geometric inconsistency in his hat. The back and left-facing sprites depict a hat leaning backwards, but the front-facing sprite suggests it leans forward.



tion, a common technique that successively replaces the current image (called a sprite) with another that depicts the next pose in the sequence. In that case, the artists need to draw each character a number of times equal to the number of directions times the sum of the lengths of all animation clips. That is a challenging task and can take up a large portion of the project schedule. As exemplified earlier, if a change to a character is necessary, artists might need to revisit every sprite, which becomes more monotonous than creative. We aim to mitigate the effort required in such repetitive activities by introducing some automation to the creation of characters.

The [PCG](#) problem we address in this dissertation is the generation of pixel art characters in a target pose (e.g., right) given one or more images of them facing other directions (e.g., front). We found research from related work approaching similar tasks, but they either did not follow the pixel art style and imposed artificial restrictions on the shapes of characters, greatly reducing their variability [[Hong et al., 2019](#); [Choi et al., 2022](#)], or worked only on a dataset of modularly assembled characters [[Reed et al., 2015](#)], which also diminishes character diversity.

The proposed task is challenging for different reasons. First, it is ambiguous, as not all of the necessary information to generate a character in a target direction might be available from the source pose (e.g., creating a front-facing image from the back pose does not encode any facial features such as the eye colors). Second, the pixel art style is a stylistic, low fidelity representation of whatever it depicts [[Silber, 2015](#)]. For that reason, it is more susceptible to artistic license than more realistic styles and may not conform to geometric rules. To illustrate, Figure 1.2 shows a cook character facing back, right, and front from which the shape of his hat seem to be leaning towards different angles at the same time. The occurrences of such shape mismatch can hinder attempts to use geometric information to translate characters between different directions.

1.2 Objectives

Our main objective in this dissertation is to investigate the use of deep generative models for the task of creating characters in the context of game development. In special, we approach the generation of pixel art characters in a target pose given one or more images of it in source directions, using generative adversarial networks. In pursuit of that target, we refine the main objective into the following specific goals:

1. propose **image generation models** based on **GANs** for the task of generating characters;
2. investigate the task as both an **image-to-image translation problem**, as well as a **missing image imputation problem** (when more images are available);
3. explore the **integration of domain-specific knowledge** from the pixel art style with the generative model to potentially improve the quality of the generated images;

1.3 Methodology Overview

We conducted our work by considering potential requirements for a possible mixed-initiative tool that could help designers in the task of pixel art character creation. Then we proposed generative models that could at least partially fulfill such requirements.

Regarding the generative models that have been proposed recently with Machine Learning, we chose to investigate Generative Adversarial Networks (**GANs**) instead of Variational Autoencoders (**VAEs**) and Diffusion Models. We did not focus on **VAEs** as the images generated by vanilla gaussian-based models can become blurry due to their likelihood-based objectives [Oord et al., 2017; Isola et al., 2017]: Theis et al., 2015 show that high log-likelihood does not guarantee sharp-looking generated samples. Additionally, as we envision the use of the character generation inside a mixed-initiative tool, the interaction between artist and the system should be as close as possible to real-time. However, whereas **VAEs** and **GANs** can generate images in a single forward pass of their neural networks (usually taking a fraction of a second), Diffusion Models need to progressively refine a noisy image into the final one, which takes longer to complete (usually a few seconds) [Rombach et al., 2022]. For that reason, we refrained from investigating diffusion models.

On the quality of the images generated by the proposed models, we investigated four hypotheses related to pixel art and the use of a reduced number of colors: representing pixels as indices in a palette, adding a histogram loss term to the generator, post-processing the generated image to quantize it to an intended palette, and embedding that quantization inside the model by making all operations differentiable.

1.4 Main Results

We investigated the proposed specific goals at different depth levels. Most of our results have been published in conference proceedings or journals with an important impact. Next, we briefly describe what we have achieved, following the publication chronology.

Our first endeavor [Coutinho and Chaimowicz, 2022a] was to approach the generation of characters with different poses through the proposition of a GAN based on a general-purpose image-to-image translation architecture called Pix2Pix [Isola et al., 2017]. We modified the original architecture and trained models with four small datasets ($\leq 1k$ examples each) that featured artistically cohesive characters. The images generated by the models achieved different quality levels, with near perfect images generated from a dataset with modularly assembled characters, but barely recognizable images produced from characters that were very different to the ones seen on the training set of more challenging datasets. We found that the small dataset sizes hindered the model ability to generalize to unseen characters. Most of this work is described as part of Chapter 4 (Model Proposition) and Chapter 5 (Experiments).

To improve the generalization capacity of the model, we investigated particularities of the pixel art style and raised two hypotheses for integrating domain-specific knowledge into the training process of the Pix2Pix-based architecture [Coutinho and Chaimowicz, 2022b]. Grounded on the idea of having a reduced number of colors in the generated characters, we proposed: representing the pixels of the images as indices in a palette (instead of using RGBA) and adding a histogram loss while training the generator to steer it towards producing images with the same colors of the input image. The outcome of investigating both ideas is described in Chapter 6 (Hypotheses on Pixel Art Generation).

The initial work was invited for extension and published³ with the investigation of different data augmentation techniques [Coutinho and Chaimowicz, 2024b]. We also proposed and evaluated a post-processing step to quantize the images to the color palette of the input image, here presented on Chapter 6 (Hypotheses on Pixel Art Generation).

³The work “Pixel art character generation as an image-to-image translation problem using GANs”, published in Graphical Models, was awarded as the best paper of the journal during the year of 2024.

As an attempt to increase the capacity to generalize well on unseen data, we evaluated having more data available for training. Thus, we assembled a diverse dataset with 14k paired images of characters (12k/2k in the train/test partitions) and observed that when the model learned from much more data, it yielded better results when validating with the more artistically cohesive individual datasets. However, even though we increased the training data size in approximately 1.5 orders of magnitude, the quality of the generated images improved only marginally.

Another progress we have made was the proposition of multi-domain architectures based on StarGAN [Choi et al., 2018] and CollaGAN [Lee et al., 2019] for the character pose translation and imputation tasks. As multi-domain models, a single generator can generate characters in any of the four poses, which allows it to learn from more data. In the case of the CollaGAN-based architecture, the models can use all of the available poses of a character (e.g., back, left, and right) to generate the missing one (e.g., front), which frames the task as a missing image data imputation problem. We modified each architecture achieving better results than the original implementations. In addition, we also compared all three propositions. The work on the CollaGAN-based architecture was published⁴ in [Coutinho and Chaimowicz, 2024a] using the other two proposed architectures as baseline. Both multi-domain models are presented in Chapter 4 (Model Proposition) and Chapter 5 (Experiments).

We were invited to write an extension to that work and we submitted and published it in [Coutinho et al., 2025], where we introduced our fourth hypothesis to use domain-specific knowledge from the pixel art style: to have the generated images strictly adhere to an intended color set, we devised a palette quantization method that is fully differentiable, allowing models to train end-to-end to produce images with colors constrained by the palette. It involves the use of a softmin with temperature operation and a weighted-average pixel replacement procedure. We also proposed a palette coverage loss term that steers the generator towards producing images that use the full set of colors in the palette. This is described in Chapter 6.

After that, we also proposed a model based on the ReMIC architecture [Shen et al., 2021], which is presented in Chapters 4 (Model Proposition) and 5 (Experiments). Similarly to CollaGAN, it also tackles the missing-data imputation problem as the model receives images from one or more available domains. But instead of producing only a single missing image at a time, it can generate images from any number of missing domains. Additionally, it disentangles the image representation into a domain-specific style code and a content code.

We conclude our set of proposed models with a novel architecture, SpriteGAN, introduced in Chapter 4 (Model Proposition). It was designed from the ground up for the

⁴The work “A Missing Data Imputation GAN for Character Sprite Generation” was awarded as the best full paper of the Computing Track of SBGames 2024.

specific task of generating pixel art characters in different poses. SpriteGAN combines characteristics of the previously discussed architectures with newly introduced methods and training strategies, endowing it with capabilities not present in the others. In addition to imputing missing domains, it can generate multiple plausible outputs for a single input (multi-modality) and inpaint missing regions within existing domains.

Although SpriteGAN has not yet outperformed the other architectures in our preliminary experiments—and is therefore not included in the evaluation presented in Chapter 5 (Experiments)—we chose to include it in this dissertation, as we believe that a comprehensive hyperparameter search could yield competitive performance while retaining its unique abilities in multi-modality and inpainting.

1.5 Contributions

The results we achieved allowed us to make important progress in the fields of Procedural Content Generation with Machine Learning, but also some minor contributions to general Machine Learning. From a theoretical perspective in the PCG area, our work has been the first to tackle the generation of pixel art characters as pose translation and imputation problems. We have proposed five different GAN architectures that can, with different quality levels, generate pixel art characters in a target pose from one or more images of them facing other directions. We have also proposed and evaluated different hypotheses for integrating domain-specific knowledge of pixel art into the generation of this type of image. Lastly, we made a dataset (14k paired examples of pixel art characters in four poses) available to the public⁵.

As contributions to the ML field, while proposing architectures based on Pix2Pix [Isola et al., 2017], StarGAN [Choi et al., 2018], CollaGAN [Lee et al., 2019], and ReMIC [Shen et al., 2021] we introduced important changes to the original versions that improved either the quality of the generated images or the training efficiency. Our preliminary results with SpriteGAN showed that it is also possible for small-scale generative models to exhibit the ability to inpaint and to produce different outputs for the same source images, although still with insufficient diversity.

As a more pragmatic contribution, by making the architectures and source code publicly available⁶, any interested party can use or continue the evolution of the work presented here. For instance, game developers could already use the proposed architec-

⁵PAC Dataset: <https://github.com/plucksquire/pac/>

⁶Pix2Pix-based architecture (<https://github.com/fegemo/palette-and-histo-gan>), StarGAN, CollaGAN, ReMIC-based architectures and SpriteGAN (<https://github.com/fegemo/multi-domain>), and the weights of the trained models (<https://github.com/fegemo/pixel-sides-models>).

tures to train models using their own character dataset and improve their asset creation pipeline.

1.6 Dissertation Outline

The remainder of this dissertation is organized in the following chapters:

Chapter 2 (Background) explains the concepts of generative adversarial networks, and image-to-image translation and missing-data imputation problems. It also characterizes some specificities of generating pixel art.

Chapter 3 (Related Work) presents our survey on work related to the generation of game art using [PCGML](#).

Chapter 4 (Model Proposition) starts by presenting the methodology we used to propose the architectures for the deep generative models, then goes on to introduce each of them. It finishes with a summary of how they compare to each other and how they meet the desired requirements.

Chapter 5 (Experiments) introduces the methodology, results, and analysis conducted for model selection, defines some general architecture decisions, and evaluates the proposed models in the tasks of generating characters from a single image, multiple images, as well as some preliminary results on image inpainting and producing varied outputs for the same source images.

Chapter 6 (Hypotheses on Pixel Art Generation) presents our investigation on proposing and evaluating four hypotheses to improve the quality of the images generated with the proposed models.

Chapter 7 (Conclusion) closes this dissertation with some final remarks and some suggested new lines of research stemming from this work. It also includes brief discussion on the ethics around deep generative models.

Chapter 2

Background

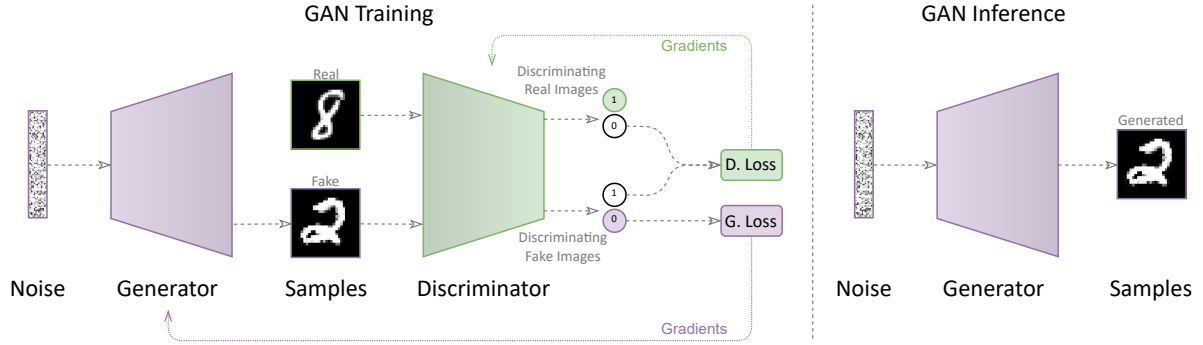
This chapter initially introduces the Generative Adversarial Networks (GAN), a framework for generating synthetic data that follows some distribution, such as images, based on deep neural networks. In this work, we investigate the use of GANs to generate pixel art characters. Following this, is the presentation of the image-to-image translation and missing-data imputation problems and how they have been addressed using GANs. We formulate the problem of generating pixel art character sprites in a target pose, given a single image of the same character in a source direction, as an image-to-image translation task. And when images from multiple domains are available, the problem can be addressed as a missing-data imputation task. This chapter concludes with a description of the pixel art style, highlighting the challenges faced by generative models when synthesizing images that follow it, in contrast to the creation of photorealistic images.

2.1 Generative Adversarial Networks

Goodfellow et al. [2014] introduced the concept of Generative Adversarial Networks (GAN) as a framework for generating content through an adversarial training process. It consists of having two models playing different roles in a minimax game: a generator G that evolves to be able to produce new content similar to the ones presented during training, and a discriminator D that learns to distinguish between real and generated (fake) examples. If an optimal state is reached, G captures the distribution of the training data and can generate new samples that are indistinguishable from the real ones, while D cannot tell whether an observation is real or fake.

Both players can be seen as functions that map their inputs to outputs. The generator receives some noise prior z (e.g., a vector of 100 random numbers) and learns to estimate a distribution p_{fake} that can generate a new sample $\hat{x} = G(z)$ which follows as close as possible the distribution p_{real} of the observed data x . The discriminator, in turn, receives either an image from p_{real} or from p_{fake} , and yields a probability $D(x)$ of x com-

Figure 2.1: Diagram of a GAN during training and at inference time. **Left** shows the model layout during training, with the generator’s output being used as the discriminator’s input, and error gradients flowing back: through the discriminator to the generator (purple dotted arrow), and through the discriminator only (green dotted arrow). **Right** shows the model at inference time consisting of only the generator, as the discriminator is not necessary for synthesizing images.



ing from p_{real} rather than p_{fake} , that is, it should be 1 for real images and 0 for fake. If both G and D are modeled using neural networks with differentiable objective and operations, they can evolve their weights towards minimizing the error using backpropagation [Linnainmaa, 1970]. Figure 2.1 (left) shows the generator and discriminator roles during training.

The training algorithm traverses the set of examples for a number of epochs. At each step, the weights of D update k times (a hyperparameter typically set to 1), and the weights of G update once. At D ’s turn, it is called once to discriminate a mini-batch of fake samples and a second time with real ones. The discriminator’s loss function $\mathcal{L}_D$ is the mean value (of all examples) of (the log of¹) the probability of incorrectly labeling an input example x as fake and (the log of) the probability of incorrectly labeling a fake example $G(z)$ as real. D updates its weights *ascending* the stochastic gradient of the following value function, while G updates *descending* it:

$$\min_G \max_D \mathcal{L}(G, D) = \mathbb{E}_x[\log D(x)] + \mathbb{E}_z[\log (1 - D(G(z)))]. \quad (2.1)$$

Considering that optimization algorithms usually strive to *minimize* the error, the particular loss function $\mathcal{L}_D$ for the discriminator is used as $\mathcal{L}_D = -\mathcal{L}(G, D)$, while the

¹Log-probabilities are used in place of probabilities themselves for two reasons. First, for each example x^i , D outputs the probability $P(x^i)$ of it being real. As we need to penalize D according to its prediction of m multiple examples (independent co-occurring events), we would multiply all such probabilities $P(x^0) \cdot P(x^1) \cdots P(x^i) \cdots P(x^m)$ together to calculate the error. However, probabilities are in the range of $[0, 1]$ and multiplying many such numbers (say, 100) approaches zero and eventually causes an underflow due to the finite capacity of the floating point number representation. Second, by the product property of logarithms, the log of the product of probabilities equals the sum of the individual log-probabilities ($\log(\prod_i P(x^i)) = \sum_i \log P(x^i)$). And a sum has much easier differentiation rules compared to multiplication, which is important for backpropagating the error throughout the networks’ weights.

one for the generator, $\mathcal{L}_G$ needs to consider only² the term that contains the function G (i.e., the right part), making $\mathcal{L}_G = \mathbb{E}_z[\log(1 - D(G(z)))]$.

After a successful training, G has converged to $p_{\text{fake}} \approx p_{\text{real}}$ allowing it to generate images that are similar to the ones from the training set. At that time, the discriminator is no longer necessary. Figure 2.1 (right) shows the GAN model after training has finished, consisting of only the generator.

Even though the proposed model is capable of generating new content, it does not allow defining which type of data it creates. For instance, if G was trained to generate images of handwritten digits $[0, 9]$, for each random input z , it will create an image of some random digit, without the opportunity of choosing which to create. Tackling that problem, Mirza and Osindero [2014] proposed a way to condition the model into producing specific modes of content, directing the data generation process. They called it **Conditional Generative Adversarial Networks (cGAN)**.

To control the generated data in some direction, in addition to the noise prior z , the authors added a second input y to G with some sort of data, which can be a class label or data from another modality (e.g., text describing images, or images). In the case of training with images of handwritten digits, they used one-hot encoded labels for numbers. D also receives the additional y input, turning it into a conditional discriminator: it distinguishes whether the image is real or fake *while considering* the provided condition (e.g., is this a real or fake *digit 9*?). The minimax value function being optimized by G and D becomes the following, with differences from Equation 2.1 colored in blue:

$$\min_G \max_D \mathcal{L}(G, D) = \mathbb{E}_x[\log D(x, y)] + \mathbb{E}_z[\log(1 - D(G(z, y), y))].$$

Odena et al. [2017] proposed a different approach to condition a GAN into generating a particular class of content. In the **Auxiliary Classifier GAN (ACGAN)**, the generator also receives some extra information y (e.g., a class label) that it should use to drive the synthesis towards a desired direction. However, instead providing the class label in tandem to the discriminator as in the cGAN, the authors propose assigning the task of finding the class y of samples x by estimating $P(y|x)$ through a second output branch. Hence, the discriminator receives only the generated $\hat{x}$ or a real image x and outputs both the discrimination between real and fake through a branch D_{adv} and a probability distribution over the class labels of the input through D_{cls} . In that scenario, the minimax value function becomes (with differences to Equation 2.1 highlighted):

$$\begin{aligned} \min_G \max_D \mathcal{L}(G, D) = & \mathbb{E}_x[\log D_{\text{adv}}(x)] + \mathbb{E}_z[\log(1 - D_{\text{adv}}(G(z, y)))] \\ & + \mathbb{E}_x[\log D_{\text{cls}}(y|x)] + \mathbb{E}_z[\log D_{\text{cls}}(y|G(z, y))]. \end{aligned}$$

²The left term in $\mathcal{L}_D$ does not contain the parameters of G , so even if it was present in $\mathcal{L}_G$, the gradient of the error for this part would be zero as it is calculated with regards to G 's weights only (and the derivative of a constant term is always zero).

Another challenge when defining a GAN is the amount of different decisions to make and configurations to choose from for the involved networks. Depending on how they are assembled, training can converge or not. In that regard, Radford et al. [2015] investigated different architectural choices for image generators that use convolutional layers and presented which yielded converging training results in most settings. They called the set of constraints the **Deep Convolutional GAN (DCGAN)**.

The proposed constraints consist of four guidelines. First, the authors suggest replacing the network pooling layers with regular or transpose convolutions, as these allow learning customized down/upsampling functions (and pooling does not involve any learnable parameters). Second, using batch normalization [Ioffe and Szegedy, 2015] in both G and D to normalize the input to each hidden unit to have zero mean and unit variance, which helps the flow of gradients through the network and avoids the vanishing gradients problem. The third is to remove fully connected layers in favor of convolutional, which makes the learned patterns shift-invariant and contain much less parameters, as the convolution kernels are reused throughout the whole input [LeCun et al., 1998]. And the fourth guideline suggests using Rectified Linear Unit (ReLU) activations [Nair and Hinton, 2010] in the generator’s layers with tanh in the last one, and Leaky ReLU [Maas et al., 2013] throughout the discriminator, except for the last.

An important challenge while training GANs is to keep the stability of the game between generator and discriminator. If a player starts to win too frequently, the other might not receive useful error gradients, and that can halt learning. Training stability was reported as a challenge in the original implementation, with the authors suggesting the possibility of using different objectives than the proposed minimax value function. For instance, early in training, the discriminator may easily distinguish between real and fake images, and that can lead to the generator receiving poor gradients, as $\log(1 - D(G(z)))$ (from Equation 2.1) saturates. To help preventing that effect, they propose using a non-saturating version in the generator:

$$\mathcal{L}_G = -\mathbb{E}_z[\log(D(G(z)))]. \quad (2.2)$$

Albeit helpful, the non-saturating loss does not solve the training stability challenge. Arjovsky et al. [2017] tackled the issue by replacing the minimax value function altogether by the Earth-Mover distance, which is a measure of separation between distributions (e.g., real from fake images). Renamed to *critic* in their Wasserstein GAN (WGAN), the discriminator outputs not a probability of an image being real, but rather an unbounded number $D(x) \in \mathbb{R}$ that is a “realness score”: it tries to assign the furthest numbers for real and fake images, increasing the distance between their distributions.

Hence, the loss becomes:

$$\mathcal{L}_D = \mathbb{E}_x[D(x)] - \mathbb{E}_z[D(G(z))], \quad (2.3)$$

$$\mathcal{L}_G = -\mathbb{E}_z[D(G(z))]. \quad (2.4)$$

One of the benefits of this loss is that it is continuous and differentiable everywhere, allowing the critic to train until optimality without hurting the gradient flow to the generator. In addition, the authors found that the use of the Wasserstein loss makes training less susceptible to the networks' topology and choice of hyperparameters. In turn, it is necessary for the critic to train more frequently (e.g., 5 times) than the generator and to enforce it to be 1-Lipschitz compliant³, which is done by clipping the weights of the critic to some small interval $[-c, c]$, $c \in \mathbb{R}$. Even though clamping to critic's weights to a small range works, it unnecessarily reduces the capacity of the network.

To overcome that limitation, a follow up work proposed the Wasserstein GAN with Gradient Penalty (**WGAN-GP**) [Gulrajani et al., 2017], which replaces weight clipping by an additional loss term to the critic that penalizes large gradients. It works by encouraging the norm of the gradient critic to go towards 1 through the following loss function (changes to Equation 2.3 highlighted):

$$\mathcal{L}_D = \mathbb{E}_x[D(x)] - \mathbb{E}_z[D(G(z))] + \lambda \mathbb{E}_{\dot{x}}[(\|\nabla_{\dot{x}} D(\dot{x})\|_2 - 1)^2], \quad (2.5)$$

where λ is a scaling hyperparameter for the gradient penalty term, and $\dot{x}$ is a random convex combination of real x and fake $\hat{x}$ samples.

Also tackling the training stability of GANs, Mao et al. [2017] proposed the Least Squares GAN (**LSGAN**) to avoid the vanishing gradients problem for the samples that lie on the correct side of the decision boundary between real and fake images. It replaces the minimax value function for the discriminator with a mean squared error, becoming:

$$\mathcal{L}_D = \mathbb{E}_x[(D(x) - 1)^2] + \mathbb{E}_z[(D(G(z)) - 0)^2], \quad (2.6)$$

$$\mathcal{L}_G = \mathbb{E}_z[(D(G(z)) - 1)^2].$$

Even though the presented alternative objective functions help stabilize the training of GANs, there is not a single solution to all scenarios and care must be taken into choosing the best option for each task.

In this work, we propose different GAN models to generate pixel art character sprites. In particular, we condition the image generation of a character in one or more images of it in source directions. That can be framed as either an image-to-image translation or a missing-data imputation problem.

³A function is considered 1-Lipschitz iff the magnitude of its gradient is at most 1 everywhere.

The next section defines the image-to-image translation and missing-data imputation problems and presents some of the deep generative architectures using GANs that have been proposed to address them.

2.2 Image-to-Image Translation and Missing-Data Imputation

Pang et al. [2022] define image-to-image translation as the process of converting an input image x_a in a source domain a to a target b while keeping some intrinsic content from a and transferring it to the extrinsic style of b . The meaning of a domain, style and content differ according to the task. To illustrate, if we want to create a cartoon version (domain b) from pictures of faces (domain a), we are translating faces x_a to b , keeping the person’s identity (intrinsic content) but using cartoonish techniques (extrinsic style). Different problems have been approached as image-to-image translation using deep generative models (GANs, VAEs etc.), such as image colorization [Jiang and Sweetser, 2022; Gonzalez et al., 2020], color transfer [Afifi et al., 2021], semantic image synthesis [Isola et al., 2017; Serpa and Rodrigues, 2019; Serpa and Rodrigues, 2022], style transfer [Zhu et al., 2017b], attribute manipulation [Choi et al., 2018; Choi et al., 2020; Lee et al., 2019], and pose translation [Hong et al., 2019; Choi et al., 2022; Loftsdottir and Guzdial, 2022].

The diversity of the presented problems involves different characteristics of the task and the proposed solution. A first important property is the use of **supervision** (label/annotated examples) for training, which largely depends on the availability of such data. For example, in a translation from grayscale to colored pictures, it is easy to have pixel-wise aligned examples; but that is not the case if we want to transform horses into zebras, as the cost of acquiring completely registered pairs of photos of horses and zebras in the same position in the same environment is impractical. Hence, when paired data is available to some task, it can use **supervised** training [Isola et al., 2017; Lee et al., 2019], whereas when it is not, the algorithm needs to train in an **unsupervised** manner [Zhu et al., 2017b; Choi et al., 2018; Afifi et al., 2021].

A second characteristic of the tasks is the **number of domains** involved in the translation, and how the proposed architecture can deal with them. For instance, many problems consist of only two domains (e.g., grayscale to color, photo to painting, semantic labels to photographs), whereas others involve multiple (e.g., translating a neutral face to one smiling, angry, or crying). Hence, the proposed architectures can be **two-domain**

[Isola et al., 2017; Zhu et al., 2017b] or **multi-domain**, supporting the translation among all directions [Choi et al., 2018; Choi et al., 2020; Lee et al., 2019]. Additionally, the architectures for two-domain translation can generate images in a single direction [Isola et al., 2017] or in both [Zhu et al., 2017b; Zhu et al., 2017a].

A third relevant feature of the problems is the diversity of samples generated for the same input [Pang et al., 2022]. Some models yield a **single-modal output**, always translating an input to the same output image [Isola et al., 2017; Zhu et al., 2017b; Choi et al., 2018; Lee et al., 2019]. However, translation between domains is an intrinsically ambiguous task in most cases, hence benefiting from the creation of multiple outputs for the same input. Taking that into consideration, other models can yield **multi-modal outputs**, as they are able to capture a distribution of potential images in the target domain for the same input [Zhu et al., 2017a; Huang et al., 2018; Choi et al., 2020].

Next, we briefly introduce some GAN architectures that perform image-to-image translation or missing-data imputation, which influenced the models we propose in Chapter 4 (Model Proposition). Compared to the unconditional GANs presented in Section 2.1 (Generative Adversarial Networks), the models we present hereon take one or more images as input, instead of a noise vector.

A seminal work in the recent approach of using deep generative models for the image-to-image translation problem is **Pix2Pix** [Isola et al., 2017]. It is a general purpose, conditional architecture for translating images x_a from one domain a to another domain b (unidirectionally) using paired examples (supervised training), i.e., $x = \langle x_a, x_b \rangle$. The generator $G_{a \rightarrow b}$ follows a U-net topology [Ronneberger et al., 2015], as it receives and produces images with the same resolution, but in a different domain: $\hat{x}_b = G_{a \rightarrow b}(x_a)$. The discriminator follows a PatchGAN format [Isola et al., 2017], as instead of providing a single real/fake logit, it yields one classification per rectangular patch (of some size). Both networks follow the strategies outlined by the DCGAN [Radford et al., 2015] to improve the training stability. During training, the generator is penalized by creating images that are pixel-wise L_1 -distant to the target images of the paired examples, as well as adversarially by the discriminator, using the non-saturating loss (Equation 2.2). The architecture has been used in different tasks, such as translating photographs from day to night, grayscale to colored images, and labeled to realistic facades.

Tackling a similar task to Pix2Pix, but in the much more difficult setting where paired examples are unavailable, Zhu et al. [2017b] propose **CycleGAN**, an architecture for unsupervised image-to-image translation between two domains, bidirectionally. To be able to translate from domain a to b and b to a , it trains two generators $G_{a \rightarrow b}$ and $G_{b \rightarrow a}$, each capable of translating images in one direction. Similarly, two discriminators, D_a and D_b are trained to distinguish between real and fake examples from each domain. As paired images are not available, the authors introduce a cyclic loss concept: during training, an image x_a is first translated in one direction, yielding $\hat{x}_b = G_{a \rightarrow b}(x_a)$, then translated back

to the original domain, producing $\tilde{x}_a = G_{b \rightarrow a}(\hat{x}_b)$; then, the generators are penalized by the pixel-wise L_1 distance between the cyclically generated $\tilde{x}_a$ and the original images x_a . The discriminators are also used as adversarial losses for the generators using the [LSGAN](#) function (Equation 2.6).

When a task contains more than two domains, the architectures proposed for two-domains translation require training multiple models, one (or two) for each direction pair. Additionally, there is much information that could be potentially reused between those models. With that in mind, [Choi et al. \[2018\]](#) proposed **StarGAN**, a multi-domain image-to-image translation architecture that uses a single generator and discriminator pair. The generator G receives the input image x_s in the source domain, as well as a label defining the target domain t of the translation, yielding $\hat{x}_t = G(x_s, t)$. The model trains without supervision, and the generator uses a cyclic loss such as in CycleGAN. In turn, the discriminator tries to distinguish between real and fake images (using [WGAN-GP](#), Equation 2.5) and, at the same time, to classify what is the domain of its input image, such as in [ACGAN \[Odena et al., 2017\]](#). The latter part works as a classification loss for the generator, steering it towards creating images that could belong to the *target* domain.

Back to the scenario where paired images are available, in the multi-domain setting, when the image of only a single domain is absent, the task is called missing image data imputation. [Lee et al. \[2019\]](#) approached that problem by proposing the Collaborative GAN (**CollaGAN**). The authors argue that a single generator model (with a discriminator counterpart) that can see the images from all the S available domains could reduce ambiguity and produce more reusable mappings among them. The generator G takes in a tuple x_S with the available images $x_s, \forall s \in S$ and zeros 0 for the missing one, and a label representing the missing domain t , yielding $\hat{x}_t = G(x_S, t)$. The objective function uses a different version of the cyclic loss, called multiple cycle consistency, calculated by generating an image $\hat{x}_t$ in the forward direction (i.e., domain t), and then generating images $\{\tilde{x}_s, \forall s \in S\}$ backwards for each of the source domains while replacing the original x_t with the one generated in the forward step, $\hat{x}_t$. The discriminator uses [LSGAN](#) to train adversarially (Equation 2.6).

Extending the idea of missing-data imputation to handle arbitrary missing subsets, [Shen et al. \[2021\]](#) introduced Representational disentanglement scheme for Multi-domain Image Completion (**ReMIC**). In this architecture, the available images $x_s, \forall s \in S$ are first decomposed into a shared *content* code c and domain-specific *style* codes s_i respectively through a unified content encoder E^{content} and style encoders E_i^{style} for each domain $i \in S$. The decoder G_i for each missing domain $i \in T$ then reconstructs the missing images $\hat{x}_i = G_i(s_i, c), \forall i \in T$ by combining the content representation with the corresponding style embedding, yielding domain-consistent but semantically aligned images. To regularize the disentanglement, the architecture introduces latent cycle-consistency losses that require the re-encoded content and style of generated images to match the originals, alongside

adversarial training with discriminators to ensure realism (using [LSGAN](#), Equation 2.6). This disentangled formulation allows ReMIC to flexibly complete any subset of missing domains while preserving cross-domain consistency.

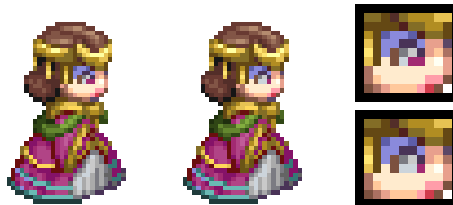
We present, in Chapter 4 (Model Proposition), a more detailed explanation of Pix2Pix, StarGAN, CollaGAN, and ReMIC along with their network topology, training objectives and procedures, as they are used as the base for the first four models we propose for the task of generating pixel art character sprites facing different directions. In that chapter we also present SpriteGAN, an architecture proposed from the ground up for the tasks we address in this research. In the next section, we introduce some of the characteristics of the pixel art style that pose specific challenges to their creation using deep generative models.

2.3 Pixel Art Characterization

Synthesizing photorealistic images using deep generative models requires solving challenging problems, such as achieving crisp images with anti-aliased edges, correct shape and texture with feasible lighting, and so on [[Karras et al., 2018](#)]. At first glance, generating pixel art seems to be a simpler objective, but here we show some of its characteristics that introduce new and specific challenges.

Because of the typical lower spatial resolution, pixel art images usually encode much more information per pixel than photos [[Kopf and Lischinski, 2011](#)]. To illustrate, Figure 2.2 depicts a sprite in which a single pixel can change the expression of a character. This characteristic makes noise and shape deviation much more harmful to the perceived quality of the generated images than in the case of photos.

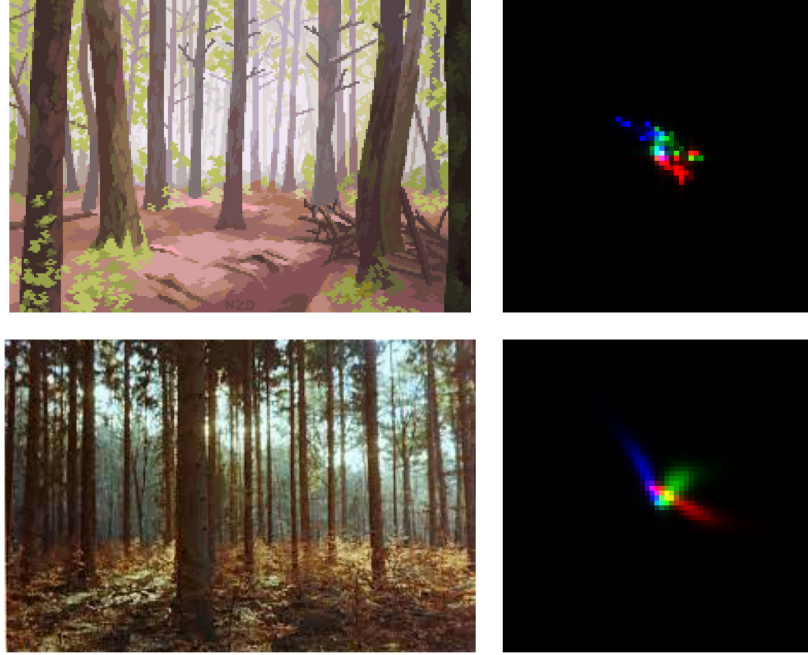
Figure 2.2: How the value of a single pixel changes the character’s facial expression. A change in a single pixel results in a happier expression.



Due to historical reasons of hardware limitations, artists create scenery and character sprites using a limited number of colors that constitute the palette of that piece of art. When generating such images using deep generative models, especially in the case of small datasets, the resulting images will likely include many more colors due to statisti-

cal uncertainty, extrapolating the intended palette. To illustrate the difference between photorealistic and pixel art images, Figure 2.3 shows the color histograms from a pixel art image and a photo of a similar landscape. We note that the pixel art histogram has a much more discrete color distribution, compared to the photo.

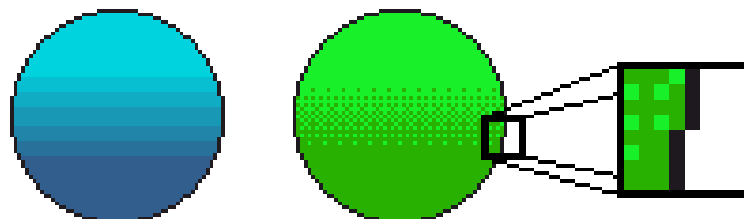
Figure 2.3: **Top.** Landscape of a forest in pixel art with its histogram in the RGB-uv space. **Bottom.** Photo and histogram of a similar landscape.



Regarding the form, pixel art is a stylistic, low-fidelity representation of whatever it depicts [Silber, 2015]. That might hinder the performance of reusing learned weights from non-pixel art datasets. In addition, different shading and texturing patterns are typically used by artists, such as color ramps (gradients) and dithering (color mixing – Figure 2.4), and those will be likely absent from such datasets.

The reduced size also impacts the frequency distribution throughout the sprite. Real pictures typically have large regions with texture bounded by shape edges, but pixel art presents both patterns interchangeably and very close to each other. For example,

Figure 2.4: **Left.** Shading with color ramps. **Center.** Shading with dithering. **Right.** Zooming in the intentional crisp edges.



a person’s face in a picture might have a large region with similar color (low frequency) for its cheek, which can be bounded on one side by the face edge (high frequency). In turn, a pixel art character, typically a lot smaller, uses much fewer pixels for its cheek, interleaving low and high frequency much more often.

Another more circumstantial characteristic that presents challenges to the generation of pixel art is the scarcity of open and large datasets, especially in comparison to the ones used in Computer Vision tasks. Table 2.1 presents the sizes of some popular datasets with which generative models are trained with. In contrast with the ones used in some related work that proposed pixel art generation models, their sizes are 1-4 orders of magnitude higher. Adding to that problem, some data augmentation techniques are inappropriate, such as rotation, since no matter whether it is used with or without interpolation, the operation results in unpleasant artifacts in the case of pixel art.

Table 2.1: Sizes and augmentation techniques of datasets used in Computer Vision tasks.

Dataset	Size	Augmentation	Used by
CMP Facade	606	Cropping, Mirroring	Pix2Pix, CycleGAN
Horse to Zebra	≈ 1060	Cropping	CycleGAN
Cityscapes	3,475	Cropping, Mirroring	Pix2Pix, CycleGAN
RaFD	4,824	Mirroring	StarGAN, CollaGAN
Night2Day	20,120	Cropping, Mirroring	Pix2Pix
FFHQ	70,000	Mirroring	StyleGAN
CelebA	202,599	Mirroring	StarGAN
LSUN Bedrooms	$\approx 3,000,000$	None	StyleGAN, DCGAN

With regard to the peculiarities of the pixel art style, Chapter 6 (Hypotheses on Pixel Art Generation) presents our investigation of four hypotheses aimed at improving the quality of the generated images: (a) representing images as indices within their color palette, (b) incorporating a histogram loss during generator training, (c) applying a post-processing step to map each pixel to its closest color in the input palette, and (d) embedding a palette quantization operation within the generative model itself, thereby enabling end-to-end training to produce images that strictly adhere to the intended palette. The following chapter discusses work related to ours.

Chapter 3

Related Work

This chapter presents related works that used deep generative models to create procedural images in the context of games, using various architectures and tackling different tasks. We also discuss about how some works approached the particularities of the pixel art style.

3.1 Generation of Game Art through Deep Learning

Different recent works employ deep generative models to create art, in particular, two dimensional images for games. We are specially interested in pixel art character generation, however we also included in our survey research on the creation of landscapes and effects, as well as non-pixel art characters. Hereon, we analyze such works on the dimensions of (a) the type of generated content, (b) the approached task, (c) the specific strategies for pixel art generation used, if any, and (d) the image representation used. Table 3.1 summarizes the related work included in our survey.

Starting with the (a) **type of content being created**, most of the works we found procedurally generate *characters*, such as ours, with exceptions of the production of landscapes [Yang et al., 2022], visual effects [Kim et al., 2023], and texture maps [Vecchio et al., 2024; Nikolov, 2025]. Yang et al. [2022] proposed an architecture that trains without paired examples based on CycleGAN [Zhu et al., 2017b] and UNIT [Liu et al., 2017] to translate photos of landscapes into pixel art. The translated images contain less colors than the source photos, but still a lot more than intended to be considered pixel art. Also, they use brighter colors than the source images and the edges of objects are blurry. In turn, Kim et al. [2023] translated images of 2D visual effects to another effect that follows style of a target image from a predefined set during training (i.e., fire, smoke, water, and maple). They proposed an architecture based on DCGAN [Radford et al., 2015] and showed that the generated images can follow the target style, but omits texture details when the input shape is small.

Vecchio et al. [2024] introduced MatFuse, a generative model that can produce reflectance properties (i.e., diffuse, normal, roughness and specular maps) for arbitrary materials by being conditioned on a textual description, a palette, and a sketch image. The system consists of a compression network that is responsible to translate images to and from a latent space, and a diffusion model that learns the distribution of the latent feature vectors and conditions them to guide the generation process. Nikolov [2025] developed a feed-forward fully convolutional network that follows a U-net topology [Ronneberger et al., 2015] and generates a light map from a pixel art image and some light configuration data representing its type and position/direction.

Regarding the (b) **task tackled in each research**, they comprise unconditional generation of characters [Horsley and Liebana, 2017], style transfer of visual effects [Kim et al., 2023] or landscapes [Yang et al., 2022], colorization of grayscale characters [Jiang and Sweetser, 2022], shading and segmentation [Serpa and Rodrigues, 2019; Serpa and Rodrigues, 2022], conditional character generation [Saravanan and Guzdial, 2022; Gonzalez et al., 2020; Löwenström, 2024; Merino et al., 2023; Gallotta et al., 2025], pose interpolation [Loftsdottir and Guzdial, 2022], and pose translation [Reed et al., 2015; Hong et al., 2019; Choi et al., 2022; Hsieh et al., 2025].

Horsley and Liebana [2017] proposed an architecture based on DCGAN [Radford et al., 2015] to generate pixel art sprites of human-like characters, faces, and creatures. The proposed model is unconditional, as it generates images from noise and does not allow defining which type of content to generate. The resulting images resemble the training set but present a lot of noise even in background areas. The system also suffered from mode collapse, a common challenge in unconditional GANs in which most noise values map to the same resulting image.

All other works present an architecture that is conditional on a source image and optionally more content, such as a class label or a text caption. Hence, because they have images as input and output, they perform image-to-image translation. Yang et al. [2022] and Kim et al. [2023], already described, both trained without paired examples and resort to a cycle consistency loss to transfer the style of the source image into the target. As proposed by CycleGAN [Zhu et al., 2017b], during a training step, an image is translated to the target style, then the result is translated back to the original, and the latter image must be as close as possible to the original one, which is enforced by the cycle consistency loss. The work of Kuang et al. [2021] is closely related to that of Yang et al. [2022], but it focused on translating characters from a cartoon style to pixel art and introduces modifications to the base CycleGAN architecture.

Using paired examples, Jiang and Sweetser [2022] approached the colorization of grayscale images of pixel art characters by proposing an architecture based on Pix2Pix [Isola et al., 2017]. The resulting colored images sometimes display colors that approach the ground truth, but usually contain many different hues to characters that should have

only a few. In some situations, however, the produced images have more desaturated colors than intended. The authors found that representing images as YUV instead of RGB yielded images with colors closer to the ground truth.

[Serpa and Rodrigues \[2019\]](#) also proposed an architecture based on Pix2pix [\[Isola et al., 2017\]](#) but in a task to shade and to segment body parts of higher-resolution (256×256) pixel art characters. It generates a grayscale-shaded sprite and another one that segments characters' body parts from rough line art sketches of animation frames from a fighting game. The generated grayscale sprites were close to the ground truth, but the colored ones diverged, especially for characters in less common poses. In a later iteration of the work [\[Serpa and Rodrigues, 2022\]](#), the authors got improved results by framing the problem as a semantic segmentation task and changing the architecture accordingly. The proposed model dropped the adversarial training and employed dense connections to increase the network's depth, deep supervision to provide gradients to every layer block, and a class-weighted focal loss to overcome the class imbalance in the training data.

Tackling the generation of characters conditioned on another content in addition to the source image, [Gonzalez et al. \[2020\]](#) proposed a Convolutional Variational Autoencoder (CVAE) to transfer the domain of a character from a source type (e.g., fire-type Pokémon) to a target type (e.g., grass). However, the reconstruction of the original images yielded blurry results with shapes faintly resembling the original but with noisy colors. In addition, the domain transfer task (type swap) had considerable shape degradation, usually with high-frequency noise and dangling pixels outside the intended character shape. Due to the lack of examples in the dataset (974 before augmentation), the authors experimented with pretraining in another dataset and then fine-tuning the model, which yielded better results.

[Saravanan and Guzdial \[2022\]](#) also conditioned the generation of new pixel art Pokémon characters based on a source character and two target types (e.g., water and electrical Pokémon types). In their work, they proposed an adaptation to the Vector-Quantized Variational Autoencoder (VQ-VAE) to the task of first learning a representation of the Pokémon sprites. In a second stage, they trained an auto-regressive model to create new characters using the learned representations. The images that are produced sometimes resemble real creatures, but most of the time, they cannot be considered characters, as they look more like contiguous blobs of colored pixels.

Both [Merino et al. \[2023\]](#) and [Gallotta et al. \[2025\]](#) proposed generative models conditioned on text to produce character images that follow some description. While the former introduces a small and specialized feed-forward network model that generates 8×8 characters and 16×16 emoji, the latter is based on pre-trained latent diffusion models. In particular, [Gallotta et al. \[2025\]](#) evaluated the effects on the image quality of conditioning a character generation process on different additional contexts (e.g., background image description, dominant background colors, no context, etc.) besides the prompt describing

the character. It uses a pre-trained Stable Diffusion model [Rombach et al., 2022] and two Low-Rank Adaptations (LoRAs)[Hu et al., 2021] to produce images in a desired art style. The authors found that the added conditioning information did change the results but no approach stood out from the others.

Also integrating text and image, Löwenström [2024] proposed using latent diffusion models [Rombach et al., 2022] with text-image similarity models [Radford et al., 2022] to allow editing or generating new images of pixel art characters from a text caption. The author shows that using text to condition the image generation increases the quality, but the resulting images can have missing or merged body parts. In addition, they contain a high number of colors and blurred edges.

Other tasks approached in the literature are pose interpolation and translation. Targeting the generation of in-between frames of animated sketches, Loftsdottir and Guzdial [2022] proposed SketchBetween: a model that takes the initial and final frames of a character animation and sketches of the internal frames, and generates colored versions of the frames in the middle. It takes five images of an incomplete animation as input and provides five images with the rendered sprite animation. Trained on a dataset of animated cartoon animals, it had promising results on shapes and poses similar to the ones from the training set. However, even the higher-quality examples presented the blurriness typical of how VAEs optimize to reduce the average reconstruction error.

Approaching the translation of poses in which an image of a character in a target pose is generated from the image in a source one, Reed et al. [2015] proposed a deep learning model that can solve visual analogies in the images it trains upon. For instance, given the relation between images of a knight character holding a sword still and swinging it, and an image of a mage with a scepter, the network can understand the relationship between the knight images and produce an image of the mage swinging the scepter. As useful as the ability to make such analogies are, the authors do not demonstrate that the model can generalize to unseen characters or character poses. Additionally, the model trained with a completely modular dataset, in which every character is formed by assembling individual body parts (e.g., head, arms, body, accessories, weapons) together, which represents a much easier task than with other datasets with much more variability.

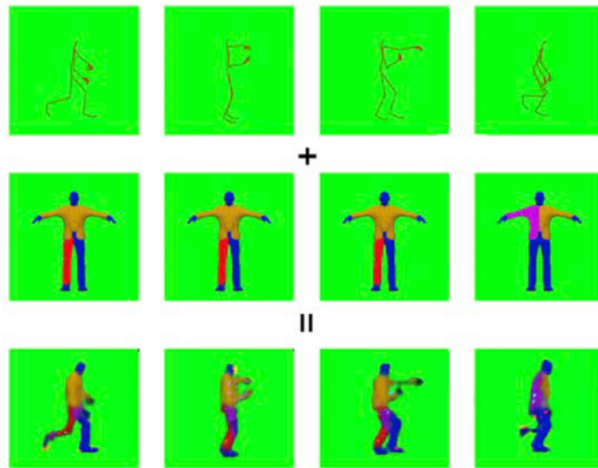
Also tackling pose translation are the works from Hong et al. [2019], Choi et al. [2022], and Hsieh et al. [2025]. Hong et al. [2019] proposed a multiple discriminators GAN (MDGAN) that translates non-pixel art character images from a source pose into a target. The generator receives two images, one representing a shape and color sprite and another comprising a target pose (bone-graph sprite). The system is trained with two discriminators, one responsible for determining whether two images have the same color and shape, and the other deciding if the pose of a character sprite is correct compared to some target bone-graph. An important limitation of the proposed work is that the model requires all characters to have basically the same shape, varying only their colors.

Figure 3.1 presents some of their results.

Similarly, [Choi et al. \[2022\]](#) created a database of character sprites in walking and running animations by feeding video frames of real people into body segmentation networks. Then, they trained a model to generate characters from the body-segmented sprites in arbitrary poses created by users. Albeit successful in their proposed experiments, both systems require tailored datasets that match the positions of characters, making it challenging for the models to generalize, especially for games with different character shapes and movements. In addition, both works use real images in their training sets, which do not conform to characters in typical 2D games, especially in the pixel art style.

Leveraging pre-trained models, [Hsieh et al. \[2025\]](#) proposed a system to create animation frames for character sprites from a single reference image in an initial pose and a sequence of skeleton images depicting some movement. The authors fine-tuned the Animate Anyone architecture [[Hu et al., 2024](#)] with a character sprite dataset that includes associated skeleton images for a few action sequences. It uses the pre-trained weights from Stable Diffusion [[Rombach et al., 2022](#)] and consists of three modules: the network ReferenceNet responsible for encoding the visual appearance of the reference image, the PoseGuider network, which encodes the desired movements from the skeleton images, and the Denoising U-net, which uses Temporal layers to encode time-related relationships and produce the final images of the reference character performing the desired motion. The resulting animation frames generally follow the intended poses, with better results for characters that are present in the training set. Due to the use of many large networks and the diffusion process at inference, the memory and time requirements are high—42 GB of Video Random Access Memory (VRAM) and a few minutes to generate the images.

Figure 3.1: Sample results from MDGAN [[Hong et al., 2019](#)] showing the bone-graph images in the top row, the shape and color sprites in the middle and the translated characters at the bottom.



In this dissertation, we also address the task of translating character poses, but specifically for characters rendered in the pixel art style. Unlike the approaches of [Hong](#)

et al. [2019] and Choi et al. [2022], our method does not require all characters to share the exact same shape and differ only in texture. Such a constraint would severely limit artistic variability and result in highly repetitive characters. In comparison to Hsieh et al. [2025], which relies on pre-trained weights from large, general-purpose latent diffusion models, our proposed architectures operate with modest VRAM requirements and can generate images within a fraction of a second.

Furthermore, in contrast to Reed et al. [2015], who also focused on generating characters in the pixel art style, our approach is designed to function beyond datasets composed solely of highly modular characters—that is, characters assembled from pre-designed body parts. Such modularity reduces visual diversity, and if a dataset already supports assembling characters in this manner, pose translation becomes largely unnecessary. Consequently, our work targets the more general problem of generating characters across diverse artistic styles, without imposing constraints that would limit their variability, and using fewer computational resources when compared to those necessary for large and general-purpose generative models.

Most of the presented works produce pixel art images, but only a few introduced (c) **specific strategies** to drive the generation process into creating images that conform to the style. Gonzalez et al. [2020] found that pre-training their CVAE with non-pixel art images and then fine-tuning the model with the pixel art improved their results. Even so, the generated images contained a lot of high-frequency noise and blurred edges.

In turn, Saravanan and Guzdial [2022] improved the quality of the generated pixel art characters by adding a 1×1 convolution layer pair at the beginning and end of the encoder and decoder networks of their VQ-VAE model. Such operation replaces each spatial value in the input or activation maps by a linear interpolation across all channels. The proposed changes improved the quality of the image reconstruction and also reduced the blurriness, but the generated images of new characters still contained more colors than intended.

In this dissertation, we also focus on the generation of pixel art, and as an attempt to improve the quality of the generated images, we investigate different hypotheses: representing the pixels of images by indices in their palette, adding a histogram loss term to penalize the generation of images with a different histogram than intended, and quantizing the pixels of the generated image to the palette of the source image.

Lastly, in terms of the (d) **image representation**, most works use RGB to describe the images, with exceptions to YUV [Jiang and Sweetser, 2022], HSV [Gonzalez et al., 2020] and as indices in a palette [Merino et al., 2023]. On the colorization of grayscale creatures, Jiang and Sweetser [2022] conducted experiments showing that representing images in YUV yielded better-colored images than when represented with RGB. Such improvement is due to the fact that when working in YUV, the Y component represents exactly the grayscale version of the character, which simplified the task of the network

to estimate only the U and V channels. Even though [Gonzalez et al. \[2020\]](#) represented images using HSV, they did not present an evaluation of the image representation, neither did they justify their decision.

In turn, [Merino et al. \[2023\]](#) proposed a very small feed-forward network that generates 8x8 characters and 16x16 emoji. It represents images as indices in a global palette with 16 colors. The generation is conditioned on text embeddings that describe the images and use a pre-trained sentence transformer to extract text embeddings from image captions. The dataset sizes were 100 and 663 for the characters and emoji. The model overfit on the characters dataset, producing bad images for unseen prompts, but was still able to learn a latent space that enabled some vector arithmetic regarding colors.

In our work, we opted to use RGBA, which consists of the usual RGB components plus an alpha (A) channel that defines the transparency of pixels. Such color space was used because the datasets we used already contained the alpha channel and through an evaluation we observed that using RGBA led to better results. The approach of representing images as indices in a palette [[Merino et al., 2023](#)] was also one of our evaluated hypotheses.

Table 3.1 displays a summary of the related work we found, organized according to their characteristics. In the next chapter we present all the models investigated in this dissertation for the task of generating pixel art characters in different poses.

Table 3.1: Summary of the characteristics of the related work that generated game art.

Reference (chronolog. order)	Task	Pixel Art	Content	Repres.
Reed et al. [2015]	Generation via analogy	Yes	Character	RGB
Horsley and Liebana [2017]	Unconditional generation	Yes	Character	RGB
Hong et al. [2019]	Pose translation	No	Character	RGB
Serpa and Rodrigues [2019]	Shading and segmentation	Yes	Character	RGB
Gonzalez et al. [2020]	Conditional generation	Yes	Character	HSV
Kuang et al. [2021]	Style transfer	Yes	Character	RGB
Jiang and Sweetser [2022]	Colorization	Yes	Character	YUV
Serpa and Rodrigues [2022]	Shading and segmentation	Yes	Character	RGB
Yang et al. [2022]	Style transfer	Yes	Landscape	RGB
Choi et al. [2022]	Pose translation	No	Character	RGB
Loftsdottir and Guzdial [2022]	Pose interpolation	No	Character	RGB
Saravanan and Guzdial [2022]	Conditional generation	Yes	Character	RGB
Kim et al. [2023]	Style transfer	No	Effects	RGB
Merino et al. [2023]	Conditional generation	Yes	Character	Palette
Löwenström [2024]	Conditional generation	Yes	Character	RGB
Vecchio et al. [2024]	Conditional generation	No	Tex. Maps	RGB
Nikolov [2025]	Conditional generation	Yes	Light maps	RGB
Gallotta et al. [2025]	Conditional generation	No	Character	RGB
Hsieh et al. [2025]	Pose translation	No	Character	RGB
Ours	Pose translation/imputation	Yes	Character	RGBA

Chapter 4

Model Proposition

This chapter describes the development of the models we propose for generating images of characters in different poses, given one or more source-view images. We begin by formally defining the problem and explaining how it can be addressed using GANs.

We then present the four models we derived from existing architectures, detailing the modifications introduced to each. These models are discussed in the order in which they were proposed, with later architectures exhibiting increasing complexity. They are based on Pix2Pix [Isola et al., 2017], StarGAN [Choi et al., 2018], CollaGAN [Lee et al., 2019], and ReMIC [Shen et al., 2021]. In addition, we introduce SpriteGAN, a novel architecture designed from the ground up for the specific task of generating pixel art characters in multiple poses.

4.1 Problem Statement

We aim to build generative models capable of generating images of characters facing different directions, given one or more images of them in other poses. This task can be modeled as an image-to-image translation or missing-data imputation problem, depending on the architecture and the number of input domains it can work with, and we formally describe it as follows.

There are $N = \{a, b, c, d\}$ domains and each corresponds to a pose that a character may appear in (back, left, front, and right). A dataset $\mathbb{D}$ consists of tuples of paired examples $x_N = (x_a, x_b, x_c, x_d)$, where each image $x_i, \forall i \in N$ depicts the character in domain i .

We denote by $S \subset N$ the set of available (source) domains and by $T \subseteq N$ the set of target domains to be generated. While some models have a single input domain (Pix2Pix and StarGAN-based) and, hence, receive a single input image x_s , others receive multiple images. In the latter case, the ordered tuple of images used by a generator is

represented as x_S such that

$$x_S = (x_i : i \in N) \quad \text{where } x_i = \begin{cases} x_i, & i \in S, \\ 0, & \text{otherwise,} \end{cases} \quad (4.1)$$

and 0 denotes a zero tensor with the same dimensions as an image. This representation keeps the order of all domains while indicating missing inputs.

We group the architectures into two categories according to their input–output characteristics:

(1) Image-to-image translation models (e.g., Pix2Pix and StarGAN-based) take a single image as input and produce a single image in a target domain. They are generally described as

$$\hat{x}_t = G_{1 \rightarrow 1}(x_s, t), \quad s, t \in N,$$

where t may be fixed (Pix2Pix) or specified as a conditioning label (StarGAN).

(2) Missing-data imputation models (e.g., based on CollaGAN, ReMIC, and SpriteGAN) receive multiple images, one per available domain, and generate one or more images that belong to the target domain(s), depending on the architecture. These models operate on the tuple x_S (defined in Equation 4.1) and are described as

$$\hat{x}_t = G_{\text{many} \rightarrow 1}(x_S, t) \quad \text{or} \quad \hat{x}_T = G_{\text{many} \rightarrow \text{many}}(x_S, \psi),$$

where ψ denotes additional conditioning variables (such as random noise, inpaint masks, or palettes). The CollaGAN-based model generates one target image $\hat{x}_t$ given multiple inputs, the ReMIC-based one produces all missing-domain images $\hat{x}_{N \setminus S}$, and SpriteGAN outputs a tuple $\hat{x}_N$ containing images for all domains, including those already available, which is useful as the model performs inpainting.

These general equations provide the basis for the specific formulations of each architecture, which are introduced later in this chapter.

Next, we formally describe how we approach the posed problem with GANs.

4.1.1 Generative Adversarial Networks Approach

As introduced in Section 2.1, GANs train by reaching an equilibrium state between two competing players, the generator and the discriminator, in a zero-sum game. The generator aims to produce images $\hat{x}$ that follow the same distribution as the real data (i.e., $\hat{x} \sim p_{\text{data}}(x)$), such that the discriminator cannot distinguish them from real samples.

Conversely, the discriminator seeks to correctly identify whether an input image originates from the true data distribution or is generated.

Both players are modeled as neural networks whose parameters are optimized jointly according to an objective function $\mathcal{L}$:

$$\min_G \max_D \mathcal{L}(G, D),$$

which varies across the models proposed in this work but consistently includes two main components: an adversarial term $\mathcal{L}_{\text{adv}}$ and a reconstruction term $\mathcal{L}_{\text{rec}}$, typically defined as the mean absolute error (L_1 distance) between generated and ground-truth images.

While in the original GAN formulation [Goodfellow et al., 2014] the generator and discriminator share the same objective, in image-to-image translation and missing-data imputation settings the networks are often optimized with distinct loss functions. We therefore denote by $\mathcal{L}_G$ and $\mathcal{L}_D$ the generator and discriminator losses, respectively.

In some architectures, such as those based on StarGAN and CollaGAN, the target domain t is explicitly provided as a conditioning variable for the generator. In these cases, the discriminator is also tasked with predicting the domain label of its input, guiding the generator to synthesize images consistent with the desired target domain. This introduces an additional domain classification term, $\mathcal{L}_{\text{dmn}}$, which penalizes incorrect domain predictions.

The following section presents the models developed as part of this research.

4.2 Model Architectures

To investigate the problem of generating pixel art characters in different poses, we developed five models: four based on existing image-to-image translation or missing-data imputation frameworks, and one novel architecture designed specifically for our task. The models, presented in order of increasing complexity, are based on Pix2Pix [Isola et al., 2017], StarGAN [Choi et al., 2018], CollaGAN [Lee et al., 2019], and ReMIC [Shen et al., 2021]. The final model, SpriteGAN, was conceived from the ground up for pixel art generation.

Although SpriteGAN has not yet achieved state-of-the-art performance in our experiments, we include it in this dissertation because it introduces novel capabilities absent in the other architectures, such as generating multiple outputs for a given input and performing region inpainting. We believe these features make it a promising direction for future research.

The following subsections describe each architecture, outlining their original formulations and detailing the modifications we introduced. For SpriteGAN, we additionally discuss a few architectures that inspired its design. We also summarize the experiments conducted to select the best model configurations, including hyperparameter choices and network architecture adjustments.

Figures 4.1, 4.3, 4.8, 4.11, and 4.13 illustrate the adopted generator and discriminator architectures. In each diagram, the sequential blocks represent the resulting activation maps after a series of operations. The accompanying legend indicates the type of operation (e.g., convolution, dropout, activation) and its configuration (e.g., stride, kernel size, dropout rate).

All generators receive and produce 64×64 pixel images with RGBA channels, whose values are rescaled to the $[-1, 1]$ range.

We implemented the code for all models in Python 3.11 using TensorFlow 2.16.2. The baseline architectures follow the original papers and, when available, official reference implementations.¹ We reimplemented all architectures to ensure consistency across models and to enable fair comparison under a unified codebase. Our source code is publicly available.²

4.2.1 Based on Pix2Pix

The Pix2Pix architecture [Isola et al., 2017] performs image-to-image translation between two fixed domains in a single direction. Accordingly, each of our trained models generates character images from a source pose s to a target pose t . Formally, the generator G receives an image x_s from the source domain and produces a corresponding image $\hat{x}_t$ in the target domain:

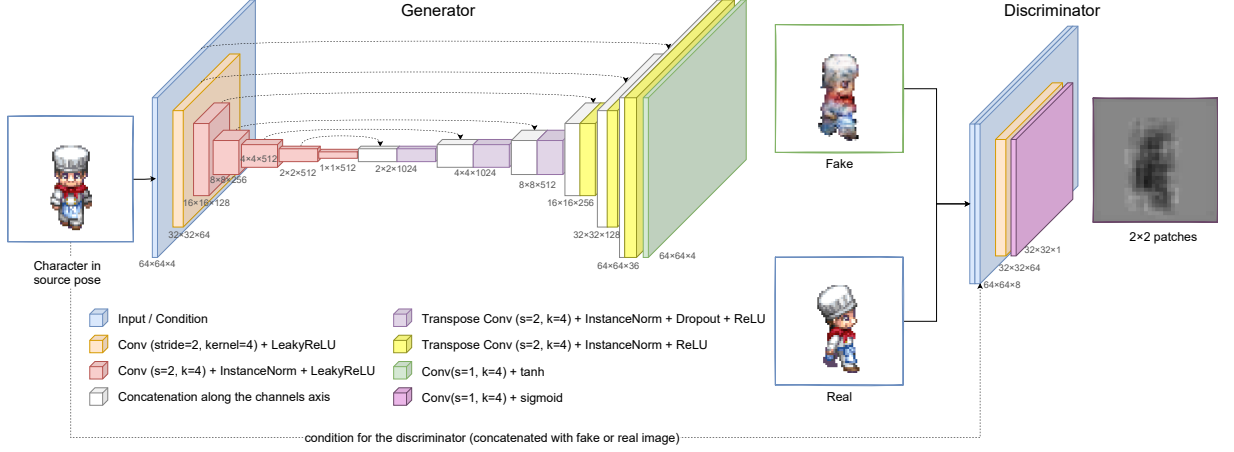
$$\hat{x}_t = G_{s \rightarrow t}(x_s), \text{ with } s, t \in N \text{ and } s \neq t.$$

The original Pix2Pix generator follows a U-Net architecture [Ronneberger et al., 2015] that processes $128 \times 128 \times 3$ images. It progressively downsamples the spatial dimensions while increasing the number of channels, followed by a symmetric upsampling path. Skip connections link each downsampling layer to its corresponding upsampling layer, and the bottleneck layer has a size of $1 \times 1 \times 512$. In our implementation, since

¹Pix2Pix, StarGAN, and CollaGAN provide official implementations in TensorFlow or PyTorch. ReMIC does not; although we contacted the authors, they did not provide their code.

²Pix2Pix-based model: <https://github.com/fegemo/palette-and-histo-gan>. StarGAN-, CollaGAN-, ReMIC-based models, and SpriteGAN: <https://github.com/fegemo/multi-domain>.

Figure 4.1: Architecture of the proposed Pix2Pix-based model. **Left.** The generator receives an input image x_s of a character in a source pose and produces an image $\hat{x}_t = G_{s \rightarrow t}(x_s)$ of the same character in a target pose. **Right.** The discriminator receives either a real target image x_t or a generated image $\hat{x}_t$, along with the source image x_s as a condition, and outputs patch-wise real/fake scores for each 2×2 region.



input and output images are 64×64 , we use one fewer downsampling block to maintain proportional network depth.

The discriminator adopts the PatchGAN design, which classifies local patches of an image as real or fake rather than evaluating the entire image at once. It is a conditional discriminator, as it considers both the source image x_s and either a real target image x_t or a generated one $\hat{x}_t$ when making its decision. While the original Pix2Pix model uses 70×70 patches (the receptive field of each patch has that size), we employ 2×2 patches, determined empirically as described in the subsection that explains the network topology of the discriminator. Figure 4.1 depicts the generator and discriminator networks we used.

4.2.1.1 Objective Function

The discriminator loss $\mathcal{L}_D$ follows the original GAN formulation but is conditioned on the source image x_s and uses the non-saturating version of the loss [Goodfellow, 2016]. That is, the discriminator evaluates whether an input image in the target domain is real or generated, given that it is a translation of x_s . Formally, the loss is

$$\mathcal{L}_D = \mathbb{E}_{x_s, x_t} [\log D(x_s, x_t)] + \mathbb{E}_{x_s} [\log(1 - D(x_s, G_{s \rightarrow t}(x_s)))],$$

where the first term penalizes the discriminator for classifying real target images as fake, and the second penalizes it for classifying generated images as real.

The generator loss $\mathcal{L}_G$ consists of adversarial and reconstruction terms, weighted by a hyperparameter λ_{rec} . The reconstruction term measures the L_1 distance between the generated and ground truth images. The generator objective is

$$\mathcal{L}_G = -\mathbb{E}_{x_s} [\log D(x_s, G_{s \rightarrow t}(x_s))] + \lambda_{\text{rec}} \mathbb{E}_{x_s, x_t} [\|x_t - G_{s \rightarrow t}(x_s)\|_1], \quad (4.2)$$

where $\|\cdot\|_1$ denotes the mean absolute difference between generated and target images (pixel-wise L_1 distance). In our experiments, λ_{rec} is set to 100.

Next, we describe the generator and discriminator networks used, highlighting the adaptations we made to their original topologies.

4.2.1.2 Generator

The generator follows a U-Net architecture [Umetani, 2017], receiving and producing images of size $64 \times 64 \times 4$ (RGBA channels), in contrast to the $128 \times 128 \times 3$ images used in the original implementation. The network consists of an encoder that progressively downsamples the input to a 1×1 spatial resolution, and a decoder that upscales the activation maps back to the original resolution.

Each downsampling block comprises a convolution (kernel size 4×4 , stride 2), instance normalization (except for the first layer, in contrast to batch normalization in the original), and a Leaky ReLU activation. Each block halves the spatial resolution while maintaining or increasing the number of channels. Given the 64×64 input resolution, the encoder contains six downsampling blocks (compared to seven in the original).

The decoder mirrors the encoder. Each upsampling block includes a transposed convolution (4×4 kernels, stride 2), instance normalization, an optional dropout layer (50% retention in the first three blocks), and a ReLU activation. The number of filters follows the encoder in reverse order. The final upsampling block uses a tanh activation, producing output pixel values in the $[-1, 1]$ range.

Strided convolutions are used exclusively for downsampling and upsampling [Radford et al., 2015], allowing the network to learn task-specific scaling operations. Skip connections link the output of the i^{th} encoder layer to the $(n - i)^{\text{th}}$ decoder layer, with $n = 6$, preserving spatial information and mitigating vanishing gradients.

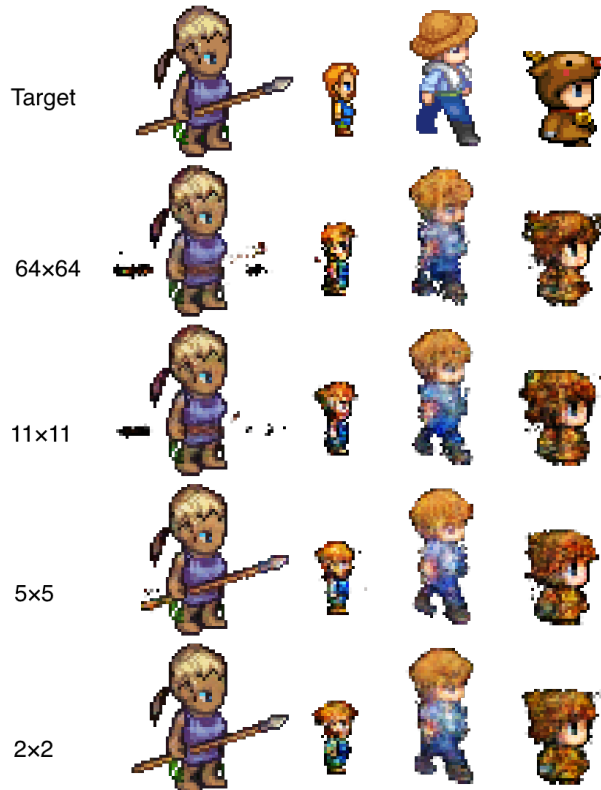
4.2.1.3 Discriminator

The discriminator follows a PatchGAN design [Isola et al., 2017], functioning as a conditional classifier. It receives a source image (e.g., a character facing front) as a condition and either a real or generated target image (e.g., facing right). The image is divided into equal-sized square patches, and the discriminator predicts whether each patch is real or fake.

The use of patch-wise discrimination enables local evaluation instead of a single global real/fake score for the entire image. This approach penalizes regions that require more refinement from the generator. While the generator’s reconstruction loss $\mathcal{L}_{\text{rec}}$ encourages the output to match the target image, it often produces blurry results; the patch-based discriminator serves as a texture and edge loss, improving visual fidelity.

We experimented with different patch sizes: 2×2 , 5×5 , 11×11 , and 64×64 (corresponding to a single output per image). Figure 4.2 presents a comparison of the outputs. The 2×2 patch size produced results closest to the ground truth. Larger patches introduced color and shape noise, and the single-patch model suffered from artifacts such as dangling pixels beyond the sprite boundaries.

Figure 4.2: Outputs of Pix2Pix-based models with varying patch sizes for the discriminator.



Smaller patches are particularly effective for pixel art, where each pixel carries

significant information due to the low resolution. Local regions often contain minimal variation, so evaluating small patches improves both texture and edge accuracy. This also mitigates color bleeding in areas that should contain uniform or nearly uniform colors. For these reasons, we adopted 2×2 patches in our implementation. Despite variations across patch sizes, all outputs roughly preserve the overall shape and color distribution of the target images.

Structurally, the discriminator uses the same downsampling blocks as the generator (Figure 4.1). Its loss function is consistent with conditional GANs [Mirza and Osindero, 2014], computed as a binary cross-entropy between real patches misclassified as fake and generated patches misclassified as real. Since the discriminator produces an output for each patch rather than a single score per image, the final loss is averaged over all patches.

4.2.1.4 Training Procedure

The generator and discriminator are trained by backpropagating the loss calculated by their respective objective functions using Adaptive Moment Estimation (Adam) [Kingma and Ba, 2014]. At each update step, a mini-batch of source-domain images is fed to the generator, which translates them to the target domain. The discriminator then receives both the generated images and real images from the target domain, and is penalized for misclassifying real images as fake or generated images as real. The generator, in turn, is penalized both for failing to fool the discriminator ($\mathcal{L}_{\text{adv}}$) and for producing images that deviate from the ground truth ($\mathcal{L}_{\text{rec}}$).

The generator has 29,307,844 parameters and performs inference in 7 ms on a GeForce GTX 1050 Ti Graphics Processing Unit (GPU)³, while the discriminator contains 9,217 parameters.

Since Pix2Pix is a single-direction, two-domain translation architecture, and our task involves four domains (i.e., $N = \{\text{back, left, front, right}\}$), we train $\|N\| \times (\|N\| - 1) = 12$ separate models to cover all possible source-target domain pairs. Training with $s = t$ yields a model that learns the identity function that trivially reproduces the input image, which is not useful for our objectives.

³The GeForce GTX 1050 Ti is a mid-range mobile GPU introduced by NVidia in 2016, with 4 GB of VRAM.

4.2.2 Based on StarGAN

Choi et al. [2018] proposed StarGAN as an architecture to translate images across multiple domains using a single generator and discriminator pair. In our problem of translating pixel art characters between poses, the generator G receives an image x_s of a character in a source domain $s \in N$ and a label $t \in N$ indicating the target domain. The generator then produces an image $\hat{x}_t$ translated to the target pose. Differing from the original implementation, we explicitly provide the generator with the source domain s label, so its mapping can be written as

$$\hat{x}_t = G_{\text{any} \rightarrow \text{any}}(x_s, t, s), \quad s, t \in N.$$

Using a single model to translate between all domains allows the network to learn shared mappings, potentially improving generation quality by leveraging information across domains.

The discriminator D (technically a *critic*, since it uses the Wasserstein loss) produces two outputs: D_{adv} , a score indicating the realness of the input image (real or generated), and D_{dmn} , a domain classification predicting the input image’s domain. Building upon the original work, we made the discriminator conditional as it also receives the image in the source domain x_s , such as in the Pix2Pix architecture, as we observed that it improved the quality of the generated images. Figure 4.3 shows the network diagrams of the generator and discriminator.

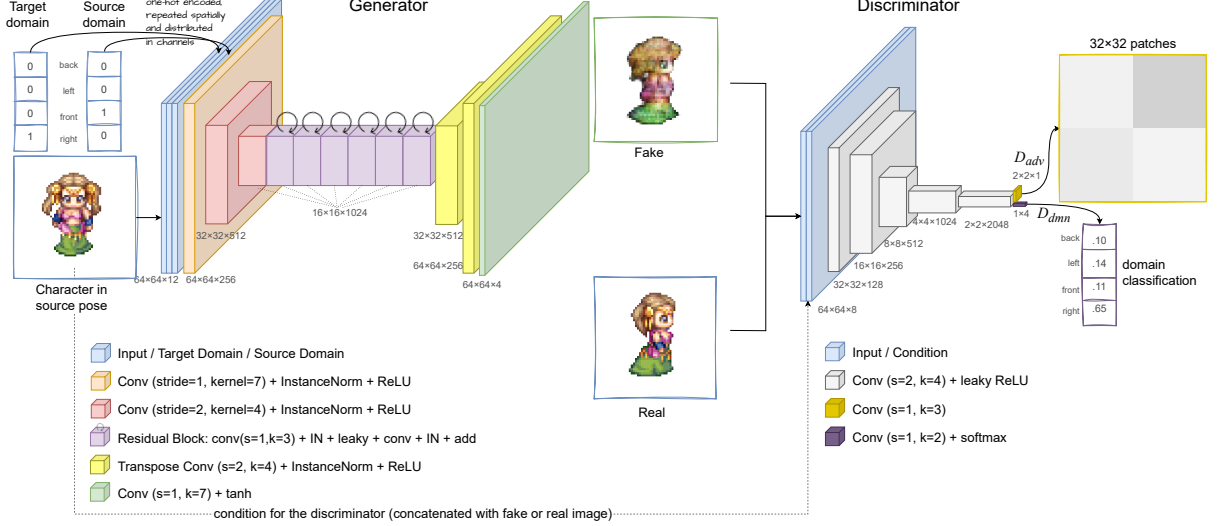
4.2.2.1 Objective Function

The generator in the original StarGAN architecture is trained with three loss terms: an adversarial component, a domain classification term, and a cyclic reconstruction term. In addition, our version includes a fourth term, $\mathcal{L}_{\text{rec}}$, that minimizes the L_1 distance between the generated image and the ground truth.

Unlike traditional GANs [Goodfellow et al., 2014], StarGAN’s adversarial term $\mathcal{L}_{\text{adv}}$ uses the Wasserstein loss with gradient penalty (WGAN-GP) [Gulrajani et al., 2017] to improve training stability. The discriminator branch D_{adv} outputs a score that should be high for real images and low for generated images. The adversarial loss is defined as:

$$\begin{aligned} \mathcal{L}_{\text{adv}} = & \mathbb{E}_{x_t, x_s} [D_{\text{adv}}(x_t | x_s)] - \mathbb{E}_{x_s, t, s} [D_{\text{adv}}(G(x_s, t, s) | x_s)] \\ & + \lambda_{\text{gp}} \mathbb{E}_{\hat{x}_t, x_s} [(\|\nabla_{\hat{x}_t} D_{\text{adv}}(\hat{x}_t | x_s)\|_2 - 1)^2], \end{aligned}$$

Figure 4.3: Architecture of the StarGAN-based model. **Left.** The generator receives an image x_s of a character in the source pose, along with the target and source domain labels. Domains are one-hot encoded, repeated, and concatenated to the image, yielding a $64 \times 64 \times 12$ input. **Right.** The discriminator receives either a real or generated image, plus the character in the source pose, and outputs a realness score D_{adv} and a domain classification D_{dmn} .



where $\hat{x}_t$ is a random convex combination of real and generated target-domain images, and λ_{gp} is the gradient penalty coefficient.

The domain classification loss $\mathcal{L}_{dmn}$ guides the generator to produce images that the discriminator classifies correctly as belonging to the target domain. The generator is penalized when it fails to achieve this, while the discriminator uses only real images to learn to classify correctly:

$$\mathcal{L}_{dmn}^{fake} = \mathbb{E}_{x_s, t} [-\log D_{dmn}(G(x_s, t, s) | x_s)],$$

$$\mathcal{L}_{dmn}^{real} = \mathbb{E}_{x_t, x_s} [-\log D_{dmn}(x_t | x_s)].$$

Following CycleGAN [Zhu et al., 2017b], a cyclic loss encourages the generator to preserve the input content (e.g., the character's identity) while modifying only domain-specific features (e.g., pose). A generated image is fed back into the generator to reconstruct the original input, and the L_1 distance between the input and reconstructed image is minimized:

$$\mathcal{L}_{cyc} = \mathbb{E}_{x_s, t, s} [\|x_s - G(G(x_s, t, s), s, t)\|_1].$$

Since paired examples are available in our dataset, we also include a pixel-wise L_1 reconstruction loss between the generated image and the ground truth:

$$\mathcal{L}_{rec} = \mathbb{E}_{x_s, x_t} [\|x_t - G(x_s, t, s)\|_1].$$

The full training objectives for the discriminator $\mathcal{L}_D$ and generator $\mathcal{L}_G$ are:

$$\begin{aligned}\mathcal{L}_D &= -\mathcal{L}_{\text{adv}} + \lambda_{\text{dmn}} \mathcal{L}_{\text{dmn}}^{\text{real}}, \\ \mathcal{L}_G &= \mathcal{L}_{\text{adv}} + \lambda_{\text{dmn}} \mathcal{L}_{\text{dmn}}^{\text{fake}} + \lambda_{\text{cyc}} \mathcal{L}_{\text{cyc}} + \lambda_{\text{rec}} \mathcal{L}_{\text{rec}}.\end{aligned}$$

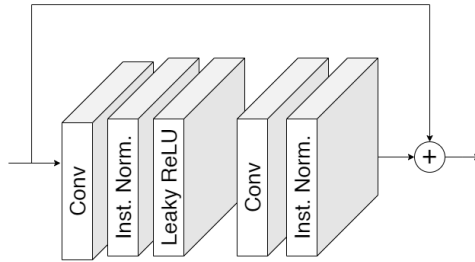
Next, we describe the generator and discriminator networks, highlighting the modifications made to their original topologies.

4.2.2.2 Generator

The generator receives an image x_s in the source domain s and a target domain label t , and outputs a translated image $\hat{x}_t$ in the target domain. Our version of the generator extends the original by explicitly informing the network of the source domain s in addition to the target, as we found that it improves the translation quality.

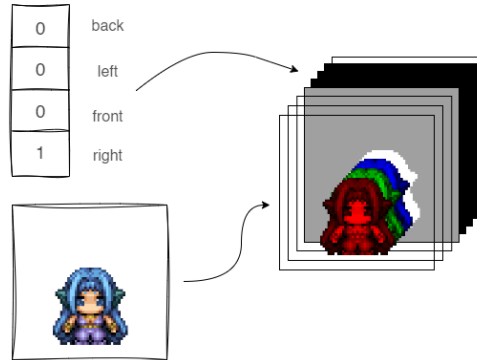
The network consists of an initial convolutional block (convolution, instance normalization, ReLU), followed by two downsampling blocks (convolution, instance normalization, Leaky ReLU), six residual blocks, two upsampling blocks (transpose convolution, instance normalization, ReLU), and a final convolutional block (convolution and tanh activation). Each residual block [He et al., 2015] contains a convolution, instance normalization, Leaky ReLU, another convolution, and instance normalization, with the input of the block added to the output of the second normalization layer (Figure 4.4).

Figure 4.4: Residual block in the StarGAN generator. The input activations are added to the output of the second instance normalization layer.



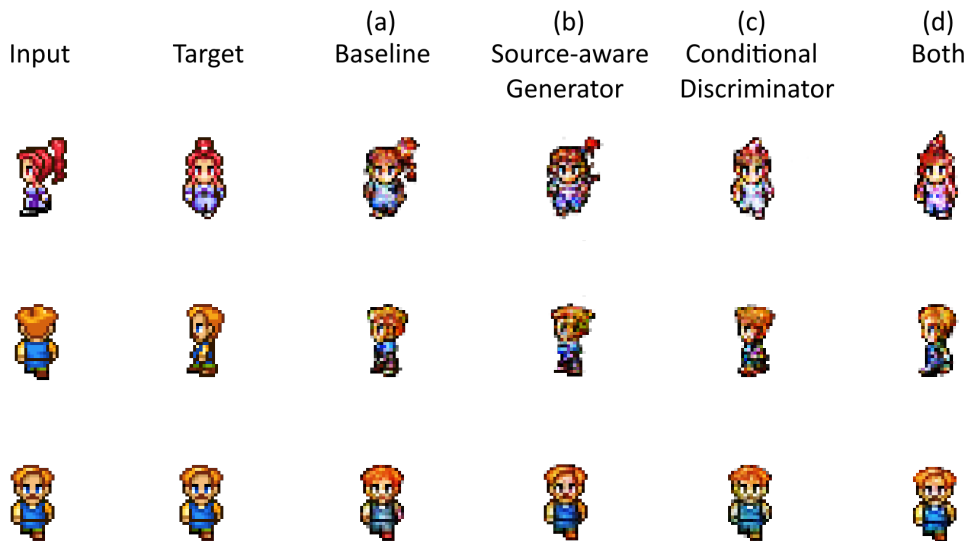
To convey domain information, the target domain label t is one-hot encoded, repeated spatially to match the input image resolution, and concatenated along the channels axis with the input image (Figure 4.5). In our modification, the source domain s is processed similarly and concatenated along the channels. Therefore, the input tensor to the generator has shape $64 \times 64 \times 12$, with the first four channels corresponding to the RGBA image, the next four to the target domain t , and the final four to the source domain s .

Figure 4.5: One-hot encoding of the domain labels, repeated across the spatial dimensions, then concatenated to the input image along the channel axis.



Making the source domain explicit improves translation results, as the generator no longer needs to infer the source pose. Figure 4.6 shows example outputs: (a) using the original generator, and (b) using the generator aware of the source domain. The improvement is most evident when the source and target domains are the same (third row).

Figure 4.6: Pose translation results using StarGAN generators with different input configurations. **Rows** show example translations: left to front, back to left, and front to front. **Columns** present the output of the following models: (a) original networks, (b) generator receiving the domain of the source image, (c) discriminator receiving the image in the source domain (becoming conditional), and (d) generator and the discriminator receiving the additional inputs.



To increase network capacity, we expanded the number of channels. While the original generator uses 64, 128, and 256 filters in successive layers, our implementation multiplies these depths by four, improving its ability to model complex image transformations.

4.2.2.3 Discriminator

The original discriminator receives an image and outputs two types of information: a realness score for image patches and a classification of the image’s domain. In the original work, the input is a single image, and the network contains six downsampling blocks. The realness score is computed per patch, and our implementation uses patches of size 16×16 .

Following our modifications to the generator, we also feed the discriminator an additional input: the character image x_s in the source pose s . This turns the discriminator into a conditional network, similar to the Pix2Pix models, where it evaluates whether a target image $\hat{x}_t$ is a realistic translation of the provided source image x_s , in addition to classifying its domain. We observed that including this additional input improves translation quality, as the generator is guided to produce images that better match the target pose. Figure 4.6 illustrates example translations for different configurations: (a) original discriminator, (c) conditional discriminator with the extra input, and (d) changes to both generator and discriminator.

The domain classification branch uses an auxiliary classifier [Odena et al., 2017] with a softmax activation, which predicts the probability of the image belonging to each domain $t \in N$. This branch steers the generator toward producing images that not only appear realistic but also correspond to the correct target domain.

4.2.2.4 Training Procedure

The model is trained using the optimizer Adam [Kingma and Ba, 2014], alternating one generator update after each discriminator update. This differs from the original StarGAN implementation, where the generator is updated only once every five discriminator updates. We observed that the extra update steps for the discriminator increased the training time without necessarily improving the quality of the resulting images.

At each update step, we select a mini-batch of images from random source domains and assign each image a random target domain. This strategy differs from the original StarGAN, which uses a single target domain for all images in a mini-batch. We found that using multiple target domains per mini-batch accelerates generator convergence, as shown in Figure 4.7. Early in training (after 1,000 update steps), images generated with a single target per batch (a) exhibit less stable translations compared to our approach with multiple targets (b). This improvement occurs because the gradient directions from multiple targets guide the generator more effectively toward the optimal solution.

Figure 4.7: Pose translation using different mini-batch sampling strategies at early training stages (1,000 generator updates). **Rows** show examples in various source/target domains. **Columns** show: (a) single target for the whole mini-batch, and (b) a different target for each image in the mini-batch.



During training, the generator creates images $\hat{x}_t = G(x_s, t, s)$ in the target domain and is penalized for differences from the ground truth via the reconstruction loss $\mathcal{L}_{\text{rec}}$. It then performs a cyclic translation back to the source domain, producing $\tilde{x}_s = G(\hat{x}_t, s, t)$, which contributes to the cyclic consistency loss $\mathcal{L}_{\text{cyc}}$. The discriminator evaluates the generated images $\hat{x}_t$, producing a realness score and predicting the domain. The generator is penalized based on these outputs using the adversarial loss $\mathcal{L}_{\text{adv}}$ and the domain classification loss $\mathcal{L}_{\text{dnn}}^{\text{fake}}$.

The discriminator is trained adversarially to assign high scores to real images and low scores to generated images ($-\mathcal{L}_{\text{adv}}$), while also classifying the domain of real images correctly ($\mathcal{L}_{\text{dnn}}^{\text{real}}$).

The network sizes are 134,448,128 parameters for the generator and 44,632,064 for the discriminator. At inference, the generator produces images in 10 ms on a GeForce GTX 1050 Ti GPU with 4 GB of VRAM.

4.2.3 Based on CollaGAN

CollaGAN [Lee et al., 2019] was proposed to address the image imputation problem, generating images for missing domains when paired examples are available. Similar to StarGAN, it employs a single generator-discriminator pair, but unlike image-to-image

translation models, the generator receives multiple input images from a subset of available domains $S \subset N$ and produces an image in a target domain $t \in N$:

$$\hat{x}_t = G_{\text{many} \rightarrow \text{any}}(x_S, t), \quad S \subset N, t \in N.$$

Here and in the next proposed models, x_S denotes the tuple of images ordered according to the domains in N , where available images are represented in their corresponding x_s and missing images as tensors filled with zeros with the same shape, 0. That is,

$$x_S = (x_i : i \in N) \quad \text{where } x_i = \begin{cases} x_s, & i \in S, \\ 0, & \text{otherwise.} \end{cases}$$

Figure 4.8 illustrates the architecture of the generator and discriminator networks used in our implementation. The generator is inspired by U-net architectures but features one encoder branch per input domain. Each branch processes its corresponding input image independently, producing intermediate feature maps. These feature maps are concatenated at the bottleneck layer and fed into a unified decoder branch, which produces the image in the target domain. Skip connections from each encoder branch to the corresponding decoder layers preserve spatial details and improve reconstruction quality. The target domain is encoded as a one-hot vector, spatially expanded, and concatenated with each input image channel-wise, guiding the generator towards producing the correct output domain.

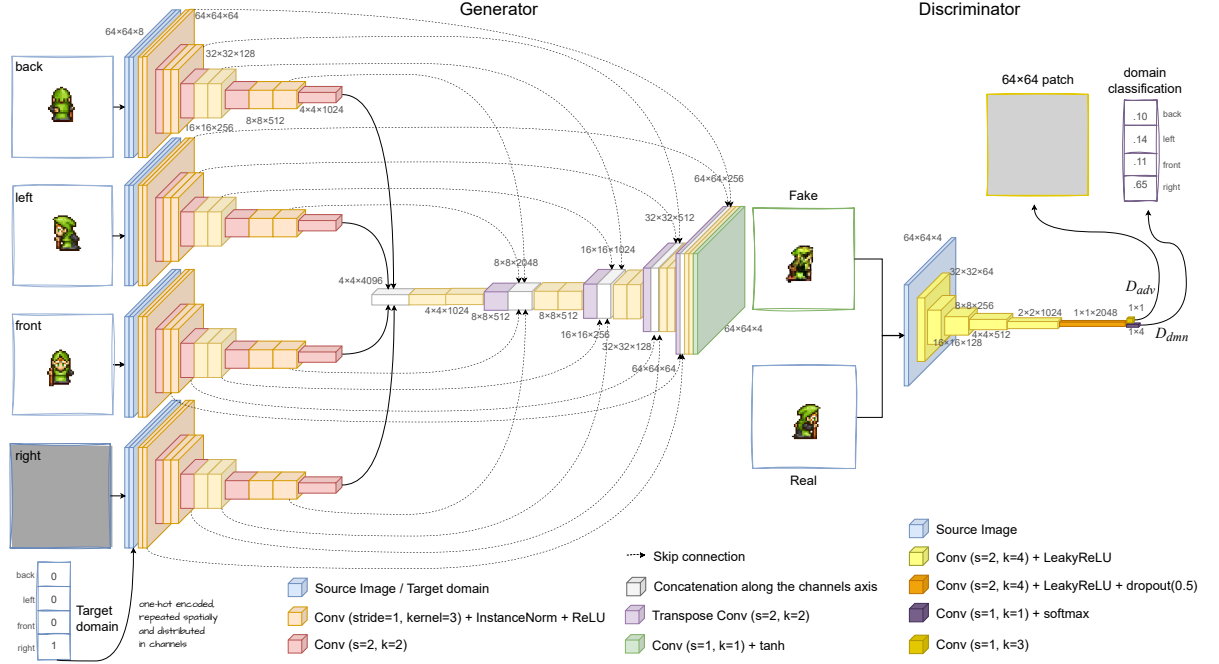
The discriminator evaluates whether an image is real or generated (D_{adv}) and simultaneously classifies its domain (D_{dmn}) through an auxiliary classifier. Its architecture is similar to StarGAN’s critic, except that it receives only the image to be evaluated, not the full set of inputs.

4.2.3.1 Objective Function

As usually done with GANs, both the generator and discriminator are trained adversarially, with additional loss terms to improve image quality and consistency. The generator’s objective comprises five terms: reconstruction, multiple cycle consistency, structural similarity, adversarial, and domain classification. The discriminator trains with adversarial and domain classification objectives.

Training involves a forward and backward pass. In the forward pass, a mini-batch of paired images with a randomly chosen missing domain t is fed to the generator G , which synthesizes the corresponding image in the missing domain. For instance, if $S = \{a, b, c\}$

Figure 4.8: Architecture of the CollaGAN-based model. **Left.** The generator receives a set of images of a character in source domains and a target domain label (one-hot encoded and spatially repeated). Each source image flows through its respective encoder branch, and the resulting feature maps are concatenated at the bottleneck to form the input to the unified decoder. Skip connections provide early outputs to the corresponding decoder layers. **Right.** The discriminator receives a single image (real or fake) and outputs D_{adv} for real/fake classification and D_{dnn} for domain classification.



and $t = d$, the available images are x_a, x_b, x_c , and the generator produces

$$\hat{x}_d = G(\langle x_a, x_b, x_c, 0 \rangle, d),$$

where 0 is a tensor of zeros matching the shape of the input images, representing the missing domain.

In the backward pass, cycle consistency is enforced by generating images for all source domains $s \in S$ using the synthesized target image $\hat{x}_d$ in place of the real x_d :

$$\tilde{x}_{a|d} = G(\langle 0, x_b, x_c, \hat{x}_d \rangle, a),$$

$$\tilde{x}_{b|d} = G(\langle x_a, 0, x_c, \hat{x}_d \rangle, b),$$

$$\tilde{x}_{c|d} = G(\langle x_a, x_b, 0, \hat{x}_d \rangle, c),$$

with the aim of reconstructing the original images x_a, x_b , and x_c . The pixel-wise L_1 losses for reconstruction and multiple cycle consistency are defined as:

$$\mathcal{L}_{\text{rec}} = \mathbb{E}_{x_t, x_S} [\|x_t - \hat{x}_t\|_1],$$

$$\mathcal{L}_{\text{mcyc}} = \mathbb{E}_{x_t, x_S} \left[\sum_{s \in S} \|x_s - \tilde{x}_{s|t}\|_1 \right].$$

An additional structural similarity loss $\mathcal{L}_{\text{ssim}}$ [Wang et al., 2004] improves perceptual quality by comparing the cyclically generated images $\tilde{x}_s$ with the original source images x_s . It uses the Structural Similarity Index Measure (SSIM) [Wang et al., 2004] to compose a loss term between the cyclically generated $\tilde{x}_{s|t}, \forall s \in S$ and the real source images $x_s, \forall s \in S$. Its formulation is the same as in the CollaGAN paper, and omitted here for brevity.

The adversarial losses for the generator and discriminator follow the one from Least Squares Generative Adversarial Networks (LSGAN) [Mao et al., 2017], which minimizes the squared errors of discriminator outputs for real and fake images. Both networks consider the images generated in the forward and in the backward pass, with the discriminator also using the real images. The losses are:

$$\begin{aligned}\mathcal{L}_{\text{adv}}^D &= \mathbb{E}_{\hat{x}_t}[(D_{\text{adv}}(\hat{x}_t) - 0)^2] + \mathbb{E}_{\tilde{x}_{s|t}}[(D_{\text{adv}}(\tilde{x}_{s|t}) - 0)^2] + \mathbb{E}_{x_t}[(D_{\text{adv}}(x_t) - 1)^2], \\ \mathcal{L}_{\text{adv}}^G &= \mathbb{E}_{\hat{x}_t}[(D_{\text{adv}}(\hat{x}_t) - 1)^2] + \mathbb{E}_{\tilde{x}_{s|t}}[(D_{\text{adv}}(\tilde{x}_{s|t}) - 1)^2].\end{aligned}$$

The domain classification objective encourages the generator to produce images that are classified as belonging to the intended target domain. For the generator, only synthesized images contribute (both from the forward and backward passes):

$$\mathcal{L}_{\text{dmn}}^{\text{fake}} = \mathbb{E}_{\hat{x}_t}[-\log D_{\text{dmn}}(\hat{x}_t)] + \mathbb{E}_{\tilde{x}_{s|t}}[-\log D_{\text{dmn}}(\tilde{x}_{s|t})],$$

while the discriminator is trained only on real images:

$$\mathcal{L}_{\text{dmn}}^{\text{real}} = \mathbb{E}_{x_t}[-\log D_{\text{dmn}}(x_t)].$$

The full objective functions for the generator and discriminator are weighted sums of the individual loss terms:

$$\begin{aligned}\mathcal{L}_G &= \mathcal{L}_{\text{adv}}^G + \lambda_{\text{rec}}\mathcal{L}_{\text{rec}} + \lambda_{\text{mcyc}}\mathcal{L}_{\text{mcyc}} + \lambda_{\text{ssim}}\mathcal{L}_{\text{ssim}} + \lambda_{\text{dmn}}\mathcal{L}_{\text{dmn}}^{\text{fake}}, \\ \mathcal{L}_D &= \mathcal{L}_{\text{adv}}^D + \lambda_{\text{dmn}}\mathcal{L}_{\text{dmn}}^{\text{real}}.\end{aligned}$$

4.2.3.2 Generator

The authors of CollaGAN evaluated their architecture in three different tasks: illumination, facial expression, and magnetic resonance contrast imputation. Each one employed a distinct configuration for both the generator and discriminator networks. In our model, we adopted the configuration from the facial expression imputation setting,

as it most closely resembles our problem. In that scenario, paired examples consist of faces exhibiting different expressions, which can introduce small spatial displacements and make the images not strictly aligned pixel-wise. Similarly, in our case, each domain represents a pose of a character from a different viewing angle.

The generator comprises four encoder branches, each processing a source image from one domain, and a channelized and spatially replicated one-hot encoding of the target domain label, as in the StarGAN-based model (Figure 4.5). Each encoder branch includes four downsampling blocks, whose outputs are concatenated at the bottleneck layer and further processed before being passed to a single decoder. The decoder consists of four upsampling blocks. Skip connections link the concatenated activation maps from the i^{th} downsampling block to the $(n - 1)^{\text{th}}$ upsampling block. Compared to the original CollaGAN generator, which contains eight encoder branches for eight domains, our variant uses four, corresponding to our number of domains.

Each downsampling block consists of two resolution-preserving convolutional sequences (convolution + instance normalization + ReLU activation), followed by a convolution that halves the spatial dimensions and doubles the number of feature channels. Conversely, each upsampling block consists of two convolutional sequences, followed by a transposed convolution that doubles the resolution of the feature maps.

The input image resolution and number of channels in our setup are $64 \times 64 \times 4$, consistent with the other proposed models, whereas the original CollaGAN used $128 \times 128 \times 3$. To increase the network capacity, we used four times more channels in each layer than in the original architecture: 64, 128, 256, 512, and 1024 for the encoder branches, and 1024, 512, 256, 128, and 64 for the decoder blocks.

4.2.3.3 Discriminator

The discriminator receives a batch of images and produces two outputs. The first, D_{adv} , is a realness score that should be close to one for real images and zero for generated ones. The second, D_{dmn} , is a probability distribution over the domains, indicating the likelihood of an image belonging to each one.

In contrast to the Pix2Pix and our StarGAN-based models, the original CollaGAN architecture does not condition the discriminator on any source image; it evaluates each image independently, whether real or synthesized. We experimented with providing the discriminator with the character’s images in the other poses as additional input, but this modification led to unstable training and prevented the generator from converging reliably.

The network topology follows the original CollaGAN design. It consists of six downsampling blocks, each composed of a convolution that halves the spatial resolution while increasing the number of feature channels, followed by a Leaky ReLU activation. The final block includes a dropout layer to regularize training. After these downsampling stages, two parallel convolutional layers produce the outputs: one yields the adversarial score D_{adv} with a linear activation, and the other provides the domain classification output D_{dmn} with a softmax activation.

4.2.3.4 Training Procedure

At each training step, we select a batch of paired images $x_s, \forall s \in S$ with random target domains t . The generator receives the tuple that represents available and missing images $\langle x_s, t \rangle$ and synthesizes the missing image $\hat{x}_t$ in the forward pass. Next, $\hat{x}_t$ is used in place of x_t to create $|N| - 1$ new batches, in which each domain in S becomes the target during the backward (cyclical) pass. The generator then produces $\tilde{x}_{s|t}, \forall s \in S$, which should closely reconstruct the corresponding original images $x_s, \forall s \in S$.

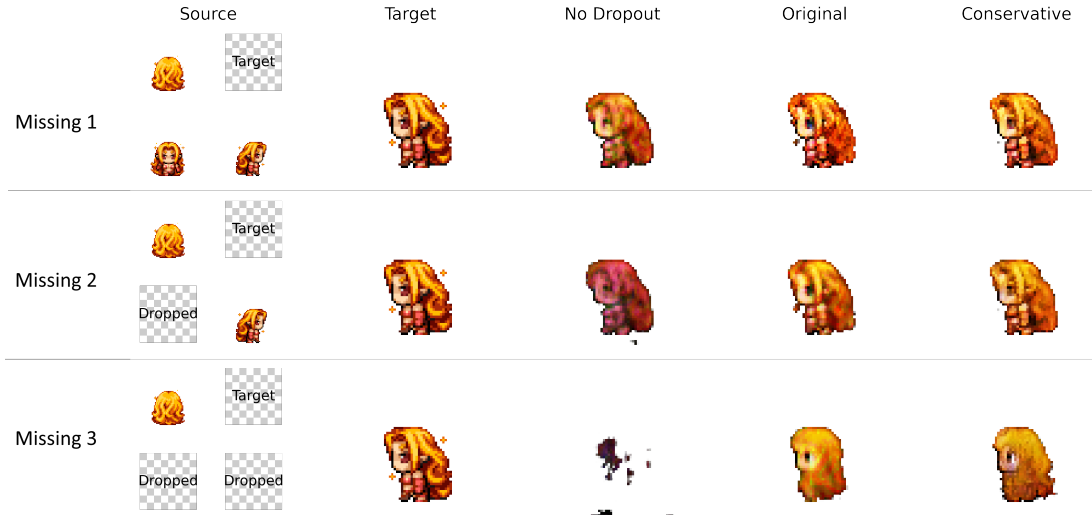
Learning proceeds by updating the network weights through Adam [Kingma and Ba, 2014], as in the other models, according to each network’s respective loss function. The generator is penalized for creating forward images $\hat{x}_t$ that deviate from the ground-truth targets x_t ($\mathcal{L}_{\text{rec}}$), and backward images $\tilde{x}_{s|t}, \forall s \in S$, that differ from the original x_s , in terms of pixel-wise absolute difference ($\mathcal{L}_{\text{mcyc}}$) and structural similarity index measure ($\mathcal{L}_{\text{ssim}}$). It also attempts to fool the discriminator into classifying its outputs as real ($\mathcal{L}_{\text{adv}}^G$) and to have them recognized as belonging to the correct target domain ($\mathcal{L}_{\text{dmn}}^{\text{fake}}$). Conversely, the discriminator learns to distinguish real from generated images ($\mathcal{L}_{\text{adv}}^D$) and to correctly classify the domains of real images ($\mathcal{L}_{\text{dmn}}^{\text{real}}$).

The original authors observed that image quality deteriorates as the number of available source domains decreases from the full set of $|N| - 1$. However, many practical scenarios involve multiple missing domains. To address this, they proposed a batch selection strategy called *input dropout*, in which the model is trained with one or more missing domains. For example, if $|N| = 4$ and $t = d$, a batch $\langle x_S, t \rangle$ sampled with input dropout may include zero, one, or two withdrawn source images, resulting in configurations such as:

$$\begin{array}{lll} x_S = (x_a, x_b, x_c, 0) & x_S = (0, x_b, x_c, 0) & x_S = (0, 0, x_c, 0) \\ & x_S = (x_a, 0, x_c, 0) & x_S = (0, x_b, 0, 0) \\ & x_S = (x_a, x_b, 0, 0) & x_S = (x_a, 0, 0, 0). \end{array}$$

In the original CollaGAN implementation, the number of dropped-out images is uniformly sampled, leading to a 33% chance of having the complete set of source domains. This strategy also improved performance in our task. However, we observed that a more conservative sampling—where the model more frequently trains with a higher number of available images—further enhances robustness in scenarios with fewer sources. Accordingly, we adopted probabilities of 10%, 30%, and 60% for dropping two, one, and zero images, respectively (in addition to the target domain). Figure 4.9 compares the three training strategies (no dropout, original dropout, and our conservative dropout) across cases with zero, one, and two missing images in x_S .

Figure 4.9: Comparison of input dropout strategies during training in the CollaGAN-based model. **Columns** show the input images to the generator, the target image, and the generated outputs using the batch selection strategies of: training without dropout, with the original version, and with our conservative strategy. **Rows** depict scenarios of generating the target image with only the target missing, with two, and with three missing source images.



We also modified the cyclical reconstruction stage of the training procedure. In the original implementation, when generating the backward images $\tilde{x}_{s|t}, \forall s \in S$ (e.g., $\tilde{x}_{a|d}, \tilde{x}_{b|d}, \tilde{x}_{c|d}$), the forward-generated image $\hat{x}_t$ replaced not only the real target image x_t , but also all input images that were dropped out by the batch selection strategy. We found that replacing only x_t with $\hat{x}_t$ leads to more stable training and improved image quality.

To illustrate the difference, consider a batch with $t = d$ and domain c dropped out. The backward-generated images $\tilde{x}_{s|t}, \forall s \in S$ for the original (left) and our modified (right) procedures are given as:

$$\begin{aligned}
 \tilde{x}_{a|d} &= G(\langle 0, x_b, \hat{x}_d, \hat{x}_d \rangle, a) & \tilde{x}_{a|d} &= G(\langle 0, x_b, 0, \hat{x}_d \rangle, a) \\
 \tilde{x}_{b|d} &= G(\langle x_a, 0, \hat{x}_d, \hat{x}_d \rangle, b) & \tilde{x}_{b|d} &= G(\langle x_a, 0, 0, \hat{x}_d \rangle, b) \\
 \tilde{x}_{c|d} &= G(\langle x_a, x_b, 0, \hat{x}_d \rangle, c) & \tilde{x}_{c|d} &= G(\langle x_a, x_b, 0, \hat{x}_d \rangle, c).
 \end{aligned}$$

Figure 4.10 compares the generated results under both training strategies. When trained using the original replacement procedure, the model tends to produce artifacts from non-target domains, often visible as duplicated or misplaced features (e.g., extra eyes), whereas our modification yields cleaner, more coherent imputations.

Figure 4.10: Comparison of the image replacement procedures for the CollaGAN-based model. **Columns** show the input images, the target, and the generated results when the model is trained replacing, during the backward pass: the forward target and all dropped-out images (original strategy), versus only the forward target (our modification). **Rows** present different examples of character generation.

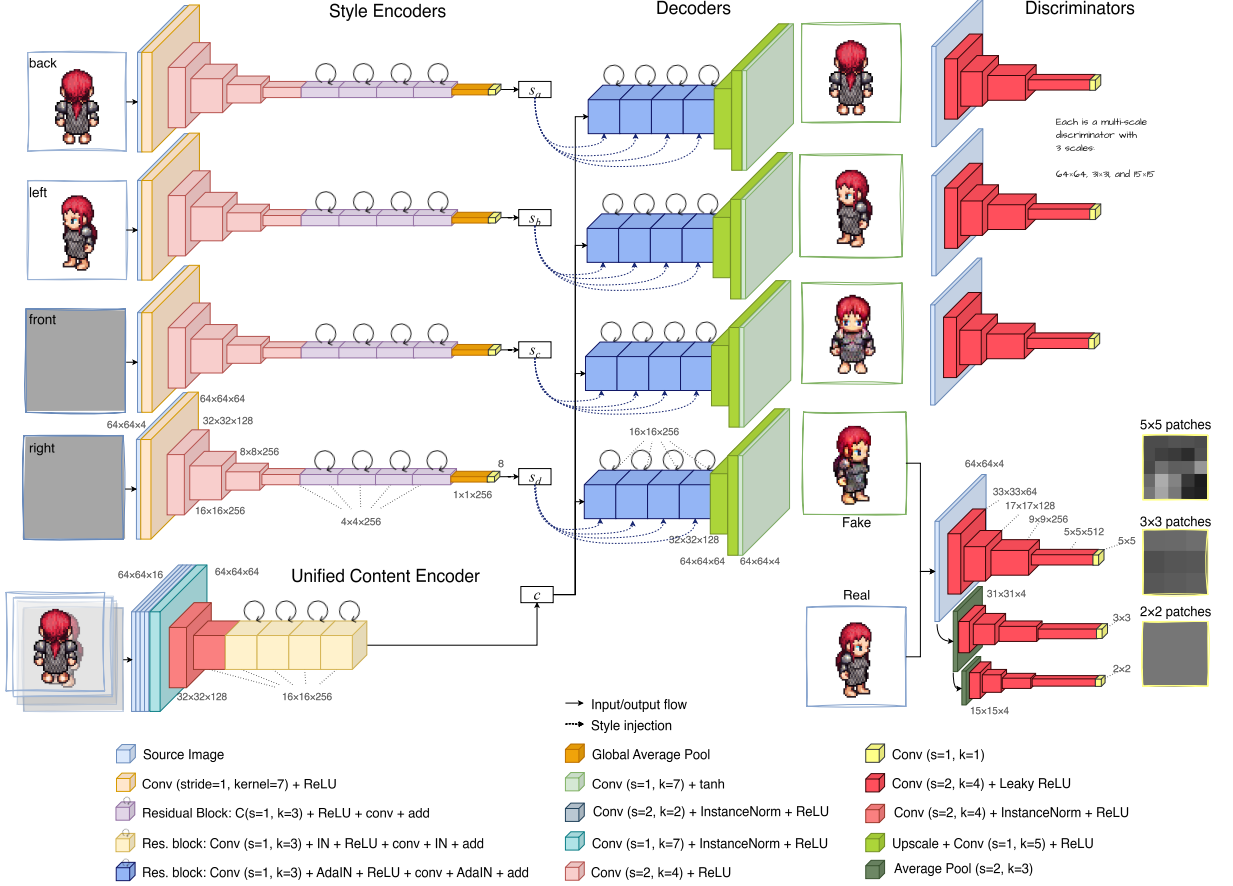


Finally, regarding model size, the generator contains 104,887,616 trainable parameters and the discriminator 44,726,272. After training, the generator performs inference in 17 ms on a GeForce GTX 1050 Ti GPU with 4 GB of VRAM.

4.2.4 Based on ReMIC

Also tackling the missing-data imputation problem, Shen et al. [2021] proposed the Representational Disentanglement Scheme for Multi-Domain Image Completion (ReMIC) architecture. However, differently from CollaGAN, a model that follows this architecture can impute images from all the missing domains at the same time, in a single feed-forward pass. Hence, it is a multi-input, multi-output architecture, that generates images in all missing domains $\hat{x}_t, \forall t \in T$ from the images in the available ones $x_s, \forall s \in S$. Its input is

Figure 4.11: Architecture of the ReMIC-based model. **Style encoders** receive an image in a pose and extract their style s_i into a tensor with length 8. The **Unified content encoder** receives all available images and outputs a content code c with shape $16 \times 16 \times 256$. Each **Decoder** processes the content code c and mix in a style code s_i in its residual blocks, generating an image in the end. Each domain has a multi-scale **Discriminator** that receives either a real or a fake image and produces three scales of patches with realness scores.



the tuple of images x_S (defined in Equation 4.1), and the output is

$$\hat{x}_T = G_{\text{many} \rightarrow \text{many}}(x_S), \text{ with } S \subset N, T = N \setminus S.$$

Another relevant characteristic from ReMIC is that it disentangles the latent representation in a domain-specific style code (different for each pose, in our case) and a domain-invariant content code (same for all poses). To achieve a disentangled representation in a multi-domain setting, it uses one style encoder E_i^{style} for each domain $i \in N$, a unified content encoder E^{content} that receives images from all of the available domains, and one decoder G_i for each domain $i \in N$. There are also one discriminator D_i for each domain. Figure 4.11 shows each network's topology in our case, where $|N| = 4$.

To illustrate the generation process, we will consider that there are two domains available, $S = \{a, b\}$, and we want to produce the images $\hat{x}_T = \{\hat{x}_c, \hat{x}_d\}$ from the other two missing. In a forward pass, the model uses random style codes s_c and s_d sampled

from $\mathcal{N}(0, 1)$. Then, the unified content encoder E^{content} receives the input image tuple $x_S = (x_a, x_b, 0, 0)$ and maps them to a content code c such that

$$c = E^{\text{content}}(\langle x_a, x_b, 0, 0 \rangle).$$

Lastly, the decoders G_c and G_d for the missing domains receive the unified content code c and the random style codes s_c and s_d and generate the images $\hat{x}_T = \{\hat{x}_c, \hat{x}_d\}$ from the missing domains:

$$\begin{aligned}\hat{x}_c &= G_c(s_c, c), \\ \hat{x}_d &= G_d(s_d, c).\end{aligned}$$

4.2.4.1 Objective Function

During training, the model aims to reconstruct the input images and impute the missing ones, but also to reconstruct the style and content latent codes of the images it generated. Even though the model does not try to cyclically reconstruct the original input images such as done by StarGAN and CollaGAN, it trains with constraints in both directions: (a) image $\rightarrow$ latent $\rightarrow$ image and (b) latent $\rightarrow$ image $\rightarrow$ latent. Hence, there is an image consistency loss $\mathcal{L}_{\text{consis}}^{\text{img}}$ and a latent consistency loss $\mathcal{L}_{\text{consis}}^{\text{lat}}$, similar to the procedure proposed for BicycleGAN [Zhu et al., 2017a].

The (a) image consistency direction requires the model to encode the available input images into domain-specific style codes and a unified content code, and then it tries to decode them back to the original images:

$$\begin{aligned}c &= E^{\text{content}}(x_S), \\ s_i &= E_i^{\text{style}}(x_i) & \forall i \in S, \\ \hat{x}_i &= G_i(s_i, c) & \forall i \in S.\end{aligned}$$

The image consistency loss $\mathcal{L}_{\text{consis}}^{\text{img}}$ penalizes the model for errors in the image reconstruction using the pixel-wise L_1 distance between the available input x_i and generated images $\hat{x}_i, \forall i \in S$. This loss component is calculated only for the available domains and is such that

$$\mathcal{L}_{\text{consis}}^{\text{img}} = \mathbb{E}_{x_i}[\|x_i - \hat{x}_i\|_1].$$

In turn, to enforce the constraint of the (b) latent consistency, the model tries to reconstruct the latent codes from the generated images. To achieve that, the model generates new images $\hat{x}_i$ from the content codes extracted in (a) but using random style

codes $\dot{s}_i$ sampled from $\mathcal{N}(0, 1)$. It then tries to reconstruct the content $\hat{c}$ and style codes $\hat{s}_i$ from the generated images $\dot{x}_i$. This loss term is calculated considering both present and absent domains, i.e., for all $i \in N$

$$\begin{aligned}\dot{x}_i &= G_i(\dot{s}_i, c) & \dot{s} &\sim \mathcal{N}(0, 1), \forall i \in N, \\ \hat{s}_i &= E_i^{\text{style}}(\dot{x}_i) & & \forall i \in N, \\ \hat{c} &= E^{\text{content}}(\langle \dot{x}_1, \dots, \dot{x}_{|N|} \rangle).\end{aligned}$$

The latent consistency loss is split in two terms, one $\mathcal{L}_{\text{consis}}^{\text{lat-cnt}}$ for the content code and another for the style codes $\mathcal{L}_{\text{consis}}^{\text{lat-sty}}$, both calculated as the L_1 distance to the original latent values c and $\dot{s}_i$, respectively:

$$\begin{aligned}\mathcal{L}_{\text{consis}}^{\text{lat-cnt}} &= \mathbb{E}_c[\|c - \hat{c}\|_1], \\ \mathcal{L}_{\text{consis}}^{\text{lat-sty}} &= \mathbb{E}_{\dot{s}}[\sum_{i \in N} \|\dot{s}_i - \hat{s}_i\|_1], \\ \mathcal{L}_{\text{consis}}^{\text{lat}} &= \mathcal{L}_{\text{consis}}^{\text{lat-cnt}} + \mathcal{L}_{\text{consis}}^{\text{lat-sty}}.\end{aligned}$$

Another training objective is the image reconstruction loss $\mathcal{L}_{\text{rec}}$, which applies for both present and absent domains and is calculated as the pixel-wise L_1 distance of the images generated with the original content but a random style (i.e., $\dot{x}_i, \forall i \in N$) to the ground truth images $x_i, \forall i \in N$:

$$\mathcal{L}_{\text{rec}} = \mathbb{E}_{x_i}[\|x_i - \dot{x}_i\|_1].$$

Lastly, there is an adversarial loss term, using the domain-specific discriminators through the **LSGAN** loss [Mao et al., 2017]. The generated images that each discriminator D_i sees are those generated with random styles, i.e., $\dot{x}$. The discriminator and generator parts are $\mathcal{L}_{\text{adv}}^D$ and $\mathcal{L}_{\text{adv}}^G$ such that

$$\begin{aligned}\mathcal{L}_{\text{adv}}^{D_i} &= \mathbb{E}[(D_i(x_i) - 1)^2] + \mathbb{E}[(D_i(\dot{x}_i) - 0)^2], \\ \mathcal{L}_{\text{adv}}^{G_i} &= \mathbb{E}[(D_i(\dot{x}_i) - 1)^2], \\ \mathcal{L}_{\text{adv}}^D &= \sum_{i \in N} \mathcal{L}_{\text{adv}}^{D_i}, \\ \mathcal{L}_{\text{adv}}^G &= \sum_{i \in N} \mathcal{L}_{\text{adv}}^{G_i}.\end{aligned}$$

The final loss functions $\mathcal{L}_G$ and $\mathcal{L}_D$ are then calculated as a sum weighted by some λ scalars:

$$\begin{aligned}\mathcal{L}_G &= \mathcal{L}_{\text{adv}}^G + \lambda_{\text{consis}}^{\text{img}} \mathcal{L}_{\text{consis}}^{\text{img}} + \lambda_{\text{consis}}^{\text{lat}} (\mathcal{L}_{\text{consis}}^{\text{lat-cnt}} + \mathcal{L}_{\text{consis}}^{\text{lat-sty}}) + \lambda_{\text{rec}} \mathcal{L}_{\text{rec}}, \\ \mathcal{L}_D &= \mathcal{L}_{\text{adv}}^D.\end{aligned}$$

4.2.4.2 Generator and Encoders

The generation of images in the ReMIC architecture involves different networks: $|N|$ domain-specific style encoders E_i^{style} , a unified content encoder E^{content} , and $|N|$ domain-specific decoders G_i .

Each style encoder E_i^{style} receives a batch of $64 \times 64 \times 4$ images and outputs a latent style code of length 8 for each image. The input passes through an initial convolution followed by a ReLU activation, then through four downsampling blocks, each composed of a convolution and a ReLU activation (128, 256, 256, and 256 channels). Next, the resulting activation maps are fed into four residual blocks composed of two sequences of convolution + ReLU activation. Finally, the activations undergo global average pooling, followed by a 1×1 convolution with eight filters, linearly mapping them to the style code representation.

The unified content encoder E^{content} processes a batch containing the tuple x_S of images from all domains, x_i for $i \in S$, or a zero tensor 0 of the same shape otherwise, and outputs a content code for each example in the batch with shape $16 \times 16 \times 256$. The input images pass through an initial convolution + instance normalization + ReLU activation sequence, followed by two downsampling blocks (convolution + instance normalization + ReLU, with 128 and 256 filters) and four residual blocks (each with 256 filters).

Lastly, each decoder G_i receives a batch with both the style code for its respective domain (8 values) and the content code ($16 \times 16 \times 256$) and outputs a batch of images. The content code is processed through four residual blocks that apply Adaptive Instance Normalization (AdaIN) [Huang and Belongie, 2017], injecting the style code into the activation maps at each stage. Each such block consists of two consecutive convolution (256 channels) + AdaIN + ReLU activation sequences. Then, the activation maps are upsampled twice (nearest-neighbor upsampling + convolution + ReLU), gradually restoring the resolution to 64×64 , with 128 and 64 channels. A final convolution followed by a tanh activation reduce the number of channels to 4, producing the final $64 \times 64 \times 4$ image.

4.2.4.3 Discriminators

There are $|N|$ discriminators, each responsible for judging the images from a specific domain. They perform multi-scale, patch-based discrimination: each one evaluates the batch of input images at three scales, obtained by average pooling the input twice (to 32×32 and 16×16), with each scale processed by a separate branch. Each branch has four

downsampling blocks consisting of a convolution + Leaky ReLU activation (64, 128, 256, and 512 channels), and a final 1×1 convolution with a single filter at the end. The outputs correspond to discrimination maps over rectangular patches of varying resolutions— 5×5 , 3×3 , and 2×2 —arising from the three scales.

4.2.4.4 Training Procedure

To disentangle the data representation, the model is trained to optimize two complementary directions: (a) $image \xrightarrow{a.1} latent \xrightarrow{a.2} image$ and (b) $latent \xrightarrow{b.1} image \xrightarrow{b.2} latent$. At each training step, batches of x_S and x_T are sampled according to an input dropout strategy, following the approach of CollaGAN.

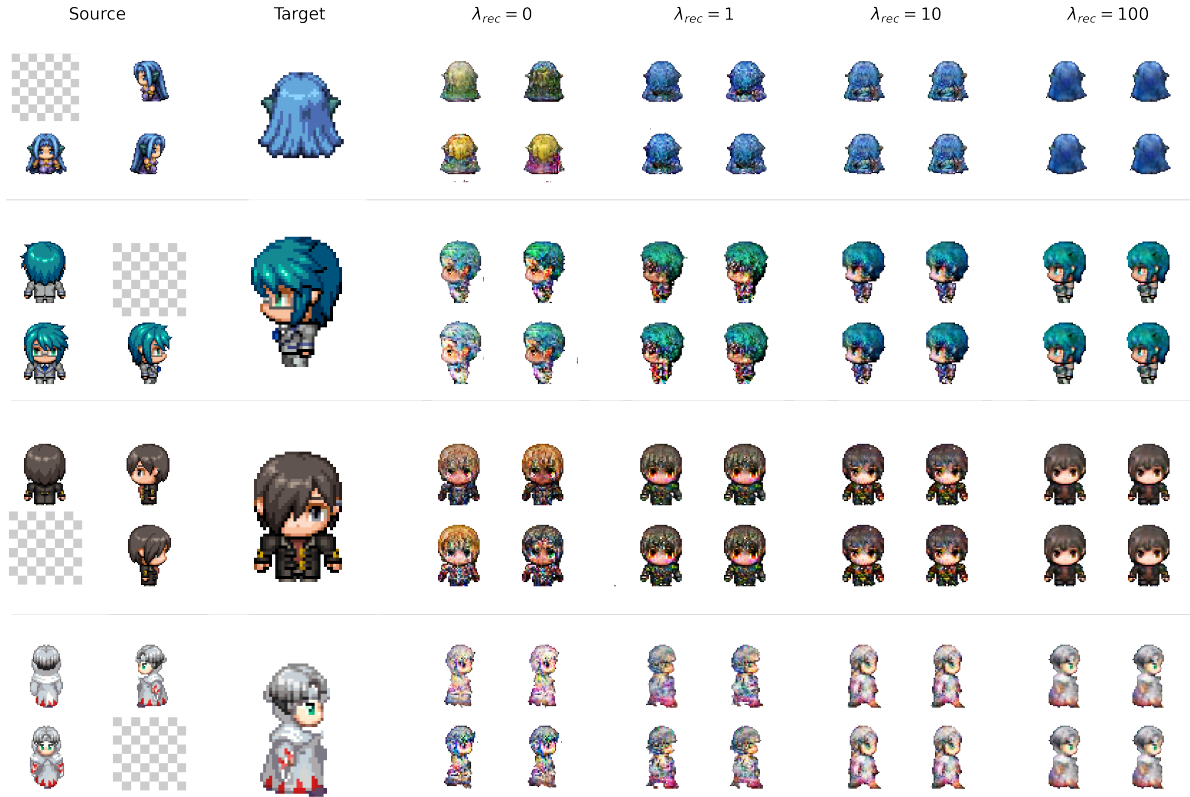
In step (a.1), the model first encodes the image tuples into a unified content code $c = E^{\text{content}}(x_S)$ and domain-specific style codes $s_i = E_i^{\text{style}}(x_i), \forall i \in S$. Then, in step (a.2), the corresponding decoders reconstruct the input images as $\hat{x}_i = G_i(s_i, c)$. The image consistency loss $\mathcal{L}_{\text{consis}}^{\text{img}}$ is computed as the reconstruction error between $\hat{x}$ and x .

To train the model in the other direction, random latent style codes $\hat{s}_i \sim \mathcal{N}(0, 1)$ are sampled and used together with the extracted content codes c to (b.1) generate new, potentially novel images $\hat{x}_i = G_i(\hat{s}_i, c)$. These images are evaluated by the discriminator D_i corresponding to their target domain, contributing to the adversarial loss term $\mathcal{L}_{\text{adv}}^{G_i}$. Next, in step (b.2), the generated images $\hat{x}$ are re-encoded to obtain reconstructed content and style codes $\hat{c}$ and $\hat{s}_i$, which are compared to their original values c and $\hat{s}_i$. These differences are captured by the latent consistency loss $\mathcal{L}_{\text{consis}}^{\text{lat}} = \mathcal{L}_{\text{consis}}^{\text{lat-cnt}} + \mathcal{L}_{\text{consis}}^{\text{lat-sty}}$. Finally, the reconstruction loss $\mathcal{L}_{\text{rec}}$ penalizes divergence between the images $\hat{x}$ generated with random style codes and their corresponding source images x .

After training the model with different λ -values for the loss terms, we observed that when λ_{rec} is large enough for $\lambda_{\text{rec}}\mathcal{L}_{\text{rec}}$ to dominate the total loss, the quality of the generated images improves. However, this occurs at the cost of the model collapsing to deterministic outputs, always producing the same image for a given input despite being conditioned on random style codes. In such cases, the $\mathcal{L}_{\text{consis}}^{\text{lat-sty}}$ term fails to decrease during training, suggesting that the model effectively ignores it. Figure 4.12 shows images produced by models trained with varying λ_{rec} values, illustrating how stochasticity decreases as λ_{rec} increases. When evaluating hyperparameters, we selected a relatively high λ_{rec} due to the resulting higher image quality.

We also evaluated different input dropout strategies when sampling batches from the training set, as done in CollaGAN. We found that uniformly choosing a random number of domains to drop yielded the best results for ReMIC.

Figure 4.12: Comparison of the images generated by ReMIC-based models trained with different λ_{rec} values and using four random style codes. Each cell displays four images with different random style codes.



The number of parameters that compose the networks in Representational Disentanglement Scheme for Multi-Domain Image Completion (ReMIC) is shown in Table 4.1. In total, there are 62,880,176 weights for the generation networks (style and content encoders and decoders) and 33,096,972 parameters for the discriminators. After training, an image can be generated in 15 ms using a GeForce GTX 1050 Ti GPU with 4 GB of VRAM.

Table 4.1: Number of parameters of the networks used by ReMIC.

	Network	Parameters	Quantity	Total Params.
Generation	Style Encoder	7,486,664	4	29,946,656
	Unified Content Encoder	5,429,120	1	5,429,120
	Decoder	6,876,100	4	27,504,400
	Total (Generator Nets)			62,880,176
	Discriminator	8,274,243	4	33,096,972

4.2.5 SpriteGAN

After presenting the four preceding models—each derived from existing image-to-image translation or missing-data imputation architectures—we shifted our focus toward designing a novel model from the ground up. This new architecture, which we call SpriteGAN, was specifically tailored for the task of generating pixel art characters in target poses given images of the same characters from one or more source directions.

Although its results have not yet reached the quality of the previous models, we believe that further exploration of hyperparameters and architectural refinements could lead to improvements. We nevertheless include SpriteGAN in this dissertation to document our design process and to provide a foundation for future research building upon this architecture.

Regarding its structure, SpriteGAN resembles ReMIC in that it is a multi-input, multi-output model. However, unlike ReMIC, SpriteGAN generates images for all domains simultaneously, rather than focusing on a target subset. This design choice arises because, in addition to imputing images for missing domains, SpriteGAN is also trained to inpaint randomly masked regions within the available source images x_S . Moreover, it is explicitly designed to be multi-modal, meaning it can produce diverse outputs from the same input.

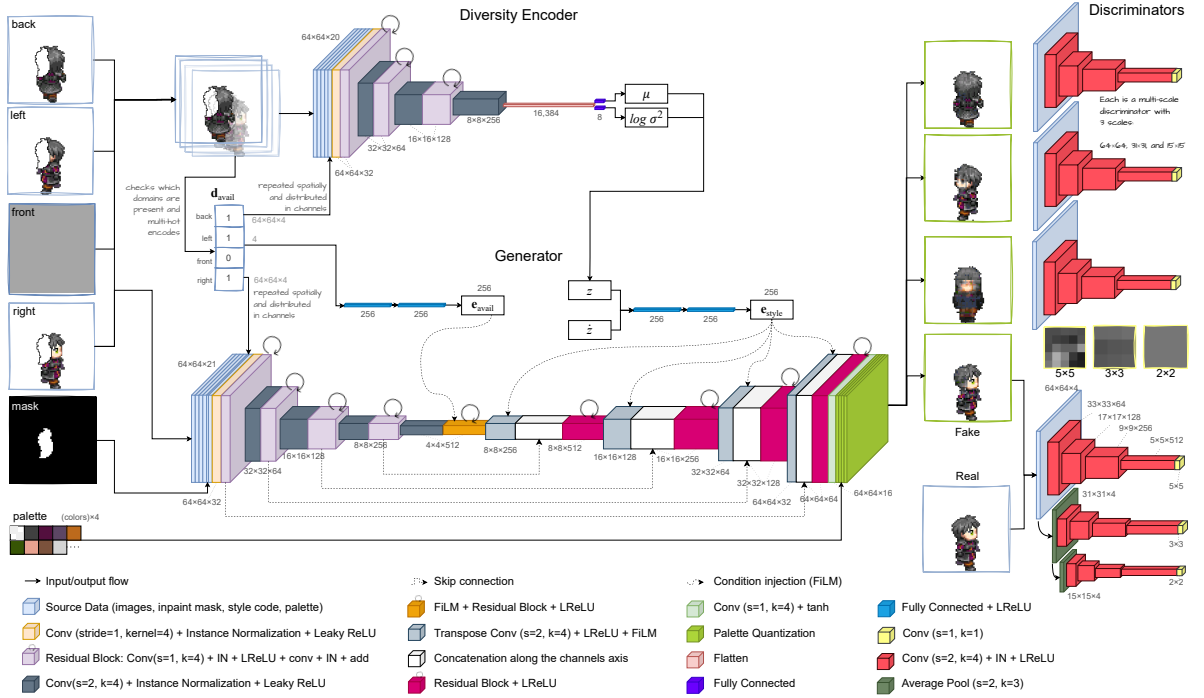
While ReMIC theoretically supports this behavior through its disentangled representation and random style codes, in practice it tends to collapse into deterministic outputs. This occurs because its regression loss enforces similarity between generated images and the ground truth even when random style codes are used. SpriteGAN addresses this limitation by treating images generated with random codes differently during training, thereby reducing mode collapse. The details of this mechanism are discussed in the following subsections describing its objective function and training procedure.

To achieve all those goals, the overall topology consists of a generator G , a diversity encoder E , and domain-specific discriminators $D_i, \forall i \in N$. Figure 4.13 shows a diagram that depicts the topology of the involved networks. The generator receives: the tuple with available and empty images x_S , some style code z , an inpaint mask M , and a target palette p ; and it produces a tuple of images $\hat{x}_N$ from all domains, with each image being either an inpainted version of the image x_i provided as input, or imputed from some missing domain. Due to a novel differentiable quantization operation we integrated into this model (which is later presented in Chapter 6 - Hypotheses on Pixel Art Generation), all generated images strictly follow the intended palette p . To formalize, a tuple of images $\hat{x}_N$ is generated as

$$\hat{x}_N = G_{\text{many} \rightarrow \text{all}}(x_S^{(M)}, z, M, p), \text{ with } S \subseteq N,$$

where $x_S^{(M)} = x_S \odot M$ represents the input images x_S that had some missing regions

Figure 4.13: Architecture of SpriteGAN. The **Diversity Encoder** receives the available domains and extracts their style z (length 8) by sampling from $\mathcal{N}(\mu, \sigma)$, which is then mapped to $\mathbf{e}_{style}$ (length 256) by a Multilayer Perceptron (MLP). Similarly, another MLP maps the multi-hot vector indicating the available domains $\mathbf{d}_{avail}$ into a $\mathbf{e}_{avail}$. The **Generator** receives all available images with erased regions $x_S^{(M)}$, the inpaint mask M , and the desired palette p , and generates images $\hat{x}_N$ in all domains. The embeddings $\mathbf{e}_{avail}$ representing the available domains conditions the bottleneck layer by Feature-wise Linear Modulation (FiLM), whereas those $\mathbf{e}_{style}$ representing the style condition the upsampling blocks. There are multi-scale **Discriminators** for each domain, and they receive either a real or a fake image and produce three grids of patches with realness scores.



erased by the inpaint mask M .

The diversity encoder E is similar to ReMIC's style encoder, as it extracts a latent code from each image. However, in our case, there is a single encoder instead of one per domain, and it sees all of the available poses of a character. Another difference to ReMIC is that it is modeled similar to a VAE encoder in that it outputs both mean (μ) and log-variance ($\log \sigma^2$) values, which are used to represent a distribution from which a $z \in \mathbb{R}^8$ can be sampled. It receives the tuple x_S with the available and missing images possibly with regions erased by the inpainting mask M :

$$\mu, \log \sigma^2 = E(x_S^{(M)}),$$

$$z = \mu + \sigma \cdot \epsilon,$$

where $\epsilon \sim \mathcal{N}(0, 1)$, as in the VAE reparameterization trick [Kingma and Welling, 2013]. Images can be produced both by using the extracted style code z or by sampling $\hat{z}$ directly

from a normal distribution $\mathcal{N}(0, 1)$, which is useful when different outputs for the same input images are required.

4.2.5.1 Objective Function

Similar to ReMIC, SpriteGAN separates the style information into a dedicated latent code, which is sampled from a normal distribution. Inspired by the objectives of BicycleGAN [Zhu et al., 2017a], SpriteGAN optimizes both image reconstruction and latent code reconstruction objectives. In addition, it introduces a cyclic reconstruction path, where images generated using random style codes are passed through the generator again to reconstruct the original inputs. This cyclic process encourages the model to produce diverse outputs when sampling random style codes, while still preserving the overall appearance and structure of the source images x_S .

The generator loss function is composed of four parts: image reconstruction $\mathcal{L}_{\text{rec}}$, latent reconstruction $\mathcal{L}_{\text{lat}}$, cyclic reconstruction $\mathcal{L}_{\text{cyc}}$, and an adversarial part $\mathcal{L}_{\text{adv}}$.

First, we select a batch of images $x_S^{(M)}$ with some randomly dropped-out domains and random regions erased by mask M . The diversity encoder E extracts their latent code $z \sim E(x_S^{(M)})$, and the generator reconstructs $\hat{x}_N = G(x_S^{(M)}, z, M, p)$. The image reconstruction loss term $\mathcal{L}_{\text{rec}}$ is a pixel-wise L_1 distance from the images $\hat{x}_N$ generated with the extracted code to their ground truth x_N :

$$\mathcal{L}_{\text{rec}} = \mathbb{E}_{x_N} [\|x_N - \hat{x}_N\|_1].$$

Next, the images $x_S^{(M)}$ are reconstructed using random codes $\dot{z}$ sampled from the normal distribution $\mathcal{N}(0, 1)$, which becomes $\dot{x}_N = G(x_S^{(M)}, \dot{z}, M, p)$. The style code is then extracted from those images, yielding $\hat{z} \equiv \hat{\mu}$ and $\hat{\mu}_- = E(\dot{x}_N)$. The latent reconstruction term $\mathcal{L}_{\text{lat}}$ then penalizes both the generator and encoders by not being able to encode the style in the generated images:

$$\mathcal{L}_{\text{lat}} = \mathbb{E}_{\dot{z}} [\|\dot{z} - \hat{z}\|_1].$$

As we presented in the ReMIC section, that architecture’s generator is penalized even for images generated with random style codes for diverging from the ground truth, which effectively makes the model ignore the style code, collapsing it into always producing the same output. In contrast, SpriteGAN uses a cyclic reconstruction term $\mathcal{L}_{\text{cyc}}$ that recreates the original images x_N from those generated with a random style $\dot{x}_N$ and their

originally extracted codes z as input, as

$$\begin{aligned}\tilde{x}_N &= G(\dot{x}_N, z, M, p), \\ \mathcal{L}_{\text{cyc}} &= \mathbb{E}_{x_N} [\|x_N - \tilde{x}_N\|_1].\end{aligned}$$

When generating the cycled images $\tilde{x}_N$, an empty mask $M = \emptyset$ is used, so the model can focus on the cyclic generation without using its capacity for inpainting. Lastly, the adversarial part uses a [LSGAN](#) loss which considers all the generated images (i.e, $\hat{x}$, $\dot{x}$ and $\tilde{x}$). In turn, the discriminator considers the real images x_N and all that were generated, $\hat{x}_N \cup \dot{x}_N \cup \tilde{x}_N$

$$\begin{aligned}\mathcal{L}_{\text{adv}}^G &= \mathbb{E}[\sum_{i \in N} (D_i(\hat{x}_i \cup \dot{x}_i \cup \tilde{x}_i) - 1)^2], \\ \mathcal{L}_{\text{adv}}^{D_i} &= \mathbb{E}[(D_i(x_i) - 1)^2] + \mathbb{E}[(D_i(\hat{x}_i \cup \dot{x}_i \cup \tilde{x}_i) - 0)^2], \\ \mathcal{L}_{\text{adv}}^D &= \sum_{i \in N} \mathcal{L}_{\text{adv}}^{D_i}.\end{aligned}$$

The encoder E has an additional Kullback-Leibler divergence loss $\mathcal{L}_{\text{KL}}$ that steers it towards using the normal distribution $\mathcal{N}(0, 1)$ over each dimension of the latent space $z \in \mathcal{R}^8$, such that

$$\mathcal{L}_{\text{KL}} = \mathbb{E}_{x_N} \left[-\frac{1}{2} \sum_{i=1}^{|z|} (1 + \log \sigma_i^2 - \mu_i^2 - \sigma_i^2) \right].$$

The final objective for each network is given by a weighted sum of their individual terms multiplied by λ scalars

$$\begin{aligned}\mathcal{L}_G &= \lambda_{\text{adv}} \mathcal{L}_{\text{adv}}^G + \lambda_{\text{rec}} \mathcal{L}_{\text{rec}} + \lambda_{\text{lat}} \mathcal{L}_{\text{lat}} + \lambda_{\text{cyc}} \mathcal{L}_{\text{cyc}}, \\ \mathcal{L}_E &= \lambda_{\text{lat}} \mathcal{L}_{\text{lat}} + \lambda_{\text{KL}} \mathcal{L}_{\text{KL}}, \\ \mathcal{L}_D &= \lambda_{\text{adv}} \mathcal{L}_{\text{adv}}^D.\end{aligned}$$

4.2.5.2 Generator and Encoder

The generator is loosely inspired by U-nets, as it contains skip connections between each resolution pair of the encoder and decoder halves, and receives and produces images with the same resolution. However, it uses different mechanisms for conditioning the generation on the available domains and also on the intended style code. It receives a batch of inputs composed of the source images $x_S^{(M)}$ which can have missing regions, a style code z , the inpaint mask M , and a target palette p . As output, it produces the images from all domains.

First, the network detects which domains are available and which are absent, and produces a domain availability multi-hot label $\mathbf{d}_{\text{avail}} \in \{0, 1\}^4$. That vector is repeated in the vertical and horizontal axes and, together with the inpainting mask M , concatenated with the input images along the channels axis. That tensor has a $64 \times 64 \times 21$ shape (16 from the images, 1 from the mask, 4 from the domain availability data) and it is fed into the generator’s encoder.

Concurrently, two **MLP** networks map the style code z and the domain availability label $\mathbf{d}_{\text{avail}}$ into embeddings $\mathbf{e}_{\text{style}}$ and $\mathbf{e}_{\text{avail}}$ in a higher-dimensional $\mathbb{R}^{256}$ space. While the style embedding is mixed-in at each decoder stage, the domain availability embedding modulates the feature maps at the bottleneck level only. Both use a **FiLM** operation [Perez et al., 2017] to condition the generation towards using the provided style and to consider only the domains that are available in each input situation, respectively. Each **MLP** consists of two sequences of a dense layer with 256 units and a Leaky **ReLU** activation.

The generator’s encoder has an initial convolution block composed of a convolution with 32 filters, instance normalization, and a Leaky **ReLU** activation. Then, the activation maps go through a sequence of four downsampling blocks (with 64, 128, 256, and 512 filters) with each consisting of a residual block (convolution, instance normalization, Leaky **ReLU**, convolution, instance normalization, addition with the input) and another convolution block that effectively downscales (stride 2) the activation maps.

Inside the bottleneck, a **FiLM** operation conditions the $4 \times 4 \times 512$ features on the domain availability embeddings $\mathbf{e}_{\text{avail}}$, followed by a residual block and a Leaky **ReLU** activation. Then, the decoder uses an analogous structure but upscales the activation maps using a transpose convolution at each stage. In addition, at each resolution level, a **FiLM** operation mixes in the style embeddings $\mathbf{e}_{\text{style}}$ right before the concatenation of the respective skip connection from the encoder.

As the last stage of the decoder, the activation maps go through a sequence of convolution, tanh activation and a palette quantization operation, producing $\hat{x}_N$, which is a $64 \times 64 \times 16$ tensor, as it contains the images from all domains. This last operation is a novel differentiable quantization method we proposed during our research, formally described in Chapter 6 (Hypotheses on Pixel Art Generation). It makes the generated images strictly follow an intended palette while using only differentiable operations, which allows training models end-to-end.

The diversity encoder receives the images $x_S^{(M)}$ from all domains, concatenates them with the spatially repeated domain availability multi-hot tensor, and processes it through a sequence of four downsampling blocks (32, 64, 128, and 256 filters), composed of the same operations (convolution, instance normalization, Leaky **ReLU**) as the blocks from the generator. When the shape of the activation maps become $8 \times 8 \times 256$, they are flattened and processed by two dense layers with 8 units each that form the network’s output: one represents the mean of a gaussian distribution and the other its log-variance.

4.2.5.3 Discriminators

As in ReMIC, $|N|$ multi-scale patch discriminators classify same-size rectangular patches for each domain. They also use three scales processed by a separate branch and receive 64×64 images. There are four downsampling blocks that use one convolution + instance normalization + Leaky ReLU activation (64, 128, 256, and 512 channels) operations, and a final 1×1 convolution that has a single filter at the end.

4.2.5.4 Training Procedure

At each training step, the generator and the diversity encoder are updated jointly but using their respective loss functions, and the discriminators are updated according to theirs.

First, a batch x_S is selected using the conservative input dropout strategy, which has higher chances (60%) of having only a single missing domain. Random inpaint masks M are created for each example, considering the combined shape of each character in all domains: only pixels that are inside its shape *in any domain* are candidates for being erased by the mask. In addition, the available images in *all domains* have the pixels erased by the generated mask. The masks M are applied to the batch of images x_S , resulting in the $x_S^{(M)} = x_S \odot M$ input tensor.

An inpaint mask generator produces a random mask for each character by selecting a number of holes from 1 to 4, each with a different radius and position inside the character’s shape. Even though all holes are circular, their union is intersected with each character’s alpha channel and the resulting shapes ends up composing highly diverse masks.

Next, the palettes p are extracted from the combined images from each character in all domains, to be used as the target palette by the quantization layer in the generator.

The networks used in the generation are trained first. The diversity encoder E extracts the style code from the masked source images, producing μ and $\log \sigma^2$. The mean and log-variance outputs are used with the reparameterization trick [Kingma and Welling, 2013] to sample a z with an $\epsilon \sim \mathcal{N}(0, 1)$, by having

$$\begin{aligned}\mu, \log \sigma^2 &= E(x_S^{(M)}), \\ z &= \mu + \sigma \cdot \epsilon.\end{aligned}$$

Next, the generator creates both $\hat{x}_N$ and $\dot{x}_N$ by using the extracted z and randomly

sampled $\dot{z} \sim \mathcal{N}(0, 1)$ from the first and second halves of the batch:

$$\begin{aligned}\hat{x}_N &= G(x_S^{(M)}, z, M, p), \\ \dot{x}_N &= G(x_S^{(M)}, \dot{z}, M, p).\end{aligned}$$

Then, the diversity encoder E extracts the style codes from the images generated with the randomly sampled values $\dot{x}_N$ so they can be compared to the sampled values $\dot{z} \sim \mathcal{N}(0, 1)$ and be used in the $\mathcal{L}_{\text{lat}}$ loss:

$$\begin{aligned}\hat{\mu}, - &= E(\dot{x}_N), \\ \hat{z} &= \hat{\mu}.\end{aligned}$$

The images $\dot{x}_N$ generated from the random codes are then used as input to the generator in a backward pass, in which it tries to cyclically reconstruct the original x_N by using the extracted z style code and an empty inpaint mask $M_\emptyset = \emptyset$:

$$\tilde{x}_N = G(\dot{x}_N, z, M_\emptyset, p).$$

The generated images $\hat{x}_N$ and $\tilde{x}_N$ are compared with the ground truth using the $\mathcal{L}_{\text{rec}}$ and $\mathcal{L}_{\text{cyc}}$ loss terms, respectively. In contrast, the images $\dot{x}_N$ are intentionally excluded from direct reconstruction losses. Including them would cause the generator to disregard the latent code entirely—an issue we observed in ReMIC when λ_{rec} was set to a non-zero value. Instead, in SpriteGAN, the generator relies on the cyclic, latent, and adversarial losses to encourage diversity, ensuring that each latent sample $z \sim \mathcal{N}(0, 1)$ yields a distinct output $\dot{x}_N$. All three categories of generated images (i.e., $\hat{x}_N$, $\dot{x}_N$, and $\tilde{x}_N$) are used by the discriminators to compute the generator’s adversarial loss $\mathcal{L}_{\text{adv}}^G$.

The final stage of the training step involves updating the discriminators. Real images x_N and generated samples $\hat{x}_N \cup \dot{x}_N \cup \tilde{x}_N$ are first randomly shuffled and then processed by each discriminator D_i , which outputs patch-level logits for every image. As in standard GAN training, the discriminators are penalized for misclassifying real patches as fake and vice versa.

The networks that compose SpriteGAN contain a number of trainable parameters described in Table 4.2. The generation networks (encoder E and generator G) contain 30,904,289 parameters while the discriminators collectively have 33,108,492 trainable variables. The time to inference, to produce images in the four domains, is 32 ms using a GeForce GTX 1050 Ti GPU with 4 GB of VRAM.

Table 4.2: Number of parameters of the networks used by SpriteGAN.

	Network	Parameters	Quantity	Total Params.
Generation	Diversity Encoder	1,650,544	1	1,650,544
	Generator	29,253,745	1	29,253,745
	Total (Generator Nets)			30,904,289
	Discriminator	8,277,123	4	33,108,492

4.3 Architecture Comparison

In this section, we summarize the main characteristics of the proposed models. Once the training of a GAN is complete, the discriminator can be discarded, as its sole purpose is to guide the generator toward producing images that resemble real ones. Therefore, the following discussion focuses on the characteristics of the generators.

Table 4.3 shows the number of trainable parameters and layers of a single generator, the time taken to create an image (i.e., the inference time), the number of generators, and the total storage size of each proposed model. We measured the performance using a GeForce GTX 1050 Ti GPU with 4 GB of VRAM and running the model using Tensorflow 2.16.2 in Python 3.9.

It is important to note that the Pix2Pix-based architecture requires training a separate model for each source–target domain pair, amounting to $|N| \times (|N| - 1)$ generators, as models with identical source and target domains are excluded. Consequently, Pix2Pix demands substantially more storage space compared to the other architectures. Regarding inference time, i.e., the duration required to generate a batch of images, it generally increases with model depth. Among the proposed models, SpriteGAN exhibits the highest inference time of 32.13 ms, whereas Pix2Pix achieves 7.23 ms due to its shallower topology.

Table 4.3: Number of generator’s trainable parameters, number of layers, inference time, number of generators and total size of the proposed models.

Base Arch.	Parameters	Layers	Inference Time (ms)	Generators	Total Size (MB)
Pix2Pix	29,307,844	20	7.23	12	1,356
StarGAN	134,448,128	63	10.03	1	514
CollaGAN	104,887,616	166	17.28	1	404
ReMIC	62,880,176	90	14.54	1	247
SpriteGAN	30,904,289	101	32.13	1	237

In addition to the storage and inference statistics, other characteristics of each model can also be examined. Table 4.4 summarizes these aspects, which are the number

of supported domains, whether the model uses a single or all available input images, whether it can generate all missing images in a single feed-forward pass, its ability to produce diverse outputs for the same input (multi-modality), and the ability to inpaint missing regions from an image.

Table 4.4: Comparison of the desired capabilities of the proposed models.

Base Model	Domains	Input	Output	Multi-Modal	Inpaint
Pix2Pix	Two	1 image	1 image	No	No
StarGAN	Multi	1 image	1 image	No	No
CollaGAN	Multi	all available	1 image	No	No
ReMIC	Multi	all available	all missing	No	No
SpriteGAN	Multi	all available	all domains	Partial	Yes

The following chapter presents the experiments conducted to develop the proposed models, tune their hyperparameters, and evaluate their performance in generating pixel art characters in different poses.

Chapter 5

Experiments

In this chapter, we present a series of experiments conducted using the models proposed in Chapter 4 (Model Proposition). Section 5.1 (Methodology) describes the experimental setup, including the datasets, evaluation metrics, and training procedures employed. Next, Section 5.2 (General Architecture Decisions) reports experiments that are not tied to a specific model but instead assess broader architectural and design choices—such as the color space used for image representation and the impact of data augmentation.

Sections 5.3 (Translation Using a Single Input) and 5.4 (Translation Using Multiple Inputs) then present the comparative evaluation of the first four proposed models. These sections focus on the tasks of translating images between domains using either a single source image or multiple source images as input. Section 5.5 (Results from SpriteGAN) presents initial findings that illustrate SpriteGAN’s performance in image imputation, multi-modality, and inpainting tasks.

Finally, Section 5.6 (Discussion) provides an overall assessment of the proposed models, highlighting their relative strengths and limitations, and suggesting potential directions for future research.

5.1 Methodology

We start by presenting the datasets used in the experiments to propose and evaluate models for generating pixel art characters in different poses. Next, we describe the L_1 (mean absolute error) and Fréchet Inception Distance (FID) metrics used to analyze the images generated by each model, and introduce the general training procedure.

Figure 5.1: Sample images from the datasets showing two examples of each of the four different sizes/art styles (**columns**), in four directions (**rows**). From left to right, the datasets are Tiny Hero, RPG Maker 2000, RPG Maker XP, and RPG Maker VX Ace.



5.1.1 Datasets

Differently from tasks that are more commonly tackled in Computer Vision research, we found no ready-to-use dataset of characters depicted in different directions and with a large number of samples. Hence, we initially assembled datasets of pixel art character sprites from four different sources: TINY HERO¹, RPG MAKER 2000², RPG MAKER XP², and RPG MAKER VX ACE². They consist of paired examples of characters in four directions, i.e. front, right, back, and left, spanning four different character sizes and art styles. Two samples from each dataset can be seen in Figure 5.1.

Images from each source had different character sizes, so the smaller ones were transparency-padded to the largest size, 64×64 . Also, we created an alpha channel with the character shape for the images that lacked one, so we could represent the images using the RGBA channels. Table 5.1 presents the individual image sizes, number of images per character pose, total number of samples, the size of the training and test partitions.

Regarding the number of images per pose, some sources (all except TINY HERO) had three frames for each character side, as they were extracted from sprite sheets of walking animations. Each frame was treated as a different example such that the samples {1, 2,

¹Dual license of GNU GPL 3.0 and CC-BY-SA 3.0.

²Custom license in which the End User License Agreement (<https://www.rpgmakerweb.com/eula>) allows the use of the assets, without the possibility of distribution, for users that own a copy of the software.

3} of each dataset depict the first character, {4, 5, 6} depict the second and so on. During the train/test split (of 85%), the last character from the training set might have 0, 1 or 2 examples present in the test set. Even though a character might have frames spread between the train and test partitions, we do not consider it as a harmful data leakage, as it happens with a single character per dataset, which has negligible impact.

Table 5.1: Description of the datasets used in the experiments.

Source	Size	Per pose	Examples	Train	Test
TINY HERO	64×64×4	1	912	776	136
RPG MAKER 2000	32×24×3	3	216	184	32
RPG MAKER XP	48×32×4	3	294	250	44
RPG MAKER VX ACE	32×32×4	3	408	347	61
LARGER (all above+more)	Assorted	Assorted	14,202	12,074	2,128

The **Tiny Hero** sprites were created in the context of the Liberated Pixel Cup³, a competition that featured different artists designing many character body parts and accessories following a cohesive art style. The sprites were modularly created by assembling such previously-drawn pieces to form each character. For this reason, it is usual for different characters to present the same or similar shapes but with different colors or a distinct combination of parts. This dataset is the largest of the four but contains the least variability due to the use of modular parts. Hence, it can be considered an easier task for models training with it to perform pose translation.

The characters sourced from **RPG Maker 2000** compose the smallest dataset, but its test split consists of mostly color variations of characters present in the training set. Therefore, the reduced variability between the train and test splits, present mostly on differences in the color palettes, makes the task of translating sprites of this dataset an indicator of the ability of the trained models to perform color transfer of the characters. That makes this a task easier than translating completely unseen characters to different poses.

The datasets created from the remaining **RPG Maker XP** and **RPG Maker VX Ace** sources constitute a more challenging task of performing pose translation of very different characters, as they present large variability between the train and test partitions, while having only small numbers of training examples.

Due to the limited training sizes of the four datasets, we also investigated training models with all of them together, plus a collection of 12,372 assorted pixel art sprites. These comprise primarily humanoid characters in different sizes and art styles but also a few sprites of animals, vehicles, and monsters. By including all four initial datasets

³The website of the Liberated Pixel Cup (<https://lpc.opengameart.org/>), which was mostly active during 2012 and 2013.

plus the 12k assorted sprites, this eclectic dataset contains 14,202 paired examples, and we called it the **Larger dataset** (Table 2.1, shown in the last row). We made it publicly available by the name Pixel Art Characters (PAC) Dataset.⁴

5.1.2 Evaluation Metrics

The evaluation of generative models is an active research problem with different metrics proposed over the recent years [Buzuti and Thomaz, 2023]. For the image-to-image translation and missing-data imputation tasks tackled in this work, we evaluate the quality of a model by how close the generated images are from their ground truth. However, we observe that a qualitative analysis is important as the metrics do not always converge.

Hence, we analyze the results qualitatively through visual inspection and quantitatively using the L_1 distance to the target images and the Fréchet Inception Distance (FID) [Heusel et al., 2017]. While the L_1 distance measures the pixel-wise absolute difference of colors between two images, FID measures the distance between the feature vectors of the generated and the real images, using the Inception v3 network (proposed for image classification) [Szegedy et al., 2016]. Both metrics are zero for identical images, so lower numbers indicate more similar generated and target images.

5.1.3 Training Procedure

We used different models and datasets for each proposed experiment. Initially, we investigated some general architecture decisions using the first model, based on Pix2Pix. After selecting the configurations that yielded the best results, we used the LARGER dataset to compare the quality of the images generated by all proposed models.

The training procedure optimizes the networks with Adam for a number of generator update steps instead of full epochs. The models trained for 40,000 when using the individual datasets and 240,000 for the LARGER one (approximately 80 epochs). We found that when training with the smaller datasets ($\leq 1k$ samples), the models converged at approximately the same number of generator updates, but it takes longer for the LARGER dataset ($\approx 12k$) due to its much higher variability.

⁴PAC Dataset: <https://github.com/plucksquire/pac/>.

At each update step, the generator receives a mini-batch of 4 training examples. We use early stopping to select the model that had the best metrics on its test set, instead of getting the one in the end, to prevent overfitting. Each model’s hyperparameters are shown in Table 5.2.

Table 5.2: Hyperparameters used for training each model.

Base Architecture	Hyperparameter	Value
Pix2Pix	λ_{rec}	100
	learning rate	0.0002
StarGAN	λ_{rec}	100
	λ_{dmn}	1
	learning rate	0.0002
CollaGAN	λ_{rec}	100
	λ_{ssim}	10
	λ_{dmn}	10
	λ_{mcyc}	10
	learning rate	0.0001
ReMIC	$\lambda_{\text{consis}}^{\text{img}}$	10
	$\lambda_{\text{consis}}^{\text{lat}}$	1
	λ_{rec}	100
	learning rate	0.0001
All models	Adam β_1	0.5
	Adam β_2	0.999

5.2 General Architecture Decisions

Here, we describe some experiments conducted to help us decide on the training procedure, such as choosing the image representation (RGB vs. RGBA), the impact of using a larger and more varied dataset, and the effect of different types of data augmentation techniques. We targeted only the Pix2Pix-based models in these tests, as they were the first proposed and developed. As such models translate between only two domains, we selected the front-to-right translation direction because it represents a challenging task, as it cannot be solved through a horizontal flip, such as in the right-to-left case (or vice versa). We observed which settings yielded the best results for those models, and we carried their configurations into the other architectures.

5.2.1 RGB vs RGBA

We experimented with training the model to receive, process, and output images with the RGB components and the added alpha channel, RGBA. In character sprites, to create transparency, the images typically have either an alpha channel or a background key color, which the game engine ignores when rendering.

Although the alpha channel is mainly used as a boolean mask (as is the case in the majority of the examples from the datasets used in this work), and a color key would also fulfill the same purpose, we found that our GAN models do use the mostly redundant information of the alpha channel, allowing them to generate shapes with higher fidelity to the ground truth and avoid dangling pixels outside the expected shape region. Figure 5.2 shows a comparison of images generated by models using RGB and RGBA representation.

Figure 5.2: Comparison of model outputs when training with RGB vs. RGBA channels.



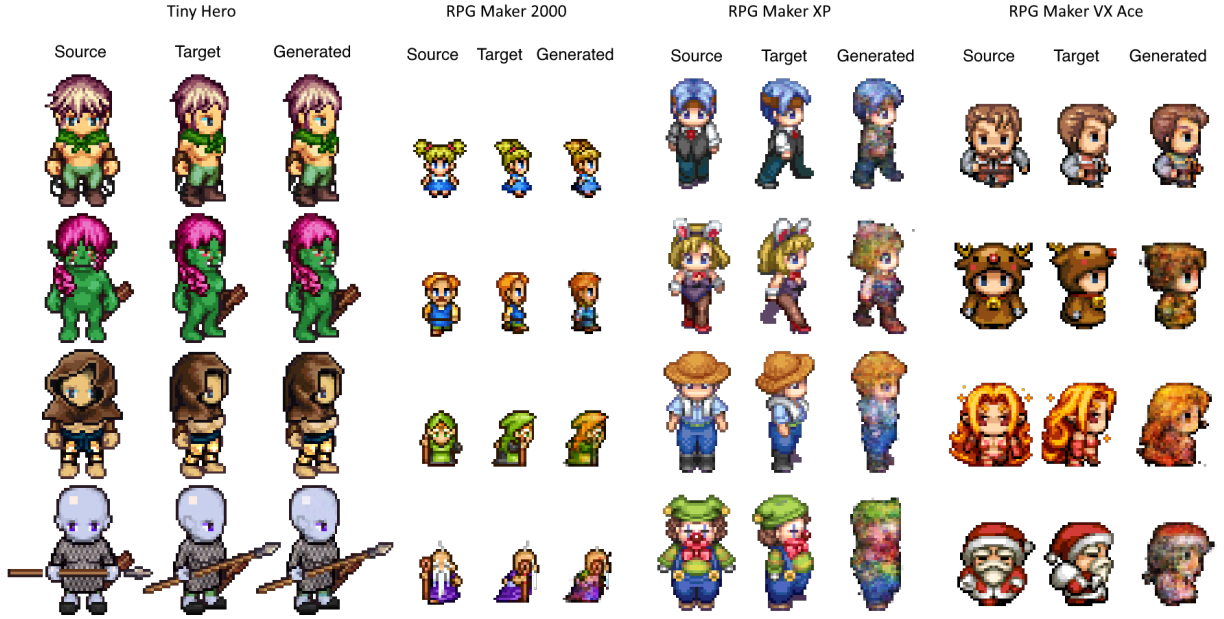
5.2.2 Different Datasets

Using the Pix2Pix-based model as our initial testbed, we trained separate models on each smaller dataset and evaluated the quality of their generated images (Figure 5.3). The results, which varied significantly across datasets, revealed different model behaviors:

Tiny Hero: The generated images closely match the ground truth. This high fidelity suggests the model successfully learned to recombine seen character components (e.g., arms, heads, accessories) into novel assemblies. However, this was the least challenging task, as the test partition consisted of similar modular combinations.

RPG Maker 2000: Output quality remains good, with primary deviations in color usage. Since the test set usually contains palette-swapped variants of characters from the

Figure 5.3: Output images generated with models trained on individual datasets in the order TINY HERO, RPG MAKER 2000, RPG MAKER XP, and RPG MAKER VX ACE.



training partition, this indicates a partial ability to transfer colors from known (or similar) characters to new palettes.

RPG Maker XP: Quality is inconsistent, ranging from poor (e.g., the clown) to resembling the ground truth. Higher-quality generations, such as the character in the first row, occur for test samples similar to training data. Distinctive, low-frequency features (e.g., bunny ears, farmer hats) are often lost. This pattern suggests limited generalization capacity on a test set containing truly novel characters, not mere color variations.

RPG Maker VX Ace: Results reinforce the insights from RPG MAKER XP. Image quality degrades as character features become more distinctive and less frequent in the training set, further indicating the model’s generalization limitations.

Given the observed limitations in generalization, we hypothesized that training on a larger, more diverse dataset could enhance model performance. To test this, a Pix2Pix-based model was trained on the combined LARGER dataset, and Table 5.3 shows the observed metrics. First, we established a baseline for in-distribution performance by evaluating it on its own test partition (i.e., data following the same distribution as its training set), which yielded the metrics FID with 3.319 and an L_1 with 0.05375. Next, to explicitly assess the impact of the increased scale and diversity on out-of-distribution generalization, we then evaluated this same model on the test partitions of the four individual smaller datasets. These metrics, which capture the model’s ability to handle

Table 5.3: Training on the larger dataset but validating on the individual ones (test set).

Calculated with test set from	Best FID		Best L_1	
	Indiv.	LARGER	Indiv.	LARGER
TINY HERO	0.136	0.553	0.00754	0.01724
RM2K	2.908	4.713	0.01168	0.01468
RMXP	9.623	8.582	0.06068	0.05971
RMVX	4.990	4.804	0.04704	0.04490
LARGER	—	3.319	—	0.05375

distinct art styles and character types not seen during its training, are presented in the first four rows of Table 5.3.

Training the model on the LARGER dataset yields FID and L_1 values that are worse for TINY HERO and RPG MAKER 2000 but better for RPG MAKER XP and RPG MAKER VX ACE, which represent more complex tasks. In the case of TINY HERO, having contact with many diverse art styles from the large set hindered the model’s ability to memorize individual character parts. Regarding RPG MAKER 2000, using the LARGER dataset to train also degraded the results. That can be attributed to the fact that RPG MAKER 2000’s test partition is mostly comprised of color variations of the training set, and seeing much variance during training made focusing on color transfer more difficult.

Regarding RPG MAKER XP and VX ACE, training with much more examples yielded better results. As their test sets have a lot more variability than the other datasets, when trained with the increased variation in shapes and colors, the model might have been able to understand better how it can “turn” characters from one pose to another.

Figure 5.4 shows some images generated by the model trained with the LARGER dataset (4th row) and by the corresponding original individual dataset (3rd row). We can observe that some results improved with the additional training data (such as in columns 2, 5, 6), some might have as well (1, 7), but others just changed without necessarily enhancing (3, 4, 8).

To summarize, we can observe that having a larger dataset with more variability hinders the output quality for tasks where color transfer or memorization of parts is sufficient. But introducing more variation in the training data can enhance the results especially for the more challenging tasks. That leads us to hypothesize that having an even larger dataset could improve the quality even more.

Figure 5.4: Comparison of images generated when training using the original individual dataset (3rd row) and the larger one (4th row). Examples come originally from RPG MAKER 2000 (column 1), RPG MAKER XP (columns 2-4, 8), and RPG MAKER VX ACE (columns 5-7).



5.2.3 Data Augmentation

Especially useful for small datasets, data augmentation can help attenuate the situation where the model overfits to the training data, yielding better results for unseen data [Krizhevsky et al., 2017; Karras et al., 2021]. However, due to some characteristics of pixel art and from the task approached in this work, some augmentation options cannot be used. For instance, rotation by oblique angles and scaling transformations produce degraded images that do not conform to the art style. Moreover, flipping and rotation by non-oblique angles could change the domain of the images (e.g., character facing right, when horizontally flipped, starts facing left), or it could put characters in a non-existent pose (e.g., upside down).

We implemented two transformations: hue rotation and translation of the character inside the 64×64 region, and studied their impact when applied together and separately. In our experiment, there is an 80% probability for an input to undergo augmentation during training. The sprite’s hue is rotated by a random angle between -180 and 180 degrees, while the translation moves the character inside the 64×64 area by some random offset.

We trained Pix2Pix-based models with the front-to-right direction on each dataset

Table 5.4: FID and L_1 values of the generated images per augmentation technique.

Dataset	Augmentation	Best FID	Best L_1
TINY HERO	none	0.135	0.00754
	hue rotation	0.174	0.00973
	translation	0.917	0.02056
	both	0.895	0.02086
RPG MAKER 2000	none	2.908	0.01168
	hue rotation	2.056	0.00888
	translation	7.874	0.01826
	both	7.325	0.01630
RPG MAKER XP	none	9.623	0.06068
	hue rotation	9.687	0.05934
	translation	9.814	0.06002
	both	9.400	0.05887
RPG MAKER VX ACE	none	4.990	0.04704
	hue rotation	5.237	0.04497
	translation	6.537	0.04640
	both	5.808	0.04617
LARGER	none	3.319	0.05375
	hue rotation	3.623	0.05343
	translation	3.862	0.05593
	both	3.948	0.05624

and compared the effects of both transformations together, using only hue rotation, only translation, and no augmentation. Table 5.4 presents the values of FID and L_1 for each experiment.

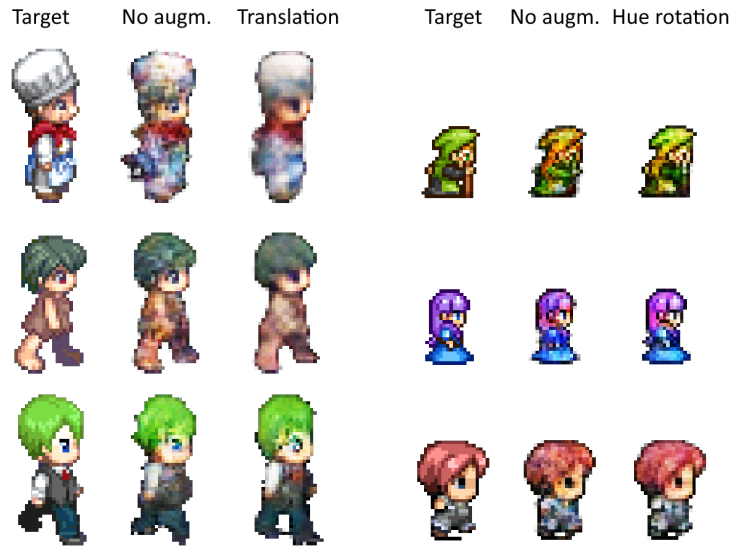
Quantitatively, none of the experimented scenarios (no technique, only hue rotation, only translation, both) scored the best values for all datasets, indicating that its use must be carefully considered. However, we observed some repeating behaviors:

- Hue rotation improved the quality of the generated sprites that had their shape seen during training but in a different color palette (color transferring). RPG MAKER 2000 had various color variation sprites in its test split—hence, its more significant gain with this augmentation technique. The right side of Figure 5.5 shows some examples.
- Translation acted as a strong regularizer for the model, hampering its ability to memorize (and hence, overfit to) the training data. This led the generated images to have less noise at the expense of some detail, even on the training set. The left side of Figure 5.5 shows examples.
- Using both augmentation techniques together generated better images than with translation alone but worse than with hue rotation only (as in TINY HERO, RPG

MAKER 2000, and RPG MAKER VX ACE).

- In TINY HERO, all augmentation techniques produced worse results, probably due to the dataset’s modular nature. We conclude that any regularization added to the model (in the form of augmentation) hindered the generator from being able to memorize individual character parts.

Figure 5.5: Example images generated after training with data augmentation techniques of translation (**left**) and hue rotation (**right**). Examples generated with translation (left) come from RPG MAKER XP (first and second rows) and the LARGER dataset (third). The ones generated with hue rotation (right) come from RPG MAKER 2000 (first and second rows) and RPG MAKER VX ACE (third).



5.3 Translation using a Single Input

We trained the proposed models following their respective procedures, described in Chapter 4 (Model Proposition), and choosing the best architectural decisions from the previous section: RGBA representation, use of the LARGER dataset, and hue rotation as data augmentation.

In this section, we evaluate the quality of the generated images in the scenario where each proposed model receives a single input image and must perform translation into a target pose. Both the Pix2Pix and StarGAN-based models can only receive a single image as input, but the CollaGAN and ReMIC-based ones can receive images from all available domains. Nonetheless, we included them in this experiment to assess their

performance when only one image is provided. To indicate the number of images used as input to those models, we write CollaGAN-**X** and ReMIC-**X**, with **X** referring to the number of input domains they are using.

Table 5.5: FID and L_1 values for Pix2Pix, StarGAN, CollaGAN-1, and ReMIC-1 models. Each cell represents the value of images generated from a source pose (**row**) into a target (**column**). We highlighted the best average values for each target pose among the different models.

	Target↘ ↓ Source	FID				L_1			
		Back	Left	Front	Right	Back	Left	Front	Right
Pix2Pix	Back	—	3.575	4.184	4.059	—	0.06510	0.05240	0.06385
	Left	6.728	—	5.692	0.991	0.05891	—	0.06161	0.02468
	Front	3.033	2.662	—	3.368	0.04393	0.05783	—	0.05788
	Right	7.601	0.902	6.301	—	0.05921	0.02508	0.06223	—
	Average	5.788	2.380	5.392	2.806	0.05402	0.04934	0.05875	0.04880
StarGAN	Back	—	1.517	2.621	1.542	—	0.07958	0.06386	0.07838
	Left	3.620	—	3.574	0.996	0.07025	—	0.07657	0.03919
	Front	3.108	1.303	—	1.565	0.05192	0.07127	—	0.07061
	Right	3.408	0.930	3.274	—	0.07069	0.03946	0.07747	—
	Average	3.378	1.250	3.156	1.368	0.06429	0.06344	0.07263	0.06273
CollaGAN-1	Back	—	8.581	2.735	7.085	—	0.07463	0.05559	0.07835
	Left	16.401	—	13.345	1.764	0.07254	—	0.07456	0.03480
	Front	2.707	5.168	—	5.562	0.04971	0.06940	—	0.07217
	Right	19.117	2.157	16.093	—	0.07630	0.03516	0.08066	—
	Average	12.741	5.302	10.725	4.803	0.06150	0.05750	0.07027	0.06177
ReMIC-1	Back	—	4.624	5.978	5.441	—	0.06743	0.05625	0.06820
	Left	8.520	—	7.654	1.134	0.05931	—	0.06273	0.02917
	Front	5.517	3.703	—	3.722	0.04616	0.05962	—	0.06035
	Right	8.731	1.073	8.992	—	0.05938	0.02971	0.06353	—
	Average	7.589	3.133	7.541	3.432	0.05495	0.05225	0.06084	0.05257

Table 5.5 shows the values of FID and L_1 distance between the generated images and the ground truth for each model. The rows indicate the source pose and, the columns, the target domain. We omitted the translation of images from and to the same domain from the table (i.e., the empty diagonals) as the models learn an identity function that simply reproduces the input image. We will analyze the table from two perspectives: comparing the performance of each model and the difficulty of translating between different domains (the translation direction).

First, let us consider the metric values averaged among all source domains to each target pose, indicated by the four aggregate rows (labeled “Average”) at the end of each model grouping. The metrics do not converge: while StarGAN has the lowest (best) FID

values (i.e., 3.378, 1.250, 3.156, and 1.368 for the targets back, left, front, and right), Pix2Pix has the lowest (best) L_1 distances (i.e., 0.05402, 0.04934, 0.05875, and 0.04880). We also note that the CollaGAN-1 model has much higher FID and higher L_1 values than the other models, indicating that its generated images are further from the ground truth, while ReMIC-1 ranks in the third place, with better metrics than CollaGAN-1.

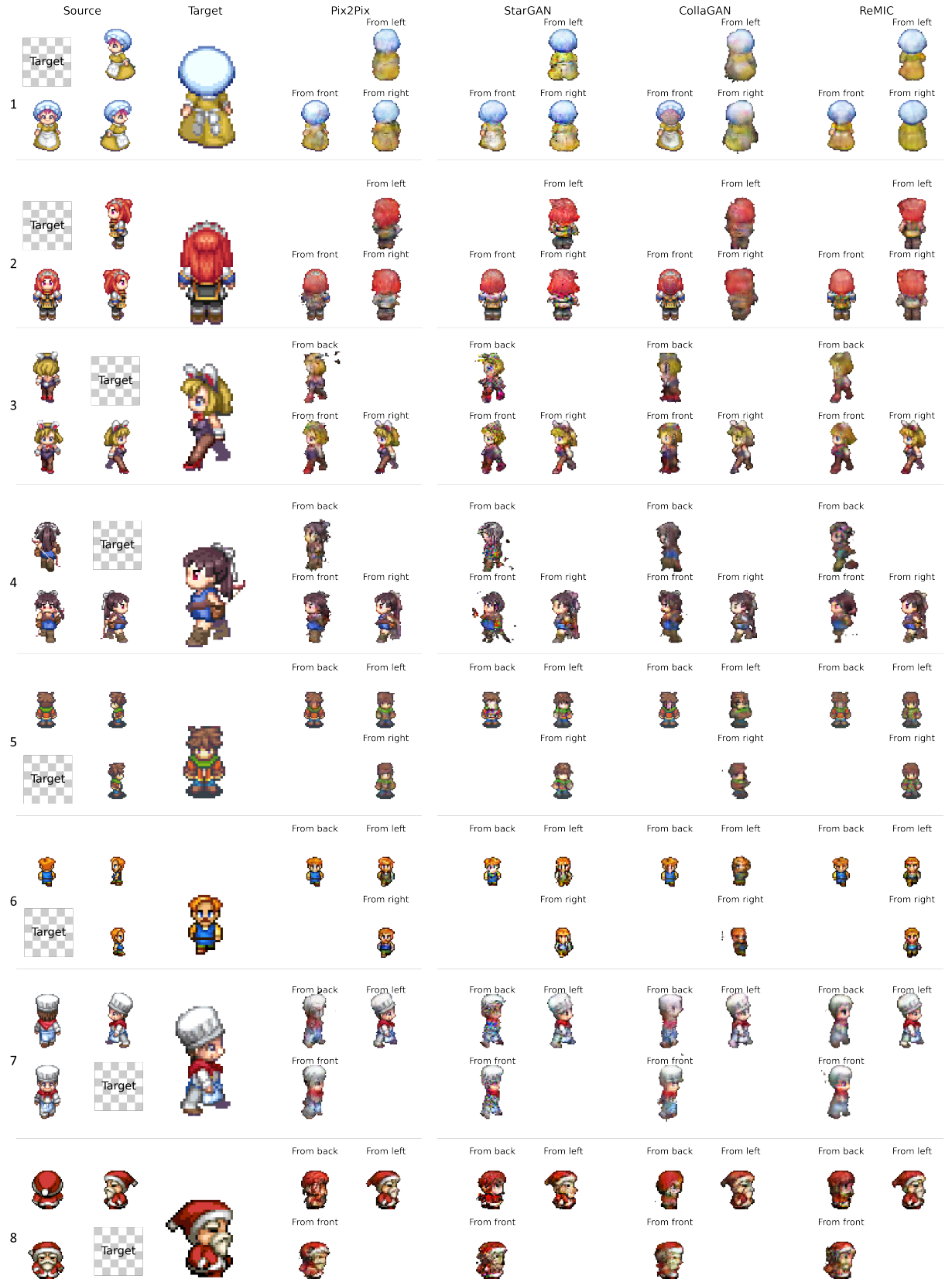
Analyzing from another perspective, we can observe the metrics values according to the target domain (i.e., looking at each column), as an indication of which poses are more difficult to generate. We can note that both metrics are consistently lower when generating lateral poses (i.e, left and right) than front and back. For instance, StarGAN has an average FID of 1.250 and 1.368 for left and right generated images, contrasted to 3.378 and 3.156 for back and front. That indicates that generating front and back poses constitute harder tasks. Additionally, looking at each row, we can determine which are the best sources for each target pose. That reveals that left and right are the best sources for each other, as well as front and back, due to the lower values.

Figure 5.6 shows some characters generated by each model, using a single input image, from all source poses other than the target. We depict the image generated for the target pose from each of the other domains. The quality of the generated images varies with the model and the translation direction. We analyze the results qualitatively according to the distribution of colors and the generated shape.

Regarding the color usage, all models generate images with colors in meaningful positions but employ many variations of the same tones instead of restricting to a small palette. Such undesired behavior can be attenuated by quantizing the colors to the palette of the input images, such as we suggest in Chapter 6 (Hypotheses on Pixel Art Generation). Other common issues related to the colors are the blurred pixels (mostly present in Pix2Pix and CollaGAN-1), which dissolves the characters' inner edges, and the high-frequency color noise (especially in StarGAN), that introduces colors with more variation than intended even on regions that should have the same tones. An example featuring both artifacts is the cook character (7th row in Figure 5.6): when generated from back or front by Pix2Pix, CollaGAN-1, and ReMIC-1, the resulting image is blurred; when generated from back or front by StarGAN, it contains high-frequency color noise. Another interesting artifact can be observed in back-facing characters (1st and 2nd rows) generated by CollaGAN-1: when the source pose is front, it is possible to partially identify the characters' faces. That does not happen with the other models.

When analyzing the resulting shapes, Pix2Pix, ReMIC-1, and StarGAN generated characters more similar to the target than CollaGAN-1, which is more easily visible for translations between back/front to right/left. To illustrate, the red-haired warrior (2nd row in Figure 5.6) displays very disproportional shapes when translated from left/right to back by CollaGAN-1, but not as much by the other models. Another artifact related to the shape is the presence of blobs outside the character region, as illustrated by the

Figure 5.6: Images generated by the proposed models using a single input image. **Rows** contain examples with different target domains. **Columns** depict the source images, the target, and the generated images from different source poses by each model.



archer lady (4th row) generated by StarGAN from the back/front poses.

By inspecting Figure 5.6 we can also verify that the tasks of translating between left and right, and back and front produce images of higher quality, endorsing the lower FID and L_1 values for those translation directions as seen in Table 5.5.

Lastly, by triangulating the visual analysis with the quantitative results, we can observe that the metrics diverge because of the different characteristics they evaluate. While FID measures the distance between the target and the generated images in the feature space of a classification network, the L_1 distance is the pixel by pixel difference between the two sets of images. Regarding the latter, if a generated image has the correct colors, but a shape mismatch, the L_1 distance can become much higher. In summary, FID evaluates the distance between distributions and L_1 penalizes color differences but mostly shape discrepancy.

5.4 Translation using Multiple Inputs

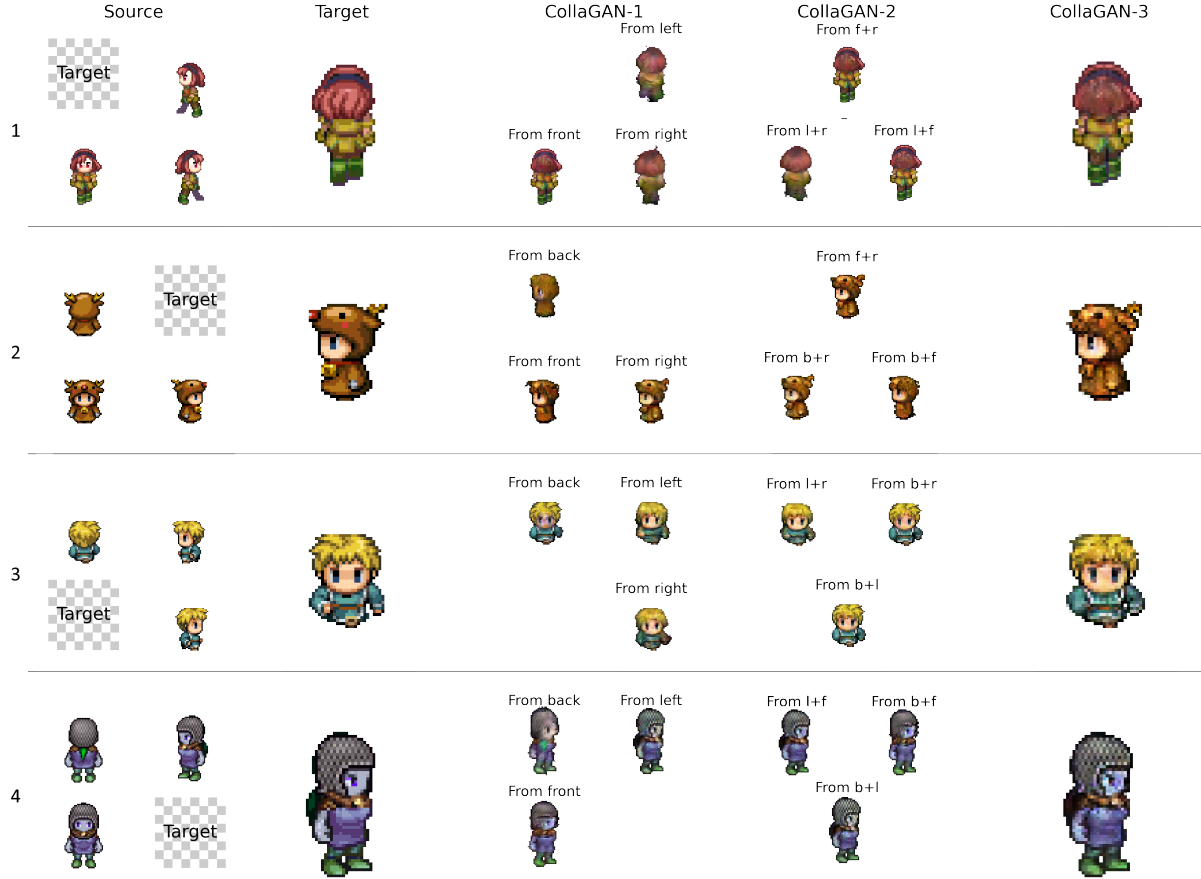
We experimented generating images with more than a single input image. Among the proposed models, the CollaGAN and the ReMIC-based ones can receive multiple images. We trained the CollaGAN-based model with our proposed conservative input dropout strategy and the forward replacer technique, and ReMIC with CollaGAN’s original input dropout strategy (as those yielded the best results), and evaluated them on the scenarios of having one (CollaGAN-1/ReMIC-1, the same as in the previous section), two (CollaGAN-2/ReMIC-2), and three (CollaGAN-3/ReMIC-3) images available as input.

Figures 5.7 and 5.8 portray example characters generated by CollaGAN and ReMIC with each possible source domain combination. We can see how the generated images get closer to the ground truth as the number of input domains increases. At the same time, some examples have promising images even in the scenario of a single input domain, especially for the easier left/right and front/back translation directions.

For a quantitative analysis, Tables 5.6 and 5.7 show the FID and L_1 distance between the generated images and the ground truth, with different source domain combinations on the rows and the target domains in the columns, for the CollaGAN and ReMIC-based models respectively. The first row grouping shows the metrics for the model evaluated with a single input image, the second with two images, and the third with three. Each section ends with a row with the average values for all source combinations for each target domain.

Analyzing both tables by sweeping the columns downwards (following the increase in the number of input images), we observe that both metrics’ average values consistently

Figure 5.7: Images generated by the CollaGAN-based model with different source domain combinations. **Columns** show the source and target images, and the ones generated with a single input image, two images and all three. **Rows** show examples with different target poses.

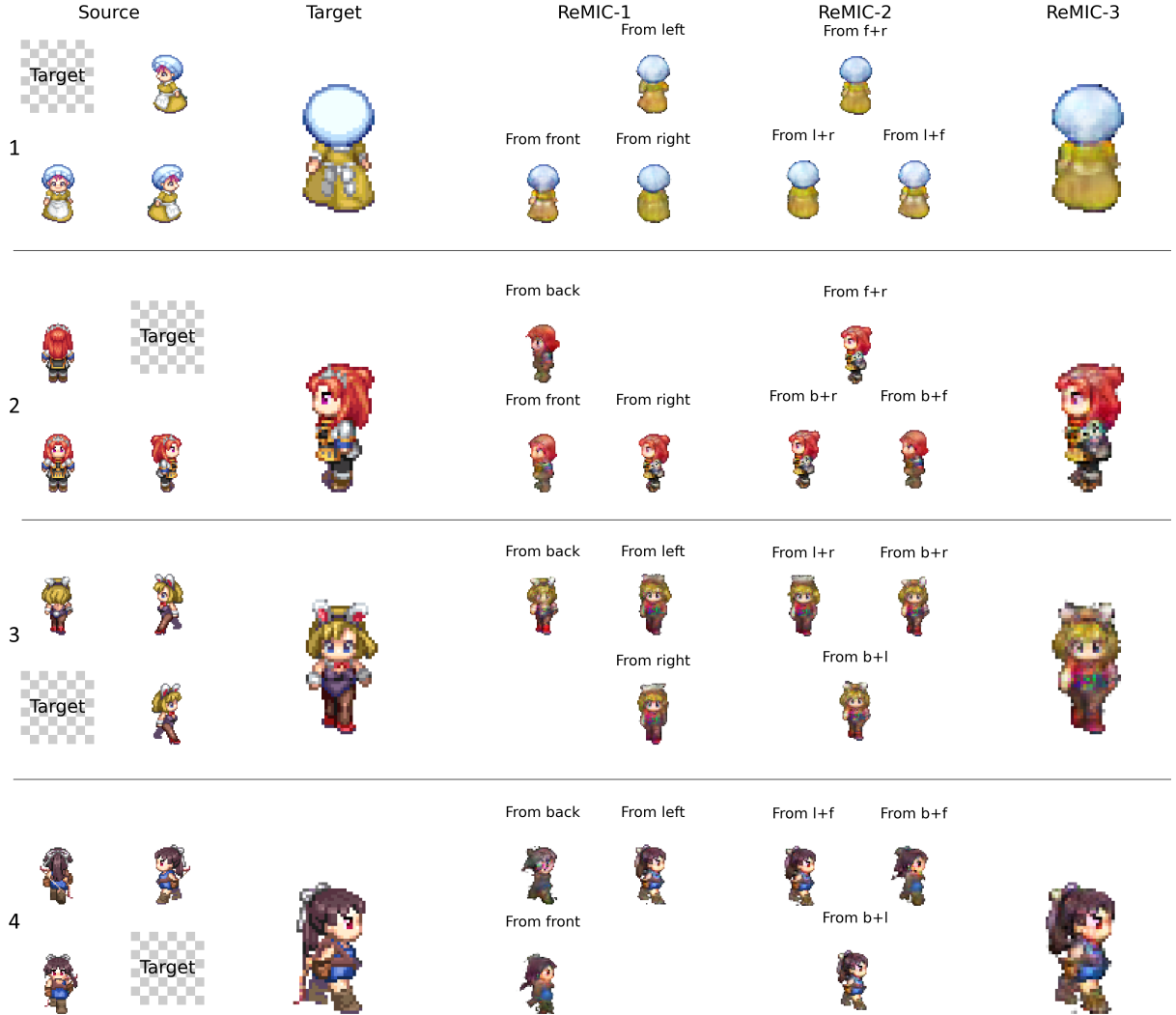


reduce (i.e., improve) as we introduce more inputs to the generators. To illustrate that behavior we can observe the [FID](#) value averaged over the source domain combinations for the right target pose: the values are 4.803, 2.769, and 0.987 for CollaGAN-1, 2 and 3 (Table 5.6) , and 3.432, 2.019, and 1.178 for ReMIC-1, 2 and 3 (Table 5.7).

However, if we look to the best source combination for each target pose (the highlighted values in the tables), there are different behaviors between the CollaGAN and the ReMIC-based models.

On the one hand, the metrics for CollaGAN improve as the generators receive more inputs in most cases, except in two situations, which we describe later. To illustrate, looking at the [FID](#) values when the target is the back pose, the best source directions are front (2.707, for CollaGAN-1), left+front (2.236, for CollaGAN-2), and left+front+right (2.054, for CollaGAN-3). In addition, the sources that yielded the highest (worst) values for each target pose also improved as the number of inputs increased. In the same example of generating the back pose, the worst sources are right (19.117, CollaGAN-1), left+right (13.574, CollaGAN-2), and left+front+right (2.054, CollaGAN-3).

Figure 5.8: Images generated by the ReMIC-based model with different source domain combinations. **Columns** show the source and target images, and the ones generated with a single input image, two images and all three. **Rows** show examples with different target poses.



Conversely, the best source combinations for each target pose from ReMIC do not come from when it receives three inputs necessarily. For instance, if we look at the best sources to generate characters facing right, the FID values are 1.134, 1.181, and 1.178, and L_1 distances are 0.02917, 0.02967, and 0.03115. That indicates that the information from the extra inputs might have introduced more uncertainty. We hypothesize that the reason for the different behaviors between the CollaGAN and the ReMIC-based models stem from the use of different input dropout strategies. While CollaGAN trained more frequently with more inputs, ReMIC trained with a uniformly random number of dropped inputs.

The two exceptions to the monotonically decreasing metric values are the L_1 distance between the best values for CollaGAN-2 and CollaGAN-3 when generating left and

Table 5.6: FID and L_1 distances for all input images (**rows**, with each pose represented by its initial letter) and target domain (**columns**) combinations for the **CollaGAN-based** model. The highlighted numbers indicate the best combination of source domains for each target pose.

Target $\searrow$		FID				L_1			
$\downarrow$ Source		Back	Left	Front	Right	Back	Left	Front	Right
CollaGAN-1	b	—	8.581	2.735	7.085	—	0.07463	0.05559	0.07835
	l	16.401	—	13.345	1.764	0.07254	—	0.07456	0.03480
	f	2.707	5.168	—	5.562	0.04971	0.06940	—	0.07217
	r	19.117	2.157	16.093	—	0.07630	0.03516	0.08066	—
	Average	12.741	5.302	10.725	4.803	0.06619	0.05973	0.07027	0.06177
CollaGAN-2	b+l	—	—	2.463	1.771	—	—	0.04996	0.03342
	b+f	—	4.909	—	4.977	—	0.06728	—	0.07140
	b+r	—	1.730	2.620	—	—	0.03395	0.05028	—
	l+f	2.236	—	—	1.559	0.04637	—	—	0.03284
	l+r	13.574	—	11.155	—	0.06894	—	0.07033	—
	f+r	2.602	1.725	—	—	0.04607	0.03334	—	—
	Average	6.137	2.788	5.413	2.769	0.05380	0.04485	0.05686	0.04589
CollaGAN-3	b+l+f	—	—	—	0.987	—	—	—	0.03360
	b+l+r	—	—	1.955	—	—	—	0.04985	—
	b+f+r	—	1.037	—	—	—	0.03439	—	—
	l+f+r	2.054	—	—	—	0.04530	—	—	—
	Average	2.054	1.037	1.955	0.987	0.04530	0.03439	0.04985	0.03360

right poses. When going from two to three inputs, the L_1 metric increases (worsens) when the target is left (0.03334 for f+r to 0.03439 for b+f+r) and right (0.03284 for l+f to 0.03360 for b+l+f). To investigate the divergence between **FID** and L_1 in this situation, we can visually analyze the generated images. Figure 5.9 depicts left-facing characters created from the right+back, right+front, and right+back+front source domains. Although the differences are subtle, the images generated from three inputs are crispier and present more fine-grained details (e.g., the better separation of hair locks). The lower L_1 values for the images generated from only two poses can be attributed to the more blurry pixels, as that metric evaluates the pixel-wise average color distance and does not account well for the crispy nature of pixel art.

To analyze ReMIC’s behavior, we can visually inspect the images generated by the best source combinations for each input number grouping. Figure 5.10 shows two examples of right-facing characters generated from: left (FID: 1.134, L_1 : 0.02917), left + front (FID: 1.181, L_1 : 0.3024), and left + front + back (FID: 1.178, L_1 : 0.03115). We can observe that if the left pose is present in the input, adding the front or even back and front directions worsens the metrics and the image quality a little, illustrated by the

Table 5.7: FID and L_1 distances for all input images (**rows**, with each pose represented by its initial letter) and target domain (**columns**) combinations for the **ReMIC-based** model. The highlighted numbers indicate the best combination of source domains for each target pose.

	Target $\searrow$ $\downarrow$ Source	FID				L_1			
		Back	Left	Front	Right	Back	Left	Front	Right
ReMIC-1	b	—	4.624	5.978	5.441	—	0.06743	0.05625	0.06820
	l	8.520	—	7.654	1.134	0.05931	—	0.06273	0.02917
	f	5.517	3.703	—	3.722	0.04616	0.05962	—	0.06035
	r	8.732	1.073	8.992	—	0.05938	0.02971	0.06353	—
	Average	7.589	3.133	7.541	3.432	0.05495	0.05225	0.06084	0.05257
ReMIC-2	b+l	—	—	5.283	1.141	—	—	0.05028	0.02967
	b+f	—	3.606	—	3.735	—	0.05878	—	0.05935
	b+r	—	1.032	5.591	—	—	0.02978	0.05091	—
	l+f	5.898	—	—	1.181	0.04521	—	—	0.03024
	l+r	7.937	—	7.520	—	0.05745	—	0.06122	—
	f+r	5.502	1.105	—	—	0.04512	0.03027	—	—
	Average	6.446	1.914	6.131	2.019	0.04926	0.03961	0.05414	0.03975
ReMIC-3	b+l+f	—	—	—	1.178	—	—	—	0.03115
	b+l+r	—	—	5.232	—	—	—	0.05026	—
	b+f+r	—	1.082	—	—	—	0.03116	—	—
	l+f+r	5.589	—	—	—	0.04532	—	—	—
	Average	5.589	1.082	5.232	1.178	0.04532	0.03116	0.05026	0.03115

little bit more chaotic use of colors as the number of input images grows. Nevertheless, the perceived quality loss is mostly negligible.

We can also compare both models to observe their performance when faced with easier and harder tasks. We consider that tasks for which the closest source domain is available to generate some target domain are easier, and those where it is absent as harder. To illustrate, as Tables 5.6 and 5.7 numerically indicate and Figures 5.7 and 5.8 show, left is the best source input to generate a character facing right (and vice versa), and front is the most useful domain to produce a back-facing character (vice versa too).

Figure 5.11 shows the images generated by CollaGAN and the ReMIC-based models for easier tasks and Figure 5.12 for the harder ones. We can observe that both models generate promising images, especially for the easier tasks, but with ReMIC producing usually cleaner results. An interesting observation is that some images created by CollaGAN contain some alien-domain artifacts, such as a faint face when the character is facing back (see the back-facing maid in line 1 of Figure 5.11). The CollaGAN model also has trouble generating the back pose of characters that have a cape (as in the example 4 of the same figure). We believe that the main reason for this type of artifact is due to

Figure 5.9: Comparison of the CollaGAN-based model using 2 and 3 images as input. When the target is either the left or right poses, introducing a third source input (i.e., from CollaGAN-2 to CollaGAN-3) results in crispier, more detailed images.

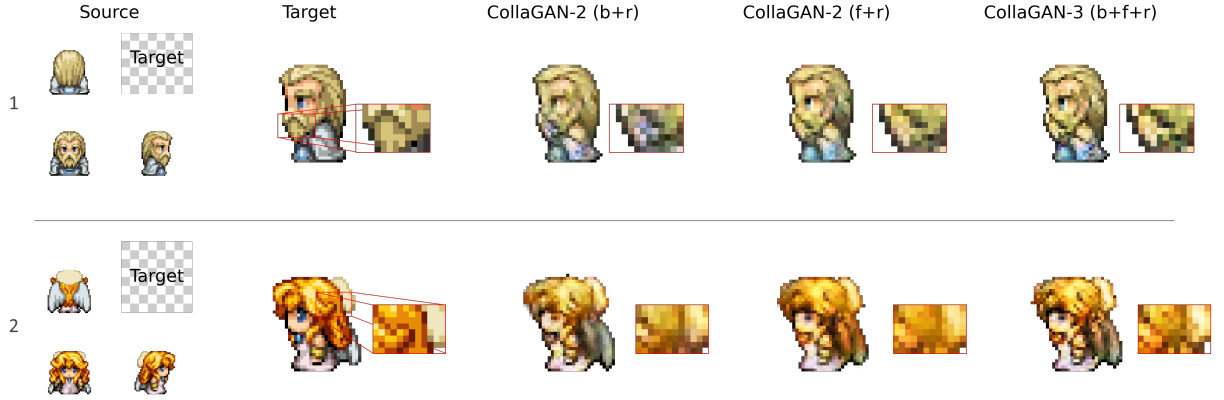
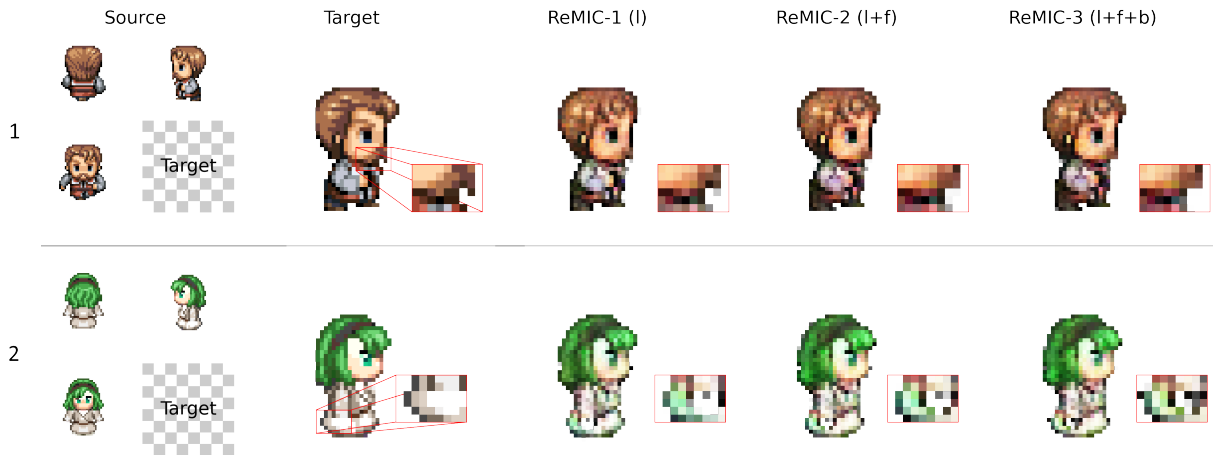


Figure 5.10: Comparison of the best source combinations for a target pose from ReMIC. When the target is either the left or right poses, introducing more inputs (i.e., from ReMIC-2 to ReMIC-3) does not necessarily improve the results.



the low number of examples in the dataset, which hinders the generalization capacity of the models. But ReMIC is not affected by that (or at least much less) because it uses separate decoders for each domain, allowing it to better separate domain-specific features.

We can also note by inspecting Figure 5.12 that the ReMIC-based model produces better images than CollaGAN on harder tasks, which can be due to its input dropout strategy which shows more sparse examples (more images missing) than CollaGAN’s conservative approach.

Next, we evaluate the impact of different input dropout strategies when training the models that can receive input images from multiple domains.

Figure 5.11: Comparison of the CollaGAN and ReMIC-based models in easier tasks: when the most useful source domain is available to generate some target pose. Aside from the first column, each pair shows the images generated by CollaGAN and ReMIC for a different task.



5.4.1 Input Dropout Strategies

We evaluated the impact of different batch selection strategies for training the proposed models that can receive multiple inputs by changing how the character examples are presented to their generators: should they always see the three available domains, or should they sometimes be omitted?

We investigated always showing all available domains (none—no input dropout), the original input dropout strategy proposed in [Lee et al., 2019], a curriculum learning approach suggested by Sharma and Hamarneh [2020], and our proposed conservative tactic. The original approach has an equal chance of presenting three, two, or a single image in a training step. The curriculum learning approach starts training with easier tasks (using three images) and progressively makes it harder (using two, then a single image) until half of the training, then it randomly chooses between the number of domains to drop out for the second part. Lastly, our conservative approach randomly selects the number of images to drop, but with higher probabilities to keep more images: 60% having

Figure 5.12: Comparison of the CollaGAN and ReMIC-based models in harder tasks: when the most useful source domain is not available to generate some target pose. Aside from the first column, each pair shows the images generated by CollaGAN and ReMIC for a different task.



3 images, 30% having 2, and 10% having a single image shown to the generator. The paper that introduced the ReMIC architecture [Shen et al., 2021] does not indicate which strategy it uses.

We also experimented with an aggressive strategy that is the inverse of the conservative and a balanced one that has an equal probability of presenting each combination of inputs at each batch, but both yielded worse results than the original dropout strategy.

Table 5.8 presents the results from the models trained with the different input dropout strategies (columns) in the scenarios of having three, two, or one image available as input (rows). We can observe that using any input dropout yields better results than always showing all domains (none). For the CollaGAN-based model, our proposed conservative tactic has better FID and L_1 metrics on the average of the three scenarios. In particular, regarding FID, the model trained with the conservative input dropout worsens its performance less drastically with the decrease of input domains. Regarding L_1 , its metrics are better than the other models when two and three images are available. In turn, the ReMIC-based model had consistently better metrics when trained using the CollaGAN’s original input dropout strategy, for both metrics and in the three scenarios.

Table 5.8: FID and L_1 distance of different input dropout strategies (**columns**) when the CollaGAN and ReMIC-based models receive 3, 2 or 1 images as input (**rows**). The highlights emphasize the best values for each metric among the different input dropout strategies for each scenario.

Inputs	FID				L_1			
	None	Origin.	Curric.	Conser.	None	Origin.	Curric.	Conser.
CollaGAN-1	32.676	11.162	20.303	8.393	0.12820	0.06243	0.12232	0.06449
CollaGAN-2	19.050	6.835	9.233	4.277	0.08003	0.05053	0.07389	0.05035
CollaGAN-3	4.816	1.911	2.160	1.508	0.04523	0.04277	0.04222	0.04078
Average	18.847	6.636	10.566	4.726	0.08449	0.05191	0.07948	0.05187
ReMIC-1	12.177	5.724	6.612	6.388	0.09721	0.05500	0.06287	0.05511
ReMIC-2	6.991	4.364	4.472	5.471	0.06069	0.04554	0.04838	0.04798
ReMIC-3	3.773	3.464	4.065	4.727	0.04056	0.03932	0.04375	0.04369
Average	8.754	4.818	5.331	5.758	0.07347	0.04871	0.05393	0.05042

5.5 Results from SpriteGAN

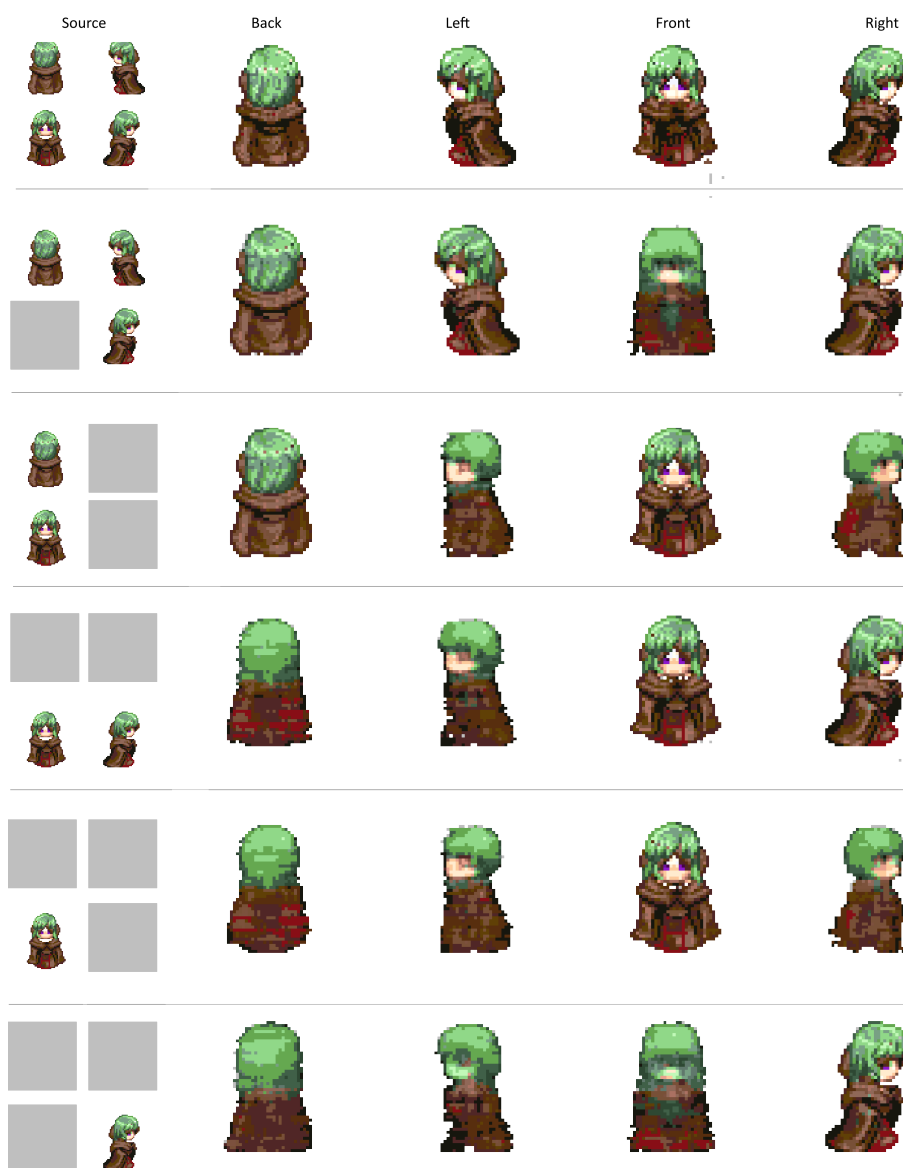
Here we present our results for SpriteGAN. The model that produced the best results trained with the following hyperparameter values: $\text{learning_rate} = 0.0001$, $\lambda_{\text{adv}} = 1$, $\lambda_{\text{rec}} = 40$, $\lambda_{\text{lat}} = 1$, $\lambda_{\text{cyc}} = 10$, and $\lambda_{\text{KL}} = 0.01$, with a mini-batch size of 4. We generated images using the model to perform a preliminary assessment of its quality on three tasks: imputing from different numbers of input domains, generating different outputs for the same input (multi-modality), and inpainting missing regions.

5.5.1 Imputation from Different Input Combinations

Figure 5.13 presents the images generated for the same character under different combinations of input domains. Each row corresponds to a distinct source domain set x_S . When an image from a particular domain is provided, the generated output reproduces it almost identically, indicating that the model has learned an identity function capable of replicating the input. This usually is undesired, as in the case of Pix2Pix—we would need to train a model that always reproduces the input. However, as SpriteGAN also performs inpainting of missing regions, it is useful for the model, as we see later in this section.

The imputed images partially resemble the ground truth, showing roughly identifiable body parts in the correct positions. However, they lack well-defined edges delineat-

Figure 5.13: Outputs from SpriteGAN showing the imputation of characters. **Rows** represent different input domain combinations. **Columns** show the generated image for each target domain.

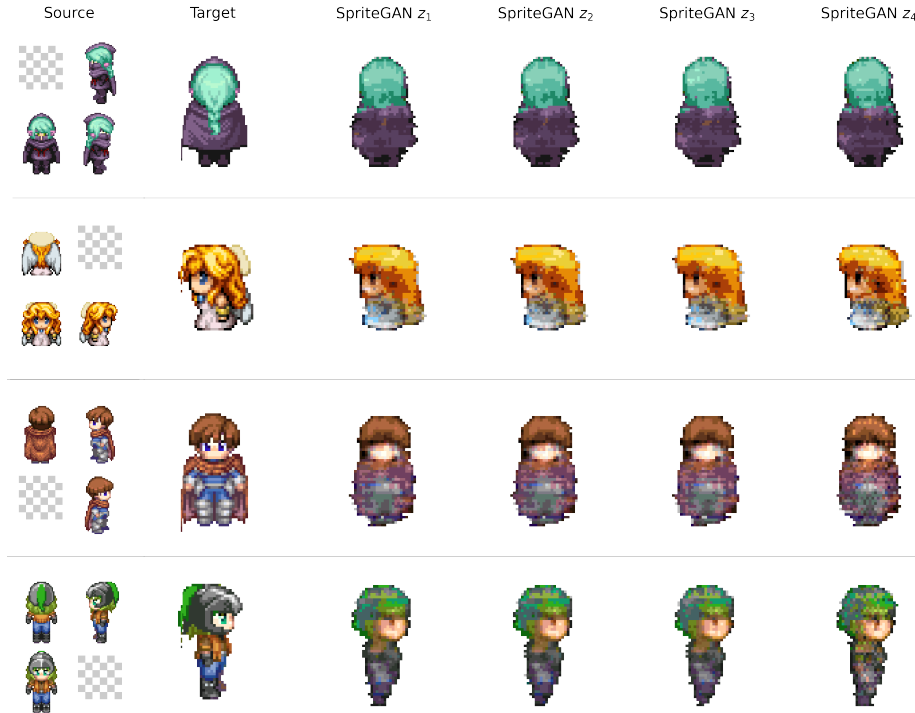


ing object or limb boundaries, and their texture details are often imprecise (for instance, the clothing shapes are not clearly depicted in the generated poses). Distinct character features—such as the character’s partially visible purple eyes—tend to be lost in some domains when it needs to be imputed. As the number of available input domains decreases, we can notice that the generated images become less precise. Nevertheless, all generated images adhere strictly to the intended target palette, owing to the palette quantization operation embedded in the generator.

5.5.2 Multi-modality for Images in Absent Domains

Figure 5.14 shows different imputed characters, for different poses, generated using four random latent codes $\hat{z}$ sampled from the $\mathcal{N}(0, 1)$ distribution. We can observe that the images are different, but only very subtly. The divergences manifest mostly in some texture parts, not having noticeable impact on the generated characters' shapes, which implies a very limited diversity.

Figure 5.14: Outputs from SpriteGAN showing characters generated with different z codes. **Rows** represent different characters with a missing domain. **Columns** show the generated images using different latent codes z sampled from $\mathcal{N}(0, 1)$.



Our approach to achieve multi-modality with SpriteGAN involved including a VAE-inspired diversity encoder E , which extracts a style code z from the source images, and is used by the generator G to reconstruct or impute the missing domains. The training loop produces three sets of images: $\hat{x}_N$, $\tilde{x}_N$, and $\tilde{\tilde{x}}_N$, respectively representing images generated with the extracted latent codes z , random latent codes $\hat{z} \sim \mathcal{N}(0, 1)$, and the same extracted z but using $\tilde{x}_N$ as input, in a cyclical reconstruction pass. Both the diversity encoder E and generator G are penalized by not being able to recover the style code that was used to produce the images, in the $\mathcal{L}_{\text{lat}}$ loss term between the random style $\hat{z}$ and the extracted $\hat{z} = E(G(\tilde{x}_N, \hat{z}, \dots))$.

By analyzing the latent loss term $\mathcal{L}_{\text{lat}}$ throughout the training procedure, we can see that it decreases, indicating that it was optimized up to some point. That behavior shows that the E and G learned to encode and decode the latent code z inside the

generated images. Indeed, different z yield different $\hat{x}_N$, but, again, the differences are barely noticeable.

We hypothesize that the palette quantization operation as the last layer of G might have restricted the output variability, as the generated images strictly follow the same set of colors, regardless of the latent code z . We believe that further changes to the training loop and objective function might be necessary to improve the diversity of the generated images.

5.5.3 Inpainting with Different Masks

Figure 5.15 shows examples of characters generated from source images in which certain regions were erased using inpainting masks. The results indicate that the model is able to reconstruct occluded regions of the characters while preserving the unmasked areas largely unchanged, as desired.

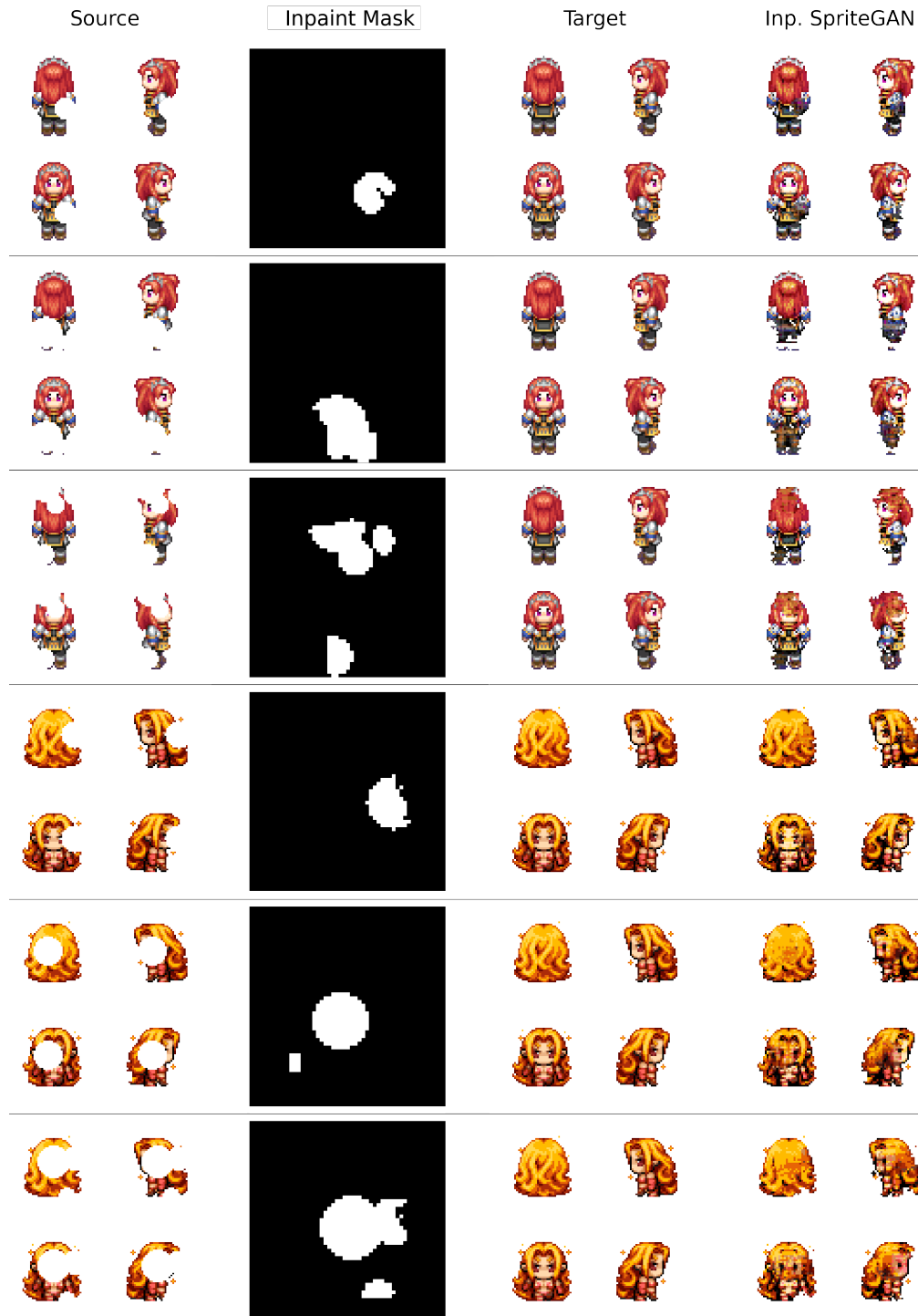
When the inpainting masks are small, the generated images exhibit visual quality close to that of the ground truth. As the mask size increases, however—particularly when it removes the same region across all available domains (e.g., the character’s face in the third row)—the reconstruction quality degrades noticeably. These results suggest that the model’s current inpainting capability is effective for small occlusions but may require architectural improvements to handle larger masked regions more robustly. An approach that was taken by [Dapogny et al. \[2019\]](#) is to have the discriminators try to reconstruct the inpaint mask that was given to the generator. Another interesting idea is the use of partial convolutions to progressively fill in the missing regions, as proposed by [Liu et al. \[2018\]](#).

5.6 Discussion

In the proposed experiments, we evaluated the models in the task of generating pixel art character sprites as an image-to-image translation problem (single input) and as a missing-data imputation task (multiple inputs).

We summarize the quantitative results on Tables 5.9 and 5.10. They contain the values for both metrics split into easier tasks, harder tasks, and an average over of all source domains and target combinations. Like in the previous section, we consider easier

Figure 5.15: Outputs from SpriteGAN showing inpainted regions of characters using different masks. **Rows** use different masks. **Columns** show the source, the inpaint mask, the ground truth, and the generated images for each target domain.



tasks those that have the most useful domain present in the source for each target (e.g., front is present in the source domains to produce a back-facing character), and harder tasks those in which that domain is absent (e.g., left is absent to generate a character facing right).

First, we analyze the generation of images from a single input (Table 5.9). As

the metrics do not converge, we consider them separately. Regarding [FID](#), StarGAN has the lowest value for the best, worst and the average cases. In turn, Pix2Pix has the lowest L_1 distance in all three cases. We understand that Pix2Pix has the lower L_1 value because this metric strongly penalizes shape mismatch, and the images generated by StarGAN sometimes displayed worse shapes than those produced by Pix2Pix. Also, both CollaGAN-1 and ReMIC-1 had worse results than the single-input models as the former focus on generating better images when more inputs are available. One last observation is that CollaGAN-1 has the largest amplitude between the metrics of easier and harder tasks.

Table 5.9: A summary of FID and L_1 averaged over all possible source and target domain combinations (except $t = s$) for the proposed models when receiving a single input.

	Task	Pix2Pix	StarGAN	CollaGAN-1	ReMIC-1
FID	Easier	2.311	1.914	2.341	3.426
	Harder	5.175	2.475	11.418	6.423
	Average	4.220	2.288	8.392	5.424
L_1	Easier	0.03589	0.04861	0.04382	0.04033
	Harder	0.05995	0.07435	0.07483	0.06257
	Average	0.05193	0.06577	0.06449	0.05515

Now, by analyzing the models that can receive multiple inputs (see [Table 5.10](#)), we note that the metrics do not converge either. In terms of FID, CollGAN-3 yields the images with the lowest value, while ReMIC-2 has the lowest L_1 distances for easier and harder tasks, and ReMIC-3 has a lower value than CollaGAN-3. Here again CollaGAN-2 and CollaGAN-3 yield a higher metric amplitude between the easier and harder cases.

Table 5.10: A summary of FID and L_1 averaged over all possible source and target domain combinations (except $t \in S$) for the proposed models when receiving multiple inputs.

	Task	CollaGAN-2	ReMIC-2	CollaGAN-3	ReMIC-3
FID	Easier	2.088	3.531	1.508	3.464
	Harder	8.653	6.030		
	Average	4.276	4.364	1.508	3.464
L_1	Easier	0.04078	0.03879	0.04078	0.03932
	Harder	0.06949	0.05904		
	Average	0.05035	0.04554	0.04078	0.03932

Quantitative results aside, a visual inspection of the generated images shows that their quality varies a lot, depending on the translation direction, the model, and the similarity of the input character to the style of those in the training set. While some of

the generated images are readily usable, others are too distant to the ground truth to be useful.

In special, when the input characters are very different from the art styles present in the training set, the generated images can have very low quality, not resembling a pixel art character. That indicates a limitation in the generalization capacity of the proposed models.

In addition, even for the generated images that are more visually appealing, they display many more colors than desired as there are lots of small variations of the same color. This happens due to the statistical nature of the models which try to approximate the probability density function of the training data and, differently from a pixel artists, they do not pick the color of each pixel from a predefined palette.

Despite the current SpriteGAN model not achieving visual quality on par with other models, it demonstrates unique capabilities, such as inpainting small missing regions of characters and generating slightly different outputs for the same source images based on a style code. Furthermore, it successfully showcases the integration of the novel differentiable palette quantization technique (detailed in Chapter 6 - Hypotheses on Pixel Art Generation), which enforces strict adherence to a target color palette—a critical requirement for pixel art generation. We believe that further investigations around architectural choices and training procedure might yield important improvements to the quality of the generated images.

In the next chapter, we investigate some hypotheses that use domain-specific knowledge of the task to improve the quality of the generated images.

Chapter 6

Hypotheses on Pixel Art Generation

After proposing different models and investigating the quality of the generated images, we observed an opportunity to increase their generalization capacity: the images produced from the training set are better than the ones from the test partition. In particular, as we are generating pixel art characters, in this chapter we investigate some hypotheses that include domain-specific knowledge from the task to potentially lead the models into generating better images.

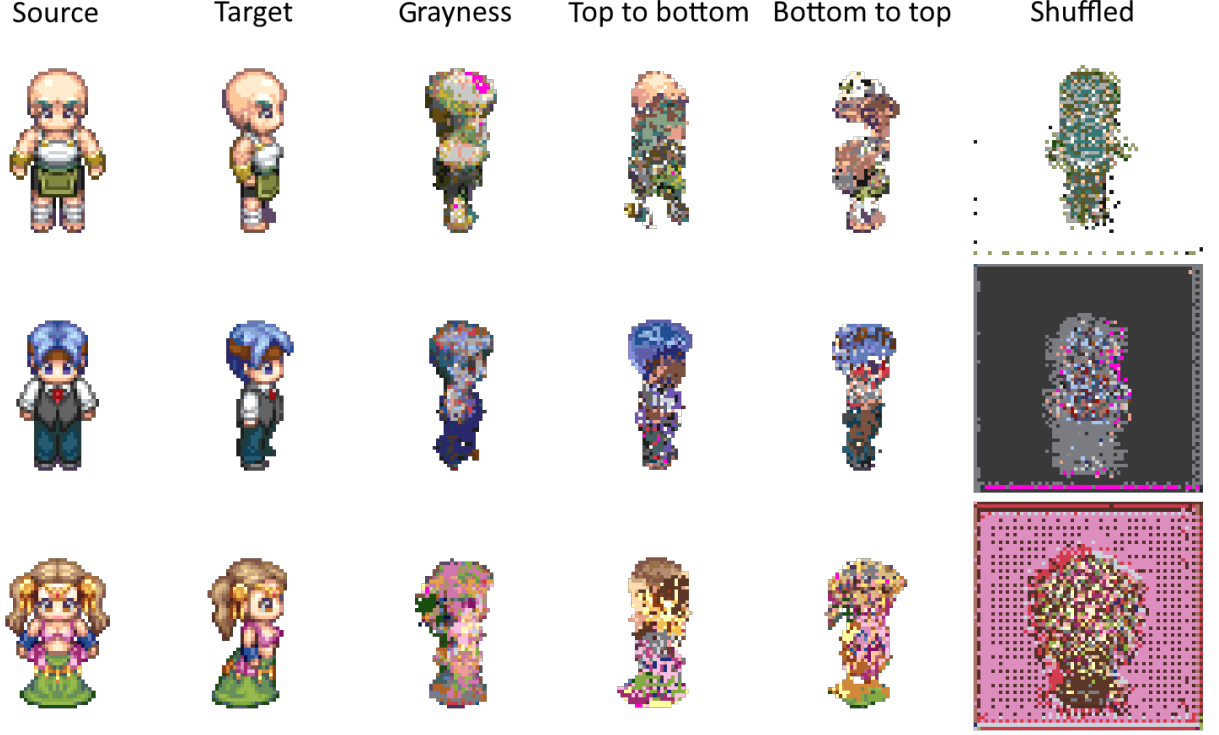
We start by presenting an architecture modification to represent images as indices in color palettes. A second idea is the use of a differentiable histogram loss term added to the generator’s objective function. We also investigate a post-processing step in which the pixels of the generated images are quantized to the closest color in the original palette. Lastly, we conclude by proposing a method that allows quantizing to the palette while still remaining differentiable, which allows end-to-end training with the palette quantization procedure.

The experiments used the Pix2Pix-based architecture, as it was the first proposed and the quickest to train in a single source/target direction. In addition, regarding the data, all experiments used the RPG MAKER XP dataset, as due to its higher variability and low number of examples, it represents a harder task, and the translation direction of front to right, as it has a medium to hard difficulty.

6.1 Palette Representation

Artists frequently employ a pre-selected color palette when creating pixel art, both to reduce the space of possible options for a pixel and to achieve consistency in the use of colors [Silber, 2015]. Inspired by the self-imposed restriction by pixel artists, we investigate the impact on image quality of representing images as indices in a color palette while training a deep generative model. Hence, instead of using an RGBA value to indicate the color of a pixel, we use an integer that is the index of the color in the palette of the

Figure 6.1: Early-training output of the generators on validation data from palette-indexed colors varying order in which colors appear in the palette.



source image. This frames the problem as a semantic segmentation task, in which we want to assign a class (i.e., a color index) to each pixel of an image.

We adapted both networks of the Pix2Pix-based architecture. We changed the last layer of the generator to have a number of units equal to the maximum palette size (i.e., 64) and a softmax activation instead of tanh. Both discriminator and generator networks process images using one-hot encoding of the color indices. Furthermore, as the color indices do not represent color distances, we replaced the $\mathcal{L}_{\text{rec}}$ term from the generator loss (Equation 4.2) with a categorical cross-entropy $\mathcal{L}_{\text{cce}}$ with the indices of the target image, scaled by some factor λ_{cce} :

$$\mathcal{L}_G = -\mathbb{E}_{x_s} [\log D(x_s, G(x_s))] + \lambda_{\text{cce}} \mathcal{L}_{\text{cce}}$$

The model trained for 40,000 steps in batches of 4 on the RPG MAKER XP dataset. The discriminators and generators were optimized using Adam ($\beta_1 = 0.5$ and $\beta_2 = 0.999$) with a learning rate of 0.0002. After only a few epochs, the generator could perfectly reproduce the data seen during training (zeroed FID and L_1 for the train partition). However, at the same time, the quality of the images generated from the test set started degrading very early in the process, indicating that the model memorized the training data even under dropout regularization on both networks.

We extracted each image’s palette by picking the unique colors and filling the remaining color slots with hot pink, a color absent from the train/test data. We found

that the order in which the colors appear in the palette impacts the quality of the resulting images. Regarding that order, we experimented with sorting the colors by their gray tone (i.e., $0.2989 \times R + 0.5870 \times G + 0.1141 \times B$), by the order of appearance in the image from top to bottom, from bottom to top, and completely at random (shuffling the palette).

The model that trained with shuffled palettes did not converge, producing unusable noise, while all others overfit to the training data. We could observe that when there was some semantics to the color indices in the palettes, the images generated from the test set presented different results. Figure 6.1 illustrates this observation in early iterations (1.2k generator updates, before overfitting): when the first indices of all palettes share the semantics of corresponding to either the characters' heads or their feet (4th and 5th columns, respectively), the quality of those parts stand out in comparison to the remainder of their bodies.

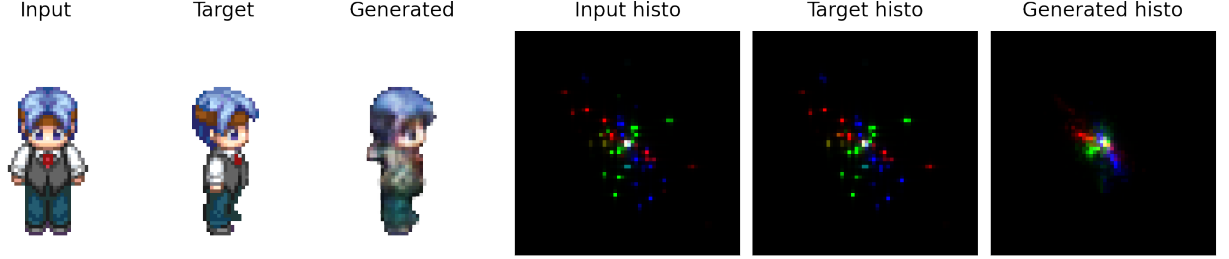
Considering the worse results when training with shuffled palettes, our main takeaway from evaluating the palette representation hypothesis is that moving out of the RGBA space into an indexed representation deprives the model of the semantics embedded into the original color space. That leads us to hypothesize whether we can define a new space, potentially with a higher dimension than RGBA, in which colors are represented by embeddings that encode their role inside the dataset. For instance, such a representation could include the relation of colors that appear adjacently or in different patterns, allowing the networks to represent and reproduce textures and shading techniques more easily.

6.2 Histogram Loss

Also inspired on the pixel art premise of having a limited number of colors, we investigate whether the quality of the generated images improve when the generator trains with a penalization for producing images with a histogram different than the one from the source image.

One challenge with this approach is that to enable the backpropagation of errors through gradient descent, all terms of the cost functions must be differentiable, but calculating histograms naively is not (due to thresholding in bins). We then resorted to a different method to derive an image histogram that replaces thresholding with a kernel weighted contribution to each bin [Afifi et al., 2021]. It uses the log-chroma space, in which two of the RGB channels are used to normalize the third, resulting in two parameters (u, v) for each original color component. By doing so, it is possible to have the color histogram visualized in 2D (Figure 6.2 shows examples). Because of the color

Figure 6.2: Example character generated by the model trained with a histogram loss. The **columns** show the input, target, and generated images, with their respective RGB-uv histograms.



representation used, the resulting image is called an RGB-uv histogram.

To add some control to the colors of generated images, the authors of HistoGAN [Affi et al., 2021] added the target histogram mapped to the latent space of the last two upsampling blocks from StyleGAN [Karras et al., 2018]. In addition, the generator tries to minimize the distance from the target histogram to the one from the generated image.

Since we intend to encourage the generator to use the same colors from the source image in our task, we added the histogram loss to the generator cost function. However, we refrain from adding it as input to the last layers of the generator as we do not intend to change the histogram. The generator’s objective function uses a scaling factor λ_{his} and has the form:

$$\mathcal{L}_G = -\mathbb{E}_{x_s}[\log D(x_s, G(x_s))] + \lambda_{\text{rec}}\mathcal{L}_{\text{rec}} + \lambda_{\text{his}}\mathcal{L}_{\text{his}}$$

where the $\mathcal{L}_{\text{his}}$ is the Hellinger distance [Simpson, 1987] between the histograms of the source H_s and target image H_t :

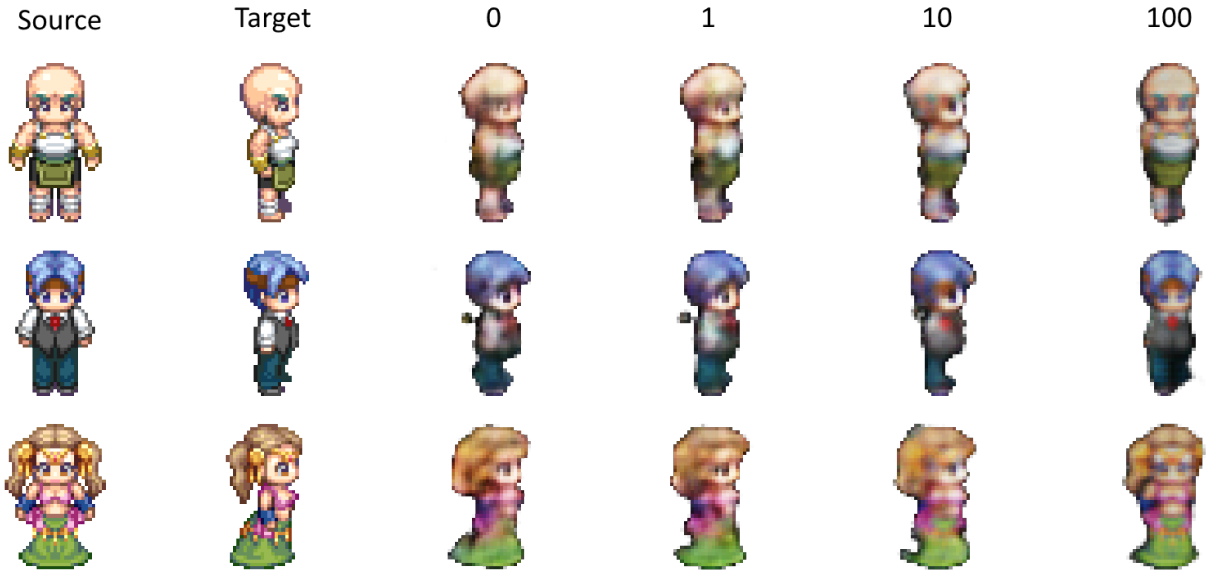
$$\mathcal{L}_{\text{his}}(H_s, H_t) = \frac{1}{\sqrt{2}} \|H_s^{1/2} - H_t^{1/2}\|_2$$

Table 6.1: FID and L_1 metrics comparing the baseline and the histogram loss hypothesis.

Model	FID	L_1
Baseline	7.876	0.05792
Histogram Loss	7.349	0.05758

We tested different combinations of the λ_{his} hyperparameter, fixing $\lambda_{\text{rec}} = 100$ (Figure 6.3). When the histogram term is higher than $\lambda_{\text{rec}}\mathcal{L}_{\text{rec}}$, the generator fails to translate the character to the target side but more rapidly reduces the histogram loss. In turn, for too low λ_{his} , the loss acts mainly as a regularizer, preventing the model from memorizing the training data but adding little to no gain over the quality of the images on the test set.

Figure 6.3: Images generated by the model trained with the histogram loss, varying the scaling factor. The **columns** show the source, target, and the images generated by models using different λ_{his} values. The **rows** present different examples.



After the hyperparameter selection, we picked $\lambda_{\text{his}} = 1$, $\lambda_{\text{rec}} = 100$ and trained a model with Adam ($\beta_1 = 0.5$ and $\beta_2 = 0.999$) and a learning rate of 0.0002 for 40,000 steps. The results can be seen in Figure 6.4 and quantitatively on Table 6.1. The figure shows example outputs from the model trained with the histogram loss compared to the Pix2Pix baseline. We can observe that the images generated by the model with histogram loss yielded slightly better images, which is endorsed by the lower FID and similar L_1 distance. However, even though there is some quality improvement with the proposed model, the histogram loss does not restrict the model to use only the colors in the palette. Hence, we investigate a third hypothesis of a post process step in the next section.

6.3 Palette quantization Post-Process

To avoid having small variations of the same color in a sprite, we created an experiment that adds a post-processing step that enforces the generated images to only have colors from the source images' palettes. As it typically reduces the number of colors in the final image, we called it palette quantization. It works by replacing each pixel of the generated image by its closest color (in some color space) in the palette of the source image. As a post-process step, it does not participate in the learning process, and is executed in an ad-hoc manner outside the deep generative model.

Figure 6.4: Images generated by the model with the histogram loss. The **columns** show the source, target, the Pix2Pix baseline, and the histogram loss hypothesis. **Rows** show different examples.



We ran different experiments to determine which color space would yield better results when finding the closest colors, comparing: RGB, YUV, and CIELAB. YUV is a color model that represents luminance information separately from the chroma of the color, with the Y term encoding a grayscale value, and U and V being a blue-green and yellow-red projections [Jiang and Sweetser, 2022]. CIELAB, in turn, is a color space intended to represent colors in a perceptually uniform space, with the L component encoding the lightness, and A and B encoding green-magenta and blue-yellow spectra [Fairchild and Berns, 1993].

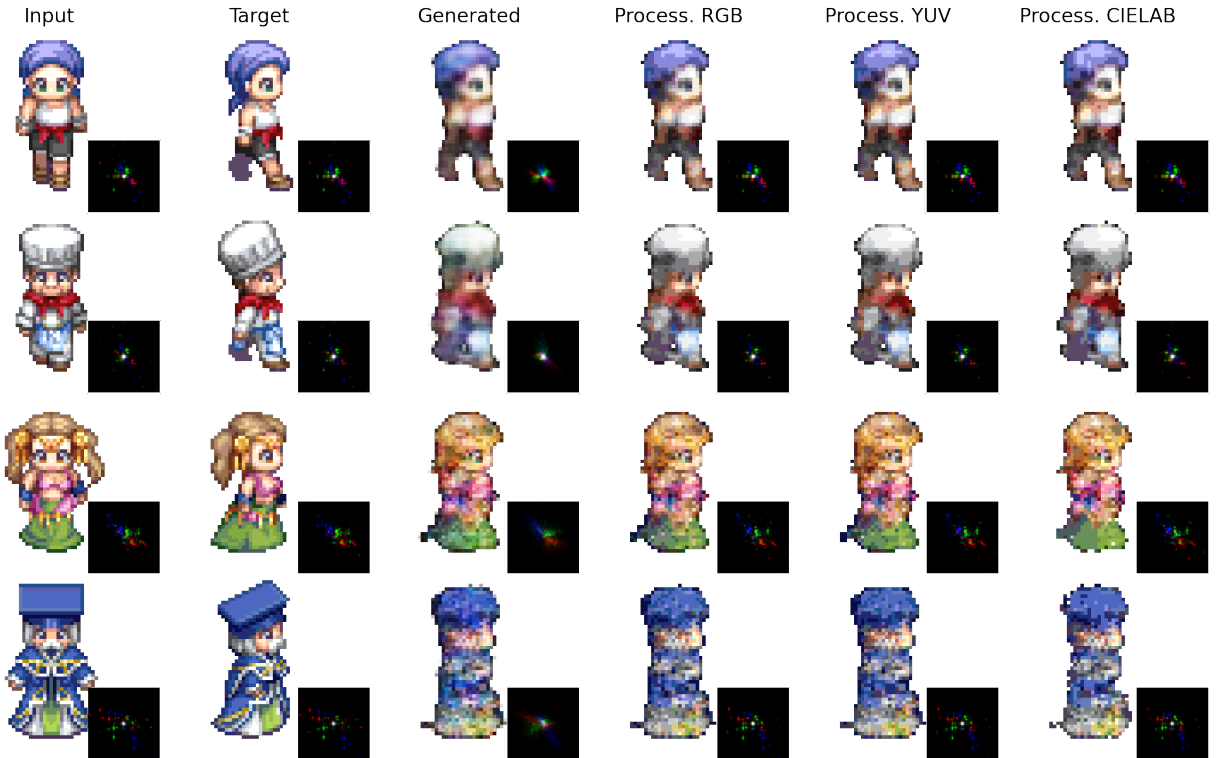
To evaluate the palette quantization in different scenarios, we experimented with models trained with and without data augmentation (hue rotation and translation). Table 6.2 presents the FID and L_1 metrics of the resulting images, and Figure 6.5 shows examples generated by the models before and after the palette quantization step using the different color spaces, together with their respective RGB-uv histograms.

Quantitatively, FID and L_1 diverge: the palette quantization using the CIELAB space to find the closest colors yields the best FID, while not using any post-processing yields the best L_1 for the experiments with and without augmentation. In these experi-

Table 6.2: FID and L_1 for images after post-processing using different color spaces.

Post-process	With Augm.		Without Augm.	
	FID	L_1	FID	L_1
None	9.400	0.05887	9.623	0.06068
Using RGB	8.778	0.05922	9.713	0.06324
Using YUV	8.892	0.05921	9.526	0.06322
Using CIELAB	8.176	0.05973	8.306	0.06403

Figure 6.5: Resulting images after the palette quantization post-processing step for test images from the RPG Maker XP dataset. The **columns** show the input, target, and generated image from left to right, followed by the post-processed versions using RGB representation, YUV, and CIELAB. **Rows** 1-2 are from the model trained with augmentation, and 3-4 are from the model without augmentation.



ments, we understand **FID** as a better indicator of quality, as it intends to compare the feasibility of the generated images compared to the target ones. On the other hand, L_1 becomes a less reliable metric as it can be more heavily impacted by colors being replaced by wrong ones. For instance, a generated character can have a region of pixels with slightly different colors that are close to the intended tone from the target image (hence, very low L_1 distance for that region). Then, after the palette quantization, a few pixels change to the wrong color from the palette: the resulting L_1 might be greater than before post-processing.

Regarding the color space to find the closest color from the palette, using CIELAB yielded the best FID results: 8.176 (with augmentation) and 8.306 (without augmentation). We hypothesize that as it is a more perceptual color space, the pixels are matched to the palette in a more meaningful way. An example of this improved mapping can be seen on the cook character in row 2 of Figure 6.5: his red scarf appears more continuous if the post-processing uses the CIELAB color space than if using RGB or YUV.

Qualitatively, the three representations yield similar results: they mostly do not change the character shape (except for a few pixels), and they reduce the model uncertainty by spreading the same color in larger regions (e.g., note the sailor’s bandana in row 1, and the dancer’s skirt in row 3, especially using CIELAB). Such benefits are more expressive with the model trained with data augmentation (rows 1-2).

Another benefit is that the sprites after the palette quantization contain only colors from the intended palette, making them more useful to pixel art designers. This can be confirmed by the histograms of the generated images before and after the post-processing. Figure 6.5 shows an RGB-uv histogram by the side of each image. We can observe that prior to the palette quantization step (3rd column), the histogram is more continuous, demonstrating that many colors are used, whereas after post-processing (using any representation), the histogram is more discrete, much closer to the one from pixel art images.

6.4 Differentiable Palette Quantization

Even though quantizing the generated images to their intended palette as a post-processing step improves not only their quality, but also their utility to game artists, it cannot be integrated within the generative model as it is not differentiable and would break the gradient flow and preclude backpropagation.

The primary challenge in integrating palette quantization in a model training end-to-end lies in the non-differentiability of operations such as finding the palette color with the minimum distance to the pixel (`tf.argmax`) or assigning an untracked value to an output (i.e., replacing the color of a pixel from a generated image), which are incompatible with gradient-based optimization. To address a similar situation, [Binnering and Sorkine-Hornung \[2024\]](#) proposed leveraging the Gumbel-Softmax reparameterization trick, enabling differentiable sampling from a discrete color palette.

In this section, we introduce a related but distinct and novel approach to enforce palette adherence by employing a softmin with temperature function and a weighted-average color replacement method. Both approximate non-differentiable operations while preserving gradient flow. Furthermore, we propose an auxiliary palette coverage loss to

encourage the generator to utilize the full diversity of the target palette, mitigating color underutilization.

6.4.1 Softmin with Temperature

The seminal work of [Kirkpatrick et al. \[1983\]](#) introduced simulated annealing, a stochastic optimization method inspired by thermodynamic cooling processes, which balances exploration and exploitation through a temperature-controlled acceptance criterion. This concept was later adapted to modern deep learning [[Hinton et al., 2015](#)] in the context of knowledge distillation, where a temperature-scaled softmax enables smoother probability distributions for transferring knowledge from a teacher to a student network.

Our approach employs a softmin variant to enable differentiable color quantization. While the *argmin* operation performs a hard selection from a categorical distribution, the *softmin with temperature* provides a probabilistic alternative, allowing gradient-based optimization over discrete decisions (i.e., selecting a palette color). Given a vector of distances $\mathbf{d}$, the softmin is defined as

$$\text{softmin}(\mathbf{d}, \tau)_i = \frac{e^{-d_i/\tau}}{\sum_j e^{-d_j/\tau}},$$

where the temperature τ controls the sharpness of the distribution—high τ yields nearly uniform probabilities, while $\tau \rightarrow 0$ approaches the one-hot *argmin* operation.

This approach allows gradients to flow through sampling operations, making it suitable for end-to-end training. As with simulated annealing [[Kirkpatrick et al., 1983](#)], the temperature is initially high and progressively decays, approaching zero at the end of training.

6.4.2 Weighted-Average Replacement

After computing the softmin distances between each pixel and the palette colors, we replace each pixel’s color with a weighted average of all palette colors, where the weights are derived from their distances in RGBA space.

Formally, let $I \in \mathbb{R}^{H \times W \times C}$ denote an image and $P \in \mathbb{R}^{K \times C}$ a target palette with K colors, where $H \times W$ is the image resolution and C the number of channels. The squared L_2 distance between pixel I_{hw} and palette color P_k is given by $D_{hw,k}$:

$$D_{hw,k} = \|I_{hw} - P_k\|_2^2.$$

Each pixel $I_{hw}^{P\text{-quantized}}$ in the quantized image is then obtained via an average weighted by the distance to each color in the palette:

$$I_{hw}^{P\text{-quantized}} = \sum_{k=1}^K \text{softmax}(D_{hw,k}, \tau)_k \cdot P_k.$$

This approach ensures differentiability, as the weighted average operation admits well-defined gradients for backpropagation. The temperature τ is initialized to a moderate value (e.g., $\tau = 1$) and decays to zero during training. Consequently, the generated images transition from smooth color interpolations early in training to strict palette adherence at convergence. Notably, the softmax guarantees that all output colors lie within the convex hull of the palette, preserving at least softly the artistic constraints throughout the optimization.

Figure 6.6 illustrates the impact of the temperature parameter τ in the palette quantization with softmax temperature and weighted-average replacement. For higher τ values, the generator produces smoothly blended colors approaching the palette mean, whilst when $\tau \rightarrow 0$, the quantization forces exact palette adherence. When $\tau = 0$, the generated images employ a color palette closely approximating that of the target images, though not identical due to two key factors:

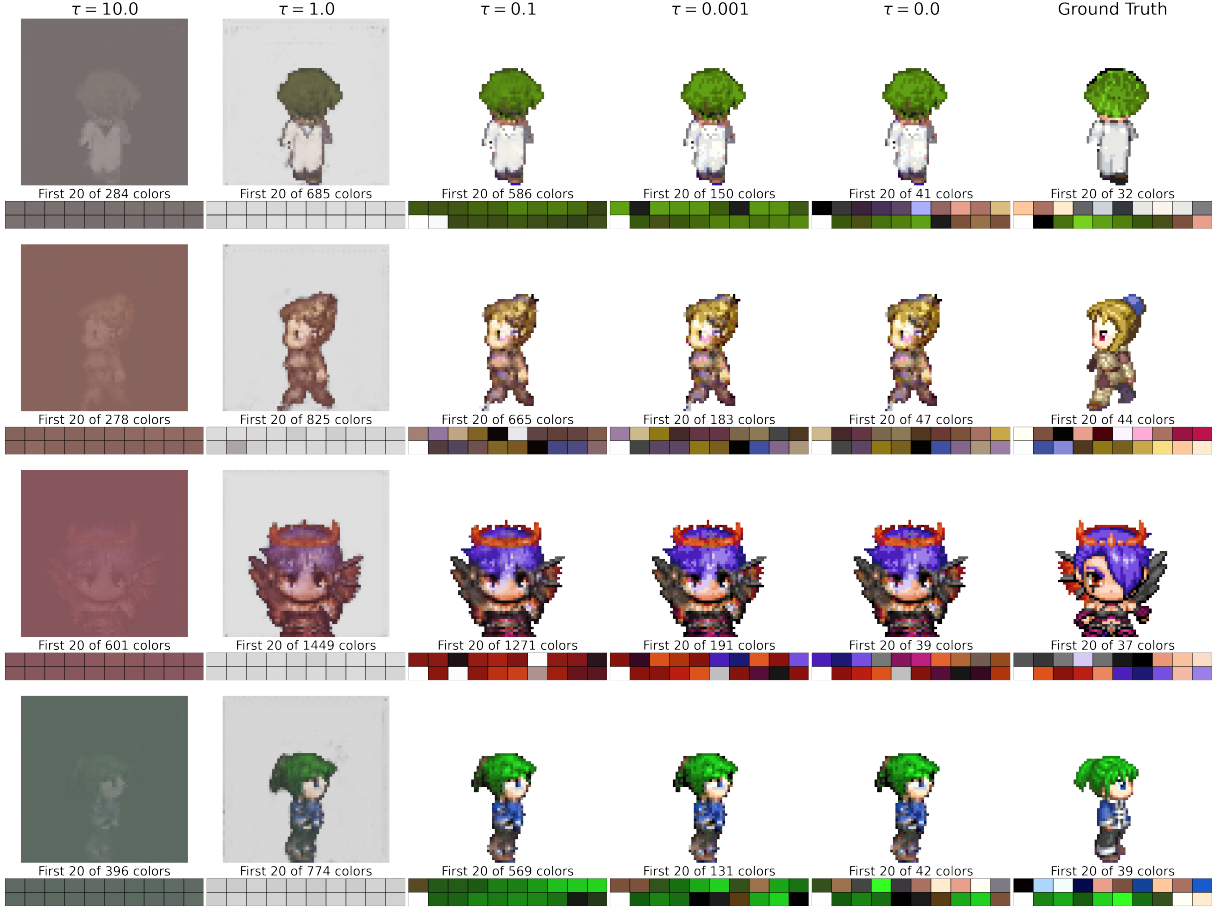
- (a) **Palette discrepancy:** we provide the generator with the desired palette extracted from the union of the known *source* images as input. This may result in a palette that is slightly different from the ground truth of the missing *target* image. For example, the target image might use a few colors not seen on any other pose used as input, or it might not use the eye colors if the target is the back pose.
- (b) **Partial palette utilization:** while our palette quantization procedure ensures that generated images exclusively employ colors from the target palette, it does not enforce the use of all palette colors, leading to potential color under-utilization.

Next, we present our strategy to address this issue of color under-use.

6.4.3 Palette Coverage Loss

While the palette quantization ensures that generated images adhere strictly to the prescribed color palette, this constraint alone does not guarantee that all palette colors

Figure 6.6: Visualization of temperature parameter (τ) effects on generated pixel art using differentiable palette quantization. High τ values produce smoothly blended colors approaching the palette mean, demonstrating the softmin operation’s probabilistic nature during early training stages when the temperature is higher. As $\tau \rightarrow 0$, the quantization becomes deterministic, forcing exact palette adherence while maintaining gradient flow. The progression illustrates our method’s annealing strategy, where τ is gradually reduced during training to achieve color diversity and strict palette compliance.



are actively utilized. For instance, the model could theoretically produce valid outputs using only a single color from the palette while entirely neglecting others.

Empirically, we observed this limitation in cases where fine details—such as a character’s eyes, which often occupy only a few pixels—were rendered without their intended palette colors. To address this, we introduce a *palette coverage loss* ($\mathcal{L}_{\text{pcov}}$) that explicitly encourages the generator to employ the full diversity of the palette.

We break down the loss calculation in three steps:

Soft Color Assignment: For each pixel I_{hw} , we compute its soft assignment probability $\alpha_{hw,k}$ to palette color P_k using the temperature-scaled softmin:

$$\alpha_{hw,k} = \frac{e^{-D_{hw,k}/\tau}}{\sum_{j=1}^K e^{-D_{hw,j}/\tau}}.$$

Color Usage Estimation: The aggregate usage β_k of each palette color P_k is obtained by summing its soft assignments across all pixels:

$$\beta_k = \sum_{h=1}^H \sum_{w=1}^W \alpha_{hw,k}.$$

Underutilization Penalty: The coverage loss penalizes underused colors ($\beta_k < 1$) via a hinge function, ensuring gradients only propagate for neglected colors. The final loss averages $\mathcal{L}_{\text{pcov},k}$ over all K palette colors:

$$\begin{aligned} \mathcal{L}_{\text{pcov},k} &= \max(1 - \beta_k, 0), \\ \mathcal{L}_{\text{pcov}} &= \frac{1}{K} \sum_{k=1}^K \mathcal{L}_{\text{pcov},k}. \end{aligned}$$

The $\mathcal{L}_{\text{pcov}}$ objective is added to the generator loss function $\mathcal{L}_G$ scaled by λ_{pcov} , which we empirically selected as 1.

Figure 6.7 compares character generations under different quantization strategies using the CollaGAN-3 model: (1) no palette quantization, (2) post-process quantization, and our end-to-end quantization approach, (3) without and (4) with the palette coverage loss.

Analysis of the generated images reveals that palette quantization significantly enhances output quality, particularly in terms of utility, by reducing the number of colors—a critical feature for pixel art. However, post-process quantization (4th column) frequently introduces incorrect color assignments. In contrast, our end-to-end quantization approach (5th column) produces results more consistent with the ground truth, though it may still underutilize certain palette colors. Including the palette coverage loss (6th column) further mitigates this under-utilization, as evidenced by more accurate eye color representation.

6.5 Discussion

In this chapter we proposed changes to the image generation process with the goal of improving the quality and utility of the images. We investigated four hypotheses based

Figure 6.7: Comparison of images generated using different palette quantization mechanisms. The **columns** display the input and target images, the image (a) from CollaGAN-3 (no quantization), (b) CollaGAN-3 with quantization as a post-process step, (c) CollaGAN-3 with the embedded quantization approach (end-to-end training), and (d) the embedded quantization with the additional palette coverage loss.



on domain-specific knowledge of pixel art, in particular, on the characteristics of its color restriction and distribution within a limited palette.

First, by representing the pixels of images as indices of colors in its palette and turning the generation problem into a semantic segmentation task, we accentuated the overfitting problem, having the model completely memorizing the training data (perfectly

reproducing it) while producing images from the test partition that got progressively further from the target as the training progressed. After observing that the order in which colors appear in the palette impact on the quality of the generated images we raised a new hypothesis of representing images, instead of in RGBA or integer indices, in a learned higher-dimensional space that captures the relation between colors, similar to how embeddings encode word relations.

Second, we investigated adding a loss term to the generator that penalizes it for creating images with histograms that are different from the source sprites'. We evaluated different scaling factors for the term in the generator's objective function and found that if it is set too high, it hinders the translation to the target domain. Although it led to the generation of images with a closer histogram to the ground truth, the quality improved only slightly.

Third, we proposed a post-processing step that replaces each pixel of the generated image with the closest color from the source image's palette. We found that comparing the distances in the CIELAB space yielded the best results. Quantitatively, the palette quantization improves the FID metric but worsens L_1 . In practice, it makes the resulting images more useful as they strictly follow the source image's palette without adding many new colors.

Lastly, we introduce a novel differential palette quantization technique that is friendly to gradient-based optimization and allows training models end-to-end to produce images that strictly adhere to their intended palette. It uses a softmax with temperature approach that replaces pixels by a weighted average of the palette colors. It also involves a training penalty for the generator to steer it towards using all of the available colors.

Table 6.3: FID and L_1 for the models that implement the proposed hypotheses.

Model	FID	L_1
Palette Representation	9.551	0.07415
Baseline	7.876	0.05792
Baseline + Post Process	7.594	0.05845
Histogram	7.349	0.05758
Histogram + Post Process	6.907	0.05818
End-to-End + Palette Coverage	5.653	0.06153

As a comparison of the results, Table 6.3 summarizes the values of FID and L_1 of the investigated hypotheses, compared to the Pix2Pix baseline. The palette representation model used $\lambda_{cce} = 0.01$, the histogram loss model trained with $\lambda_{his} = 1$, the post-processing used the CIELAB space to compare color distances, and the end-to-end model using $\lambda_{pcov} = 1$.

The palette representation has the highest (worst) metrics, while the model trained with end-to-end palette quantization has the lowest FID but a high L_1 . The histogram loss

with the post processing step has the second smallest FID and a low L_1 . For a qualitative comparison, Figure 6.8 shows images generated by the baseline and the models following each of the investigated hypotheses. A visual inspection of the generated images shows the slightly better defined results of the model trained end-to-end and the histogram + palette quantization model (7th and 8th columns) compared to the others. We can also see that the post processing step improves the baseline result (5th vs 4th column), and also that the histogram loss model (6th column) is slightly better than the baseline (3rd column).

Overall, even though our proposed hypotheses of the histogram loss, the post-processing step, and the differentiable palette quantization yielded better results, the generated images are still not good enough to exempt from manual work from artists that might use them. Hence, there is room for the investigation of new hypotheses that might improve even more the quality of the generated images, especially for characters that are very different from the ones seen during training.

Figure 6.8: Comparison of the images generated following each of the investigated hypotheses. The **columns** show the source, target, the palette-represented hypothesis, the Pix2Pix baseline, the post processed baseline, the histogram loss hypothesis, its post-processed version, and the model trained end-to-end with palette quantization. **Rows** show different examples.



Chapter 7

Conclusion

In this dissertation, we approached the procedural generation of pixel art character in different poses as both an image-to-image translation and a missing-data imputation problem, and proposed five different GAN architectures. The first four were based on existing architectures—Pix2Pix, StarGAN, CollaGAN, and ReMIC, and the models achieved results with varying quality, which is better when creating images from characters that are similar to those seen during training, but not so good for those with much deviation from the training set.

Going from simpler to more complex architectures, the proposed models have different characteristics. While the most straightforward, based on Pix2Pix, can perform translation from only a single to a target domain, StarGAN and the others are multi-domain, hence, can generate images in any target domain. In addition, the models based on CollaGAN and ReMIC perform missing-data imputation and can receive more than one input image, when available, to generate a potentially better image in the target domain. Lastly, ReMIC can produce all of the missing images at once instead of having to be run once for each one not in the source set.

We proposed changes to the underlying architectures and training procedures that yielded improved results in our task compared to their original works. Such changes include differences in the networks' topologies, parameter number, the discriminators' role, batch selection strategies, etc.

Even when the models can generate images that are close to their ground truth, the number of colors displayed in the images is much larger as they contain many small variations of the same color. That is a very undesired characteristic, as the pixel art style requires hard constraints in the number of colors used in an image.

We also introduced a fifth model, which we called SpriteGAN, and was specifically tailored for the task of generating pixel art characters in different poses. Even though the quality of the generated images is not on par with those from the first models, SpriteGAN exhibits interesting characteristics: it can inpaint small missing regions from characters as well as produce slightly different outputs for the same source images. Additionally, as it uses a differentiable palette quantization operation, every image produced strictly follows a desired palette, making them more useful.

We sought to improve the quality and utility of the generated images by proposing four different strategies of integrating domain-specific knowledge from the pixel art style into the generation process. While representing images as indices in a color palette resulted in strong overfitting to the training data, decreasing the image quality, the hypotheses of penalizing the generator for creating images with an undesired histogram and quantizing the resulting image to the colors of the intended palette (both as a post-process step and integrated into the model) yielded images with higher quality.

The fact that a model can train end-to-end with palette quantization through our proposed differentiable method makes the generated images potentially much more useful to artists, as even if they might require some cleanup or adaptation, they are guaranteed to strictly adhere to the intended color palette.

Finally, we conclude by discoursing about the limited generalization capacity exhibited by the models, which produce low-quality images when the input characters are too different from those seen during training. We believe that the low number of training examples ($\sim 12k$) is the major hindrance to achieving good-enough generalization capacity. We have implemented numerous different ideas that did not yield better results and, hence, did not appear in this dissertation. That leads us into thinking that we might have reached a ceiling in the image quality in our problem setting: low number of examples, simple feed-forward inference of a network to generate images, and small and specialized models.

Nevertheless, we have become aware of commercial products¹ that are apparently based on Diffusion Models [Ho et al., 2020] and can generate pixel art characters in a different pose given an image in a source, and they exhibit great quality. However, we were unable to find any publications regarding their methods, and they did not answer any of our communication attempts. Nonetheless, we can argue that as their architectures use pre-trained Diffusion Models, they have been trained with datasets in the hundreds of millions to billions of images, which is 4-5 orders of magnitude higher than what we have available. Of course, the vast majority of that is not pixel art imagery, which was probably used only to fine-tune checkpoints.

Another major difference between ours and their approaches is that their models are likely much larger than ours, requiring much more powerful hardware. Lastly, due to the online optimization nature of the Diffusion Models, they produce images through a denoising process that takes much longer to execute than a simple feed-forward of a generator network.

Even though we did not explore fine-tuning large and general Diffusion Models due to the constraints we chose to follow, we inspired ourselves to propose some of the new lines of investigation, which are presented next.

¹PixelLab: <https://www.pixellab.ai/> and Retro Diffusion: <https://www.retrodiffusion.ai/>.

7.1 Future Work

Now, we describe a few directions which we believe this research can pursue.

Model selection for SpriteGAN: keep investigating the SpriteGAN architecture by doing a thorough hyperparameter sweep to find models that produce better images on the imputation, inpainting and the multi-modality tasks.

Specialized evaluation metrics: we have relied on the L_1 distance and [FID](#) to evaluate the quality of the generated images, but neither contemplate the particularities of the pixel art style. Hence, we propose an investigation of better metrics that can evaluate important aspects of the generated images in our task, such as the shape of the characters, the number of colors used, and the adherence to some desired palette.

Feature fusion mechanisms: the missing-data imputation architectures we implemented (i.e., based on CollaGAN and on ReMIC) mix the features from the available images by concatenating them along the channels axis and they rely on convolution operations to understand which domains are available and to combine them. SpriteGAN, in turn, uses an auxiliary [MLP](#) to produce domain availability embeddings that could, in theory, allow the model to learn the relationships between the domains.

More recently, researchers have proposed other methods, such as DFUM [[Zhang et al., 2024](#)] and RTF [[Cao et al., 2025](#)], which leverage attention layers besides convolutions. Hence, it is worth exploring how the features from the available domains can be fused together to produce the target image using different feature fusion mechanisms other than concatenation.

Multi-modal architectures: even though ReMIC uses a disentangled representation for an image’s style and content, the use of a regression loss between the generated images and the ground truth leads the model into completely ignoring the style code, hampering its ability to produce different outputs for the same input. SpriteGAN, in turn, exhibited some ability to produce varied outputs, but still with very limited diversity.

Nonetheless, we consider that multi-modal models are more useful to artists as they can provide different suggestions instead of always the same, letting the users choose which best fits their needs. Hence, we propose investigating other multi-modal architectures for the task we tackled in this work.

Textual information: the large and general-purpose Diffusion Models such as Stable Diffusion [[Rombach et al., 2022](#)] and Flux [[Labs et al., 2025](#)] train by conditioning

each image with a textual caption. That has at least two advantages: first, it allows a trained model to produce new images conditioned on text. Second and more interesting to us is that by using the textual embeddings from language models trained with lots of text, the model can learn a much more structured latent space.

Comparing to our situation, having a dataset of only $\sim 12k$ examples, the model does not have sufficient data to learn many concepts. For instance, some of our proposed models had trouble generating characters with capes as they do not understand that a cape covers most of the back; however, if captions were available and associated with each example in our training set, that could lead the model into understanding that a cape must cover most of the body when looking from the back. Hence, we believe that integrating textual information can contribute to our models having a more structured latent space and produce better images.

Additional conditioning: there are different specialization methods for pre-trained Diffusion Models, such as regular fine-tuning, DreamBooth [Ruiz et al., 2022], Low-Rank Adaptation (LoRA [Hu et al., 2021]), and ControlNets [Zhang et al., 2023]. Even though they cannot be used for GANs, simply conditioning the generation on more information might help the models learn a more useful latent space.

For instance, ControlNets have allowed Stable Diffusion to generate photos of people in a custom pose encoded in a skeleton image. If we can acquire or produce the extra information of each character’s body parts/skeleton in some pixel art character dataset, that might improve the quality of the generated images.

Integrate with an image editor: a seminal goal of this project was to deliver the proposed generative models inside image editors. However, that task unfortunately did not fit in our scope.

Doing this would have a practical contribution of making the use of the models more accessible to digital artists, as they would be available inside the environment they are comfortable working in. But it could also create interesting testing grounds where research on mixed-initiative systems could take place. Hence, we suggest integrating the proposed models inside an image editor as a plugin that can execute on the own user’s machine, due to their much smaller size compared to general purpose generative models.

7.2 Ethical Considerations

The development of deep generative models carries important ethical implications, particularly regarding their training data. The prevailing practice of training large-scale models on massive, indiscriminately scraped datasets raises significant concerns about intellectual property and the rights of content creators. Many artists and illustrators have rightly questioned the use of their copyrighted works without consent, attribution, or compensation, especially to train commercial systems.

Additionally, the computational costs associated with performing inference on very large generative models raise further concerns. High computational demands require substantial energy and water consumption, which can negatively impact both infrastructure and the environment.

This dissertation explicitly considered these issues in its methodological choices. Rather than employing or building upon general-purpose models such as Stable Diffusion [Rombach et al., 2022] or Flux [Labs et al., 2025], which are models that very likely relied on unlicensed image data during training, we focused on models trained from scratch on limited-scale datasets with appropriately permissive licenses. While this approach ensures an ethical use of data, it also constrains the model’s generalization capacity.

Possible directions for advancing the field responsibly include the development of specialized legislation governing energy and water consumption and the fair use of creative content. Such measures could encompass mandatory opt-in or opt-out mechanisms for data collection and royalty systems to compensate content authors who consent to share their work.

Looking forward, the field needs to balance the advances in generative models with sustainable and ethical practices. By highlighting these issues and demonstrating that small, specialized models can be effective, at least up to some point, and responsible, this work aims to contribute to a more ethical paradigm for the research and application of generative models.

References

- Activision (1982). *River Raid*. [Atari 2600 cartridge]. Activision, 1982.
- Afi, Mahmoud, Marcus A. Brubaker, and Michael S. Brown (2021). “HistoGAN: Controlling Colors of GAN-Generated and Real Images via Color Histograms”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. Virtual: IEEE Computer Society, Nov. 2021, pp. 7941–7950. ISBN: 9781665445092. DOI: [10.48550/arXiv.2011.11731](https://doi.org/10.48550/arXiv.2011.11731).
- Akenine-Möller, Tomas, Eric Haines, and Naty Hoffman (2018). *Real-Time Rendering*. 4th edition. Boca Raton, FL: A K Peters/CRC Press, 2018. ISBN: 9781315365459. DOI: [10.1201/9781315365459](https://doi.org/10.1201/9781315365459).
- Arjovsky, Martin, Soumith Chintala, and Léon Bottou (2017). “Wasserstein Generative Adversarial Networks”. In: *Proceedings of the 34th International Conference on Machine Learning*. PMLR, Jan. 2017, pp. 214–223.
- Aycock, John (2016). “Procedural Content Generation”. In: *Retrogame Archeology*. Cham: Springer International Publishing, 2016, pp. 109–143. DOI: [10.1007/978-3-319-30004-7_6](https://doi.org/10.1007/978-3-319-30004-7_6).
- Baldwin, Alexander, Steve Dahlskog, Jose M. Font, and Johan Holmberg (2017). “Towards pattern-based mixed-initiative dungeon generation”. In: *ACM International Conference Proceeding Series*. Vol. Part F1301. New York, New York, USA: Association for Computing Machinery, Aug. 2017, pp. 1–10. ISBN: 9781450353199. DOI: [10.1145/3102071.3110572](https://doi.org/10.1145/3102071.3110572).
- Binninger, Alexandre and Olga Sorkine-Hornung (2024). “SD-piXL: Generating Low-Resolution Quantized Imagery via Score Distillation”. In: *SIGGRAPH Asia 2024 Conference Papers*. Ed. by Alexandre Binninger and Olga Sorkine-Hornung. New York, USA, Oct. 2024, pp. 1–12. ISBN: 9798400711312.
- Buzuti, Lucas F. and Carlos E. Thomaz (2023). “Fréchet AutoEncoder Distance: A new approach for evaluation of Generative Adversarial Networks”. In: *Computer Vision and Image Understanding* 235 (Oct. 2023), pp. 1–11. ISSN: 10773142. DOI: [10.1016/j.cviu.2023.103768](https://doi.org/10.1016/j.cviu.2023.103768).
- Cao, Bing, Guoliang Qi, Jiaming Zhao, Pengfei Zhu, Qinghua Hu, and Xinbo Gao (2025). “RTF: Recursive TransFusion for Multi-Modal Image Synthesis”. In: *IEEE Transactions on Image Processing* 34 (2025), pp. 1573–1587. ISSN: 19410042. DOI: [10.1109/TIP.2025.3541877](https://doi.org/10.1109/TIP.2025.3541877); PAGE:STRING:ARTICLE/CHAPTER. URL: <https://dl.acm.org/doi/10.1109/TIP.2025.3541877>.

- Choi, Jong-In, Soo-Kyun Kim, and Shin-Jin Kang (2022). “Image Translation Method for Game Character Sprite Drawing”. In: *Computer Modeling in Engineering & Sciences* 131.2 (2022), pp. 747–762. ISSN: 1526-1506. DOI: [10.32604/cmes.2022.018201](https://doi.org/10.32604/cmes.2022.018201).
- Choi, Yunjey, Minje Choi, Munyoung Kim, Jung-Woo Ha, Sunghun Kim, and Jaegul Choo (2018). “StarGAN: Unified Generative Adversarial Networks for Multi-domain Image-to-Image Translation”. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. Salt Lake City: IEEE, June 2018, pp. 8789–8797. ISBN: 978-1-5386-6420-9. DOI: [10.1109/CVPR.2018.00916](https://doi.org/10.1109/CVPR.2018.00916).
- Choi, Yunjey, Youngjung Uh, Jaejun Yoo, and Jung-Woo Ha (2020). “StarGAN v2: Diverse Image Synthesis for Multiple Domains”. In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. Seattle: IEEE, June 2020, pp. 8185–8194. ISBN: 978-1-7281-7168-5. DOI: [10.1109/CVPR42600.2020.00821](https://doi.org/10.1109/CVPR42600.2020.00821).
- Coutinho, Flávio and Luiz Chaimowicz (2022a). “Generating Pixel Art Character Sprites using GANs”. In: *2022 21st Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)*. Natal, Brazil: IEEE, Oct. 2022, pp. 1–6. ISBN: 978-1-6654-6156-6. DOI: [10.1109/SBGAMES56371.2022.9961120](https://doi.org/10.1109/SBGAMES56371.2022.9961120).
- Coutinho, Flávio and Luiz Chaimowicz (2022b). “On the Challenges of Generating Pixel Art Character Sprites Using GANs”. In: *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment* 18.1 (Oct. 2022), pp. 87–94. ISSN: 2334-0924. DOI: [10.1609/AIIDE.V18I1.21951](https://doi.org/10.1609/AIIDE.V18I1.21951).
- Coutinho, Flávio and Luiz Chaimowicz (2024a). “A Missing Data Imputation GAN for Character Sprite Generation”. In: *Proceedings of the XXIII Brazilian Symposium on Digital Games and Entertainment (SBGames 2024)*. Vol. 1. Manaus, Brazil, Sept. 2024, p. 10. URL: <http://arxiv.org/abs/2409.10721>.
- Coutinho, Flávio and Luiz Chaimowicz (2024b). “Pixel art character generation as an image-to-image translation problem using GANs”. In: *Graphical Models* 132 (Apr. 2024), p. 101213. ISSN: 1524-0703. DOI: [10.1016/J.GMOD.2024.101213](https://doi.org/10.1016/J.GMOD.2024.101213).
- Coutinho, Flávio, Lucas G. S. Chaves, and Luiz Chaimowicz (2025). “Imputation of missing pixel art character poses with differentiable palette quantization”. In: *Entertainment Computing* 55.Special Issue on Best Papers of the SBGames 2024 (Sept. 2025), p. 13. ISSN: 18759521. DOI: [10.1016/j.entcom.2025.101021](https://doi.org/10.1016/j.entcom.2025.101021).
- Dapogny, Arnaud, Matthieu Cord, and Patrick Perez (2019). “The Missing Data Encoder: Cross-Channel Image Completion with Hide-And-Seek Adversarial Network”. In: *AAAI 2020 - 34th AAAI Conference on Artificial Intelligence* (May 2019), pp. 10688–10695. ISSN: 2159-5399. DOI: [10.1609/aaai.v34i07.6696](https://doi.org/10.1609/aaai.v34i07.6696). URL: <https://arxiv.org/pdf/1905.01861>.
- Fairchild, Mark D. and Roy S. Berns (1993). “Image color-appearance specification through extension of CIELAB”. In: *Color Research & Application* 18.3 (June 1993), pp. 178–190. ISSN: 1520-6378. DOI: [10.1002/COL.5080180308](https://doi.org/10.1002/COL.5080180308).

- Gallotta, Roberto, Antonios Liapis, and Georgios N Yannakakis (2025). “The Importance of Context in Image Generation: A Case Study for Video Game Sprites”. In: *Proceedings of the 14th International Conference on Artificial Intelligence in Music, Sound, Art and Design (EvoMUSART)*. Trieste, Italy: Springer, Cham, 2025, p. 16. URL: <https://github.com/gallorob/>.
- Gonzalez, Adrian, Matthew Guzdial, and Felix Ramos (2020). “Generating Gameplay-Relevant Art Assets with Transfer Learning”. In: *Proceedings of the AIIDE Workshop on Experimental AI in Games*. Worcester: ArXiv, Oct. 2020, pp. 1–7.
- Goodfellow, Ian (2016). “NIPS 2016 Tutorial: Generative Adversarial Networks”. In: (Dec. 2016). URL: <http://arxiv.org/abs/1701.00160>.
- Goodfellow, Ian J., Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio (2014). “Generative Adversarial Nets”. In: *NIPS’14: Proceedings of the 27th International Conference on Neural Information Processing Systems*. Vol. 29. 5. Cambridge: MIT Press, 2014, pp. 2672–2680. DOI: [10.5555/2969033.2969125](https://doi.org/10.5555/2969033.2969125).
- Gulrajani, Ishaan, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville (2017). “Improved Training of Wasserstein GANs”. In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. Vol. 2017-December. Red Hook, NY, USA: Curran Associates Inc., 2017, pp. 5769–5779. ISBN: 9781510860964.
- He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun (2015). “Deep Residual Learning for Image Recognition”. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* 2016-December (Dec. 2015), pp. 770–778. ISSN: 10636919. DOI: [10.1109/CVPR.2016.90](https://doi.org/10.1109/CVPR.2016.90).
- Heusel, Martin, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter (2017). “GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium”. In: *Advances in Neural Information Processing Systems* 2017-December (June 2017), pp. 6627–6638. ISSN: 10495258.
- Hinton, Geoffrey, Oriol Vinyals, and Jeff Dean (2015). “Distilling the Knowledge in a Neural Network”. In: *NIPS 2014 Deep Learning Workshop*. Montreal, Canada, 2015, p. 9.
- Ho, Jonathan, Ajay Jain, and Pieter Abbeel (2020). “Denoising Diffusion Probabilistic Models”. In: *Advances in Neural Information Processing Systems* 2020-December (June 2020). ISSN: 10495258. URL: <https://arxiv.org/pdf/2006.11239>.
- Hong, Seungjin, Sookyun Kim, and Shinjin Kang (2019). “Game sprite generator using a multi discriminator GAN”. In: *KSII Transactions on Internet and Information Systems* 13.8 (2019), pp. 4255–4269. ISSN: 22881468. DOI: [10.3837/tiis.2019.08.025](https://doi.org/10.3837/tiis.2019.08.025).
- Horsley, Lewis and Diego Liebana (2017). “Building an automatic sprite generator with deep convolutional generative adversarial networks”. In: *2017 IEEE Conference on*

- Computational Intelligence and Games, CIG 2017* (2017), pp. 134–141. DOI: [10.1109/CIG.2017.8080426](https://doi.org/10.1109/CIG.2017.8080426).
- Hsieh, Cheng-An, Jing Zhang, and Ava Yan (2025). *Sprite Sheet Diffusion: Generate Game Character for Animation*. Mar. 2025. URL: <http://arxiv.org/abs/2412.03685>.
- Hu, Edward, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen (2021). “LoRA: Low-Rank Adaptation of Large Language Models”. In: *ICLR 2022 - 10th International Conference on Learning Representations* (June 2021). URL: <https://arxiv.org/pdf/2106.09685>.
- Hu, Li, Xin Gao, Peng Zhang, Ke Sun, Bang Zhang, and Liefeng Bo (2024). “Animate Anyone: Consistent and Controllable Image-to-Video Synthesis for Character Animation”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. Seattle, USA, June 2024, pp. 8153–8163.
- Huang, Xun and Serge Belongie (2017). “Arbitrary Style Transfer in Real-time with Adaptive Instance Normalization”. In: *Proceedings of the IEEE International Conference on Computer Vision 2017-October* (Mar. 2017), pp. 1510–1519. ISSN: 15505499. DOI: [10.1109/ICCV.2017.167](https://doi.org/10.1109/ICCV.2017.167). URL: <https://arxiv.org/pdf/1703.06868>.
- Huang, Xun, Ming Yu Liu, Serge Belongie, and Jan Kautz (2018). “Multimodal Unsupervised Image-to-Image Translation”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 11207 LNCS (Apr. 2018), pp. 179–196. ISSN: 16113349. DOI: [10.1007/978-3-030-01219-9_11](https://doi.org/10.1007/978-3-030-01219-9_11).
- Ioffe, Sergey and Christian Szegedy (2015). “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. In: *International conference on machine learning*. Lille, France: PMLR, June 2015, pp. 448–456.
- Isola, Phillip, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros (2017). “Image-to-Image Translation with Conditional Adversarial Networks”. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Vol. 2017-Janua. Honolulu: IEEE, July 2017, pp. 5967–5976. ISBN: 978-1-5386-0457-1. DOI: [10.1109/CVPR.2017.632](https://doi.org/10.1109/CVPR.2017.632).
- Jiang, Zhouyang and Penny Sweetser (2022). “GAN-Assisted YUV Pixel Art Generation”. In: *Australasian Joint Conference on Artificial Intelligence*. Sydney, NSW, Australia: Springer International Publishing, 2022, pp. 595–606. DOI: [10.1007/978-3-030-97546-3_48](https://doi.org/10.1007/978-3-030-97546-3_48).
- Karavolos, Daniël, Anders Bouwer, and Rafael Bidarra (2015). “Mixed-Initiative Design of Game Levels: Integrating Mission and Space into Level Generation”. In: *Proceedings of the 10th International Conference on the Foundations of Digital Games*. 2015. ISBN: 9780991398249.
- Karras, Tero, Samuli Laine, and Timo Aila (2018). “A Style-Based Generator Architecture for Generative Adversarial Networks”. In: *IEEE Transactions on Pattern Analysis*

- and Machine Intelligence* 43.12 (Dec. 2018), pp. 4217–4228. ISSN: 19393539. DOI: [10.48550/arxiv.1812.04948](https://doi.org/10.48550/arxiv.1812.04948).
- Karras, Tero, Samuli Laine, and Timo Aila (2021). “A Style-Based Generator Architecture for Generative Adversarial Networks”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 43.12 (Dec. 2021), pp. 4217–4228. ISSN: 0162-8828. DOI: [10.1109/TPAMI.2020.2970919](https://doi.org/10.1109/TPAMI.2020.2970919).
- Kim, Jae Won, Kyo Hoon Jin, Soo Jin Jang, Shin Jin Kang, and Young Bin Kim (2023). “Game effect sprite generation with minimal data via conditional GAN”. In: *Expert Systems with Applications* 211 (Jan. 2023). ISSN: 09574174. DOI: [10.1016/j.eswa.2022.118491](https://doi.org/10.1016/j.eswa.2022.118491).
- Kingma, Diederik P. and Jimmy Lei Ba (2014). “Adam: A Method for Stochastic Optimization”. In: *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings* (Dec. 2014).
- Kingma, Diederik P. and Max Welling (2013). “Auto-Encoding Variational Bayes”. In: *2nd International Conference on Learning Representations, ICLR 2014 - Conference Track Proceedings* (Dec. 2013). DOI: [10.61603/ceas.v2i1.33](https://doi.org/10.61603/ceas.v2i1.33). URL: <https://arxiv.org/pdf/1312.6114>.
- Kirkpatrick, S., C. D. Gelatt, and M. P. Vecchi (1983). “Optimization by Simulated Annealing”. In: *Science* 220.4598 (May 1983), pp. 671–680. ISSN: 00368075. DOI: [10.1126/SCIENCE.220.4598.671](https://doi.org/10.1126/SCIENCE.220.4598.671).
- Kopf, Johannes and Dani Lischinski (2011). “Depixelizing pixel art”. In: *ACM SIGGRAPH 2011 papers on - SIGGRAPH '11*. Vol. 30. 4. New York, New York, USA: ACM Press, July 2011, p. 1. ISBN: 9781450309431. DOI: [10.1145/1964921.1964994](https://doi.org/10.1145/1964921.1964994).
- Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton (2017). “ImageNet classification with deep convolutional neural networks”. In: *Communications of the ACM* 60.6 (May 2017), pp. 84–90. ISSN: 0001-0782. DOI: [10.1145/3065386](https://doi.org/10.1145/3065386).
- Kuang, Hailan, Nan Huang, Shuchang Xu, and Shunpeng Du (2021). “A Pixel image generation algorithm based on CycleGAN”. In: *2021 IEEE 4th Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC)*. IEEE, June 2021, pp. 476–480. ISBN: 978-1-7281-8535-4. DOI: [10.1109/IMCEC51613.2021.9482118](https://doi.org/10.1109/IMCEC51613.2021.9482118).
- Labs, Black Forest, Stephen Batifol, Andreas Blattmann, et al. (2025). “FLUX.1 Kontext: Flow Matching for In-Context Image Generation and Editing in Latent Space”. In: (June 2025). URL: <https://arxiv.org/pdf/2506.15742>.
- LeCun, Yann, Léon Bottou, Yoshua Bengio, and Patrick Haffner (1998). “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2323. ISSN: 00189219. DOI: [10.1109/5.726791](https://doi.org/10.1109/5.726791).
- Lee, Dongwook, Junyoung Kim, Won-Jin Moon, and Jong Chul Ye (2019). “CollaGAN: Collaborative GAN for Missing Image Data Imputation”. In: *2019 IEEE/CVF Con-*

- ference on Computer Vision and Pattern Recognition (CVPR)*. Vol. 2019-June. Long Beach: IEEE, June 2019, pp. 2482–2491. ISBN: 978-1-7281-3293-8. DOI: [10.1109/CVPR.2019.00259](https://doi.org/10.1109/CVPR.2019.00259).
- Liapis, Antonios, Gillian Smith, and Noor Shaker (2016). “Mixed-initiative content creation”. In: *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*. Springer, 2016. Chap. 11.
- Linnainmaa, Seppo (1970). “The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors”. PhD thesis. Helsinki: University of Helsinki, 1970, p. 66.
- Liu, Guilin, Fitsum A. Reda, Kevin J. Shih, Ting Chun Wang, Andrew Tao, and Bryan Catanzaro (2018). “Image Inpainting for Irregular Holes Using Partial Convolutions”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 11215 LNCS (Apr. 2018), pp. 89–105. ISSN: 16113349. DOI: [10.1007/978-3-030-01252-6_6](https://doi.org/10.1007/978-3-030-01252-6_6). URL: <https://arxiv.org/pdf/1804.07723>.
- Liu, Ming-Yu, Thomas Breuel, and Jan Kautz (2017). “Unsupervised Image-to-Image Translation Networks”. In: *Advances in Neural Information Processing Systems*. Curran Associates, Inc., Mar. 2017, p. 9. DOI: [10.5555/3294771.3294838](https://doi.org/10.5555/3294771.3294838).
- Loftsdottir, Dagmar and Matthew Guzdial (2022). “SketchBetween: Video-to-Video Synthesis for Sprite Animation via Sketches”. In: *Proceedings of the 17th International Conference on the Foundations of Digital Games*. New York: ACM, Sept. 2022, pp. 1–7. ISBN: 9781450397957. DOI: [10.1145/3555858.3555928](https://doi.org/10.1145/3555858.3555928).
- Löwenström, Paul (2024). “Guiding generation of 2D pixel art characters using text-image similarity models: A comparative study of generating 2D pixel art characters using PixelDraw and Diffusion Model guided by text-image similarity models”. PhD thesis. KTH, School of Electrical Engineering and Computer Science (EECS), 2024, p. 44.
- Maas, Andrew L, Awni Y Hannun, and Andrew Y Ng (2013). “Rectifier Nonlinearities Improve Neural Network Acoustic Models”. In: *ICML Workshop on Deep Learning for Audio, Speech, and Language Processing* *ICML Workshop on Deep Learning for Audio, Speech, and Language Processing (WDLASL 2013)*. 2013, p. 6.
- Mao, Xudong, Qing Li, Haoran Xie, Raymond Y.K. Lau, Zhen Wang, and Stephen Paul Smolley (2017). “Least Squares Generative Adversarial Networks”. In: *2017 IEEE International Conference on Computer Vision (ICCV)*. Venice: IEEE, Oct. 2017, pp. 2813–2821. ISBN: 978-1-5386-1032-9. DOI: [10.1109/ICCV.2017.304](https://doi.org/10.1109/ICCV.2017.304).
- Merino, Timothy, Roman Negri, Dipika Rajesh, M Charity, and Julian Togelius (2023). “The Five-Dollar Model: Generating Game Maps and Sprites from Sentence Embeddings”. In: (Aug. 2023). URL: <https://arxiv.org/abs/2308.04052v1>.

- Mirza, Mehdi and Simon Osindero (2014). “Conditional Generative Adversarial Nets”. In: *arXiv preprint arXiv:1411.1784* (Nov. 2014).
- Nair, Vinod and Geoffrey E Hinton (2010). “Rectified Linear Units Improve Restricted Boltzmann Machines”. In: *Proceedings of the 27th International Conference on International Conference on Machine Learning*. Haifa, Israel, June 2010, pp. 807–814. DOI: [10.5555/3104322.3104425](https://doi.org/10.5555/3104322.3104425).
- Nikolov, Ivan Adriyanov (2025). “Light the Sprite: Pixel Art Dynamic Light Map Generation”. In: *Annual Conference of the European Association for Computer Graphics*. London, United Kingdom: The Eurographics Association, 2025, p. 4. ISBN: 978-3-03868-268-4. DOI: [10.2312/EGS.20251032](https://doi.org/10.2312/EGS.20251032). URL: <https://vbn.aau.dk/en/publications/light-the-sprite-pixel-art-dynamic-light-map-generation>.
- Nintendo (1998). *The Legend of Zelda: Ocarina of Time*. [Nintendo 64 cartridge]. Nintendo, 1998.
- Nintendo (2017). *The Legend of Zelda: Breath of the Wild*. [Nintendo Switch]. Nintendo, 2017.
- Novak, Jeannie (2022). *Game Development Essentials: An Introduction*. 4th edition. Santa Monica: NOVA UNLIMITED, INC, 2022, p. 484.
- Odena, Augustus, Christopher Olah, and Jonathon Shlens (2017). “Conditional Image Synthesis With Auxiliary Classifier GANs”. In: *34th International Conference on Machine Learning, ICML 2017*. Vol. 6. International Machine Learning Society (IMLS), Aug. 2017, pp. 4043–4055. ISBN: 9781510855144. DOI: [10.5555/3305890.3305954](https://doi.org/10.5555/3305890.3305954).
- Oord, Aaron van den, Oriol Vinyals, and Koray Kavukcuoglu (2017). “Neural Discrete Representation Learning”. In: *Advances in Neural Information Processing Systems*. Vol. 30. Long Beach: Curran Associates, Inc., 2017, pp. 1–10.
- Pang, Yingxue, Jianxin Lin, Tao Qin, and Zhibo Chen (2022). “Image-to-Image Translation: Methods and Applications”. In: *IEEE Transactions on Multimedia* 24 (2022), pp. 3859–3881. ISSN: 1520-9210. DOI: [10.1109/TMM.2021.3109419](https://doi.org/10.1109/TMM.2021.3109419).
- Perez, Ethan, Florian Strub, Harm De Vries, Vincent Dumoulin, and Aaron Courville (2017). “FiLM: Visual Reasoning with a General Conditioning Layer”. In: *32nd AAAI Conference on Artificial Intelligence, AAAI 2018* (Sept. 2017), pp. 3942–3951. ISSN: 2159-5399. DOI: [10.1609/aaai.v32i1.11671](https://doi.org/10.1609/aaai.v32i1.11671). URL: <https://arxiv.org/pdf/1709.07871>.
- Radford, Alec, Jong Wook Kim, Chris Hallacy, et al. (2022). “Learning Transferable Visual Models From Natural Language Supervision”. In: *Proceedings of the 38th International Conference on Machine Learning*. Baltimore, Maryland, EUA: PMLR, July 2022.
- Radford, Alec, Luke Metz, and Soumith Chintala (2015). “Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks”. In: *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings* (Nov. 2015). URL: <http://arxiv.org/abs/1511.06434>.

- Reed, Scott E., Yi Zhang, Yuting Zhang, and Honglak Lee (2015). “Deep Visual Analogy-Making”. In: *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*. Vol. 28. Montreal, Canada: MIT Press, Dec. 2015, pp. 1252–1260. DOI: [10.5555/2969239.2969379](https://doi.org/10.5555/2969239.2969379).
- Rombach, Robin, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Bjorn Ommer (2022). “High-Resolution Image Synthesis with Latent Diffusion Models”. In: *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. Vol. 2022-June. New Orleans, Louisiana, USA: IEEE, June 2022, pp. 10674–10685. ISBN: 978-1-6654-6946-3. DOI: [10.1109/CVPR52688.2022.01042](https://doi.org/10.1109/CVPR52688.2022.01042).
- Ronneberger, Olaf, Philipp Fischer, and Thomas Brox (2015). “U-net: Convolutional networks for biomedical image segmentation”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 9351 (2015), pp. 234–241. ISSN: 16113349. DOI: [10.1007/978-3-319-24574-4_28/COVER](https://doi.org/10.1007/978-3-319-24574-4_28/COVER).
- Ruiz, Nataniel, Yuanzhen Li, Varun Jampani, Yael Pritch, Michael Rubinstein, and Kfir Aberman (2022). “DreamBooth: Fine Tuning Text-to-Image Diffusion Models for Subject-Driven Generation”. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition 2023-June* (Aug. 2022), pp. 22500–22510. ISSN: 10636919. DOI: [10.1109/CVPR52729.2023.02155](https://doi.org/10.1109/CVPR52729.2023.02155). URL: <https://arxiv.org/pdf/2208.12242>.
- Saravanan, Akash and Matthew Guzdial (2022). “Pixel VQ-VAEs for Improved Pixel Art Representation”. In: *Experimental AI in Games Workshop (EXAG) 2022*. Pomona: ArXiv, Mar. 2022, pp. 1–9. DOI: [10.48550/arxiv.2203.12130](https://doi.org/10.48550/arxiv.2203.12130).
- Schreier, Jason (2017). *Blood, Sweat, and Pixels: The Triumphant, Turbulent Stories Behind How Video Games Are Made*. New York: HarperCollins, 2017, p. 304. ISBN: 9780062651242.
- Serpa, Ygor Rebouças and Maria Andréia Formico Rodrigues (2019). “Towards Machine-Learning Assisted Asset Generation for Games: A Study on Pixel Art Sprite Sheets”. In: *2019 18th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)*. Vol. 2019-Octob. Rio de Janeiro: IEEE, Oct. 2019, pp. 182–191. ISBN: 978-1-7281-4637-9. DOI: [10.1109/SBGames.2019.00032](https://doi.org/10.1109/SBGames.2019.00032).
- Serpa, Ygor Rebouças and Maria Andréia Formico Rodrigues (2022). “Human and machine collaboration for painting game assets with deep learning”. In: *Entertainment Computing* 43 (Aug. 2022), p. 100497. ISSN: 1875-9521. DOI: [10.1016/J.ENTCOM.2022.100497](https://doi.org/10.1016/J.ENTCOM.2022.100497).
- Sharma, Anmol and Ghassan Hamarneh (2020). “Missing MRI Pulse Sequence Synthesis Using Multi-Modal Generative Adversarial Network”. In: *IEEE Transactions on Medical Imaging* 39.4 (Apr. 2020), pp. 1170–1183. ISSN: 0278-0062. DOI: [10.1109/TMI.2019.2945521](https://doi.org/10.1109/TMI.2019.2945521).

- Shen, Liyue, Wentao Zhu, Xiaosong Wang, et al. (2021). “Multi-Domain Image Completion for Random Missing Input Data”. In: *IEEE Transactions on Medical Imaging* 40.4 (Apr. 2021), pp. 1113–1122. ISSN: 1558254X. DOI: [10.1109/TMI.2020.3046444](https://doi.org/10.1109/TMI.2020.3046444).
- Silber, Daniel (2015). *Pixel Art for Game Developers*. Boca Raton: CRC Press, 2015, p. 252. ISBN: 9781482252316.
- Simpson, Douglas G. (1987). “Minimum Hellinger distance estimation for the analysis of count data”. In: *Journal of the American Statistical Association* 82.399 (1987), pp. 802–807. ISSN: 1537274X. DOI: [10.1080/01621459.1987.10478501](https://doi.org/10.1080/01621459.1987.10478501).
- Smith, Gillian, Jim Whitehead, and Michael Mateas (2011). “Tanagra: Reactive Planning and Constraint Solving for Mixed-Initiative Level Design”. In: *IEEE Transactions on Computational Intelligence and AI in Games* 3.3 (Sept. 2011), pp. 201–215. ISSN: 1943-068X. DOI: [10.1109/TCIAIG.2011.2159716](https://doi.org/10.1109/TCIAIG.2011.2159716).
- Summerville, Adam, Sam Snodgrass, Matthew Guzdial, Christoffer Holmgård, Amy K. Hoover, Aaron Isaksen, Andy Nealen, and Julian Togelius (2018). “Procedural content generation via machine learning (PCGML)”. In: *IEEE Transactions on Games* 10.3 (June 2018), pp. 257–270. ISSN: 24751510. DOI: [10.1109/TG.2018.2846639](https://doi.org/10.1109/TG.2018.2846639).
- Szegedy, Christian, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna (2016). “Rethinking the Inception Architecture for Computer Vision”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Vol. 2016-December. IEEE, June 2016, pp. 2818–2826. ISBN: 978-1-4673-8851-1. DOI: [10.1109/CVPR.2016.308](https://doi.org/10.1109/CVPR.2016.308).
- Theis, Lucas, Aäron Van Den Oord, and Matthias Bethge (2015). “A note on the evaluation of generative models”. In: *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings* (Nov. 2015). URL: <https://arxiv.org/pdf/1511.01844>.
- Togelius, Julian, Emil Kastbjerg, David Schedl, and Georgios N. Yannakakis (2011). “What is procedural content generation?” In: *Proceedings of the 2nd International Workshop on Procedural Content Generation in Games*. New York, NY, USA: ACM, June 2011, pp. 1–6. ISBN: 9781450308724. DOI: [10.1145/2000919.2000922](https://doi.org/10.1145/2000919.2000922).
- Togelius, Julian, Noor Shaker, and Mark J Nelson (2016). “Introduction”. In: *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*. Springer, 2016. Chap. 1.
- Umetani, Nobuyuki (2017). “Exploring generative 3D shapes using autoencoder networks”. In: *SIGGRAPH Asia 2017 Technical Briefs*. New York, NY, USA: ACM, Nov. 2017, pp. 1–4. ISBN: 9781450354066. DOI: [10.1145/3145749.3145758](https://doi.org/10.1145/3145749.3145758).
- Vecchio, Giuseppe, Renato Sortino, Simone Palazzo, and Concetto Spampinato (2024). “MatFuse: Controllable Material Generation with Diffusion Models”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. Seattle, USA, June 2024, pp. 4429–4438.

- Wang, Zhou, Alan Conrad Bovik, Hamid Rahim Sheikh, and Eero P. Simoncelli (2004). “Image quality assessment: From error visibility to structural similarity”. In: *IEEE Transactions on Image Processing* 13.4 (Apr. 2004), pp. 600–612. ISSN: 10577149. DOI: [10.1109/TIP.2003.819861](https://doi.org/10.1109/TIP.2003.819861).
- Yang, Runtian, Yudi Wang, Yiran Wang, and Michelle Xu (2022). “Application of Neural Network in Pixel Art Creation: Bi-directional Conversion between Photo and Pixel Art with GAN Base Model”. In: (Feb. 2022), pp. 855–858. DOI: [10.1109/ICCECE54139.2022.9712735](https://doi.org/10.1109/ICCECE54139.2022.9712735).
- Zhang, Lvmin, Anyi Rao, and Maneesh Agrawala (2023). “Adding Conditional Control to Text-to-Image Diffusion Models”. In: *Proceedings of the IEEE International Conference on Computer Vision* (Feb. 2023), pp. 3813–3824. ISSN: 15505499. DOI: [10.1109/ICCV51070.2023.00355](https://doi.org/10.1109/ICCV51070.2023.00355). URL: <https://arxiv.org/pdf/2302.05543>.
- Zhang, Yue, Chengtao Peng, Qiuli Wang, Dan Song, Kaiyan Li, S Kevin Zhou, Kevin Y Zhou Zhang, K Li, and S K Zhou (2024). “Unified Multi-Modal Image Synthesis for Missing Modality Imputation”. In: *IEEE TRANSACTIONS ON MEDICAL IMAGING* XX.1 (July 2024), pp. 4–18. DOI: [10.1109/TMI.2024.3424785](https://doi.org/10.1109/TMI.2024.3424785). URL: <https://arxiv.org/pdf/2304.05340v2><http://dx.doi.org/10.1109/TMI.2024.3424785>.
- Zhu, Jun Yan, Richard Zhang, Deepak Pathak, Trevor Darrell, Alexei A. Efros, Oliver Wang, and Eli Shechtman (2017a). “Toward Multimodal Image-to-Image Translation”. In: *Advances in Neural Information Processing Systems*. Vol. 2017-December. Long Beach, California, USA: Neural information processing systems foundation, Nov. 2017, pp. 466–477. DOI: [10.5555/3294771.3294816](https://doi.org/10.5555/3294771.3294816).
- Zhu, Jun-Yan, Taesung Park, Phillip Isola, and Alexei A. Efros (2017b). “Unpaired Image-to-Image Translation Using Cycle-Consistent Adversarial Networks”. In: *2017 IEEE International Conference on Computer Vision (ICCV)*. Venice: IEEE, Oct. 2017, pp. 2242–2251. ISBN: 978-1-5386-1032-9. DOI: [10.1109/ICCV.2017.244](https://doi.org/10.1109/ICCV.2017.244).