Vinícius Luis Trevisan de Souza

ESTUDO SOBRE O USO DE REDES ADVERSÁRIAS GENERATIVAS PARA TRANSFORMAÇÃO DE CONTORNOS EM IMAGENS E SÍNTESE DE IMAGENS

Santo André, 2022

Universidade Federal do ABC

Centro de Matemática, Computação e Cognição

Vinícius Luis Trevisan de Souza

Estudo sobre o uso de redes adversárias generativas para transformação de contornos em imagens e síntese de imagens

Orientador: Prof. Dr. João Paulo Gois

Coorientador: Prof. Dr. Bruno Augusto Dorta Marques

Dissertação de mestrado apresentada ao Centro de Matemática, Computação e Cognição para obtenção do título de Mestre em Ciência da Computação

Santo André, 2022

Trevisan de Souza, Vinícius Luis
Estudo sobre o uso de redes adversárias generativas para transformação de contornos em imagens e síntese de imagens / Vinícius Luis Trevisan de Souza. — 2022.

196 fls.

Orientador: João Paulo Gois
Coorientador: Bruno Augusto Dorta Marques

Dissertação (Mestrado) — Universidade Federal do ABC, Programa de Pós-Graduação em Ciência da Computação, Santo André, 2022.

1. Redes Adversárias Generativas. 2. Aprendizado Profundo. 3. Processamento de Imagens. 4. Visão Computacional. 5. Computação Gráfica. I. Gois, João Paulo. II. Augusto Dorta Marques, Bruno. III. Programa de Pós-Graduação em Ciência da Computação, 2022. IV. Título.

Este exemplar foi revisado e alterado em relação à versão original,
de acordo com as observações levantadas pela banca examinadora no dia da defesa,
sob responsabilidade única do(a) autor(a) e com a anuência do(a) (co)orientador(a).

MINISTÉRIO DA EDUCAÇÃO
Fundação Universidade Federal do ABC
Avenida dos Estados, 5001 – Bairro Santa Terezinha – Santo André – SP
CEP 09210-580 · Fone: (11) 4996-0017

FOLHA DE ASSINATURAS

Assinaturas dos membros da Banca Examinadora que avaliou e aprovou a Defesa de Dissertação de Mestrado do candidato, VINÍCIUS LUIS TREVISAN DE SOUZA realizada em 30 de Novembro de 2022:

Documento assinado digitalmente
gov.br MARCELO FERREIRA SIQUEIRA
Data: 01/12/2022 00:06:03-0300
Verifique em https://verificador.iti.br

Prof.(a) MARCELO FERREIRA SIQUEIRA
ALIGN TECHNOLOGY

Documento assinado digitalmente
gov.br WALLACE CORREA DE OLIVEIRA CASACA
Data: 30/11/2022 18:20:45-0300
Verifique em https://verificador.iti.br

Prof.(a) WALLACE CORREA DE OLIVEIRA CASACA
UNIVERSIDADE ESTADUAL PAULISTA JÚLIO DE MESQUITA FILHO

Prof.(a) HARLEN COSTA BATAGELO
UNIVERSIDADE FEDERAL DO ABC

Documento assinado digitalmente
gov.br JOAO PAULO GOIS
Data: 30/11/2022 17:59:42-0300
Verifique em https://verificador.iti.br

Prof.(a) JOAO PAULO GOIS
UNIVERSIDADE FEDERAL DO ABC - Presidente

* Por ausência do membro titular, foi substituído pelo membro suplente descrito acima: nome completo, instituição e assinatura

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001

RESUMO

As áreas de processamento de imagens, computação gráfica e visão computacional recentemente viram avanços significativos em diversas aplicações, muitos dos quais se devem a técnicas baseadas em Redes Adversárias Generativas (*Generative Adversarial Networks* - GANs). Dentre essas aplicações, podemos ressaltar a super-resolução, transformação de domínio, transferência de estilo, restauração de fotos, e síntese de imagens, dentre outras. Neste trabalho estudamos o uso de GANs em aplicações de transformações de contornos em imagens e síntese de imagens de faces.

A primeira aplicação envolve transformar esboços de carros em imagens realistas, e esboços de personagens de desenho em uma versão colorida e texturizada. Analisamos e discutimos o uso das arquiteturas Pix2Pix e CycleGAN nessa tarefa, experimentando com diferentes configurações de parâmetros como melhorar a imagem sintetizada.

Na aplicação de síntese, inspirados por trabalhos de manipulação de imagens, exploramos como gerar imagens sintéticas de faces humanas utilizando GANs condicionais como *autoencoders*, de forma que gerem vetores latentes que permitam a manipulação das imagens diretamente no espaço latente das características. Testamos diversas combinações de geradores, discriminadores, funções de *loss* e técnicas de treinamento para entender quais elementos contribuem na melhoria da imagem sintetizada.

Palavras-chave:Aprendizado Profundo, Redes Adversárias Generativas, Síntese de Imagens, *Deep Image Rendering*, GANs, Processamento de Imagens

ABSTRACT

Significant advances in image processing, computer vision, and computer graphics applications were recently achieved, much due to techniques based on Generative Adversarial Networks (GANs). Among those applications we can cite super resolution, image to image translation, style transfer, picture restoration and image synthesis. In this work we study the use of GANs in contour-to-image translation and face image synthesis applications.

The first task involves transforming car sketches into realistic images, and cartoon sketches into a colorized and texturized version. We analyze and discuss the use of both Pix2Pix and CycleGAN architectures in this task, performing experiments with different parameters and configurations to enhance the quality of the synthetic image.

In the synthesis application, inspired by image manipulation works, we explore how to generate synthetic images of human faces using conditional GANs as autoencoders, so that they can generate latent vectors that allow image manipulation directly on the feature latent space. We tested many combinations of different generators, discriminators, loss functions and training techniques to understand which elements contribute to generate better images.

Keywords:Deep Learning, Generative Adversarial Networks, Deep Image Synthesis, Deep Image Rendering, GANs, Image Processing

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

1 INTRODUÇÃO

Redes Adversárias Generativas (*Generative Adversarial Networks* - GANs) são técnicas baseadas no chamado "treinamento adversário" que consiste em dois componentes, um gerador que é treinado para criar um conjunto de dados sintético que seja o mais próximo possível de um conjunto de dados real, e um discriminador cuja função é distinguir (ou discriminar) dados reais de dados falsos [20, 19]. Normalmente esses componentes são redes neurais, e seu treinamento é feito como uma competição entre os dois módulos, sendo o objetivo do gerador sintetizar dados tão realistas que consigam enganar o discriminador, ao passo que o objetivo do discriminador é não se deixar enganar.

Essa é uma técnica de aprendizagem profunda (*deep learning*), área do aprendizado de máquina que estuda o uso de modelos multicamadas treináveis em diversas tarefas [19]. Redes de aprendizado profundo realizam o que se chama de aprendizado de representação [19], ou seja, o modelo consegue aprender a representação dos atributos (ou a representação da distribuição dos atributos) que compõem os dados de treino. Uma característica desse tipo de modelo que torna sua escolha interessante é que, apesar do treinamento ser computacionalmente custoso, as redes treinadas mantém a representação dos dados dentro da sua estrutura, e por isso conseguem fazer inferência mais rapidamente, o que pode ser útil para aplicações interativas.

Um tipo de rede comumente empregada em tarefas que envolvem processamento de imagens são as Redes Neurais Convolucionais (*Convolutional Neural Networks* – CNNs), que utilizam camadas com operadores de convolução para reconhecer as características das imagens através de mapas de atributos (*feature maps*) [37, 19]. Os atributos reconhecidos

pelas CNNs podem ser mais genéricos, como diferenciar um humano de um gato, ou podem ser atributos mais específicos, como as características da face de uma pessoa que a tornam diferente das demais.

Como a rede tem dentro de si uma representação dos atributos das imagens usadas no treinamento, diversas aplicações podem ser derivadas, tais como a transformação de imagens [28, 80], a composição de imagens a partir de colagens desses atributos [28, 72], detecção de objetos em tempo real [68, 23, 52, 53, 10], reconhecimento facial [65], transferência de estilo [80, 33, 34, 17], super resolução [29, 38, 60], modelagem baseada em esboço [40], restauração de fotografias [73].

Recentemente as GANs baseadas em CNNs tem recebido maior destaque por alcançarem bons resultados em diversas dessas tarefas [75, 21], com métodos que frequentemente superavam o estado da arte [49, 28, 72].

Essas aplicações eram restritas a computadores com alto poder de processamento, mas com a recente evolução das unidades de processamento gráfico (GPUs) e a redução de custo em relação ao poder de processamento desses componentes, as aplicações de inteligência artificial e aprendizado de máquina puderam chegar também nos computadores pessoais e dispositivos móveis. Paralelamente a isso, uma maior quantidade de dados passou a ser gerada e armazenada globalmente, efeito da disseminação do uso de dispositivos com internet embarcada, como *smartphones*, a digitalização dos serviços através de aplicativos, e o uso de redes sociais. A união dessa maior disponibilidade de melhores dados com o maior poder computacional propicia a aplicação de técnicas de aprendizado de máquina para as mais diversas tarefas.

Como exemplo, aplicações baseadas em GANs podem ser usadas por artistas que queiram ter uma primeira visão de como objetos esboçados em duas dimensões ficariam se transformados em modelos tridimensionais [51], ou então usar uma aplicação interativa que construa cenas e paisagens em tempo real a partir de uma representação mais simples como um mapa semântico [72, 46, 45]. Um artista pode fazer uma ilustração base com seu

traçado e aplicar um estilo diferente na sua imagem, como aplicar a uma foto o estilo de Monet ou de Van Gogh [80] (Fig. 1.1).

Figura 1.1: Exemplos de transformações artísticas em uma foto. Fonte [80]. Uma foto de uma paisagem, à esquerda, tem seu estilo transformado pela CycleGAN para se assemelhar ao estilo de Monet, Van Gogh, Cezanne e Ukiyo-e, respectivamente.

Outros tipos de transformações podem ser feitas, como uma interpolação linear a nível dos atributos, e não dos pixels, em que seria possível criar híbridos de gatos e cachorros [39] ou mesclar a *Tower Bridge* com a Torre Eiffel [79], como na Figura 1.2.

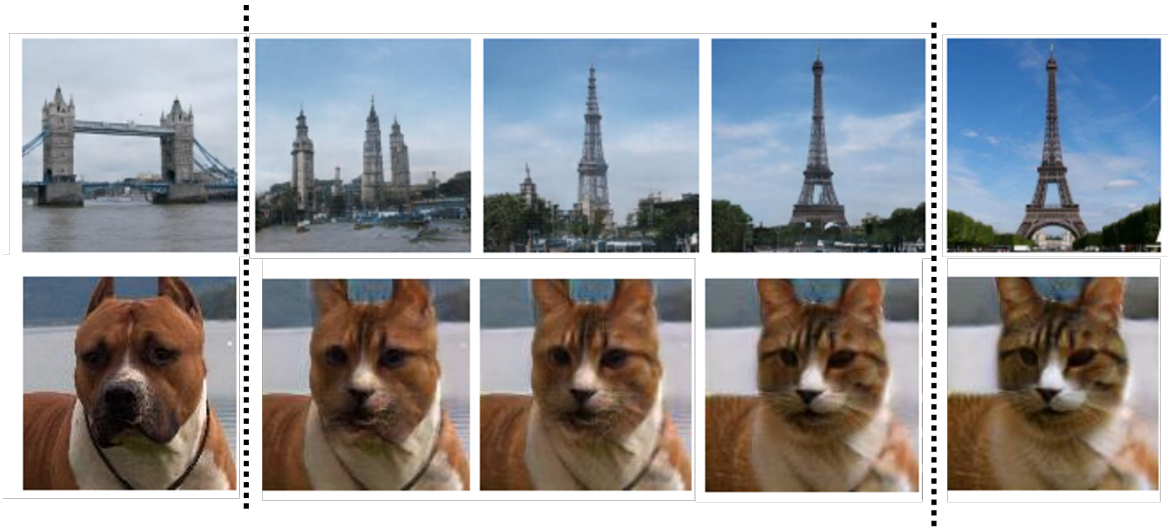

Figura 1.2: Exemplos de transformações de imagens naturais. Na linha superior, o trabalho de Zhu *et al.* [79] interpola de maneira contínua e natural a *Tower Bridge* à esquerda com a Torre Eiffel à direita. Na linha inferior, Lira *et al.* [39] transformam a imagem de um cachorro gradualmente até que ele se torne um gato completamente sintético à direita.

Isso é feito através da manipulação das características das imagens diretamente no espaço latente aprendido pela GAN, por meio de operações vetoriais no vetor latente que codifica a informação da imagem criada. Um desafio é descobrir para uma determinada imagem, qual é o vetor latente que a representa dentro do espaço de uma GAN específica, e geralmente as soluções para esse problema envolvem a criação de novas redes [79, 54, 2, 3].

1.1 Motivação

Ferramentas baseadas em IA, como o Nvidia Canvas[1] que é baseado na GauGAN [45], ou o Neural Filters[2] presente no Adobe Photoshop, podem auxiliar artistas nas suas criações.

Uma aplicação que transforma esboços em imagens colorizadas, texturizadas e com maior quantidade de detalhes, pode auxiliar na etapa de ideação de um produto ou de *storyboarding* de uma animação, por exemplo. Se uma GAN puder ser treinada para fazer essa transformação, ela pode ser incluída em um software que permita aos artistas manipularem suas criações e ver uma versão melhorada de forma interativa.

Outra aplicação interessante pode ser a manipulação dos atributos já existentes em uma imagem. Um exemplo pode ser alterar as características faceais de uma pessoa, como idade, pose, gênero ou até presença de barba ou óculos, em uma foto ou vídeo. Softwares com essa capacidade já são usados em produções audiovisuais, e também podem auxiliar aos criadores de conteúdo digital em suas criações.

1.2 Objetivo

Neste trabalho pretendemos estudar o uso das GANs em duas aplicações distintas: a transformação de contornos em imagens, e a síntese de imagens de faces.

Na primeira aplicação estudamos o uso das arquiteturas Pix2Pix [28] e CycleGAN [80] na tarefa de transformar esboços de carros em imagens realistas, e a transformação de esboços de personagens humanoides em um desenho colorido e com textura. Uma aplicação desse tipo pode permitir uma prototipagem mais rápida, com o artista esboçando enquanto assiste,

1 Disponível em: https://www.nvidia.com/pt-br/studio/canvas/
2 Disponível em: https://www.adobe.com/products/photoshop/neural-filter.html

em instantes, sua criação tomar um aspecto mais próximo do produto final. Avaliamos, portanto, a qualidade das imagens geradas e o tempo de inferência, para garantir que o processo possa ocorrer de forma interativa.

Na aplicação de síntese de imagens estudamos a aplicação de GANs condicionais como *autoencoders* treinados de forma adversária para gerar imagens de faces. Uma vantagem desse método é que o gerador que cria as imagens sintéticas possivelmente também tem a capacidade de gerar o vetor latente que corresponde a uma imagem específica, sem a necessidade de uma rede adicional para isso. Também comparamos diversas combinações distintas dos elementos que compõem essas GANs para entender o impacto de cada um deles na síntese da imagem final.

1.3 Estrutura

O Capítulo 2 terá uma introdução ao referencial teórico, em que serão abordados conceitos de aprendizado de máquina, redes neurais, e outros necessários para o entendimento deste trabalho, seguido por uma revisão de trabalhos relacionados no Capítulo 3. Nosso estudo de uma aplicação de transformação de esboços em imagens complexas será aprofundada no Capítulo 4, e o Capítulo 5 irá discorrer a respeito de nosso estudo sobre como um *autoencoder* baseado em GANs pode gerar um vetor latente do domínio do gerador que permita controlar a síntese da imagem final. Finalmente no Capítulo 6 será feita uma discussão dos resultados obtidos, uma conclusão e possíveis trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

A área de síntese de imagens por aprendizado profundo é relativamente recente, mas foi construída a partir de um longo desenvolvimento do ramo da Inteligência Artificial. A Figura 2.1 ilustra a relação entre a inteligência artificial, o aprendizado de máquina, o aprendizado de representação e o aprendizado profundo, que será detalhado mais adiante.

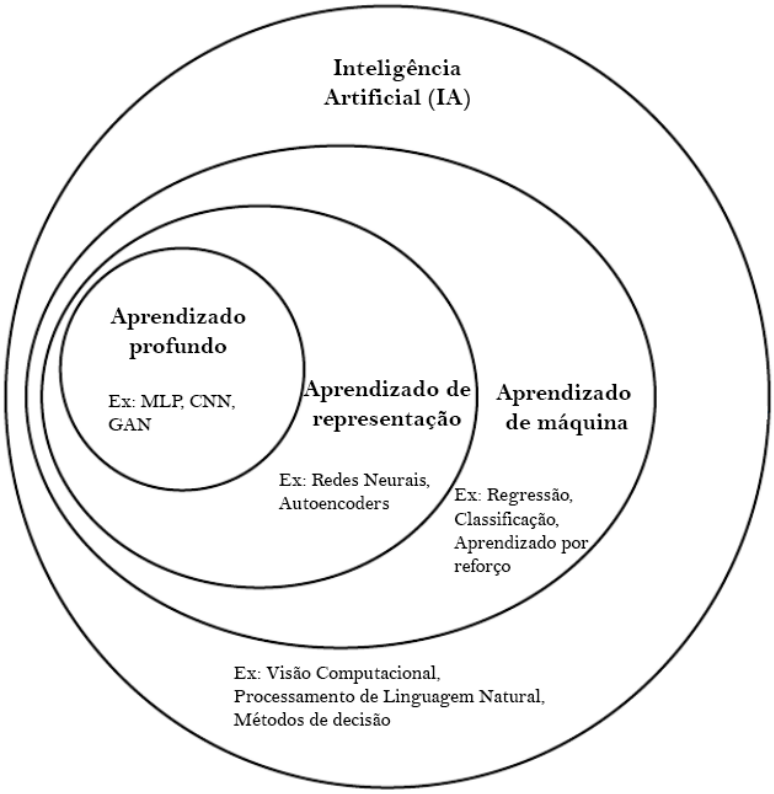

Figura 2.1: Diagrama de Venn da Inteligência Artificial. Adaptado de: [19]. O ramo da inteligência artificial (IA) engloba diversas sub-áreas como o aprendizado de máquina e outras não representadas na ilustração. Da mesma forma, aprendizado de representação é uma sub-área do aprendizado de máquina e o aprendizado profundo é uma sub-área do aprendizado de representação.

Este capítulo inicia com uma breve abordagem do que é aprendizado de máquina, e os tipos mais comuns de aprendizado e de tarefas que podem usar essa técnica. Na sequência será apresentado como as redes neurais passaram de uma tentativa de modelar o funcionamento do cérebro humano para um dos mais importantes modelos de estimação estatística da atualidade.

Essas redes foram então adaptadas para trabalhar com diferentes tipos de dados, como imagens, e novas técnicas surgiram disso. Dessas, a que aparece mais recorrentemente na área de síntese de imagens são as Redes Adversárias Generativas (*Generative Adversarial Networks* - GANs) que usam um treinamento baseado em competição para alcançar resultados verossímeis. As GANs serão abordadas mais profundamente na Seção 2.4.

2.1 Aprendizado de Máquina

O Aprendizado de Máquina é uma sub-área da Inteligência Artificial que usa algoritmos que representam modelos adaptativos que conseguem ser otimizados a partir de dados em um processo de "treinamento" ou "aprendizagem" [9]. Especificamente, dada uma tarefa e uma métrica de desempenho, um algoritmo de aprendizado de máquina usa a experiência passível de ser extraída dos dados nos dados para se aperfeiçoar na execução dessa tarefa [19].

Chamamos esses algoritmos treinados de modelos, e ao se modificar o algoritmo, a base de dados ou a métrica de performance, é possível criar diversos tipos de modelos diferentes que são especializados em alguma tarefa específica, como prever valores futuros de uma série temporal, reconhecer atributos em imagens, analisar a linguagem natural e responder de acordo, dentre tantas outras [19].

Uma dessas tarefas é o aprendizado de representação, que consiste no modelo extrair a representação latente dos dados do conjunto de treino [19]. Em outras palavras, os parâme-

tros treinados do algoritmo modelam a representação ou a distribuição dos atributos desses dados. Um exemplo disso é como modelos de detecção de faces aprendem naturalmente a extrair características específicas, como olhos, narizes e bocas, das imagens que participaram do seu treinamento.

2.1.1 Tipos de Tarefas

O objetivo de empregar um modelo de aprendizado de máquina é resolver uma tarefa que seria complexa, demorada ou até impossível para um ser humano resolver em tempo hábil, como aprender o comportamento de milhares de clientes de uma loja online, ou calcular o resultado de um sistema que emprega centenas de variáveis.

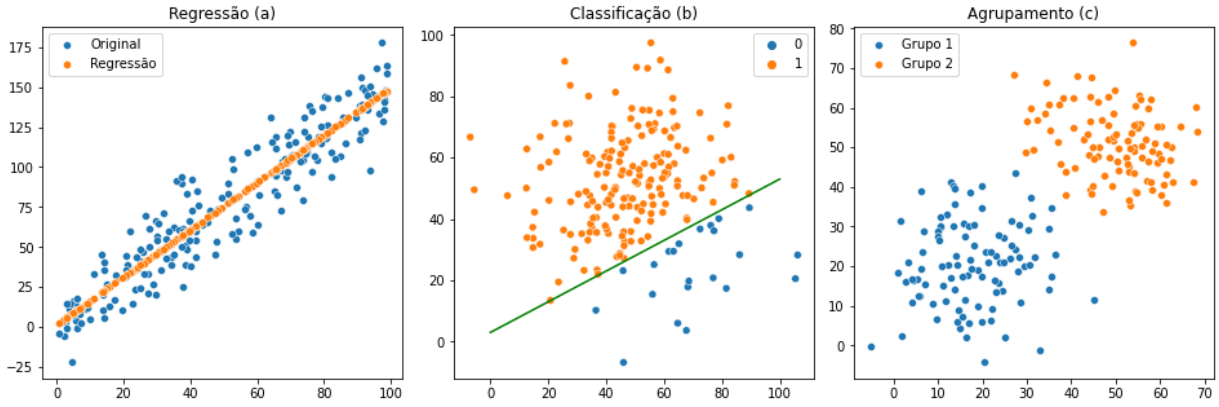

Figura 2.2: Principais tipos de tarefas de aprendizado de máquina. À esquerda (a) um modelo de regressão em que os pontos azuis representam observações reais e os pontos laranjas correspondem às saídas do modelo. Ao centro (b) um classificador binário representado pela linha verde divide o espaço em dois, correspondendo às classes 0 e 1. À direita (c) um modelo de agrupamento separou os exemplos em dois grupos diferentes.

Dentre as tarefas que podem ser executadas por sistemas baseados em aprendizado de máquina, pode se ressaltar três principais, representadas na Figura 2.2: a regressão, a classificação e o agrupamento. Essas três tarefas serão detalhadas adiante nesta seção.

Além das citadas, existem outros tipos de tarefas, como processamento de linguagem natural (NLP), detecção de anomalias, sistemas de recomendação, transcrição, ou síntese de dados [19].

Regressão

Os Métodos de Regressão Linear encontram uma função $f : \mathbb{R}^n \to \mathbb{R}$ a partir de um conjunto de dados de treino $X \subsetneq \mathbb{R}^n \times Y \subsetneq \mathbb{R}$ com o objetivo de, dado um $x_i \in \mathbb{R}^n$, a função retorna $f(x_i)$ como uma aproximação, ou *previsão*, de Y [19, 9]. Essa capacidade de prever informações desconhecidas chamamos de capacidade de generalização do modelo.

Classificação

A tarefa de classificação se assemelha à regressão, mas em vez de modelar uma função f que produza uma saída contínua, a classificação modela uma função $h : \mathbb{R}^n \to C$ em que o conjunto $C = \{c_1, \ldots, c_k\}$ corresponde a todas as k possíveis classes que os exemplos podem assumir [19, 9].

Similarmente à regressão, o modelo é treinado utilizando uma base de treino em que cada exemplo tem sua classificação conhecida, e o objetivo do modelo é classificar corretamente os exemplos da base de treino e generalizar para exemplos não conhecidos.

Quando um algoritmo de classificação deve decidir entre apenas duas classes diferentes, chamamos essa tarefa de classificação binária, como representado na Figura 2.2(b). Tarefas com mais de duas classes são chamadas de classificação multiclasse.

Agrupamento

O agrupamento modela uma função $g : \mathbb{R}^n \rightarrow P$ em que $P = \{p_1, \ldots\ p_k\}$ representa o conjunto de grupos a qual cada elemento pode ser associado [9].

Normalmente os algoritmos de agrupamento particionam o espaço dos exemplos de forma a manter dentro do mesmo grupo os elementos mais similares entre si. Essa similaridade (ou dissimilaridade) pode ser calculada através de medidas de distância entre os vetores que representam cada objeto. O objetivo do agrupamento é criar partições que maximizem a coesão entre os elementos de um mesmo grupo ao mesmo tempo que maximizam a separação entre grupos diferentes, como ilustrado pela Figura 2.2(c).

2.1.2 Tipos de Aprendizado

Existem diversas maneiras de se realizar o treinamento de um algoritmo de aprendizado de máquina. Dentre essas, pode-se ressaltar o aprendizado supervisionado, o aprendizado não supervisionado e o aprendizado por reforço.

Aprendizado Supervisionado

No aprendizado supervisionado, sabemos o resultado pretendido para cada exemplo da base de dados de treino. Esse resultado pode ser um objetivo (*target* ou *label*) de acordo com a tarefa a ser executada, como ilustrado na Figura 2.3 (esquerda). Os objetivos são definidos anteriormente ao passo de treinamento por um supervisor, o que dá o nome ao método [9]. Classificação e Regressão são exemplos de tarefas em que normalmente se aplica o aprendizado supervisionado.

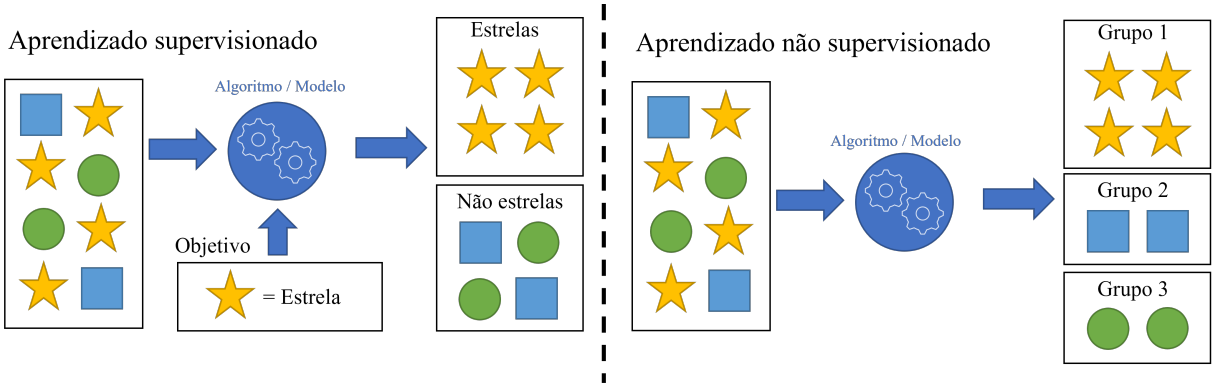

Figura 2.3: Comparação entre aprendizado supervisionado e não supervisionado. À esquerda, um algoritmo baseado em aprendizado supervisionado aprende a classificar "estrelas" de "não estrelas" com base no objetivo pré-definido por um supervisor. À direita um algoritmo baseado em aprendizado não supervisionado agrupa exemplos da base de dados de acordo com a similaridade de seus atributos.

No caso de um algoritmo de classificação, por exemplo, cada observação X_i deve ter um rótulo (*label*) c_i que corresponde à sua classificação pretendida. Em cada iteração do processo de treinamento do algoritmo, um ou mais exemplos são introduzidos como entrada do algoritmo que deve se ajustar para que sua saída seja a classe definida no objetivo para aquele exemplo.

Em uma regressão, para cada exemplo X_i uma saída y_i, que corresponde ao valor que a função modelada deve retornar ao se aplicar esse exemplo na entrada. Em outras palavras, em um modelo ideal tem-se $y_i = f(X_i)$.

Uma vantagem da abordagem supervisionada é que, por saber exatamente o objetivo que se quer alcançar, a convergência do modelo costuma ser mais rápida e normalmente para a direção de um mínimo local. Com o conhecimento do objetivo também é possível calcular funções de erro que permitem uma medida mais clara do erro que o modelo tem de representar os dados a cada iteração.

Uma característica importante desse tipo de aprendizado é que ele exige uma base de dados que contenha os objetivos, ou que eles sejam preparados previamente, o que pode ser custoso para tipos de dados não estruturados como imagens ou áudios. Além disso,

caso exista uma função matemática ou um algoritmo que possa calcular o objetivo de cada exemplo da base, a abordagem de aprendizado torna-se desnecessária.

Aprendizado Não Supervisionado

O aprendizado não supervisionado, por outro lado, não possui um objetivo claro definido para cada exemplo. Por conta disso o algoritmo utiliza padrões inerentes aos próprios dados para cumprir a tarefa [9], como no caso do agrupamento, ilustrado na Figura 2.3 (direita).

No agrupamento, o algoritmo precisa definir quais elementos pertencem a cada um dos grupos, e portanto deve calcular uma dissimilaridade entre objetos, ou uma divergência entre distribuições de dados, para decidir em qual grupo colocar cada objeto.

Os modelos de agrupamento podem criar grupos diferentes a depender do seu funcionamento ou da sua inicialização. Por conta disso, para dois modelos diferentes treinados nos mesmos dados podem trazer resultados bem diferentes, e normalmente é necessária uma interpretação humana dos resultados para dizer qual deles é melhor para uma tarefa.

Ainda assim, modelos não supervisionados são úteis, principalmente quando não há bases de dados estruturadas e com os objetivos definidos.

Aprendizado por Reforço

O aprendizado por reforço funciona através de um sistema de recompensas. O modelo é criado a partir de regras básicas e não é treinado com um conjunto de dados fixo, mas interage com o ambiente sendo reforçado de acordo com as recompensas ou punições que recebe de acordo com as interações que executa [19].

O problema que modelos de reforço tentam resolver é quais ações tomar em cada situação de forma a maximizar a recompensa obtida, através de um processo de tentativa e erro [9]. Normalmente esses problemas envolvem um agente sujeito às variáveis do ambiente, e cujas ações também impactam esse ambiente, e as respostas às sequências de ações podem não vir imediatamente.

2.2 Aprendizado Profundo

O aprendizado profundo é uma sub-área do aprendizado de máquina (vide Fig. 2.1) que consiste em aprender representações complexas que são expressadas em termos de outras representações mais simples [19]. Normalmente esses modelos são chamados de *redes*, justamente pela forma como os elementos simples são interligados formando estruturas mais complexas, como as Redes Neurais Artificiais, em inglês *Artificial Neural Networks* (ANNs), que formam uma estrutura inspirada na forma como os neurônios se interligam em nosso cérebro [14].

As redes neurais modernas são, em última instância, estimadores de funções projetados para realizar generalização estatística a partir dos dados de treino [19].

Baseadas nas ANNs, outros métodos foram criados, especializados em tipos específicos de tarefas, como por exemplo as Redes Neurais Convolucionais (*Convolutional Neural Networks* - CNNs) utilizadas para tarefas que exigem processamento de imagens ou sons, ou as Redes Neurais Recorrentes (*Recurrent Neural Networks* - RNNs), utilizadas para modelar séries temporais, ou modelos de sistemas em que um estado anterior tem efeito no estado atual.

2.2.1 Redes Neurais Artificiais - ANNs

O cérebro humano é composto de bilhões de neurônios conectados entre si, e que se comunicam transmitindo sinais através de sinapses. Em 1943, McCulloch e Pitts [42] modelaram um neurônio matemático, que serviu de base para Rosenblatt criar o Perceptron [57], um modelo neurônio capaz de ser treinado para realizar uma classificação binária, ou seja, dada uma entrada o neurônio aprenderia a classificá-la corretamente como 0 ou 1 [14].

A equação do Perceptron é expressa por:

$$y = step\left(b + \sum_{i=0}^{n} w_i x_i\right), \tag{2.1}$$

em que uma entrada vetorial $X = [x_1, x_2, \ldots, x_n]$ tem cada i-ésimo elemento ponderado por um peso w_i, correspondente à sua importância na ativação do neurônio, e b é um termo que insere um viés no neurônio, similar ao intercepto de uma função linear. A função degrau $step(x)$, que retorna 1 quando $x \geq 0$ ou 0 caso contrário, é a função de ativação do Perceptron. Em outras palavras, o que define se o neurônio será ativado ou não é o resultado da média ponderada das suas entradas aplicado à função degrau.

Por essa característica, o Perceptron só pode aprender a modelar funções que são linearmente separáveis, já que ele efetivamente representa um hiperplano que separa o espaço em duas regiões que correspondem à classificação 0 e à classificação 1 [14].

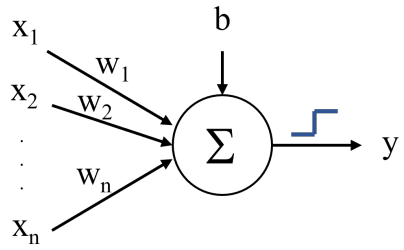

Figura 2.4: Modelo de um Perceptron.

O aprendizado do Perceptron é feito por meio do ajuste dos pesos do neurônio. Dado um conjunto de m pares (X_k, y_k) com $k = \{1, \ldots, m\}$ chamado *conjunto de treino*, em que sabemos a classificação pretendida y_k para cada entrada X_k, pode-se calcular o erro de classificação do perceptron $e = y_k^{perceptron} - y_k^{real}$ e então ajustar os pesos através da iteração:

$$w_{i+1} \leftarrow w_i + \alpha x_i e, \tag{2.2}$$

em que α é uma taxa de aprendizado definida pelo usuário. Os pesos normalmente são iniciados de forma aleatória, com valores amostrados de uma distribuição uniforme no intervalo $[-1, 1]$, e para cada novo exemplo apresentado ao perceptron ele calcula o erro da sua predição e reajusta seus pesos, idealmente convergindo à classificação correta [14, 19]. A Figura 2.4 ilustra esse modelo de neurônio.

Multilayer Perceptron - MLP

A maior limitação do Perceptron original é que ele só consegue separar o espaço em duas regiões distintas delimitadas por uma reta, e por isso não resolve problemas relativamente simples, mas não lineares, como modelar a função lógica OU exclusivo (XOR), que envolve separar o espaço em mais do que duas regiões distintas.

Assim, após anos de desenvolvimento, uma evolução do Perceptron foi criada, em que um ou mais neurônios eram dispostos em camadas, de forma que os neurônios da mesma camada não se interligam, mas podem se comunicar com neurônios da camada anterior e da próxima, formando enfim o que chamamos de rede neural. Esse modelo é chamado Perceptron Multicamadas (*Multilayer* Perceptron - MLP) [58].

A Figura 2.5 apresenta um MLP com quatro camadas, sendo uma entrada, uma saída e duas camadas ocultas. Cada círculo representa um nó, ou neurônio, e as saídas dos neurônios de uma camada podem se ligar às entradas dos neurônios da camada seguinte. Chamamos de densa ou *fully-connected* a rede ou seção da rede em que todos os neurônios

de uma camada estão ligados a todos os neurônios da camada seguinte. A quantidade de neurônios em cada camada e a quantidade de camadas podem variar, e esses parâmetros são definidos pelo projetista da rede.

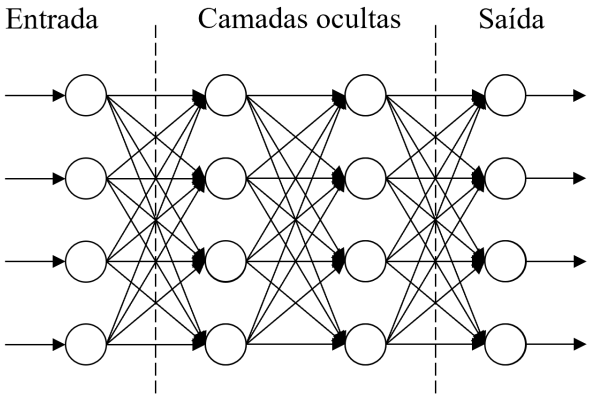

Figura 2.5: Esquema de um MLP genérico.

MLPs são um tipo de modelo de aprendizado profundo [19] por serem redes complexas, mas de elementos simples, e a profundidade da rede é determinada pela quantidade de camadas que ela tem. Frequentemente, quando se fala de redes neurais artificiais, na realidade está se falando de variações de modelos MLP.

Considerando-se uma camada de entrada com n elementos, de forma que x_i com $i = \{1, \ldots, n\}$ corresponda ao i-ésimo elemento da camada, e considerando uma camada oculta com m elementos de forma que y_j com $j = \{1, \ldots, m\}$ seja a saída do j-ésimo neurônio, podemos modelar cada neurônio j individualmente pela expressão

$$y_j = \sigma\left(\sum_{i=0}^{n} w_{ij}x_i - b_j\right), \tag{2.3}$$

onde o índice j se refere ao neurônio modelado, que recebe cada um dos elementos x_i da camada anterior como entrada. O peso w_{ij} se refere à importância da entrada x_i para a ativação do neurônio j, e b_j é um elemento de viés desse nó. Os pesos e viéses são geralmente

iniciados com valores aleatórios [14]. Por fim, σ é a *função de ativação do neurônio*, e no caso do MLP a função utilizada é do tipo sigmóide:

$$\sigma(x) = \frac{1}{1 + e^{-x}}, \tag{2.4}$$

que pode ser considerada uma aproximação contínua e diferenciável da função degrau [14].

Em modelos modernos outros tipos de funções de ativação costumam ser usados, mais comumente a ReLU (*Rectified Linear Unit*) [18], a tangente hiperbólica (tanh), funções lineares, ou mesmo a função degrau, com seus respectivos gráficos representados na Figura 2.6.

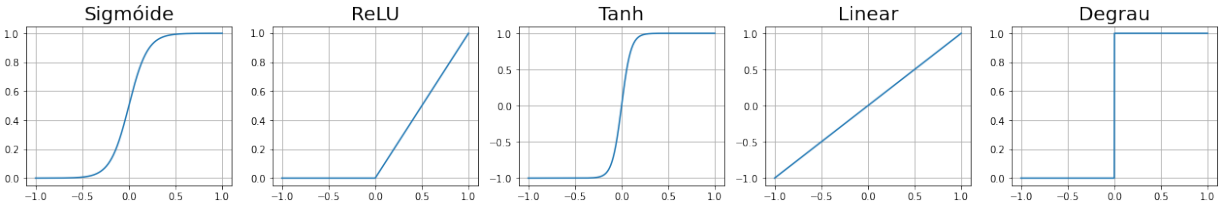

Figura 2.6: Exemplos de diferentes funções de ativação.

Retropropagação

O mecanismo de treinamento de uma MLP é chamado de retropropagação (*backpropagation*), e cada iteração do treinamento da rede é composto de duas etapas. Primeiramente se executa a etapa de *feedforward*, cujo objetivo é apresentar uma entrada X_i para a rede e obter a saída y_i^{rede}. Na sequência ocorre a retropropagação, em que se calcula um gradiente de erro para cada nó a partir da saída da rede em relação aos seus pesos, e então os pesos são atualizados com algum método de otimização normalmente baseado em gradiente descendente, como o *Stochastic Gradient Descent* (SGD) [58] ou o ADAM [35].

Para ilustrar esse mecanismo, a Figura 2.7 mostra a superfície de uma função de erro de um nó que tem dois pesos, w_1 e w_2. O eixo vertical é a magnitude do erro e os eixos horizontais representam os valores dos pesos daquele nó. As setas representam o ajuste

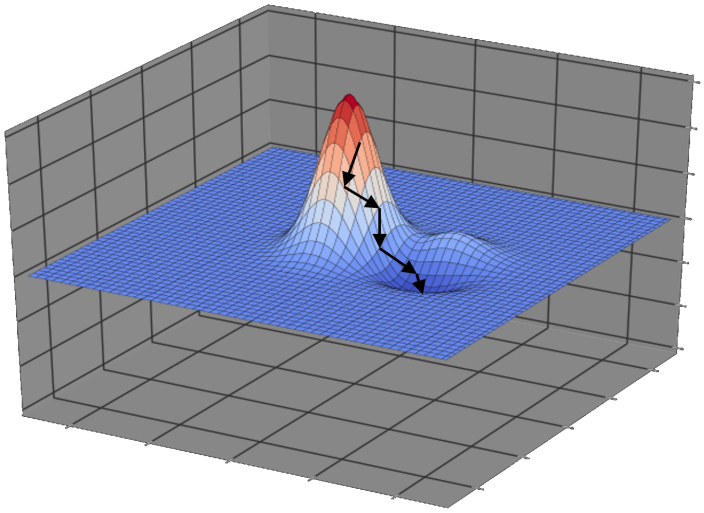

Figura 2.7: Mecanismo de otimização por gradiente descendente. A partir de uma posição inicial aleatória, o método busca convergir a um mínimo local no gráfico da função de *loss*.

dos pesos a cada iteração, que tem o efeito de reduzir o erro na direção mais íngreme da descida, ou seja, com maior gradiente, até que se encontre um ponto de mínimo na função.

Em modelos modernos, esse erro é normalmente chamado de *loss* e pode ser calculado de diversas formas, como a *Binary Cross-Entropy* ou a acurácia para tarefas de classificação binária, o erro médio absoluto (MAE) ou o erro médio quadrático (MSE) para tarefas de regressão, ou inclusive com *losses* mais complexas em casos específicos.

Um problema dessa abordagem é que, a se depender da função de *loss* e dos parâmetros utilizados, o treinamento pode não convergir ou convergir para um mínimo local de pouca relevância.

2.3 Redes Neurais Convolucionais - CNNs

As Redes Neurais Convolucionais (CNN) ou Redes Convolucionais [37] utilizam a operação de convolução em ao menos uma de suas camadas [19].

Esse tipo de rede é muito efetiva em aplicações em que os dados são dispostos de forma que a relação de vizinhança entre os elementos é relevante, como no caso de imagens que são representadas por matrizes bidimensionais de pixels, ou no caso de séries temporais ou dados de áudio, que são sequências unidimensionais de dados amostrados em intervalos de tempo regulares.

Por serem eficazes em extrair atributos das imagens, as CNNs são amplamente empregadas em aplicações de detecção e reconhecimento de objetos, reconhecimento facial, segmentação semântica, processamento e manipulação de imagens, dentre outras [13]. Filtros inteligentes de câmeras também usam as CNNs para modificar a face da pessoa, sistemas de navegação de robôs e carros autônomos utilizam CNNs para detectar obstáculos e elementos do trajeto, e sistemas mais modernos de visão computacional também as empregam.

Assim como as MLPs, as CNNs também são treinadas através de um mecanismo de retropropagação, normalmente baseado em gradiente descendente estocástico, como o SGD [58] ou o ADAM [35].

2.3.1 *Convolução*

A convolução é uma operação matemática entre duas funções. Sejam $x(t)$ e $w(a)$ funções reais, respectivamente chamadas de *entrada* e *kernel*. Considerando que $x(t)$ é uma função contínua dependente do tempo t e que $w(a)$ é uma função de peso que decai com maiores valores de a, podemos calcular uma média ponderada de x de forma que os valores mais recentes tenham maior relevância que os valores mais antigos através da expressão

$$s(t) = (x * w)(t) = \int x(a)w(t-a)da, \tag{2.5}$$

que é a operação de convolução entre x e w, representada por $(x * w)$. A saída $s(t)$ é normalmente chamada de *feature map*, ou mapa de atributos.

A operação pode ser definida também para funções discretas, que é o caso mais comum utilizado nos algoritmos de CNNs unidimensionais. A convolução se torna

$$s(t) = (x * w)(t) = \sum_{a} x(a)w(t-a). \tag{2.6}$$

No caso de uma convolução 2D, muito utilizada no processamento de imagens, e considerando I como a imagem de entrada e K como o *kernel* bidimensional, a convolução pode ser representada em um espaço discreto, conforme a expressão:

$$S(i,j) = (I * K)(i,j) = \sum_{m}\sum_{n} I(m,n)K(i-m, j-n). \tag{2.7}$$

Normalmente os *kernels* são muito menores do que as imagens, e através da operação de convolução, um único *kernel* é utilizado para processar toda uma imagem, e portanto dizemos que os parâmetros desse *kernel* são compartilhados com toda a imagem [19]. Por conta dessa representação reduzida em *kernels*, cada camada da rede convolucional deve aprender uma quantidade menor de parâmetros do que uma camada de MLP que processe a mesma imagem.

As ANNs, por outro lado, utilizam matrizes de pesos para representar as relações entre os elementos da camada atual com as saídas da camada anterior. Como cada conexão é independente das outras, cada peso representa especificamente uma única conexão, o que faz com que a rede precise aprender uma grande quantidade de parâmetros. Comparativamente, por usarem menos parâmetros, as CNNs costumam ser mais eficientes em termos de memória e são treinadas mais rapidamente do que as ANNs [19].

2.3.2 Convolução em imagens

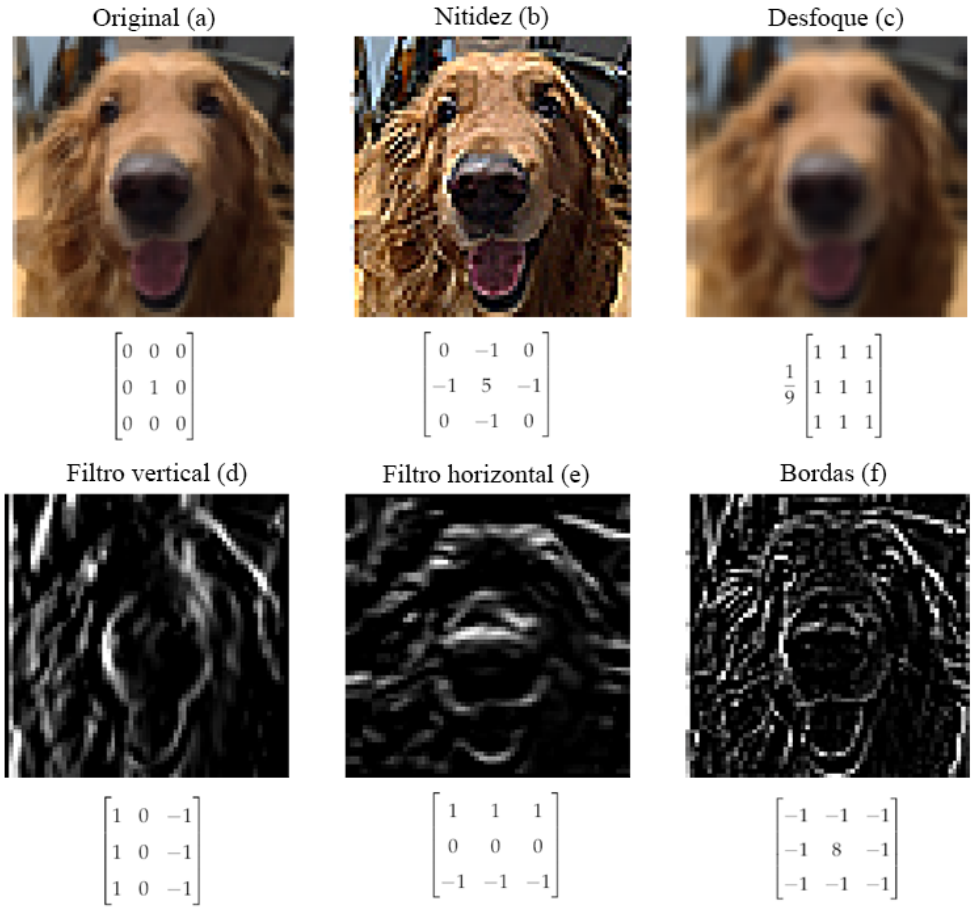

Figura 2.8: Aplicações de diferentes *kernels* em uma imagem. A imagem original é representada em (a), juntamente com o *kernel* identidade, que não faz modificações. Em (b) a imagem passou por um *kernel* que aumenta a nitidez reforçando o pixel central e aumentando a diferença dele em relação à sua vizinhança. Em (c) foi usado um *kernel* de desfoque, que substitui um pixel pela média dele com seus vizinhos. A imagem em (d) é o resultado de aplicar na imagem em escala de cinza um *kernel* que reforça suas linhas verticais, enquanto (e) reforça suas linhas horizontais. O *kernel* em (f) reforça os contornos (bordas) da imagem.

A convolução é amplamente utilizada em processamento de imagens. A depender do *kernel* utilizado, é possível manipular a nitidez e o desfoque, estilizar a imagem, ou inclusive detectar as arestas dos objetos presentes. A Figura 2.8 exemplifica a aplicação de alguns diferentes tipos de *kernels* em uma imagem base.

O *kernel* é aplicado em cada pixel da imagem, e o resultado da convolução entre o pixel e a região da imagem substitui o pixel central na imagem resultante, como ilustrado na Figura 2.9. O *kernel* não pode ser aplicado nos pixels de borda, pois parte dele não teria correspondência com nenhum pixel, e como consequência a imagem gerada é um pouco menor do que a imagem original.

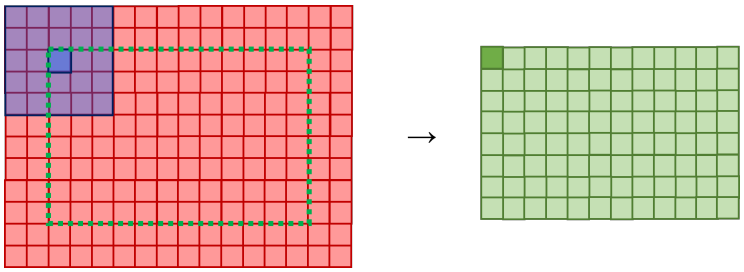

Figura 2.9: Convolução bidimensional em imagens: O *kernel* (azul) é convolucionado sobre a imagem (vermelha), ou seja, operação é feita nos pixels da imagem que correspondem à posição em que o *kernel* está. O resultado da operação de convolução determina o valor do pixel central dessa região, e o conjunto dos resultados da aplicação do *kernel* em cada pixel forma a imagem convolucionada (verde), que é menor do que a imagem original.

No exemplo da Figura 2.9 ocorre a convolução de uma imagem (vermelha) de 16 pixels de largura (w_{in}) por 12 pixels de altura (h_{in}) com um *kernel* de 5 pixels de largura (w_{kernel}) por 5 pixels de altura (h_{kernel}). A imagem resultante (verde) tem dimensões 12×8. O cálculo da largura (w_{out}) e da altura (h_{out}) da imagem resultante é feito pelas expressões:

$$w_{out} = w_{in} - w_{kernel} + 1, \tag{2.8}$$

$$h_{out} = h_{in} - h_{kernel} + 1. \tag{2.9}$$

Em CNNs, normalmente consideramos a convolução como uma camada da rede, e cada camada normalmente tem mais do que um *kernel*, produzindo a mesma quantidade de mapas de atributos.

Por exemplo, uma camada convolucional com 3 *kernels* que recebe como entrada uma imagem em escala de cinza de dimensões $(w_{in} \times h_{in} \times 1)$ irá produzir como saída uma matriz de dimensões $(w_{out} \times h_{out} \times 3)$. Essa saída não é mais considerada uma imagem, mas sim um conjunto de três mapas de atributos que correspondem aos atributos, também chamados de canais, reconhecidos pelos três *kernels* da camada. Encadeando os mapas de atributos através da estrutura de camadas da rede, ela pode capturar atributos mais gerais ou mais detalhados da imagem, e esse é o processo pelo qual as CNNs com diversas camadas aprendem a reconhecer objetos e características das imagens.

Em uma imagem com mais de um canal, como no caso de imagens RGB, a convolução acontece com um *kernel* em cada canal, como ilustrado na Fig. 2.10. O conjunto de canais é chamado de filtro, e cada filtro tem a mesma quantidade de *kernels* que a imagem ou mapa de atributos onde esse filtro será aplicado.

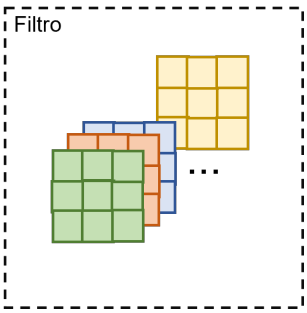

Figura 2.10: Exemplo de filtro com mais de um *kernel*. Um conjunto de n *kernels* forma um filtro que é aplicado em um mapa de atributos com n canais. A saída do filtro forma um único canal do mapa de atributos da sua camada.

Ao final da operação de convolução, um filtro de n *kernels*, aplicado a n camadas terá n mapas de atributos como saída. Uma operação de agregação como soma ou média é então aplicada nesses mapas de atributos para que o filtro tenha uma saída única. Cada camada de convolução pode ter um ou mais filtros.

Padding

Uma forma de contornar o problema convolução do *kernel* nas bordas é a partir da aplicação de *padding* na entrada da convolução, que nada mais é do que acrescentar pixels no perímetro da imagem. A Figura 2.11 ilustra três tipos de *padding*, o *zero padding* (a) que insere o valor 0 em todo o perímetro da imagem original, o *reflection padding* (b) que reflete a imagem a partir da borda original ou o *constant padding* (c) que preenche a nova borda com um valor constante, que pode inclusive ser 0.

Figura 2.11: Aplicação de *padding* em imagens. Em (a) a imagem original passou por *zero padding*, que inseriu uma borda com valor 0 (preto) ao redor dela. A imagem em (b) é resultado do *reflection padding*, que criou bordas a partir da reflexão da imagem interna. Em (c), assim como em (a) a imagem passou por um *padding* constante, mas com um valor diferente de 0, nesse caso.

A largura e altura da borda nova inserida pelo *padding* é definida pelo projetista da rede, mas normalmente é feito de forma a anular o efeito de redução da imagem convolucionada. O cálculo das novas dimensões é feito de acordo com as expressões:

$$w_{out} = w_{in} + 2 \cdot w_{padding} - w_{kernel} + 1, \tag{2.10}$$

$$h_{out} = h_{in} + 2 \cdot h_{padding} - h_{kernel} + 1. \tag{2.11}$$

Pooling

Cada camada da rede convolucional gera um ou mais mapas de atributos a partir da entrada que for apresentada. Esses mapas contém as características que a rede conseguiu obter por meio da convolução naquela camada, e o treinamento é responsável por ajustar os *kernels* de forma que os atributos presentes nos mapas sejam os mais importantes para as camadas seguintes.

Essas camadas, no entanto, podem também gerar ruído e informações que são pouco relevantes para o restante da rede, por exemplo como o *kernel* de bordas na Figura 2.8(f) detectou alguns contornos que não são relevantes para determinar a forma do cachorro.

A técnica de *pooling* substitui agrupamentos de pixels por um valor que os represente. A Figura 2.12 ilustra um `MaxPool` de tamanho 2, em que os pixels são agrupados em quadrados de tamanho 2×2 e são substituídos pelo maior valor de intensidade de pixel do grupo. Um comportamento similar pode ser feito com o `AvgPool`, em que o pixel resultante é a média dos valores de intensidade dos pixels do grupo.

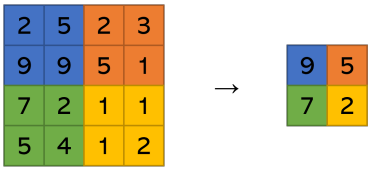

Figura 2.12: Funcionamento do `MaxPool`. Cada grupo de 2×2 pixels é representado pelo valor de maior intensidade do grupo.

Como consequência imediata da aplicação do *pooling*, a informação fica mais condensada, mantendo apenas o que é importante para as próximas camadas e reduzindo ruído, e saída é reduzida em um fator que depende do tamanho do *pooling*, o que faz com que as camadas seguintes tenham menos pixels para processar, sendo mais eficientes [19, 16]. A Figura 2.13 ilustra o efeito de se aplicar o `MaxPool` (b) e o `AvgPool` (c) na imagem de bordas detectadas do cachorro.

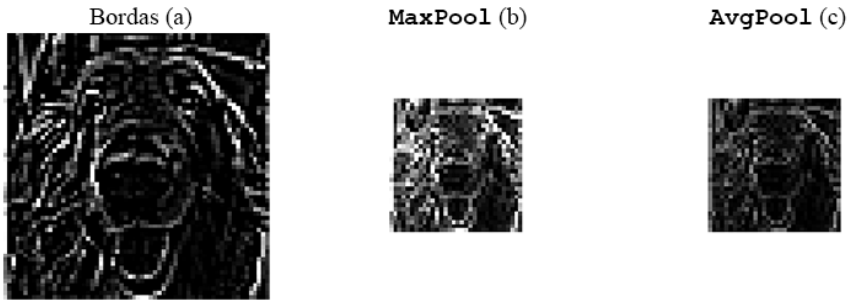

Figura 2.13: Aplicação de *pooling* em uma imagem de bordas. Em (a) a imagem de bordas detectadas por um *kernel* adequado. Em (b) e em (c) a aplicação do `MaxPool` e do `AvgPool`, respectivamente, ambos com tamanho 2. As imagens resultantes tem metade do tamanho da original.

Outro importante efeito do `MaxPool` é que ele torna a rede robusta a pequenas translações na entrada [19]. Como os pixels de uma vizinhança são representados pelo maior valor dentre eles, não há nenhuma diferença na saída se o pixel mais intenso estiver em outra posição, desde que esteja dentro da mesma vizinhança.

Stride

Uma outra forma de se reduzir a dimensão da imagem após uma convolução é utilizando o parâmetro de *stride*, que representa quantos pixels o *kernel* anda em cada iteração [16]. Nos exemplos anteriores o *stride* utilizado foi 1, o que significa que após a iteração o *kernel* se moverá para o pixel vizinho, e que quando a operação terminar a linha, ele irá para a linha seguinte. A parte superior da Figura 2.14 ilustra essa situação com um *kernel* de dimensões 5×5, e os pixels azuis correspondem ao centro do *kernel* em cada iteração.

A metade inferior da Figura 2.14, no entanto, representa o uso do mesmo *kernel*, mas com um *stride* de 2 pixels. Isso significa que o *kernel* avançará para o segundo pixel depois de cada iteração, e quando a linha terminar, ele também irá para a segunda linha. Da mesma forma, os pixels azuis representados na figura correspondem à posição do centro do *kernel* em cada iteração. Como menos regiões são visitadas, a imagem final também é menor.

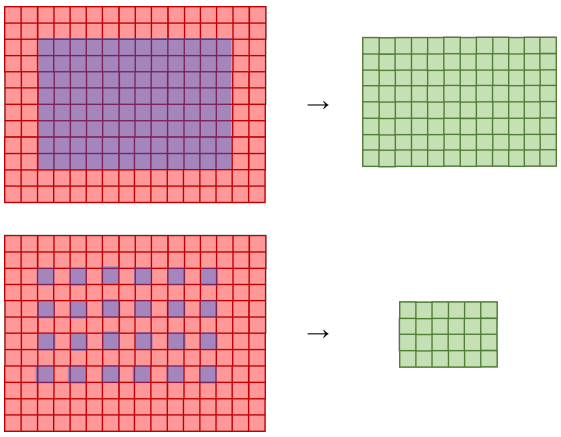

Figura 2.14: Efeito do *stride* na convolução. A parte superior da imagem mostra um caso em que o *stride* é 1, ou seja, o *kernel* anda um pixel na horizontal para cada iteração e no final da linha anda um pixel na vertical. A metade inferior ilustra um caso com valor de *stride* igual a 2, então o *kernel* anda dois pixels na horizontal e quando a linha termina, anda dois pixels na vertical. Em ambas as imagens os pixels azuis correspondem ao centro do *kernel* de tamanho 5×5 em cada iteração, e pode-se verificar que a convolução com maior *stride* gera uma saída de menor dimensão.

A redução de dimensão é importante para situações em que a dimensão da saída da rede é menor do que a dimensão da entrada, como um *encoder* que quer representar a imagem de uma forma comprimida, ou um classificador binário que tem como saída um único valor booleano que indica se a imagem faz parte da classe ou não.

O parâmetro de *stride* pode ser definido separadamente para a largura (w_{stride}) e para a altura (h_{stride}), e dessa forma o cálculo das dimensões da imagem considerando o *padding* e o *stride* é feito pelas expressões:

$$w_{out} = \left\lfloor \frac{w_{in} + 2 \cdot w_{padding} - w_{kernel}}{w_{stride}} + 1 \right\rfloor, \tag{2.12}$$

$$h_{out} = \left\lfloor \frac{h_{in} + 2 \cdot h_{padding} - h_{kernel}}{h_{stride}} + 1 \right\rfloor. \tag{2.13}$$

Convolução transposta

Em outros tipos de aplicação pode-se desejar aumentar a dimensão da imagem a cada camada. Alguns métodos generativos, por exemplo, partem de uma representação menor como um vetor latente e processam essa informação através de várias camadas até gerar uma representação maior como uma imagem [49].

Uma forma de fazer isso utilizando a operação de convolução é através da convolução transposta (muitas vezes confundida com deconvolução [49]), que consiste em utilizar *padding* e *stride* para aumentar a imagem de entrada antes de passar ela por uma convolução "tradicional" [16, 44], de forma que a imagem de saída seja maior do que a imagem de entrada. A Figura 2.15 ilustra o processo pelo qual uma imagem 3×3 passa por uma convolução transposta de *kernel* 3×3 e *stride* 1 para gerar uma imagem 5×5.

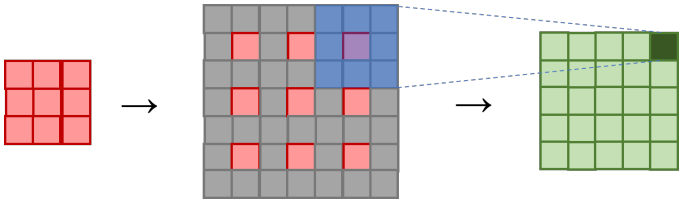

Figura 2.15: Convolução transposta. Uma convolução de *kernel* 3×3 com *stride* 1 em uma imagem 5×5 geraria uma imagem de dimensões 3×3. A convolução transposta aplicada na imagem 3×3 (vermelha) para se obter uma imagem 5×5 (verde) é feita em duas etapas. Primeiramente (centro) se insere pixels de valor 0, representados pela cor cinza, entre os pixels da imagem original, juntamente com pixels de borda. Em seguida se faz uma convolução nessa imagem alterada também usando um *kernel* 3×3 e *stride* 1.

O cálculo das dimensões da imagem gerada pela convolução transposta é feito a partir das expressões:

$$w_{out} = (w_{in} - 1) \cdot w_{stride} - 2 \cdot w_{padding} + w_{kernel}, \tag{2.14}$$

$$h_{out} = (h_{in} - 1) \cdot h_{stride} - 2 \cdot h_{padding} + h_{kernel}. \tag{2.15}$$

A convolução transposta não é a única forma de resolver o problema do aumento de dimensão. Outras arquiteturas utilizam métodos clássicos de redimensionamento, como interpolação bilinear ou interpolação do vizinho mais próximo (*nearest-neighbor interpolation*) seguidos por camadas de convolução na nova resolução de imagem [56, 30].

Artefatos quadriculados

A convolução transposta gera uma imagem que é resultado da convolução de um *kernel* sobre uma imagem menor. Uma característica dessa operação é que o *kernel* se sobrepõe quando passa sobre a imagem de entrada, e nos pontos dessa sobreposição ele reforça a importância daqueles pixels na criação da imagem de maior dimensão [44]. Como podemos ver na Figura 2.16, o resultado dessa sobreposição através de várias camadas reforça um padrão de artefatos quadriculados que aparecem periodicamente na imagem gerada.

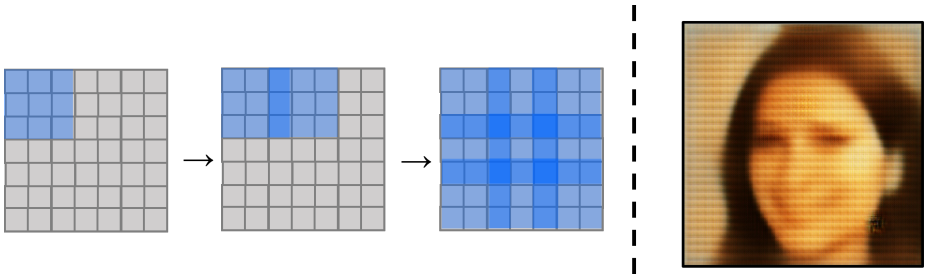

Figura 2.16: Artefatos quadriculados causados pela convolução transposta. À esquerda, uma convolução de um *kernel* de 3×3 com *stride* 2 cria regiões de sobreposição conforme ele passa pela imagem. Essa sobreposição gera artefatos quadriculados na imagem convolucionada.

Outros métodos de redimensionamento na rede convolucional, como a interpolação bilinear seguida por uma convolução, não apresentam esse efeito adverso, porém costumam gerar imagens mais desfocadas, como será discutido na parte experimental deste trabalho.

2.3.3 ResNet

A ResNet [23] (*Deep Residual Network*) é uma arquitetura de CNNs muito profundas, chegando em alguns casos a centenas de camadas, idealizada para a tarefa de reconhecimento de imagens.

Redes mais profundas têm mais mapas de atributos e conseguem extrair mais informação das imagens, em comparação a redes mais rasas [23]. Apesar disso, redes maiores também intensificam o problema da dissipação de gradiente [7] já que a informação deve atravessar muitas camadas, o que prejudica o treinamento pela retropropagação do erro.

A principal mudança na arquitetura foi a inclusão de conexões de atalho que pulam uma ou mais camadas. Através dessas conexões, a informação das camadas superiores pode passar para as camadas inferiores de forma direta, sendo somada ao que foi processado pelas camadas puladas. Esses blocos compostos de camadas convolucionais com conexões de atalho são chamados de blocos residuais e uma representação deles pode ser visualizada na Figura 2.17. A saída de cada bloco residual contém a informação processada pelas camadas do bloco somada à informação da saída do bloco anterior.

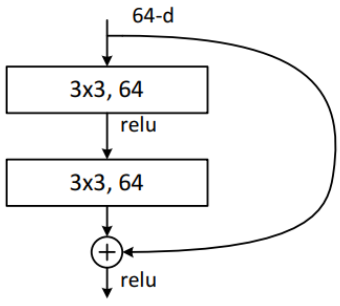

Figura 2.17: Bloco Residual. Fonte: [23]. Camadas de convolução em sequência com *kernel* de dimensões 3×3 e 64 mapas de atributos formam um bloco que é pulado por uma conexão de atalho. A saída da primeira camada de convolução passa pela ativação ReLU e a saída da segunda camada é somada com a conexão de atalho antes de passar também pela ativação ReLU.

A arquitetura ResNet normalmente é referenciada como ResNet-X em que "X" corresponde ao número de camadas da rede, em que suas formas mais comuns são a ResNet-34, a ResNet-50, a ResNet-101 e a ResNet-152. A última camada é uma `Dense`, ou seja, uma camada única de MLP completamente conectada que permite traduzir os mapas de atributos da última convolução em uma classificação.

Como a informação pode atravessar camadas através desses atalhos, ela consegue chegar com pequenas alterações às camadas mais profundas, e com isso o problema da dissipação de gradiente e da degradação da informação são minimizados, sendo possível obter os benefícios de usar redes mais profundas [23].

2.3.4 U-Net

A U-Net [56] é um tipo de CNN que foi desenvolvida com o intuito de realizar segmentação de estruturas de células neuronais em imagens biomédicas de microscópios eletrônicos. Nessa tarefa, a rede não deve apenas classificar se há uma estrutura neuronal, mas também deve dizer a posição dessa estrutura.

A sua arquitetura é baseada em uma rede convolucional que faz sucessivas convoluções na imagem até reduzi-la a um vetor de 1024 elementos, e depois reconstrói a imagem utilizando convoluções inversas, como ilustrado na Figura 2.18. A metade da rede que efetua a redução da dimensão da imagem é chamada de caminho de contração, ou *encoder*, enquanto a metade que atua na reconstrução é chamada de caminho de expansão, ou *decoder*. Entre essas duas metades existem conexões de atalho, que transportam informações do *encoder* para o *decoder*.

A informação que passa do *encoder* para o *decoder* são as características reconhecidas pelas camadas de convolução, e por isso cada camada do *decoder* recebe a informação da sua camada anterior mais os mapas de atributos da camada equivalente do *encoder*. Essa

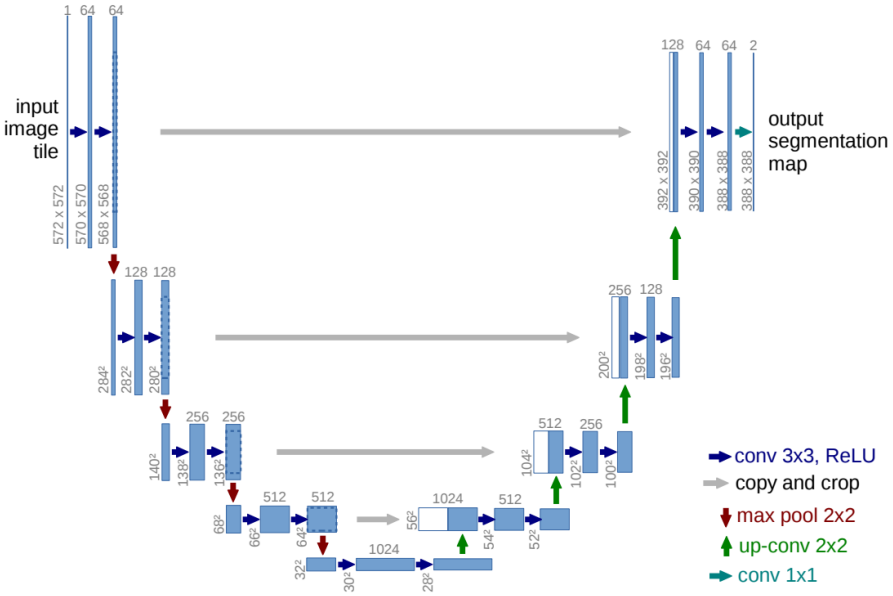

Figura 2.18: Arquitetura U-Net. Fonte: [56]. A parte do *encoder* faz sucessivas convoluções (setas azuis) e reduções de tamanho (setas vermelhas) na imagem até que ela seja codificada em um vetor de 1024 elementos. O *decoder* então reconstrói a imagem através de convoluções a partir do vetor e de informações que o *encoder* manda pelas conexões de atalho (setas cinza).

informação é rica em contexto e permite que o *decoder* gere uma imagem com a segmentação correta das estruturas celulares.

Como existem poucas imagens de estruturas celulares com a segmentação já feita manualmente, a base de dados disponível é muito pequena para permitir um treinamento supervisionado adequado da rede. A solução foi usar deformações elásticas nas imagens existentes para criar novas imagens que são suficientemente diferentes das originais de forma que a rede pudesse considerar como um exemplo diferente. Outra estratégia foi dividir as imagens em pedaços menores, que eram grandes o suficiente para conter as estruturas e seus contextos, mas pequenas o suficiente para que o treinamento não fosse limitado pela memória da GPU [56].

2.3.5 *Frameworks* para aprendizado profundo

A diversidade dos tipos de elementos utilizados para aprendizado profundo como camadas MLP ou convolucionais, blocos de normalização, *pooling*, diferentes funções de ativação e a forma como todos esses elementos se relacionam de acordo com a arquitetura utilizada, torna muito complexo criar o código de redes com muitos elementos.

Além de preparar a arquitetura da rede, ainda é necessário calcular os gradientes das funções de erro em relação aos milhares ou milhões de parâmetros para a se realizar a retropropagação, combinado a algoritmos de otimização variados.

Essa dificuldade levou grupos de pesquisa de aprendizado profundo a desenvolverem *frameworks* que permitem criar redes complexas a partir de objetos simples presentes nessas bibliotecas. As mais utilizadas atualmente são o TensorFlow [1] e o PyTorch [47], ambas baseadas na linguagem Python. Com elas é fácil criar redes utilizando blocos (ou camadas) pré-definidos, simplesmente indicando a relação entre as entradas e saídas de cada um desses blocos.

Complementando os blocos que constituem a rede, as funções de otimização mais populares estão presentes junto a mecanismos de cálculo dos gradientes para a retropropagação. Em ambos os *frameworks* também é possível escrever blocos e funções customizadas, caso o modelo criado seja uma inovação ou tenha complexidade maior.

2.4 Redes Adversárias Generativas - GANs

As Redes Adversárias Generativas, propostas por Goodfellow *et. al.* [20], têm dois componentes principais: um discriminador D, que é treinado para distinguir um conjunto

de dados de entrada (*input*) real de um falso; e um gerador G, que cria conjuntos de dados sintéticos, ou falsos, semelhantes aos reais a partir de uma entrada no formato de um vetor de ruído $z \in Z$ em que Z é o chamado espaço latente da GAN, geralmente amostrado de alguma distribuição de probabilidade. Normalmente o discriminador e o gerador são modelados com redes neurais artificiais.

Sendo p_{data} a distribuição real dos dados e p_z a distribuição da qual os vetores de entrada são amostrados, o gerador G treinado com os parâmetros θ_z atua como uma aplicação $G(z, \theta_z) : p_z \rightarrow p_g$, em que p_g é a distribuição dos dados sintéticos, criados pelo gerador. O intuito da GAN é fazer com que $p_g \sim p_{data}$ após o treinamento.

Em cada iteração, o discriminador D recebe uma amostra de dados reais x_{real} e uma amostra de dados criados pelo gerador $G(z)$. O discriminador acerta sempre que categorizar a entrada real como *verdadeira* e a entrada que veio do gerador como *falsa*. Se o discriminador erra, ele se corrige para ficar mais assertivo numa próxima iteração. Por sua vez, o gerador é reforçado sempre que o discriminador classifica $G(z)$ como *verdadeira* e é penalizado caso contrário. (Fig. 2.19).

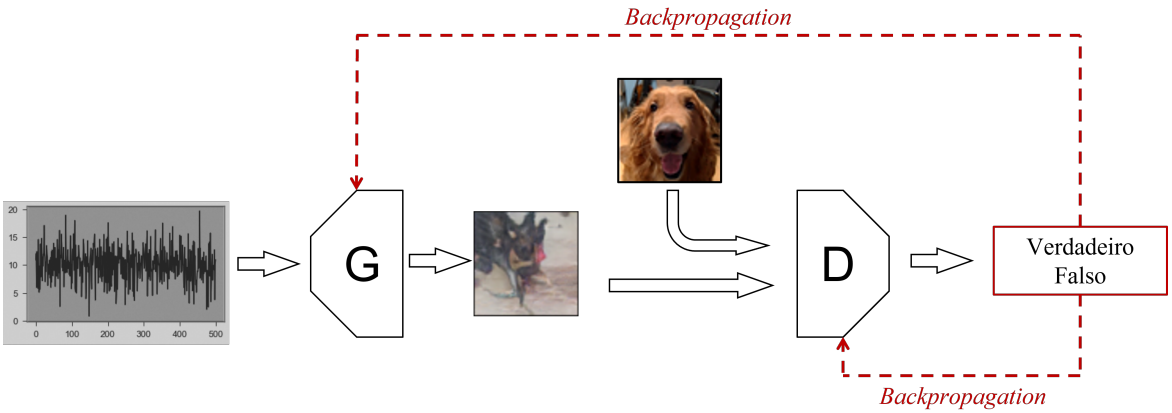

Figura 2.19: Esquema de treinamento para uma GAN de imagens. O gerador cria uma imagem sintética a partir de uma entrada de ruído. O discriminador recebe alternadamente imagens sintéticas e reais e as classifica como "reais" (1) ou "sintéticas" (0).

O gerador e o discriminador participam de um treinamento min-max como representado pela Eq. 2.17 [20]. Interpretando $D(x)$ como a probabilidade de que o dado x tenha vindo da distribuição p_{data} e não de p_g, o discriminador é treinado para maximizar a probabilidade

de discriminar corretamente as imagens reais como *verdadeiras* (1) e as sintéticas como *falsas* (0). O gerador, por sua vez, é treinado para gerar imagens sintéticas que consigam fazer com que o discriminador lhes atribua uma alta probabilidade de serem verdadeiras, minimizando $\log(1 - D(G(z)))$.

$$G^*, D^* = \min_G \max_D \mathcal{L}(D, G), \tag{2.16}$$

onde

$$\mathcal{L}(D, G) = \mathbb{E}_{x \sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))]\,. \tag{2.17}$$

O primeiro termo da Equação 2.17 é o valor esperado de que o discriminador D tenha classificado corretamente a entrada real x como sendo obtida da distribuição p_{data}, e o segundo termo é a probabilidade esperada de que o discriminador tenha classificado corretamente a entrada sintética $G(z)$ como sendo obtida da distribuição p_g, e consequentemente, que z venha da distribuição p_z. A Equação 2.17 é chamada de *loss* de GAN ou *loss* adversária.

2.4.1 DCGAN

As *Deep Convolutional GANs* (DCGANs) [49] são uma aplicação das GANs para a síntese de imagens, utilizando camadas de convolução tanto no gerador (Fig. 2.20) quanto no discriminador.

Assim como em CNNs tradicionais, o papel das convoluções é capturar a representação dos atributos das imagens. O diferencial das DCGANs foi a utilização da estratégia de treinamento adversário na tarefa.

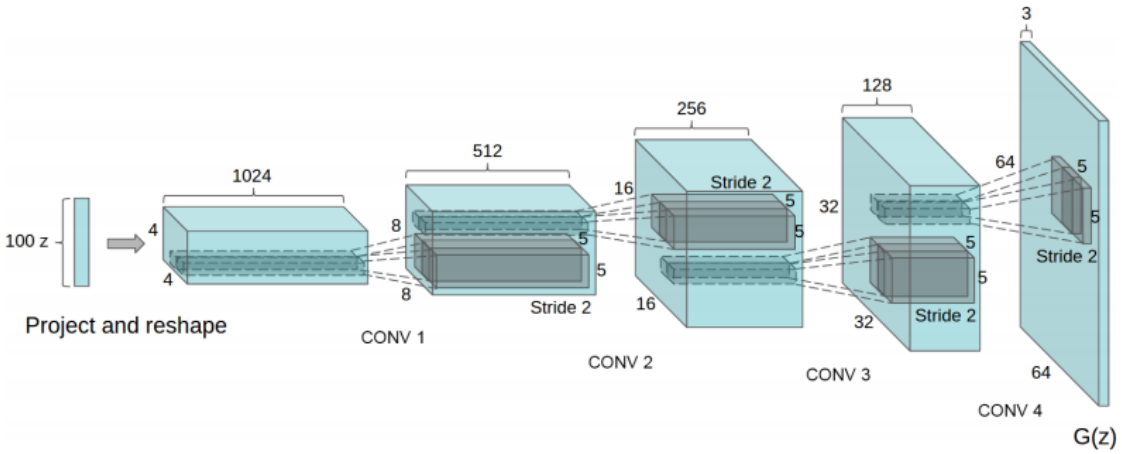

Figura 2.20: Arquitetura original do gerador DCGAN. O vetor de 100 elementos passa por uma camada MLP para se transformar em 16.384 elementos, que são redimensionados para formarem um tensor de dimensões $4 \times 4 \times 1024$. Esse tensor passa então por quatro camadas consecutivas de convolução transposta com *stride* $= 2$ e *kernel* de tamanho 5. Após a última convolução, a imagem final terá dimensões $64 \times 64 \times 3$. Fonte [49].

Os autores mostraram que essa arquitetura consegue sintetizar imagens verossímeis, e que o discriminador treinado pode ser usado para classificação de objetos de forma competitiva, quando comparado com métodos supervisionados.

2.4.2 CGAN

Uma limitação da GAN e da DCGAN originais é que a composição da imagem gerada depende quase exclusivamente da informação contida no vetor de ruído, então há pouco controle do que se quer criar.

As GANs Condicionais (CGANs) [43] incluem um vetor y que condiciona a criação do dado sintético. Esse vetor pode conter a classe à qual pertence o dado, por exemplo, ou algum outro tipo de condição como anotações em linguagem natural. Com a adição de y, a *loss* da CGAN é

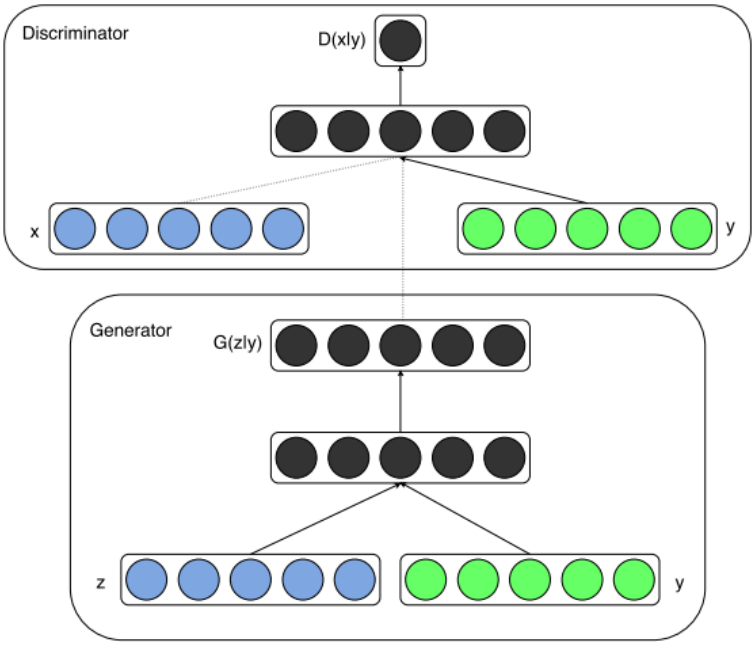

Figura 2.21: Inclusão da condição na CGAN. Fonte: [43]. O gerador recebe o vetor de condição y juntamente com o vetor de ruído z. Similarmente, o discriminador recebe o vetor de condição y junto com o dado x que será discriminado.

$$\mathcal{L}_{CGAN}(D,G) = \mathbb{E}_{x\sim p_{data}(x)}\left[\log D(x|y)\right] + \mathbb{E}_{z\sim p_z(z)}\left[\log(1 - D(G(z|y)))\right], \tag{2.18}$$

enquanto o gerador e o discriminador buscam otimizar o objetivo

$$G^*, D^* = \min_G \max_D \mathcal{L}_{CGAN}(D,G). \tag{2.19}$$

No gerador e no discriminador, o acréscimo da condição pode ser feito através da concatenação da entrada original x ou z com o vetor de condição y (Fig. 2.21).

A CGAN permitiu que fosse possível escolher quais caracteres gerar no dataset MNIST, em vez de gerar um caractere aleatório.

2.5 Problemas comuns no treinamento de GANs

Um fato notório a respeito das GANs é a dificuldade de se treina-las corretamente, principalmente por conta de problemas de convergência e estabilidade [61, 4, 75].

Um dos principais problemas é o colapso de modo, em que o gradiente do discriminador tende a orientar o gerador a criar imagens de um único modo (ou de poucos diferentes modos), gerando imagens muito similares e com pouca diferença entre si [61]. Outro problema bastante comum é a dissipação de gradiente do gerador que ocorre conforme o discriminador se aproxima da saturação [4, 5].

Partindo-se do princípio que uma das principais causas de instabilidade no treinamento das GANs é o uso da divergência KL como *loss* adversária original [4, 21], Arjovsky *et al.* propuseram usar a distância *Earth Mover* (EM) ou distância de Wasserstein na composição de uma *loss* não-saturada, chamando a rede resultante de Wasserstein-GAN (WGAN), que melhora consideravelmente os problemas de colapso de modo e dissipação de gradiente [5, 22]. Outros métodos usam diferentes funções objetivo e termos de regularização para abordar e corrigir esses problemas [21, 75].

A abordagem de treinamento progressivo também auxilia no aumento da estabilidade, por dividir o aprendizado que deveria ser feito em toda a rede simultaneamente em uma abordagem de camadas. Dessa forma, a cada nova camada adicionada, a representação aprendida pelas anteriores já deve ter convergido, fazendo com que essas camadas antigas apenas precisem refinar essa representação sem grandes atualizações, reduzindo a instabilidade da rede completa [30].

2.5.1 WGAN e WGAN-GP

A Wasserstein GAN [5], proposta por Arjovsky *et al.*, altera a *loss* adversária original e passa a usar a distância de Wasserstein no lugar da divergência KL. Sua *loss* pode ser expressa pelas equações

$$\mathcal{L}_G^{WGAN} = \frac{1}{m}\sum_{i=1}^{m} D(G(z^{(i)})), \tag{2.20}$$

e

$$\mathcal{L}_D^{WGAN} = \frac{1}{m}\sum_{i=1}^{m} D(x^{(i)}) - \frac{1}{m}\sum_{i=1}^{m} D(G(z^{(i)})), \tag{2.21}$$

em que $\mathcal{L}_G$ e $\mathcal{L}_D$ são respectivamente a *loss* do gerador e do discriminador, e m é o tamanho do lote (*batch size*), $z^{(i)}$ é o i-ésimo vetor de entrada, e $x^{(i)}$ a i-ésima imagem real de referência. Uma restrição importante para o uso da *loss* de Wasserstein é que o discriminador deve ser Lipschitz contínuo, e para garantir essa continuidade, o ajuste dos pesos das camadas da rede são limitados a um hipercubo com cada aresta variando entre $[-c, c]$, definido originalmente com $c = 0,01$.

Segundo os próprios autores, esse corte no ajuste dos pesos é uma forma terrível de se forçar a continuidade [5], e alguns problemas dessa abordagem são aprofundados no trabalho de Gulrajani *et al.* [22], que propuseram adicionar um termo de penalidade de gradiente, criando então a WGAN-GP (*Wasserstein GAN with Gradient Penalty*). Com esse novo termo, a *loss* do discriminador se torna

$$\mathcal{L}_D^{WGAN-GP} = \mathcal{L}_D^{WGAN} + \lambda\frac{1}{m}\sum_{i=1}^{m}\left[\left(||\nabla_{\hat{x}^{(i)}} D(\hat{x}^{(i)})||_2 - 1\right)^2\right], \tag{2.22}$$

em que λ é um parâmetro que regula o efeito da penalidade de gradiente na *loss*, normalmente definido como $\lambda = 10$. O elemento $\hat{x}^{(i)}$ é a interpolação entre um exemplo de imagem real $x^{(i)}$ e uma imagem sintética $G(z^{(i)})$ de forma que

$$\hat{x}^{(i)} = \epsilon x^{(i)} + (1 - \epsilon)G(z^{(i)}), \tag{2.23}$$

com ϵ sendo definido como um tensor de mesmas dimensões dos elementos $x^{(i)}$ e $G(z^{(i)})$, cujos elementos são números aleatórios amostrados de um intervalo uniforme $[0, 1]$.

O uso da *loss* de Wasserstein com penalidade de gradiente permite um treinamento mais estável do gerador e do discriminador (chamado neste contexto de crítico), evitando os problemas de colapso de modo e dissipação do gradiente [4, 5, 22]. Essa técnica pode ser utilizada em conjunto com outras técnicas [75] e é inclusive empregada em outras GANs, como a ProGAN [30] e a StyleGAN [33].

2.6 Avaliação de Qualidade

Um problema ainda em aberto na área de síntese de imagens é como se avaliar a qualidade percebida das imagens geradas [21]. Algumas das primeiras GANs que perseguiram a criação de imagens realistas, como a Pix2Pix e a CycleGAN [28, 80], usavam ferramentas como o Amazon Mechanical Turk que permitiam que pessoas dissessem se consideravam que as fotos eram realistas ou não.

Esse tipo de método é interessante por levar em conta a subjetividade da percepção humana, mas não é escalável nem prático de ser usado sempre que se quiser avaliar um experimento. Os resultados também dependem da motivação dos avaliadores e são drasticamente diferentes quando os eles recebem um retorno sobre seus erros: eles passam a

perceber melhor os erros e tendem a uma análise mais pessimista da qualidade [61]. Assim, algumas alternativas foram então propostas.

2.6.1 *Inception Score* - IS

Radford *et al.* [61] aproveitaram o bom desempenho da rede InceptionV3 em classificar imagens nas 1000 diferentes classes da ImageNet [69] e propuseram uma abordagem que consiste em passar várias imagens sintéticas x através dessa rede para obter as probabilidades condicionais $P(y^i|x)$ para cada classe y^i e calcular uma pontuação (*score*) baseada nessas distribuições de probabilidade. Quanto maior o *score*, melhor a qualidade das imagens sintéticas.

Segundo os autores, essa métrica correlaciona bem com o julgamento humano [61], o que indica que pode ser usada como uma medida de qualidade percebida. No entanto, o IS pode apresentar bons resultados mesmo quando a rede sofreu de colapso de modo [21, 24].

2.6.2 Frechét *Inception Distance* - FID

Se baseando no IS, Heusel *et al.* [24] criaram uma métrica chamada Frechét Inception Distance (FID) que usa a InceptionV3 tanto em imagens reais quanto em imagens sintéticas e criam duas distribuições multinormais, uma para cada conjunto de imagens, porém, em vez de usarem a saída da rede para calcular as probabilidades, eles usam os *logits* da penúltima camada.

Em seguida eles calculam a distância de Frechét (também chamada de Wasserstein-2) entre as duas distribuições, chegando ao valor do FID. Quanto menor essa distância, menor o FID, e mais as imagens sintéticas se assemelham das imagens reais. Apesar de a

versão original da FID usar a InceptionV3 como base para cálculo, outras redes podem ser usadas [21, 36].

No momento o FID é uma das métricas mais usadas para se comparar o resultado de diferentes GANs, mas nem sempre essa comparação é justa. Dentro de uma mesma arquitetura, uma melhoria da FID normalmente é correlacionada com uma melhoria na qualidade percebida da imagem gerada, no entanto, ao se comparar arquiteturas diferentes o valor numérico não consegue mostrar necessariamente qual delas é melhor. Isso se deve ao fato de que a FID é muito dependente das classes da ImageNet em seu cálculo, que geralmente é mais afetado por texturas do que por forma [36].

Outras medidas como o KID [8] ou o MS-SSIM [74] também podem ser usadas, embora todas tenham lados negativos. Por conta disso, a prática mais adotada para se comparar métodos diferentes é apresentando diversas métricas em conjunto.

2.7 Normalização em Síntese de Imagens

A normalização é uma ferramenta importante no contexto do aprendizado profundo [19, 27, 71, 76, 26, 62]. Elas auxiliam na estabilidade do treinamento por garantir que os sinais que passam de uma camada para a próxima vão estar dentro de um intervalo definido, mantendo-os em uma mesma escala. Argumenta-se que esse aumento de estabilidade permite um treinamento mais rápido [27]

No contexto das GANs de síntese de imagens, além dos benefícios de estabilidade e convergência, alguns tipos de normalização tem o poder de determinar o estilo da imagem final [71, 26, 33].

Nesta seção serão apresentados os principais métodos de normalização e a importância deles para síntese e manipulação de imagens.

Batch Normalization

Ioffe e Szegedy [27] hipotetizaram que o *internal covariate shift* (ICS) é um dos motivos pelos quais o treinamento de redes muito profundas tende a ser lento. Esse efeito ocorre por conta da mudança de distribuição na ativação de cada camada, decorrente da própria variação dos exemplos que são utilizados durante o treinamento. Por conta dessa variação, os parâmetros das camadas demoram para convergir, deixando o treinamento mais lento.

Os autores mostram que uma forma eficiente de estabilizar o treinamento é normalizando os exemplos dentro de cada lote (ou *batch*), o que eles chamaram de *batch normalization*, ou `BatchNorm`.

Essencialmente, a `BatchNorm` calcula a média μ_B e o desvio padrão σ_B de todos os valores de todos os exemplos dentro de um mesmo lote, e usa essas informações para normalizar cada exemplo x_i de dentro do lote de acordo:

$$\hat{x_i} = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}. \tag{2.24}$$

No contexto de redes convolucionais bidimensionais, em uma entrada de dimensões $(W \times H \times C \times N)$, em que W e H são a largura e altura da imagem ou dos mapas de atributos, C é a quantidade de canais ou mapas, e N é o tamanho do lote, a normalização acontece individualmente em cada canal, mas utilizando as estatísticas do lote inteiro.

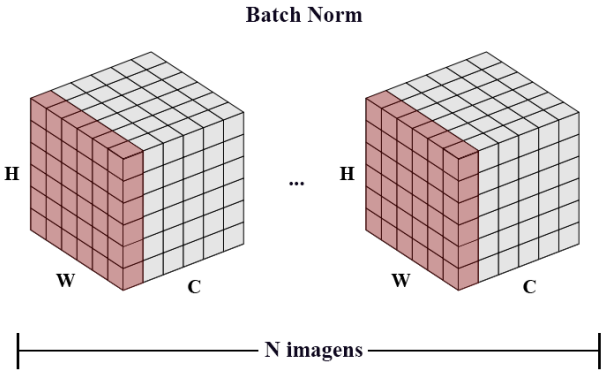

Figura 2.22: Funcionamento da `BatchNorm`. A média e desvio padrão de cada canal é calculada usando todos os pixels de todas as imagens do lote (em vermelho).

A Figura 2.22 representa como acontece esta normalização. Os elementos em vermelho correspondem a um canal C, de dimensões $W \times H$. Para o cálculo da média e desvio padrão são considerados todos os pixels de todas as imagens do lote.

O fato de uma das ativações estar sendo mais reforçada do que as outras é essencial para o aprendizado da rede, e é assim que a rede sabe selecionar quais atributos são mais importantes para cada situação. A normalização em si descarta esse efeito, então a `BatchNorm` também inclui um parâmetro de escala γ e um parâmetro de viés β, ambos treináveis, para recuperar esse efeito, como na Eq. 2.25.

$$y_i = \gamma \hat{x}_i + \beta. \tag{2.25}$$

É argumentável se a *batch normalization* realmente reduz o ICS [62], mas o efeito prático dela é real, e por isso é bastante utilizada em aplicações diversas de aprendizado profundo.

Instance Normalization

A *instance normalization* [71], ou `InstanceNorm`, se assemelha à *batch normalization*, com a diferença que em vez de normalizar cada canal dentro do lote, a *instance normalization* normaliza cada elemento individualmente. Por conta disso, ela não é afetada pelo tamanho do lote que é definido no treinamento.

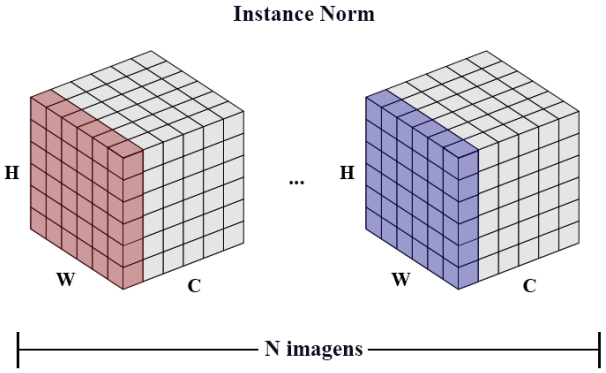

Figura 2.23: Funcionamento da `InstanceNorm`. A média e desvio padrão de cada canal é calculada usando todos os pixels de cada imagem do lote individualmente.

Na Figura 2.23 é mais fácil ver a diferença: em vez de normalizar todas as imagens (ou elementos) do *batch* através da dimensão do canal, como acontece na *batch normalization*, a `InstanceNorm` normaliza cada canal de cada imagem individualmente. Experimentalmente é demonstrado que esse tipo de normalização tem um desempenho melhor do que a *batch normalization* em tarefas de transferência de estilo [71, 26].

A Figura 2.24 compara a aplicação da `InstanceNorm` para a transferência de estilo com as técnicas existentes na época da sua publicação [71]. Nela, a primeira imagem (superior esquerda) é a base e a segunda imagem (superior central) é o estilo a ser aplicado na imagem base, e é possível ver que o resultado da transferência de estilo executada com `InstanceNorm` (inferior direita) é superior ao resultado obtido com as outras técnicas.

Figura 2.24: Uso de `InstanceNorm` na transferência de estilo. Fonte: [71].A primeira linha mostra uma imagem base (esquerda), uma imagem de estilo (centro) e a transferência de estilo conforme o método de Gatys *et. al.* [17] (direita). Na segunda linha três transferências de estilo conforme o método de estilização rápida de Ulyanov *et. al.* [70], com *zero padding* (esquerda), uma técnica mais aprimorada de *padding* (centro), e com *zero padding* e `InstanceNorm` (direita).

Layer Normalization

A *layer normalization* [6], ou `LayerNorm`, normaliza todos os canais de cada imagem do *batch* individualmente. Por conta disso, pode-se dizer que é uma normalização intra-imagem que reduz grandes variações que podem acontecer entre canais diferentes da mesma imagem.

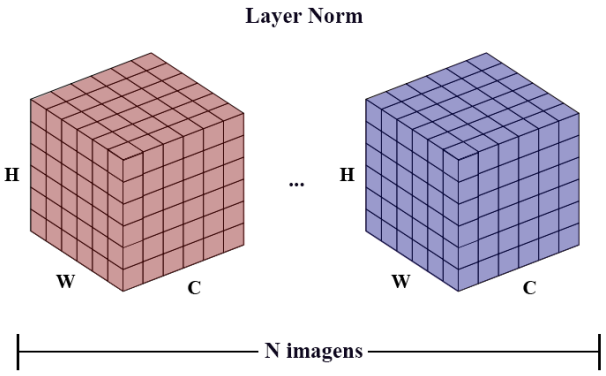

Figura 2.25: Funcionamento da `LayerNorm`. A média e desvio padrão de imagem é calculada usando todos os pixels de todos os canais dela. A normalização de cada imagem é calculada individualmente.

Na Figura 2.25 os elementos em vermelho são usados para calcular as estatísticas que serão usadas para normalizar aquela mesma imagem à esquerda, e os elementos em azul são usados para normalizar a imagem à direita.

Segundo Lei Ba *et al.*, a `LayerNorm` aumenta a performance de RNNs (*Recurrent Neural Networks*) em relação à `BatchNorm`. Esse tipo de normalização não é muito utilizado no contexto de GANs de processamento de imagem.

Group Normalization

A *group normalization* [76] (`GroupNorm`) pode ser considerada um meio termo entre a `InstanceNorm` e a `LayerNorm`.

Assim como a *layer normalization*, a `GroupNorm` também normaliza cada imagem individualmente através dos canais, como representado na Figura 2.26, mas em vez de atuar em todos os canais de uma vez esse método os divide em grupos e atua em cada grupo individualmente. Na imagem à esquerda da Figura 2.26, os canais em vermelho serão normalizados como um grupo e os canais em cinza serão normalizados como um segundo grupo.

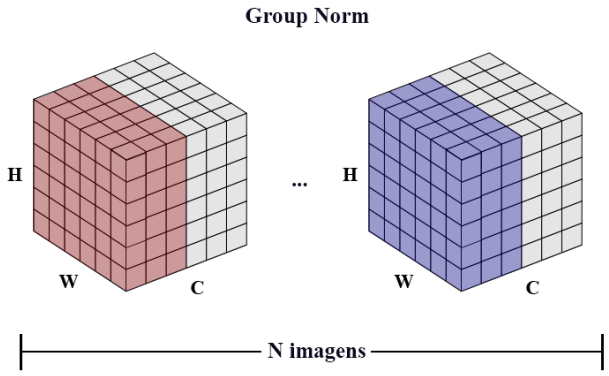

Figura 2.26: Funcionamento da `GroupNorm`. A média e desvio padrão de imagem é calculada usando todos os pixels de um grupo de canais de cada vez. A normalização de cada imagem é calculada individualmente.

No caso especial em que o número de grupos é igual ao número de canais o `GroupNorm` equivale ao `InstanceNorm`. Similarmente, no caso especial em que o número de grupos é igual a 1, o `GroupNorm` equivale ao `LayerNorm`.

PixelNorm

A *Pixelwise Feature Vector Normalization* (`PixelNorm`) foi proposta por Karras *et. al.* [30] como uma forma de evitar que a magnitude dos sinais de ativação escalem muito rapidamente.

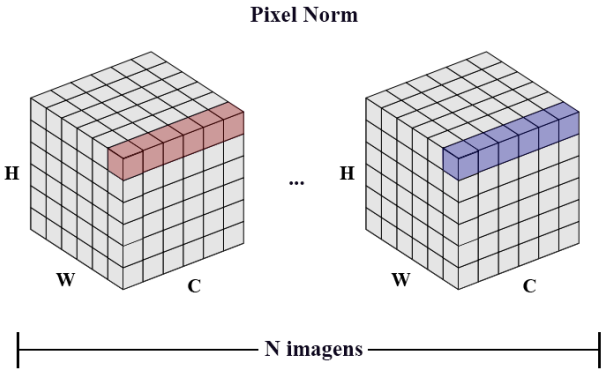

Figura 2.27: Funcionamento da `PixelNorm`. A média e desvio padrão de imagem é calculada usando todos os pixels de um grupo de canais de cada vez. A normalização de cada imagem é calculada individualmente.

A Figura 2.27 exemplifica o funcionamento da `PixelNorm`. A seção em vermelho equivale a um pixel (um elemento W × H) da imagem à esquerda. Similarmente, os elementos em azul equivalem a um pixel da imagem à direita. A normalização ocorre através dos canais, representados pela dimensão C.

Considerando que cada imagem do *batch* tem C canais, a `PixelNorm` calcula a média da ativação de todos os canais para cada pixel $p_{x,y}$ de cada elemento, como ilustrado na 2.27, e então essa média é utilizada para normalizar o pixel de acordo com:

$$\hat{p}_{x,y} = p_{x,y} / \sqrt{\frac{1}{C} \sum_{c \in C} (p_{x,y}^{c})^2 + \epsilon}, \tag{2.26}$$

com $p_{x,y}^{c}$ sendo o valor da ativação do canal c para o pixel $p_{x,y}$ e $\epsilon = 10^{-8}$.

AdaIN

O método `AdaIN` (*Adaptative Instance Normalization*) [26] é uma expansão da *instance normalization*. Ele mantém a normalização individual de cada elemento, mas se diferencia pela forma como trabalha os parâmetros de escala e de viés.

Sendo x uma imagem base, e y uma imagem de estilo e considerando que queremos aplicar o estilo de y na imagem base x, o `AdaIN` realiza a normalização da seguinte forma:

$$AdaIN(x, y) = \sigma(y) \frac{x - \mu(x)}{\sqrt{\sigma(x)}} + \mu(y). \tag{2.27}$$

Na prática, o `AdaIN` transfere a média e o desvio padrão de y para x durante todo o processo de geração da imagem sintética. Assim, a imagem final x sintetizada pelo gerador terá características da imagem y.

3 REVISÃO BIBLIOGRÁFICA

As áreas de processamento de imagens e visão computacional utilizando modelos de aprendizado profundo tem atraído muita atenção nos últimos anos, e muitos estudos têm sido publicados nessa área com diversos trabalhos acontecendo paralelamente, o que torna essa uma área desafiadora de se acompanhar [75, 21].

Alguns trabalhos foram tão importantes que, apesar de serem recentes, formaram a base de onde as novas pesquisas partem. Entre eles, podemos destacar as primeiras construções de GANs e DCGANs [20, 49, 43], e as redes residuais [56, 23, 29], que possuem tamanha importância que foram citados no Capítulo 2 como parte da teoria necessária para a compreensão deste trabalho.

Neste capítulo apresentaremos trabalhos que são importantes especificamente para a área de síntese de imagens, e muitos deles se baseiam nos artigos já citados anteriormente.

3.1 Técnicas de Transformação de Imagens

No contexto deste trabalho, denominaremos *transformações de imagens* os métodos que criam funções $G : A \rightarrow B$ que mapeiam imagens de um domínio A em imagens que podem ser consideradas como parte de um domínio B. Um exemplo desse tipo de transformação é a *image-to-image translation*, em que imagens inteiras são transformadas

em outro tipo de imagem, como cavalos em zebras, paisagens diurnas em noturnas, anotações em imagens, dentre outras [28, 80]. Outro exemplo é a transferência de estilo (*style transfer*), em que uma imagem base de um domínio A é transformada para se assemelhar em estilo a imagens do domínio B, como se transformando imagens em pinturas [80].

Esses métodos podem ter abordagens supervisionadas como a Pix2Pix [28] ou a Pix2Pix HD [72] e não-supervisionadas como a CycleGAN [80], discutidas a seguir.

Métodos supervisionados usam imagens pareadas, o que significa que para cada imagem $x_A \in A$ deve haver uma imagem objetivo $x_B \in B$, e o gerador G deve ser treinado para representar $x_B = G(x_A)$ para todo par (x_A, x_B). Isso é uma limitação que pode impedir o uso em diversas aplicações, já que são poucas as bases de dados amplamente disponíveis com essa característica, e criar bases sintéticas pode não ser a melhor estratégia para representar modelos do mundo real.

Métodos não supervisionados, no entanto, dispensam o uso de imagens pareadas, porém contam com restrições que reduzem o seu escopo de aplicação. No caso da CycleGAN, por exemplo, as imagens precisam ter proximidade estrutural, como na transformação de cavalos em zebras [80].

3.1.1 Pix2Pix

A Pix2Pix [28] é uma GAN condicional que faz transformações supervisionadas de imagens entre domínios diferentes. Como demonstrado na Figura 3.1, ela foi aplicada na tradução de anotações para fotos de ruas com carros, anotações para fachadas, coloração de imagens em escala de cinza, imagens aéreas para mapas, imagens diurnas para imagens noturnas e tradução de bordas para fotos.

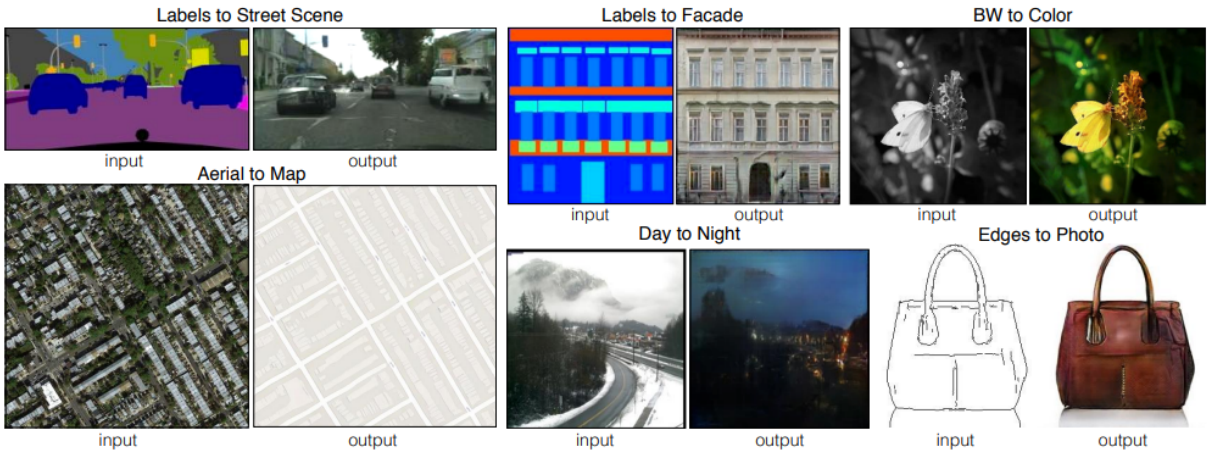

Figura 3.1: Algumas aplicações da Pix2Pix em tranformação de imagens entre domínios. Fonte: [28]

Essa arquitetura é adaptada de uma GAN condicional (CGAN [43]), em que o gerador recebe como "condição" a imagem x_A (por exemplo uma imagem de bordas), mas não recebe como entrada o vetor de ruído presente na CGAN original (Figura 3.2).

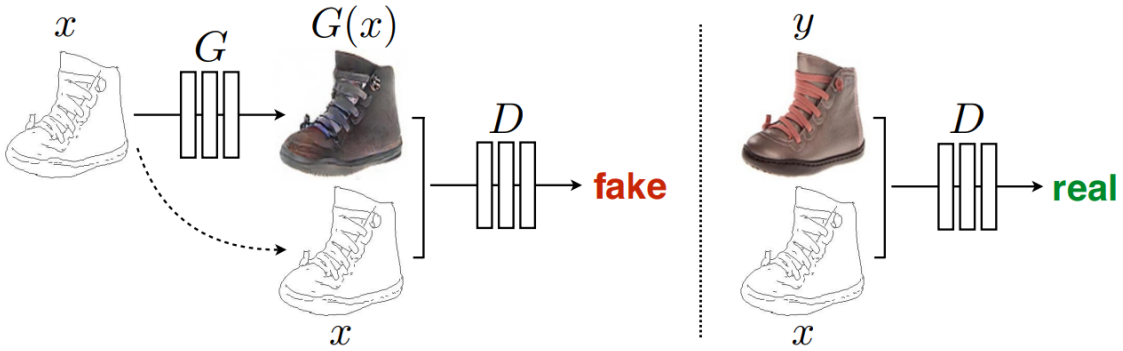

Figura 3.2: Arquitetura simplificada da Pix2Pix. Fonte: [28]. À esquerda um esboço x é apresentado ao gerador G que cria uma imagem sintética $G(x)$. O discriminador D recebe a imagem sintética junto com a entrada x e deve discriminar o conjunto como "*fake*", ou "falso". À direita uma imagem real y é apresentada ao discriminador juntamente com o esboço correspondente x. O discriminador deve então classificar o conjunto como "real". Para um gerador ideal G^*, a imagem y e a imagem sintética $y^* = G^*x$ devem ser idênticas.

O discriminador, por sua vez, tem uma etapa de discriminação de imagem real alternada com uma etapa de discriminação de imagem sintética. Quando ele pretende discriminar uma imagem real, ele recebe como entrada a imagem x_A, que é também a

entrada do gerador, e a imagem x_B, que é o objetivo. Na etapa de discriminação da imagem sintética, ele recebe $G(x_A)$ e x_B.

Por ser uma GAN condicional, a Pix2Pix também usa a *loss* da CGAN (Eq. 2.18) tanto para o gerador quanto para o discriminador, porém é acrescentada uma restrição para que a imagem sintética $G(x_A)$ contenha as características do objetivo x_B. Essa restrição é a distância L1 pixel a pixel entre a imagem sintética e o objetivo. O gerador é treinado de forma a minimizar $\mathcal{L}_G$ enquanto o discriminador busca maximizar $\mathcal{L}_D$:

$$\mathcal{L}_G = \mathcal{L}_{CGAN}(G, D) + \lambda \|x_B - G(x_A)\|_1, \tag{3.1}$$

$$\mathcal{L}_D = \mathcal{L}_{CGAN}(G, D), \tag{3.2}$$

em que $\mathcal{L}_{CGAN}(G, D)$ é a *loss* da CGAN e $\|x_B - G(x_A)\|_1$ é a norma L1 (distância *Manhattan*) das imagens. O parâmetro λ regula o efeito do termo L1 no treinamento do gerador, definido experimentalmente por Isola *et al.* como $\lambda = 100$, mas que pode variar de acordo com a arquitetura e o experimento.

Além dessas alterações, a Pix2Pix usa uma estratégia de treinamento chamada PatchGAN com o intuito de melhorar a qualidade da imagem final. A saída do discriminador nesse caso não é um único escalar que pode assumir valores no intervalo $[0, 1]$, mas uma matriz $m \times m$ em que cada elemento corresponde a uma seção da imagem original, como ilustrado pela Figura 3.3. Cada um desses elementos pode assumir valores no intervalo $[0, 1]$ e correspondem à percepção que o discriminador tem de aquele pedaço (*patch*) da imagem corresponder a uma imagem real ou sintética.

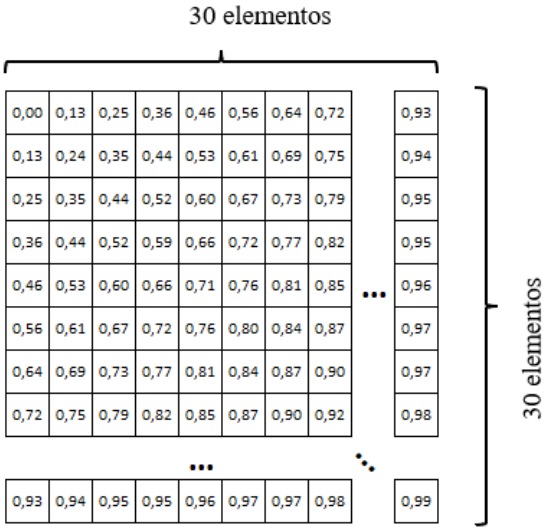

Figura 3.3: Saída de um discriminador PatchGAN 30×30. Cada elemento da matriz corresponde a uma seção (*patch*) da imagem, e seu valor é a avaliação que o discriminador faz sobre aquela seção ser parte de uma imagem real ou falsa.

3.1.2 CycleGAN

A CycleGAN [80] propõe uma abordagem não-supervisionada para a transformação de domínio, a partir de GANs condicionais que tenham consistência de ciclo, ou seja, que consigam transformar imagens de forma consistente e bidirecional entre dois domínios.

Sejam A e B dois domínios distintos, mas que tenham características similares, como cavalos e zebras, ou paisagens e seus mapas semânticos. Um gerador $G_A : A \rightarrow B$ realiza a transformação $x_B = G_A(x_A)$ transformando a imagem x_A do domínio A em uma imagem sintética x_B, que corresponde à sua equivalente no domínio B. Similarmente um gerador $G_B : B \rightarrow A$ transforma uma imagem y_B do domínio B na sua correspondente $y_A = G_B(y_B)$ no domínio A.

Para garantir que a transformação $A \rightarrow B$ ocorra corretamente, existe um discriminador D_B que classifica as imagens como sendo pertencentes ou não ao domínio B, e que é treinado com imagens reais do domínio B e imagens sintetizadas a partir de imagens do domínio A. Também existe um discriminador D_A que avalia se as imagens pertencem

ou não ao domínio A, de maneira similar. Um esboço da arquitetura pode ser visto na Figura 3.4.

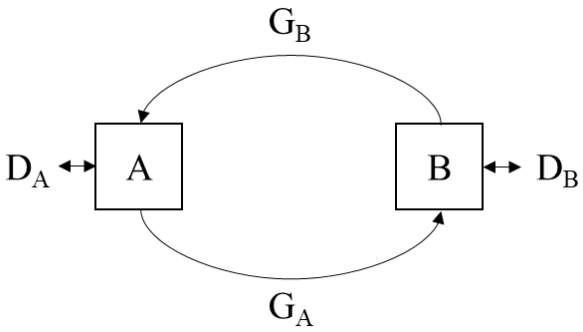

Figura 3.4: Arquitetura básica da CycleGAN.

Apenas com essas restrições, G_A pode convergir a um modo em que toda imagem $x_A \in A$ seja transformada numa mesma imagem x_B que seja discriminada por D_B como sendo do domínio B, mas que não mantenha a estrutura da imagem original. Para evitar isso os autores propuseram uma *loss* de consistência de ciclo (Fig. 3.5), que consiste em obter a imagem sintética $\hat{x}_A = G_B(G_A(x_A))$ e calcular sua distância da imagem original x_A, e repetir o procedimento para o par $(\hat{x}_B, x_B)$.

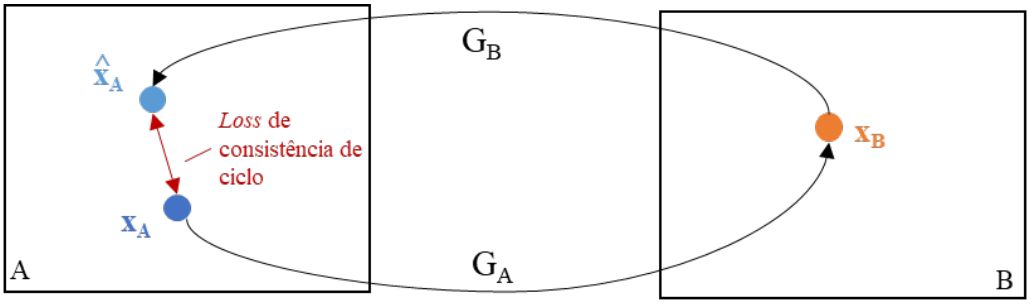

Figura 3.5: *Loss* de consistência de ciclo. A imagem x_A é transformada em x_B e de volta em $\hat{x}_A$. A diferença entre a posição original de x_A e a nova posição de $\hat{x}_A$ é a *loss* de consistência de ciclo.

Com essa nova restrição, os geradores são encorajados a fazerem um mínimo de alterações estruturais nas imagens, e transformar apenas os atributos que de fato diferenciam A de B. A *loss* de consistência de ciclo é então dada por:

$$\mathcal{L}_{cyc}(G_A, G_B) = \|G_B(G_A(x_A)) - x_A\|_1 + \|G_A(G_B(x_B)) - x_B\|_1. \tag{3.3}$$

Uma última restrição é a chamada *loss* identidade. Se o gerador G_A transforma uma imagem do domínio A para o domínio B, caso ele já receba como entrada uma imagem do domínio B ele não deve realizar nenhum tipo de alteração nessa imagem. Seja x_B uma imagem do domínio B e $\hat{x}_B = G_A(x_B)$ o resultado de se aplicar essa imagem no gerador G_A, e $\hat{x}_A = G_B(x_A)$ a aplicação simétrica no gerador G_B. A *loss* identidade é dada pela equação

$$\mathcal{L}_{id}(G_A, G_B) = \mathbb{E}_{x_B \ P_{data}(x_B)} \|G_A(x_B) - x_B\|_1 + \mathbb{E}_{x_A \ P_{data}(x_A)} \|G_B(x_A) - x_A\|_1. \tag{3.4}$$

Os autores perceberam que essa *loss* ajuda a preservar as cores da imagem original na imagem sintética, e a aplicaram na tarefa de tradução pintura $\leftrightarrow$ fotografia.

Assim, a loss completa do sistema é:

$$\begin{aligned} \mathcal{L}(G_A, G_B, D_A, D_B) = &\mathcal{L}_{CGAN}(G_A, D_B) + \mathcal{L}_{CGAN}(G_B, D_A) \\ &+ \gamma \mathcal{L}_{cyc}(G_A, G_B) + \frac{\gamma}{2} \mathcal{L}_{id}(G_A, G_B), \end{aligned} \tag{3.5}$$

em que L_{CGAN} representa a *loss* da CGAN (Eq. 2.18) e γ é um parâmetro que regula a amplitude do efeito da *loss* de consistência de ciclo e da *loss* identidade no treinamento.

Os geradores e discriminadores usados pela CycleGAN são os mesmos da Pix2Pix [28], mas os autores também criaram um novo gerador com conexões residuais [23]. A arquitetura dos geradores pode ser vista em detalhes nos artigos correspondentes [28, 80].

Apesar da vantagem de ser um método não supervisionado, a CycleGAN tem como limitação a incapacidade de fazer alterações significativas na estrutura e na forma das imagens. Ainda assim é bastante eficiente na transferência de texturas e atributos simples.

3.2 Técnicas de Síntese de Imagens

Embora as técnicas apresentadas na Seção 3.1 criem imagens sintéticas, elas partem de uma imagem base que é modificada de acordo com o objetivo do treinamento. As técnicas de síntese de imagens, no contexto deste trabalho, serão consideradas aquelas que sintetizam uma imagem inexistente a partir de um vetor latente.

Existem diversas técnicas que permitem sintetizar imagens de alta qualidade. Algumas delas, como a ProGAN [30] e as StyleGANs [33, 34, 31, 32] são especializadas em gerar imagens de um único domínio, enquanto outras, como a SAGAN [78] e a BigGAN [11] conseguem gerar imagens de vários domínios diferentes com um único gerador.

Pode-se manipular e guiar a síntese através de alterações no vetor latente de entrada. Consequentemente, para se manipular uma imagem específica, é antes necessário se obter o vetor latente que a gera em uma GAN já treinada. Algumas técnicas mostram formas de se obter esse vetor [79, 2, 3, 54].

A seguir mostramos como a ProGAN [30] e a StyleGAN [33] original sintetizam imagens de alta qualidade, e apresentaremos uma técnica de obtenção do vetor latente a partir de uma imagem específica e uma GAN treinada [79].

3.2.1 ProGAN

Karras *et. al.* [30] propuseram uma estratégia de treinamento de GANs não condicionais que conseguia gerar imagens realistas em alta resolução (1024^2). Nesse método tanto o gerador quanto o discriminador começam a ser treinados em baixa resolução (4^2) e progressivamente suas camadas vão crescendo juntamente com a resolução da imagem gerada. Por conta do crescimento progressivo, ilustrado na Figura 3.6, a GAN proposta é comumente chamada de ProGAN ou ProgGAN.

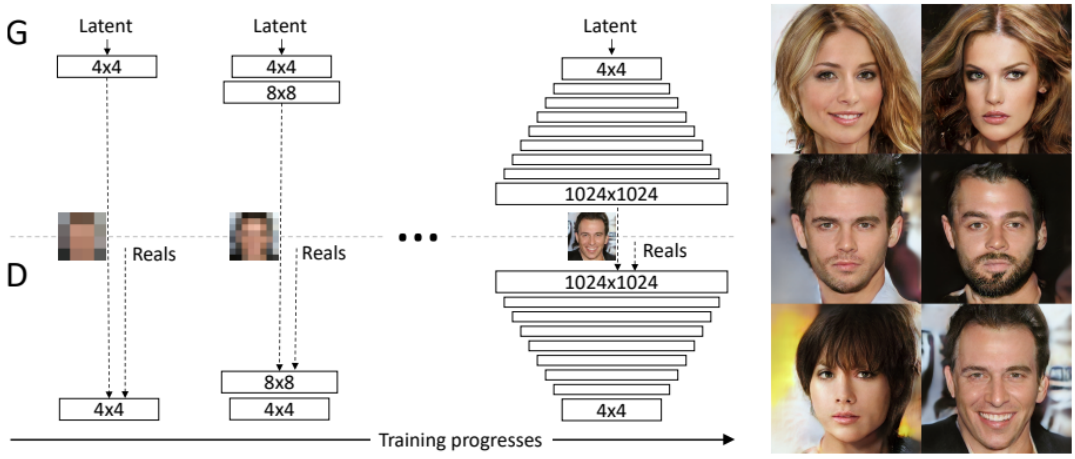

Figura 3.6: ProGAN: crescimento progressivo das GANs. Fonte: [30]

O gerador proposto pelos autores amostra vetores $z \in \mathbb{R}^n$ em que cada dimensão n vem de uma distribuição estatística, mais frequentemente a distribuição normal. Com o vetor, o gerador consegue ser treinado de forma adversária para construir imagens realistas de faces humanas. O gerador treinado tem em si a representação de um espaço latente multidimensional que contém todas as características do domínio das faces que foram usadas no treinamento.

Para que o crescimento progressivo cause transições menos bruscas no treinamento ao mudar a resolução da rede, foi proposta uma transição suavizada por um parâmetro α. Como pode ser visto na Figura 3.7, conforme α varia linearmente entre 0 e 1 o gerador

deixa gradualmente de usar a saída da camada anterior e passa a usar a saída da camada nova como referência para o discriminador.

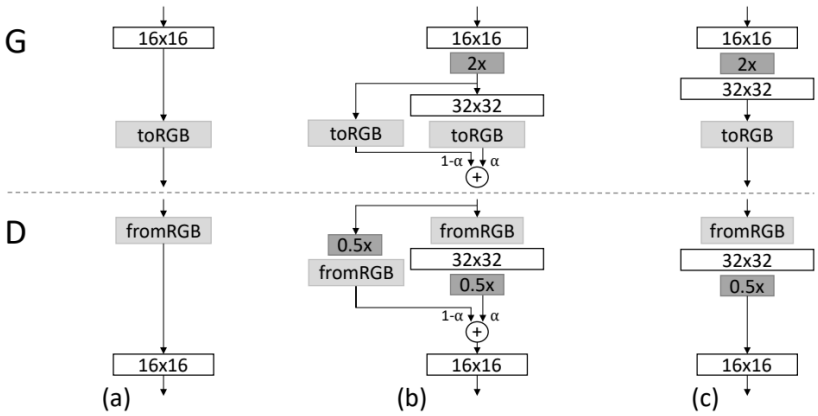

Figura 3.7: Mecanismo de transição da ProGAN. Fonte: [30]. Em (a) o gerador cria imagens 16x16 em RGB e manda para o discriminador. Em (b) uma nova camada 32x32 é criada, e a saída do gerador é uma interpolação entre a imagem 16x16 redimensionada e a imagem criada pela camada 32x32. Em (c), após a transição suavizada, a única saída do gerador é a criada pela camada 32x32.

Outras diferenças da ProGAN em relação a outras redes incluem o uso de PixelNorm como mecanismo de normalização, detalhado na Seção 2.7, e o uso de uma taxa de aprendizado equalizada, em que a atualização dos parâmetros da rede é normalizada através de todas as camadas, evitando que parâmetros oscilem muito durante o treinamento e permitindo que se ajustem mais rapidamente. A *loss* utilizada no treinamento da ProGAN é baseada na *loss* da GAN de Wasserstein com Penalidade de Gradiente, ou WGAN-GP [5, 22].

A ProGAN e a estratégia de treinamento progressivo foram uma inovação que permitiu o desenvolvimento de técnicas cada vez mais avançadas [33, 34, 31], mas que possuem um custo computacional muito elevado, tanto de memória quanto de processamento.

3.2.2 StyleGAN

A StyleGAN [33] contribuiu com algo inédito para a comunidade de *Deep Learning*: a capacidade de criar uma base de dados sintética e em alta definição de imagens de faces com grande diversidade de atributos desde idade e gênero até o uso de acessórios como óculos.

A arquitetura utilizada é similar à ProGAN, com adaptações no gerador. A Figura 3.8 compara o gerador da ProGAN com o gerador StyleGAN. Uma das principais alterações é a *mapping network*, uma rede que aumenta o nível de desemaranhamento do espaço latente. Os autores mostraram que antes de iniciar a construção da imagem pode-se passar o vetor z por um perceptron multicamadas para transformá-lo em um vetor $w \in \mathbb{R}^n$. A vantagem de se fazer isso é que o perceptron pode ser treinado junto com o gerador e, consequentemente, a rede aprende a construir o espaço latente intermediário W de forma que represente de forma desemaranhada as distribuições dos atributos das imagens da base de dados de treinamento.

Uma característica importante do espaço latente é o nível de emaranhamento (*entanglement*) das características. Ao se transitar por uma direção em um espaço muito emaranhado, várias características do produto final serão alteradas, ao passo que em um espaço muito desemaranhado ao se variar um atributo ao longo de uma direção específica do espaço, apenas uma característica é alterada. Um exemplo está representado na Figura 3.9, em que ao transitar pela linha laranja, altera-se a criação da imagem entre exemplos masculinos e femininos. Espaços latentes com alto nível de desemaranhamento são muito úteis para a síntese de imagens, pois torna possível a manipulação individual de cada característica da figura gerada [33, 79, 46].

Os atributos são linearmente separáveis no espaço latente W (Fig. 3.10). A consequência disso é que as regiões dos atributos são bem definidas, e pode-se tratar elas de forma

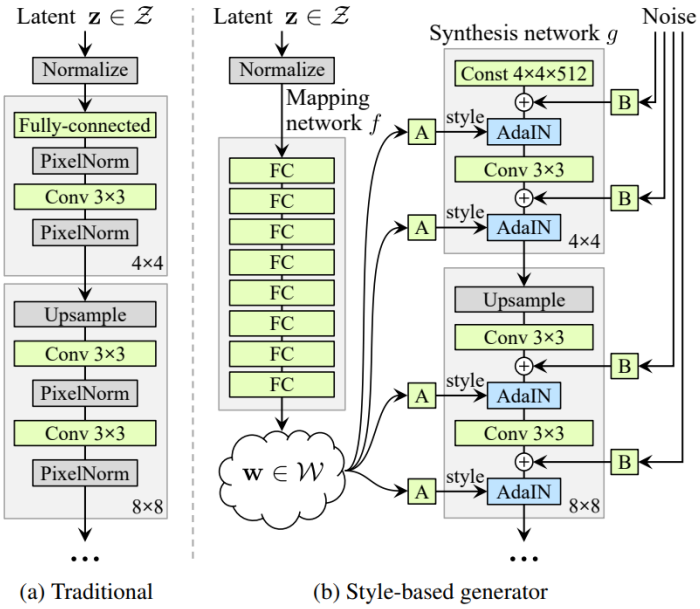

Figura 3.8: Arquitetura do gerador StyleGAN. Fonte [33]. A parte (a) mostra as primeiras camadas de um gerador ProGAN, e a parte (b) da figura mostra o gerador equivalente com arquitetura StyleGAN. Os blocos ”A”correspondem a transformações afim treináveis e os blocos ”B”aplicam transformações de escala treináveis na entrada de ruído.

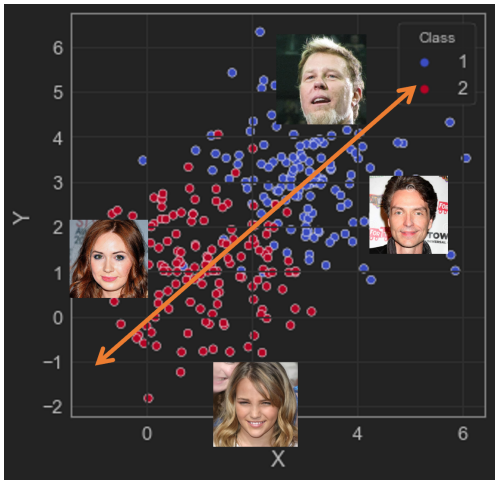

Figura 3.9: Exemplo de um espaço latente desemaranhado. Ao transitar pela linha laranja pode-se criar imagens de faces mais masculinas ou mais femininas. O espaço será considerado desemaranhado se ao alterar essa características, as outras se mantiverem inalteradas.

binária (ex. homem vs mulher, jovem vs velho) para poder manipular as características da face gerada, simplesmente fazendo transformações vetoriais dentro do espaço W [33, 79].

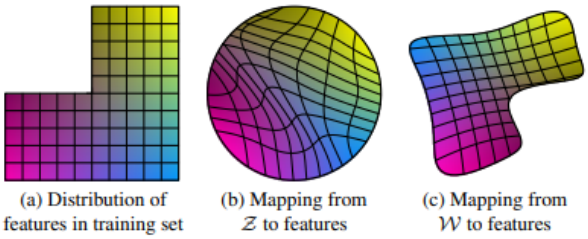

Figura 3.10: Efeito da *mapping network* da StyleGAN. Fonte: [33]. Em (a) há a representação da distribuição dos atributos na base de treino; em (b) o mapeamento a partir das características de Z; em (c) o mapeamento a partir das características de W. Verifica-se que há uma correspondência maior entre (a) e (c) do que entre (a) e (b).

Na ProGAN e em GANs clássicas, como a DCGAN, apenas a primeira camada recebe a informação original, que é propagada através dos *feature maps* das camadas de convolução. A StyleGAN, no entanto, parte de uma constante treinável e recebe o vetor w como uma entrada de estilo em uma camada de normalização AdaIN [26] presente duas vezes em cada resolução. Para que o vetor possa ser aplicado em diferentes resoluções ele passa antes por blocos treináveis de transformação afim, que ajustam a entrada para cada camada em que ela é aplicada [33].

Essa escolha de arquitetura dá o nome de *"Style"* para a StyleGAN, e permite que se faça mistura de estilos ao se mostrar vetores diferentes em camadas diferentes do gerador. Os autores mostraram que camadas de menor resolução recriam informações mais gerais como formato do rosto e pose, enquanto resoluções maiores sintetizam os detalhes e ajustes da imagem.

Antes de cada bloco AdaIN o gerador recebe uma entrada de ruído gaussiano fixo que é dimensionada para a resolução em que o bloco se encontra e somada à saída do bloco convolucional. Esse ruído insere informação que permite ao gerador criar detalhes estocásticos como fios de cabelo e barba, sardas e marcas no rosto. Com isso as imagens ficam perceptivelmente mais realistas e detalhadas.

A StyleGAN continuou sendo aprimorada nos últimos anos com novas versões como a StyleGAN2 [34], a StyleGAN2-ADA [31] e a StyleGAN3 [32]. A qualidade das imagens

geradas por esses métodos levanta uma preocupação sobre o uso indevido desse tipo de tecnologia para criar imagens e vídeos sintéticos utilizando a imagem de pessoas públicas em situações falsas, como as *deepfakes* [32]. Por fim, há uma forte limitação computacional no uso dessas técnicas, já que são necessários *clusters* de GPUs que somam juntos o equivalente anos de tempo de processamento e centenas de MWh de eletricidade para o treinamento de um modelo da StyleGAN3 [32].

3.2.3 *In Domain GAN Inversion*

Com base nos modelos não supervisionados de síntese de imagens como a StyleGAN ou a ProGAN, se torna possível manipular imagens alterando-se seu vetor latente. Apesar disso, uma das suas maiores limitações é que o método não nos diz como obter um vetor latente dada uma imagem qualquer do domínio.

Uma forma de se resolver essa questão é treinando *encoders* que obtém o vetor latente de uma imagem do domínio para o qual a GAN foi treinada, de forma que as características do vetor latente coincidam com as caracterísicas codificadas pelo espaço latente do gerador. Esse método se chama inversão de GAN intradomínio (*In-Domain GAN Inversion*) [79] e da forma que foi desenvolvido por Zhu *et. al.*, pode codificar imagens em vetores que funcionam corretamente como entrada de uma StyleGAN já treinada.

Por restringir o *encoder* a codificar uma imagem de forma consistente com o domínio do gerador pré-treinado, esse método permite aproveitar os benefícios do espaço latente desemaranhado da StyleGAN, e com isso torna possível manipular diversos atributos da imagem diretamente nesse espaço. A Figura 3.11 ilustra algumas aplicações, como a alteração de pose e expressão (a), interpolação contínua entre duas instâncias (b) e um tipo de manipulação que os autores chamaram de difusão semântica (c), em que se pode colocar a face de uma pessoa na cabeça de outra, por exemplo.

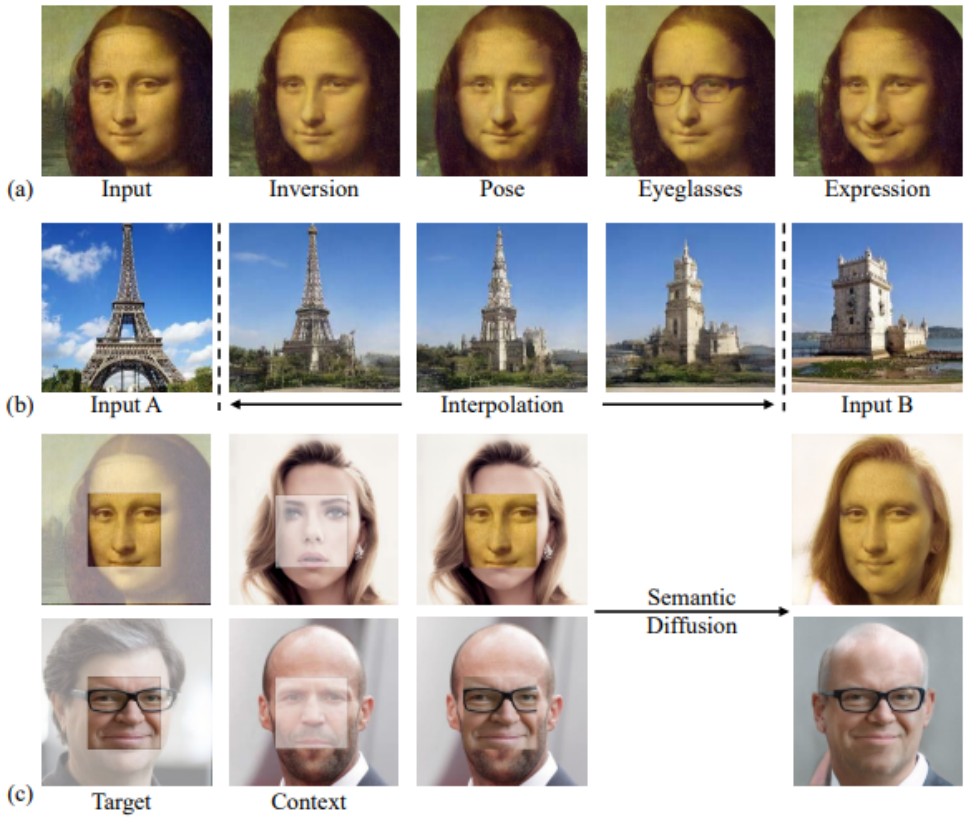

Figura 3.11: Aplicações da Inversão de GAN Intradomínio. Fonte: [79]. Em (a) a imagem "*Input*" é a entrada do *encoder* e a "*Inversion*" é a imagem criada pelo gerador a partir do vetor gerado pelo *encoder*. Na sequência, três manipulações feitas na imagem invertida, sendo elas a pose da face, a presença de óculos, e a expressão, respectivamente. Em (b) uma interpolação entre duas torres diferentes, de forma que as mudanças acontecem de forma linear. Em (c) um exemplo da difusão semântica em que a face de uma pessoa pode ser inserida no contexto da cabeça de outra e a rede gera uma saída compatível.

A parte superior da Figura 3.12 ilustra uma forma de se treinar *encoders* utilizando o gerador como referência. Nesse método um vetor $\mathbf{z}^{sam}$ passa pelo gerador, fornecendo a imagem $\mathbf{x}^{syn}$, e após passar essa imagem pelo *encoder*, obtem-se um vetor de saída $\mathbf{z}^{enc}$, e o *encoder* é treinado de forma a minimizar a diferença entre $\mathbf{z}^{syn}$ e $\mathbf{z}^{enc}$, representado na figura pela seta vermelha, enquanto o gerador se mantém fixo. Segundo os autores, apesar de coerente, esse método falha em reproduzir em $\mathbf{z}^{enc}$ as mesmas características que permitem a manipulação do espaço latente.

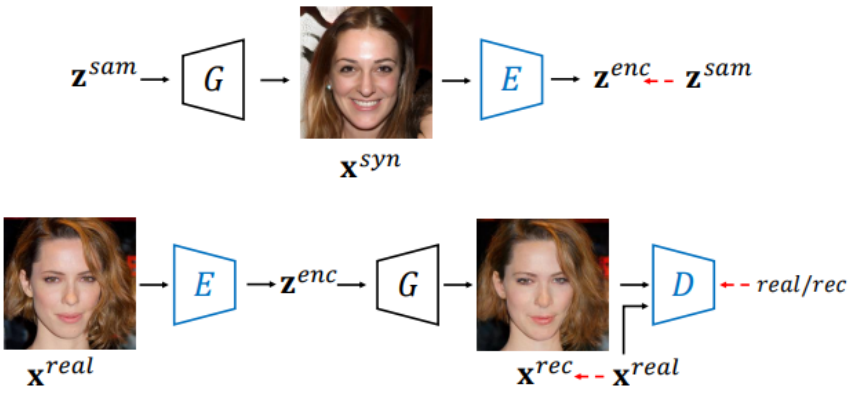

Figura 3.12: Arquitetura da Inversão de GAN Intradomínio. Fonte: [79].

Alternativamente eles propõem um *encoder* guiado pelo domínio, representado na parte inferior da Figura 3.12. Enquanto o *encoder* do método anterior é treinado com uma imagem sintética aleatoriamente criada pelo gerador, o método proposto utiliza imagens reais x^{real} como entrada do *encoder*, que gera um vetor z^{enc} que então é passado ao gerador. A saída reconstruída do gerador, x^{rec} é comparada com x^{real} utilizando-se o discriminador da GAN, cuja saída é usada para treinar o *encoder* e o discriminador, enquanto o gerador se mantém fixo. Segundo os autores um dos benefícios desse método está em usar ambos os componentes da GAN já treinada como "supervisores" do *encoder*, o que incentiva sua saída a produzir imagens coerentes com o domínio.

A *loss* do *encoder* ($\mathcal{L}_E$) é descrita por

$$\begin{aligned}\mathcal{L}_E = ||x^{real} - G(E(x^{real}))||_2 + \lambda_{per}||F(x^{real}) - F(G(E(x^{real})))||_2 \\ - \lambda_{gan}\mathbb{E}_{x^{real}\sim P_{data}}\left[D(G(E(x^{real})))\right],\end{aligned} \tag{3.6}$$

em que o primeiro termo corresponde à *loss* L2 entre a imagem real e a imagem reconstruída, o segundo termo diz respeito à *perceptual loss* [29] e $F(u)$ é o modelo de extração de atributos da VGG [66, 29], e o terceiro termo corresponde à *loss* adversária. λ_{per} e λ_{gan} ajustam os

pesos dos seus respectivos termos. A *loss* do discriminador ($\mathcal{L}_D$), por sua vez, é descrita pela expressão

$$\begin{aligned}\mathcal{L}_D = \mathbb{E}_{x^{real} \sim P_{data}} \left[D(G(E(x^{real}))) \right] - \mathbb{E}_{x^{real} \sim P_{data}} \left[D(x^{real}) \right] \\ + \frac{\gamma}{2} \mathbb{E}_{x^{real} \sim P_{data}} \left[||\nabla_x D(x^{real})||_2^2 \right],\end{aligned} \tag{3.7}$$

com o primeiro e segundo termos correspondendo à discriminação da imagem sintética e da imagem real, respectivamente, e o terceiro termo sendo uma regularização de gradiente controlada pelo hiperparâmetro γ.

3.3 Discussão

As arquiteturas Pix2Pix e CycleGAN apresentam duas formas de se realizar transformações de domínio, sendo a Pix2Pix supervisionada, usando imagens pareadas, e a CycleGAN usando uma abordagem não supervisionada. Nos próximos capítulos iremos avaliar o uso da Pix2Pix para a transformação de imagens de bordas de carros em imagens realistas. Na sequência usaremos a mesma base de dados com a arquitetura CycleGAN e compararemos os resultados. Também utilizaremos a CycleGAN com uma base de dados não pareada de esboços e desenhos finalizados de uma animação.

Por serem geradores capazes de criar imagens de boa qualidade, também exploraremos o uso de ambos os geradores numa abordagem de síntese.

4 TRANSFORMAÇÕES DE CONTORNOS EM IMAGENS

Como explicado na Seção 3.1, definimos as transformações de imagens como funções $G : A \rightarrow B$ que mapeiam imagens de um domínio A a uma equivalente no domínio B.

Uma aplicação de geração de imagens sintéticas que pode auxiliar artistas é a transformação dos esboços em imagens finalizadas [28]. Mesmo que a imagem gerada não seja perfeitamente similar a uma imagem natural, ou mesmo que a qualidade da finalização, da textura e da colorização sejam inferior ao trabalho do artista, um método que consiga construir uma imagem a partir do esboço de forma interativa pode ajudar no estágio de concepção e pré-produção do trabalho final.

Neste capítulo estudamos o desempenho das arquiteturas Pix2Pix e CyleGAN para a aplicação de transformação de esboços em imagens com maior complexidade de informação, tais como cor, textura e iluminação.

4.1 Metodologia

Apresentamos, na Seção 4.2, um estudo da utilização da Pix2Pix para a tarefa de reconstruir esboços de carros em imagens naturais (fotografias). Uma das restrições desse método é a necessidade de se utilizar bases de dados de imagens pareadas, ou seja, cada esboço deve ter seu carro correspondente. Na impossibilidade de encontrar uma base de

dados pareada de esboços e imagens naturais, criamos uma base sintética aplicando um detector de bordas [12] em uma base de dados de imagens de carros.

Essa base de dados está contaminada com imagens que não fazem parte do domínio, então um classificador binário de imagens foi treinado para limpar automaticamente a base, removendo os pares de imagens que não são coerentes com o domínio pretendido. Os resultados da aplicação da Pix2Pix com a base completa e com a base limpa são comparados.

Em seguida, na Seção 4.3, utilizamos a mesma base de dados de forma não pareada com a arquitetura CycleGAN. Seus resultados são então comparados com os obtidos na seção anterior, e também avaliamos o efeito da *loss* de consistência de ciclo na transformação da imagem de um domínio para o outro.

Na Seção 4.4, testamos a CycleGAN em outro contexto: a reconstrução de esboços de personagens humanoides em versões coloridas da mesma imagem. Com isso testamos a capacidade da rede de aprender a reconhecer os atributos semânticos da imagem (tais como olhos, cabeça, torso, roupas, e outros) e a reconstruí-los de forma consistente.

Ao final apresentamos uma discussão dos resultados e próximos passos. Alguns exemplos de imagens geradas pelos melhores experimentos estão presentes no Apêndice A.

4.1.1 Arquitetura e parâmetros

Os geradores utilizados nos experimentos são os mesmos citados pelos artigos originais. A Figura 4.1 ilustra as três diferentes arquiteturas:

- U-Net [56], como usado por Isola et al. [28];
- Gerador Pix2Pix, que é baseado no gerador U-Net e usado por Isola et al. [28];
- Gerador CycleGAN (ou gerador residual) como usado em Zhu et al. [80]. Nos experimentos a versão de seis blocos residuais será usada.

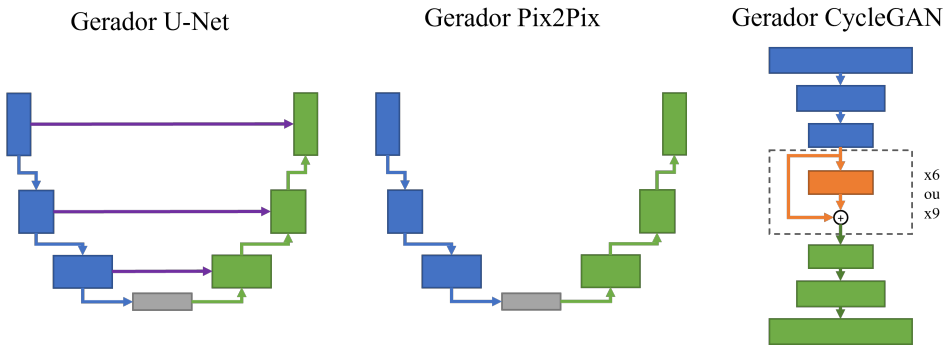

Figura 4.1: Geradores utilizados na na transformação de imagens. O gerador U-Net [56] é usado nos artigos originais tanto da Pix2Pix quanto da CycleGAN. O gerador Pix2Pix foi uma adaptação sugerida em [28] e o gerador CycleGAN [80], inspirado na ResNet, usa 6 ou 9 blocos residuais (em laranja) na sua construção. Os blocos azuis indicam as camadas do *encoder* e os blocos verdes indicam as camadas do *decoder*.

Exceto quando explicitado no experimento, as configurações utilizadas foram as seguintes:

- Quantidade de épocas de treinamento:
  - Pix2Pix com base de carros: 10 épocas;
  - CycleGAN com base de carros: 5 épocas;
  - CycleGAN com base de personagens: 20 épocas.
- Discriminador PatchGAN [28]
- *Losses* como definidas nos artigos originais:
  - Pix2Pix: Loss adversária + L1;
  - CycleGAN: Loss adversária + Loss de ciclo + Loss de identidade;
- Otimizador Adam com *Learning Rate* $\alpha = 1e-4$ e $\beta_1 = 0.5$;
- Na Pix2Pix, o parâmetro $\lambda = 100$ controla o efeito da *loss* L1;
- Na CycleGAN, o parâmetro $\gamma = 10$ controla o efeito das *losses* de ciclo e identidade;
- Uso de random jitter [28] nas imagens
- Batch sizes conforme tabela; 4.1, para usar o máximo possível da memória da GPU.

Arquitetura	**UNet**	**Gerador Pix2Pix**	**Gerador CycleGAN**
Pix2Pix	12	12	6
CycleGAN	4	-	3

Tabela 4.1: Valores de *batch size* utilizados a depender da combinação de arquitetura e gerador

4.1.2 Métricas avaliadas

Durante o treinamento, avaliamos as *losses* da arquitetura para a base de treinamento a cada iteração e para a base de validação a cada 10 iterações.

Ao final de cada época, para uma amostra da base de dados de validação, a métrica FID [24] é avaliada para ambas as arquiteturas, e a distância L1 [28] entre a imagem gerada e o objetivo para a arquitetura Pix2Pix, já que a CycleGAN não tem um objetivo, por ser não supervisionada.

Ao final do treinamento, todas as imagens da base de dados de teste serão geradas e o tempo médio de inferência (de geração) das imagens será medido. Além disso, as métricas de qualidade (FID e L1, quando aplicável) serão avaliadas para todas as imagens da base de dados de teste.

4.1.3 Ambiente de experimentação

Todos os experimentos foram executados em um computador com Windows 10, processador Intel Core i7 9700K, 32 GB de memória RAM, e uma GPU Nvidia RTX 2070 Super com 8GB de VRAM.

O ambiente utilizado foi o Python v3.8.8 com *framework* Tensorflow [1] v2.7.0, e Nvidia CUDA 11.4.

Os códigos correspondentes aos experimentos estão disponíveis no repositório do projeto no GitHub[1].

4.2 Aplicação da Pix2Pix na base de dados de carros e esboços

O objetivo deste experimento é utilizar a arquitetura Pix2Pix para sintetizar imagens naturais de algum objeto a partir de esboços desse mesmo objeto. Como é muito difícil encontrar bases de dados de correspondência entre esboços e imagens naturais, será necessário criar sinteticamente uma base pareada.

4.2.1 Criação da base de dados de carros e esboços

A base de imagens pareadas foi criada a partir da *60,000+ Images of Cars*[2] que possui aproximadamente 64,4 mil imagens de interiores e exteriores de carros em diferentes ângulos. A Figura 4.2 contém exemplos das imagens originais da base.

Além das imagens de carros, é necessário ter uma base de dados no domínio dos esboços, e ao utilizar métodos supervisionados é importante que as imagens de ambos os domínios sejam pareadas, de forma que uma imagem em A tenha um par equivalente em B, portanto as imagens de carros foram processadas por um detector de bordas [12] de forma que cada imagem de carro tivesse uma imagem de bordas equivalente.

Nesse contexto, as imagens de esboços (bordas) de carros serão o domínio A e as imagens reais de carros serão o domínio B. As 64.467 imagens foram separadas aleatoriamente de forma que 15% dos pares fossem reservados para a validação e 5% para teste.

1 https://github.com/vinyluis/Pix2Pix-CycleGAN
2 obtida em https://www.kaggle.com/prondeau/the-car-connection-picture-dataset

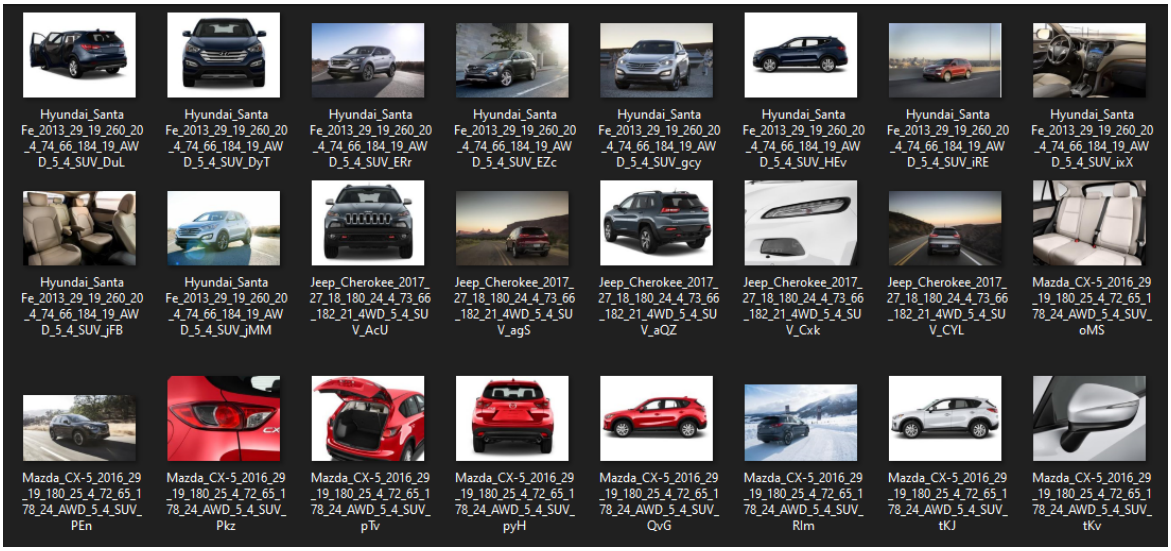

Figura 4.2: Exemplo da base de dados de carros

Baseado na arquitetura original da Pix2Pix, e por restrições de memória do equipamento utilizado, as imagens utilizadas serão nas dimensões 256 × 256, e portanto no pré-processamento todas elas passam por um corte quadrado central seguido por um redimensionamento, o que garante a dimensão correta.

4.2.2 Comparação do uso de diferentes geradores

Utilizando a arquitetura Pix2Pix com os geradores U-Net, Pix2Pix e CycleGAN, a rede foi treinada na base de dados de carros e esboços por 10 épocas com a base de dados de treinamento. Assim como no caso original, foi utilizado *random jitter* e espelhamento das imagens durante o treinamento [28].

Os gráficos da Figura 4.3 representam a *loss* total do gerador, conforme Eq. 3.1, calculada a cada iteração na base de dados de treinamento e a cada 10 iterações na base de validação, para os três diferentes geradores. Através desses gráficos é possível verificar que o gerador CycleGAN apresentou a maior redução de *loss* dentre os três geradores, sendo seguido de perto pelo U-Net. O gerador Pix2Pix teve um desempenho muito inferior

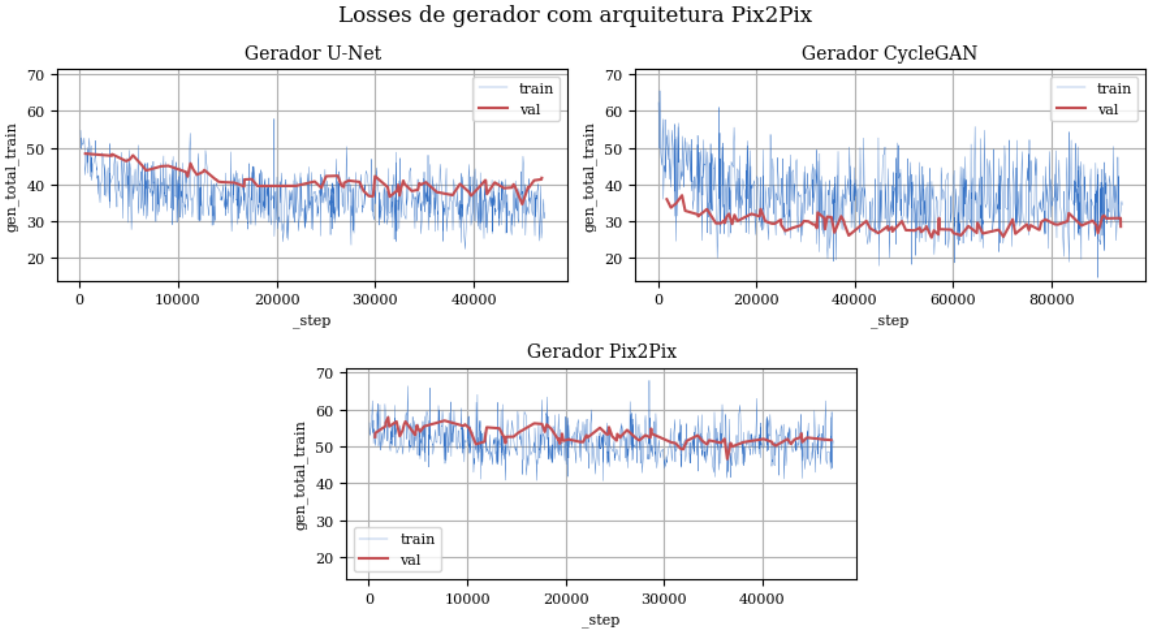

Figura 4.3: *Losses* de gerador com arquitetura Pix2Pix. A linha azul representa a *loss* total calculada para a base de treinamento e a linha vermelha representa a *loss* total calculada na base de validação.

durante o treinamento. Em nenhum dos três casos há ocorrência relevante de sobreajuste (*overfitting*), em que as métricas de validação estariam muito descoladas das de treinamento.

Em relação às métricas de qualidade calculadas na base de validação a cada época e ilustradas na Figura 4.4, as imagens geradas pelo gerador Pix2Pix foram tão inferiores que não permitiram o cálculo correto das métricas L1 (omitida nesse caso) e FID (representado no gráfico pela linha vermelha). Apesar disso, é possível verificar que os outros geradores conseguiram melhorar ambas as métricas conforme o treinamento avançava.

Ao final do treinamento, avaliamos ambas as métricas para os três geradores usando a base de dados de teste, e apresentamos esses resultados na Tabela 4.2. Na métrica FID o gerador CycleGAN apresentou o melhor resultado, embora a métrica L1 tenha apresentado resultado bastante próximo do gerador U-Net.

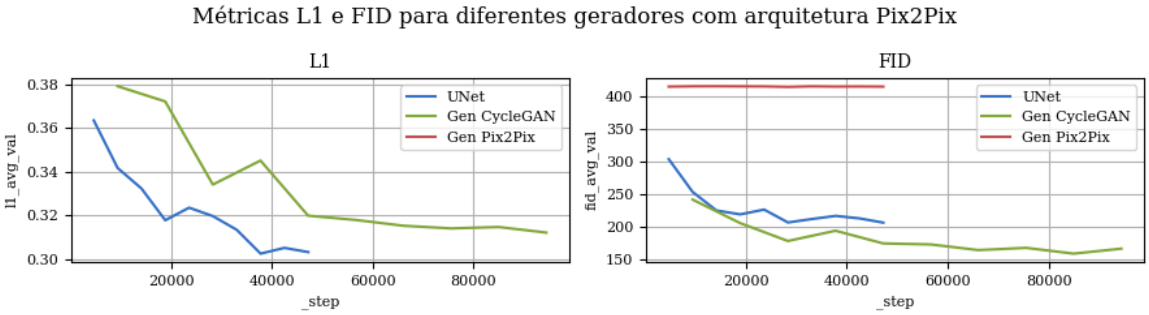

Figura 4.4: Métricas de qualidade para diferentes geradores com arquitetura Pix2Pix. O gerador Pix2Pix não conseguiu gerar imagens com qualidade suficiente para o cálculo correto das métricas L1 e FID.

Em todos os casos o tempo que o gerador demorou para gerar uma imagem (tempo de inferência) foi da ordem de décimos de segundos, o que permite seu uso em aplicações interativas. O *runtime*, que é o tempo de total do experimento, variou de 4 a 12 horas.

Experimento	Gerador	FID Teste	L1 Teste	Tempo inferência (s)	*Runtime* (h)
P01A	U-Net	214,31	**0,30**	0,14	6,53
P01B	CycleGAN	**179,84**	0,31	0,13	12,31
P01C	Pix2Pix	-	-	0,13	4,27

Tabela 4.2: Resultados dos experimentos com o uso de diferentes geradores na arquitetura Pix2Pix

Alguns exemplos de imagens geradas pelo gerador treinado no experimento P01B, a partir da base de dados de teste, podem ser observados na Figura 4.5, demonstrando uma boa capacidade de reconstrução da imagem final, inclusive com o plano de fundo.

A rede consegue, de forma autônoma, distinguir a estrutura do carro de suas rodas, calotas, vidros e inclusive a paisagem ao fundo. Previsivelmente a rede não acerta a cor da pintura do carro original porque nenhuma informação de cor foi passada juntamente com o esboço, o que permite que o gerador inclua uma cor compatível com o que a rede tenha visto em outros exemplos. Entretanto, artefatos que mantém a mesma cor em todas as imagens, como rodas e lanternas, são coloridos de forma consistente.

Apesar do bom resultado, na Figura 4.6 é possível visualizar que alguns artefatos foram reconstruídos de forma errada, e um dos motivos para isso pode ter sido a presença

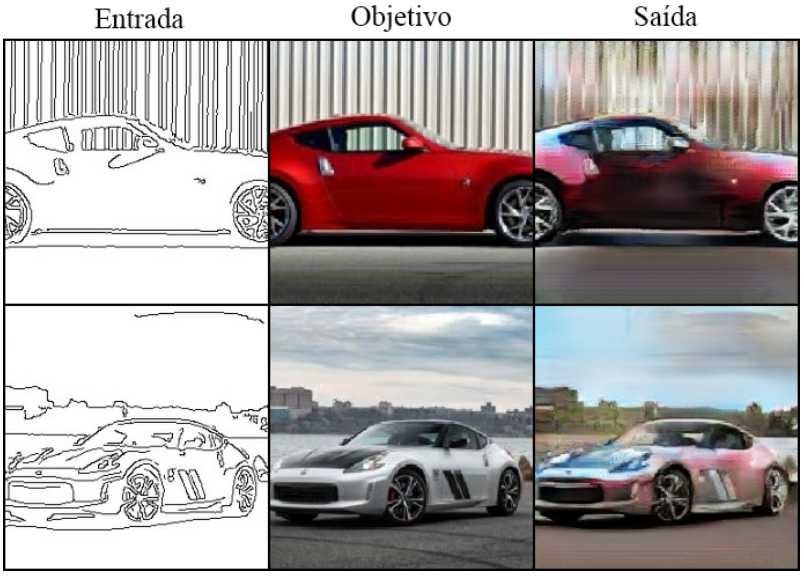

Figura 4.5: Primeiros resultados da aplicação da Pix2Pix na base de dados de imagens e esboços de carros. A coluna da esquerda contém os esboços enviados ao gerador, na coluna do meio estão os objetivos do gerador para cada entrada, e a coluna da direita mostra as imagens sintéticas geradas. Cada linha representa um exemplo transformado.

de imagens que não fazem parte do domínio, como pessoas, a parte interior dos carros ou até uma captura de uma página de internet com uma foto de carro.

Figura 4.6: Exemplos de atributos reconstruídos de forma errada, como os homens ou a página escrita, que não fazem parte de nenhum dos domínios.

4.2.3 Limpeza da base de dados e retreino

Os resultados anteriores mostraram a necessidade de se limpar a base de dados, incluindo o máximo possível de imagens que mostram apenas o exterior dos carros, e remover imagens de outros contextos, como painéis, bancos, interiores, ou até imagens externas como lanternas e capôs sem o contexto completo.

Com esse intuito, criamos com o *framework* TensorFlow [1] um classificador binário simples baseado em CNN (Fig. 4.7), que foi treinado para separar automaticamente as imagens de exteriores de carros (`car`) das outras imagens (`notcar`).

Um total de 600 exemplos de imagens da classe `car` e outros 600 exemplos de imagens da classe `notcar` foram selecionados manualmente para criar uma base de dados de exemplos de imagens de carro × não-carro.

As imagens foram separadas em dois conjuntos correspondendo às suas classes, e em seguida o classificador foi treinado por 50 épocas com *batches* de 5 imagens para aprender a classificar corretamente a base de dados. 20% das imagens foram usadas como validação e o classificador obteve uma acurácia de 93.33%.

Na arquitetura do classificador, as camadas `MaxPooling` [64] foram usadas para reduzir as dimensões das imagens de modo a conseguir um treinamento mais rápido sem prejudicar a acurácia. Já as camadas de `BatchNormalization` ajudam na convergência da rede [27, 62]. As camadas de convolução bidimensionais (`Conv2D`) aprendem a reconhecer as características das imagens e em seguida as camadas MLP completamente conectadas (`Dense`) realizam a classificação entre `car` (0) e `notcar` (1). A ativação das camadas de convolução e MLP foi feita pela função `ReLU`, exceto a última camada que usou uma ativação sigmoide para manter a saída dentro do intervalo $[0,1]$. A imagem era classificada como `car` se a predição p for $p > 0,5$ e classificada como `notcar` caso contrário.

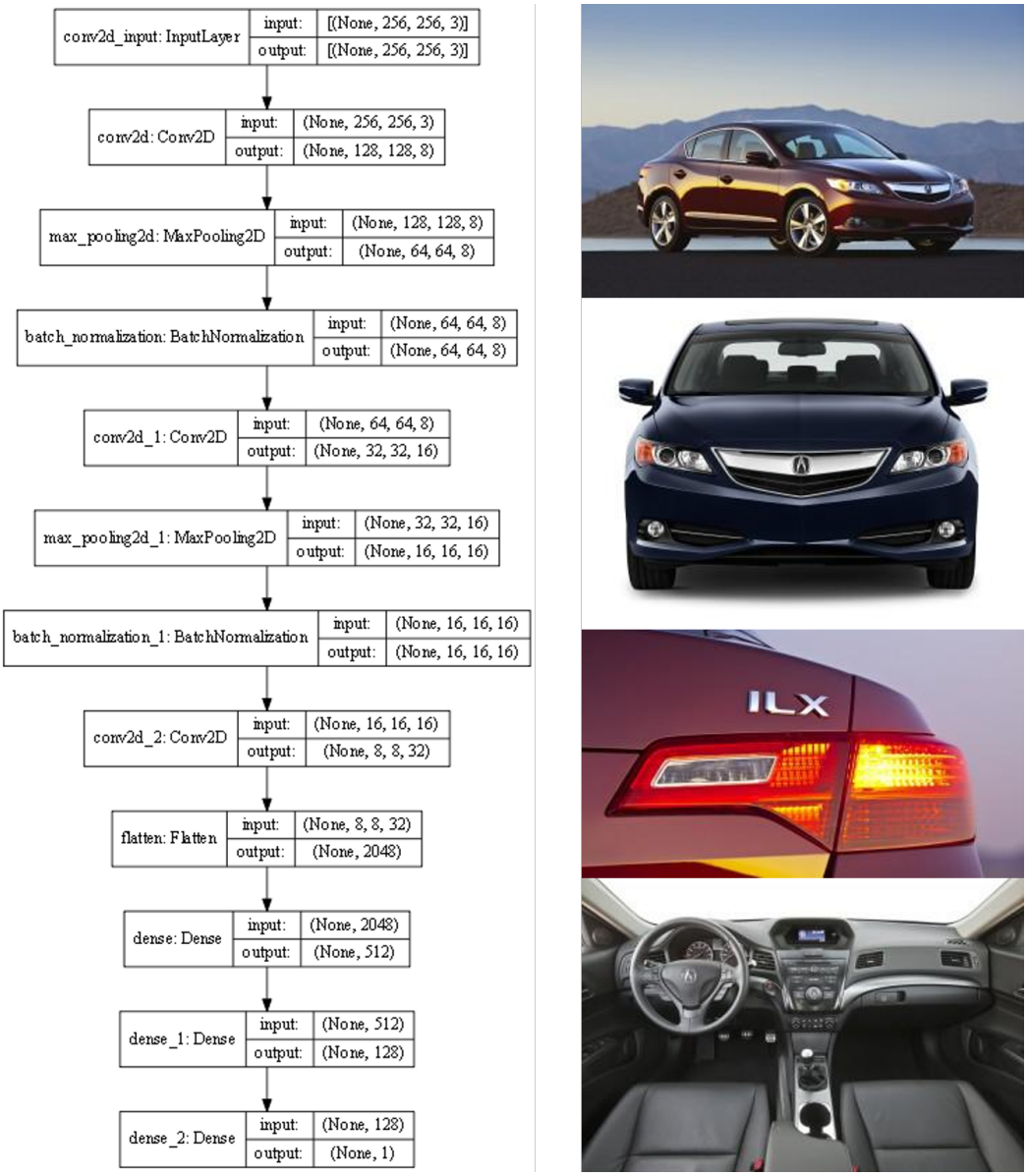

Figura 4.7: Arquitetura do classificador binário de carros. As duas imagens superiores são exemplos da Classe `car` e as duas imagens inferiores são exemplos da Classe `notcar`.

O otimizador utilizado foi um `ADAM` com taxa de aprendizado $\alpha = 0.0002$ e $\beta_1 = 0,5$, e a *loss* utilizada foi a *Binary Cross-Entropy*, por se tratar de um classificador binário.

Aplicando a classificação às mais de 64 mil imagens da base original, foram obtidas cercas de 40 mil imagens de carros, com uma quantidade muito menor de imagens da classe `notcar`.

Os três experimentos anteriores foram repetidos com essa nova base de dados, e comparando o comportamento das *losses* de gerador ao longo do treinamento destes experimentos, ilustradas na Fig. 4.8 com os anteriores, na Fig. 4.3, é possível verificar que as redes treinadas com a base limpa e reduzida conseguiram alcançar patamares menores de *loss*, o que pode indicar melhor qualidade nas imagens geradas. O mesmo ocorre com os resultados das métricas de qualidade FID e L1 (Fig. 4.9), que são claramente otimizados ao longo do treinamento e alcançam patamares menores (melhores) do que anteriormente.

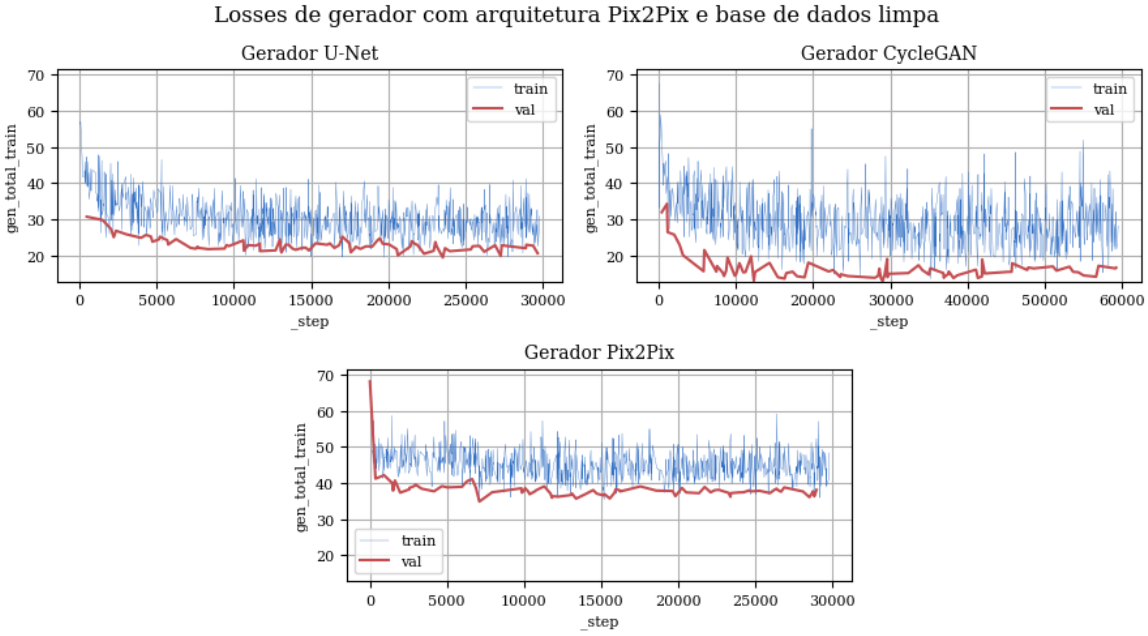

Figura 4.8: *Losses* de gerador com arquitetura Pix2Pix e base de dados limpa. A linha azul representa a *loss* total calculada para a base de treinamento e a linha vermelha representa a *loss* total calculada na base de validação.

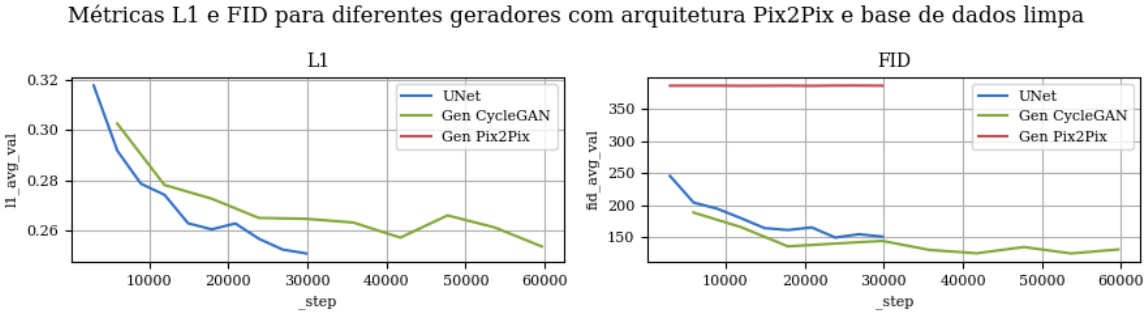

Figura 4.9: Métricas de qualidade para arquitetura Pix2Pix e base de dados limpa. O gerador Pix2Pix não conseguiu gerar imagens com qualidade suficiente para o cálculo correto das métricas L1 e FID.

Essa conclusão é reforçada pelas métricas de qualidade, presentes na Tabela 4.3. Os valores de FID e L1 dos experimentos P02A e P02B superaram seus antecessores P01A e P01B, respectivamente. Assim como no caso anterior, o gerador CycleGAN obteve os melhores resultados com um FID significativamente melhor e um L1 quase igual ao do U-Net.

Experimento	Gerador	FID Teste	L1 Teste	Tempo inferência (s)	*Runtime* (h)	Base
P01A	U-Net	214,31	0,30	0,14	6,53	Completa
P01B	CycleGAN	179,84	0,31	0,13	12,31	Completa
P01C	Pix2Pix	-	-	0,13	4,27	Completa
P02A	U-Net	140,40	**0,22**	0,14	4,06	Limpa
P02B	CycleGAN	**119,44**	0,23	0,13	7,85	Limpa
P02C	Pix2Pix	385,13	-	0,13	2,73	Limpa

Tabela 4.3: Comparação dos resultados dos experimentos da arquitetura Pix2Pix. Os experimentos que iniciam com P01 usam a base de dados completa, e os que iniciam com P02 usam a base de dados reduzida (limpa).

As imagens geradas mantiveram as características presentes nos resultados dos experimentos anteriores, como ilustrado pela Figura 4.10, que mostra imagens da base de dados de teste geradas pelo gerador da rede P02B.

4.2.4 Teste de generalização da Pix2Pix

Em aprendizado de máquina, generalização é a capacidade de um modelo de funcionar o mais corretamente possível com entradas que nunca tenham sido vistas durante o treinamento.

Com o objetivo de testar o poder de generalização da Pix2Pix treinada com os pares esboço-carro, foram obtidos do Google Imagens alguns exemplos de imagens de esboços de carros de dois tipos: com e sem hachuras, como na Figura 4.11. As imagens foram preparadas através da sequência abaixo:

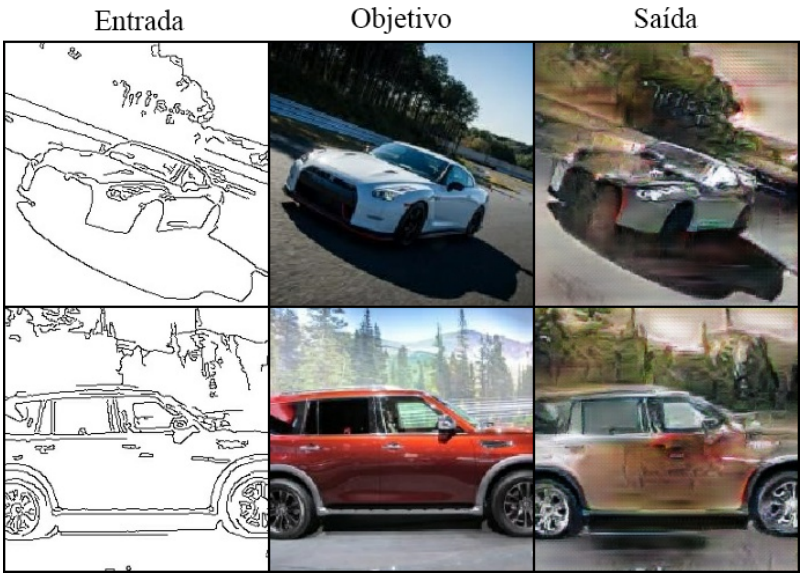

Figura 4.10: Resultados da aplicação da Pix2Pix na base de dados limpa. A coluna da esquerda contém os esboços enviados ao gerador, na coluna do meio estão os objetivos do gerador para cada entrada, e a coluna da direita mostra as imagens sintéticas geradas. Cada linha representa um exemplo transformado.

- Transformadas em escala de cinza, para evitar ruídos de compressão e pixels coloridos;
- Normalizadas, para que os valores dos pixels ficassem entre o intervalo $[-1, 1]$, como deve ser a entrada do gerador;
- Redimensionadas mantendo a razão de aspecto, de forma que a aresta maior da imagem correspondesse a 256, e o restante fosse preenchido com pixels brancos para a imagem ficar com as dimensões 256×256;
- Convertidas novamente para RGB, apenas para manterem a estrutura de três canais, necessária para a Pix2Pix.

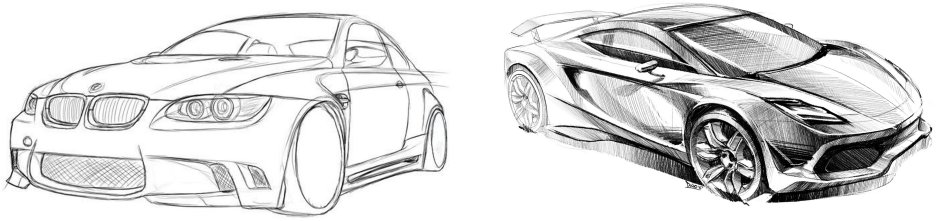

Figura 4.11: Exemplos de imagens de esboços de carros para validação. Fontes [48, 41].

A rede treinada no experimento P02B foi então utilizada para sintetizar imagens reais a partir desses esboços, e o resultado foi muito inferior ao obtido anteriormente, como

mostra a Figura 4.12. Já era esperado que a GAN não fosse capaz de generalizar bem para imagens com hachura, mas surpreendentemente ela teve um desempenho abaixo do esperado inclusive para os esboços de contornos sem hachura.

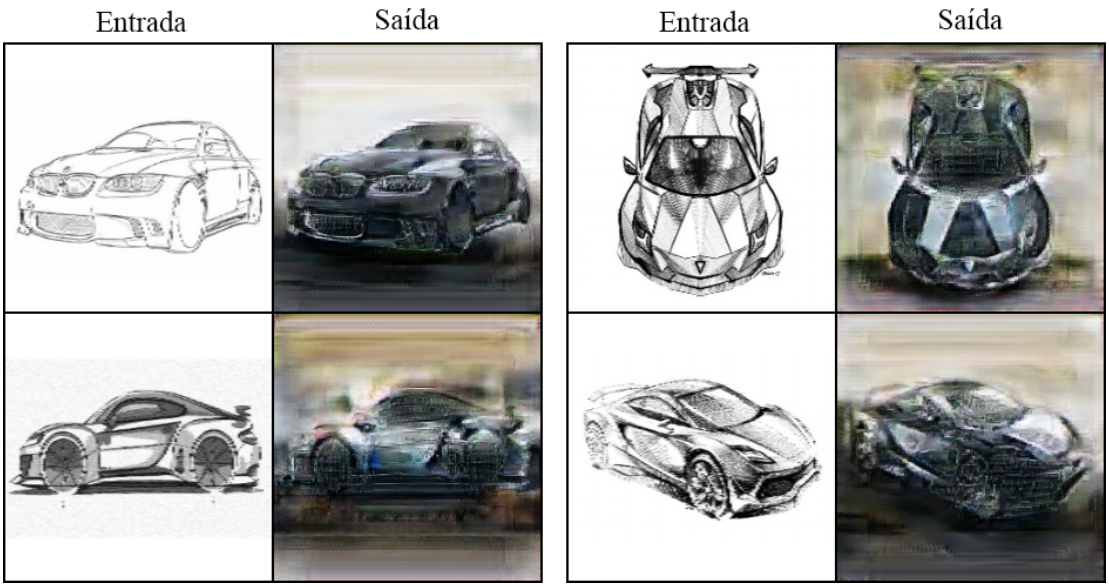

Figura 4.12: Teste de generalização da Pix2Pix com imagens de carros. Fontes [48, 41].

Pelo fato de a base de dados ser pequena, com 14 imagens, as métricas FID e L1 não seriam comparáveis com as métricas calculadas nos experimentos, com centenas ou até milhares de imagens. Por esse motivo, essas métricas não foram calculadas para o teste de generalização. Apesar disso, o resultado visual já mostra que a qualidade é, de fato, inferior ao do experimento com bordas.

4.2.5 Discussão

A arquitetura Pix2Pix se mostrou muito competente na tarefa de transformar bordas de carros em uma imagem final, inclusive com textura, iluminação e cor. Entretanto, ainda falta realismo nas imagens geradas, que são facilmente percebidas como sintéticas. Isso é um ponto que pode ser trabalhado futuramente.

A base de dados com menos imagens intrusas, e com um domínio mais bem definido (exteriores de carros) permitiu que as redes treinadas tivessem um desempenho melhor, o que fortalece um indício de que a qualidade da base de dados é superior à bruta quantidade de dados para essa tarefa. Isso leva a crer que se houvesse uma base de dados pareada com imagens de carros e seus respectivos esboços artísticos, e que fosse suficientemente ampla, a qualidade alcançada poderia ser melhor do que a obtida com estes experimentos.

De acordo com o comportamento das *losses* durante o treinamento, não há motivos para acreditar que mais épocas de treinamento levariam a um resultado substancialmente melhor do que o encontrado, já que os gráficos indicam que a tendência da *loss* estabilizou e não seria reduzida de forma consistente com mais épocas. Outro problema que isso poderia causar é o sobreajuste (*overfitting*). A decisão de se treinar os modelos por 10 épocas veio a partir dessas observações e da limitação de tempo e recursos computacionais para a execução dos experimentos.

Apesar dos pontos citados, as imagens geradas ainda não são confundidas com imagens reais, e a capacidade de generalização das redes foi fraca. A respeito disso, levantamos duas hipóteses:

1. A arquitetura Pix2Pix não é suficientemente capaz de trabalhar com generalizações para entradas de esboços / bordas. Essa hipótese será testada na Seção 4.3.
2. Há um problema de domínio, e embora para seres humanos não haja uma diferença tão grande entre bordas detectadas automaticamente e um esboço de um contorno de um carro, para a rede essa diferença é suficientemente grande a ponto de impedir a GAN de mapear corretamente entrada de um esboço o novo domínio. Essa hipótese será avaliada na Seção 4.4.

4.3 Aplicação da CycleGAN na base de dados de carros e esboços

A CycleGAN [80] é essencialmente composta por duas redes Pix2Pix que compartilham uma *loss* de consistência de ciclo. Por ser um método não supervisionado, ela adapta o discriminador de forma que não seja condicional, dispensando o uso de imagens pareadas.

Com essa arquitetura, uma base de dados de imagens pareadas com 40 mil pares pode ser usada como duas bases de 40 mil exemplos cada. Assim, os pares da base de dados limpa foram separados e as imagens foram divididas de forma que 15% dos exemplos fossem usados para validação, 5% para teste e o restante para o treinamento da rede.

Nos experimentos desta seção, o domínio A é composto pelas imagens reais de carros e o domínio B pelas imagens de bordas (esboços). O gerador $G : A \rightarrow B$ realiza a transformação carro→esboço, e o gerador $F : B \rightarrow A$ realiza a transformação esboço→carro. O discriminador D_A discrimina as imagens reais e sintéticas do domínio dos carros enquanto o discriminador D_B atua no domínio dos esboços.

Ao final, comparamos os resultados obtidos por essa abordagem com os resultados da Pix2Pix.

4.3.1 Efeito da *loss* de ciclo nas transformações da base de carros

O uso da *loss* de consistência de ciclo é o que permite à CycleGAN ser treinada de forma não supervisionada e dispensar o uso de imagens pareadas. Como consequência, o parâmetro γ, que regula a importância relativa da *loss* de ciclo com a *loss* adversária (Eq. 3.5), é ajustado experimentalmente de acordo com o domínio da aplicação.

Os geradores escolhidos para os experimentos são os usados pelos autores do artigo original [80], o U-Net e o que chamamos informalmente de "Gerador CycleGAN", que tem duas versões: o de seis e o de nove blocos residuais. Por limitações de memória VRAM não foi possível realizar o experimento com o Gerador CycleGAN em nenhuma das suas versões, mesmo se usando *batch size* unitário, portanto apenas o U-Net será utilizado.

A rede foi treinada três vezes, com valores distintos do parâmetro γ. A evolução da *loss* completa dos geradores, conforme Eq. 3.5, pode ser verificada nos gráficos da Figura 4.13. O gerador G apresentou poucas mudanças em seu comportamento ao variar γ de 5 para 10, mas sua *loss* ficou num patamar um pouco mais alto com $\gamma = 20$. A diferença no gerador F é mais visível, apresentando uma *loss* num patamar maior conforme o valor de γ foi aumentando. Apesar do comportamento similar, as *losses* do gerador F são de uma escala consideravelmente menor do que as do gerador G.

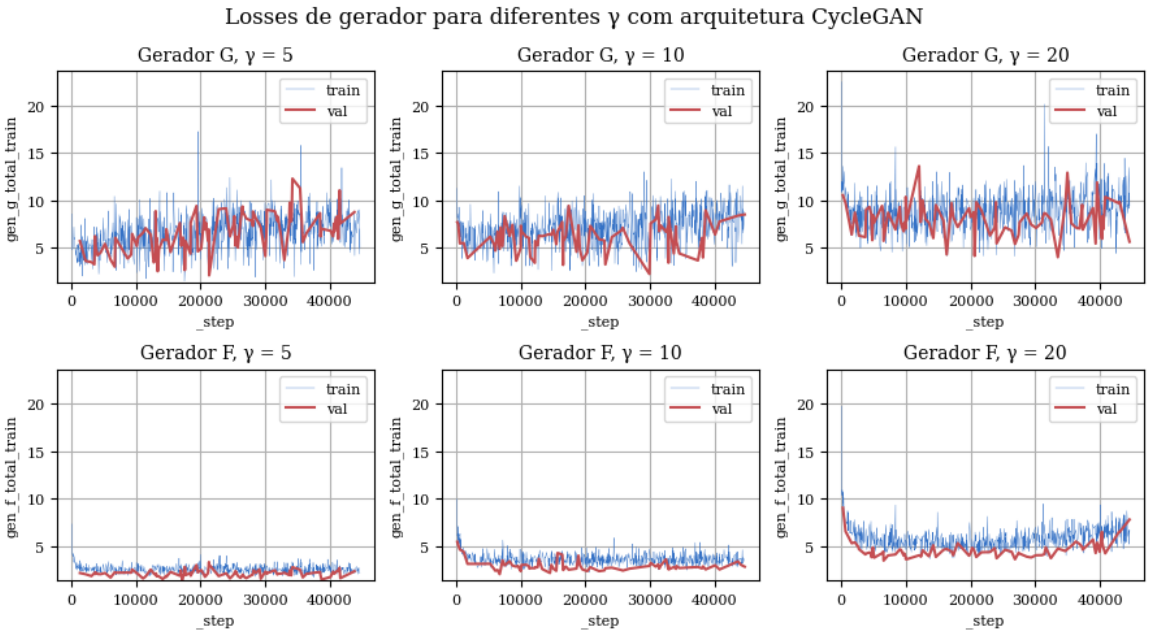

Figura 4.13: *Losses* de gerador com arquitetura CycleGAN e diferentes valores de γ. A linha azul representa a *loss* total calculada para a base de treinamento e a linha vermelha representa a *loss* total calculada na base de validação. O gerador $G : A \rightarrow B$ realiza a transformação carro→esboço, e o gerador $F : B \rightarrow A$ realiza a transformação esboço→carro.

A evolução da FID, calculada na base de dados de validação e exibida nos gráficos da Fig. 4.14, mostra que ambos os geradores tiveram um comportamento bem parecido nos três experimentos, mas ao passo que G quase não teve evolução, o F melhorou muito ao longo das épocas de treinamento.

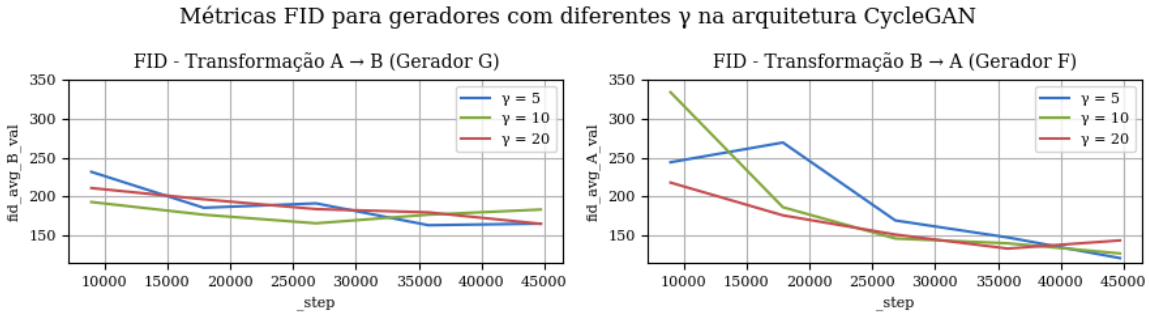

Figura 4.14: Métricas de qualidade para arquitetura CycleGAN e diferentes valores de γ. A métrica FID A representa a diferença entre imagens reais e sintéticas do domínio de carros e a métrica FID B representa a diferença entre imagens reais e sintéticas do domínio de esboços.

Por conta da abordagem não supervisionada, a CycleGAN não tem uma imagem objetivo, o que impede o uso da métrica L1 da forma como foi usada na Pix2Pix. No entanto, ao se realizar a transformação de uma imagem de A para B e transformar o resultado de volta para A, completamos um ciclo da imagem. A diferença entre a imagem original e a imagem sintetizada através do ciclo completo pode indicar o quanto a rede preserva as características originais da imagem ao realizar as transformações. Essa métrica, juntamente com a FID de ambos os domínios, estão na Tabela 4.4.

Experimento	Gama (γ)	Tempo infer. A (s)	Tempo infer. B (s)	*Runtime* (h)
C01A	5	0,1102	0,1135	7,9
C01B	10	0,1076	0,1075	8,3
C01C	20	0,1079	0,1085	8,2

Experimento	FID A	FID B	L1 A-B-A	L1 B-A-B
C01A	**115,6354**	**166,5402**	0,2387	0,1378
C01B	118,1349	182,6977	0,2082	**0,1298**
C01C	132,1669	166,9382	**0,1758**	0,1303

Tabela 4.4: Comparação dos resultados dos experimentos da arquitetura CycleGAN ao se variar o parâmetro γ.

Ao se comparar as métricas FID calculadas para a base de dados de teste completa, o experimento C01A com $\gamma = 5$ mostrou os melhores resultados em ambos os domínios, porém, o mesmo experimento apresentou os piores valores de L1 ao se realizar um ciclo. Esse comportamento é esperado pela forma como a *loss* de consistência de ciclo age, com valores menores de γ atribuindo menor importância para o ciclo (impactando negativamente a L1) e maior importância para a componente adversária (impactando positivamente a FID).

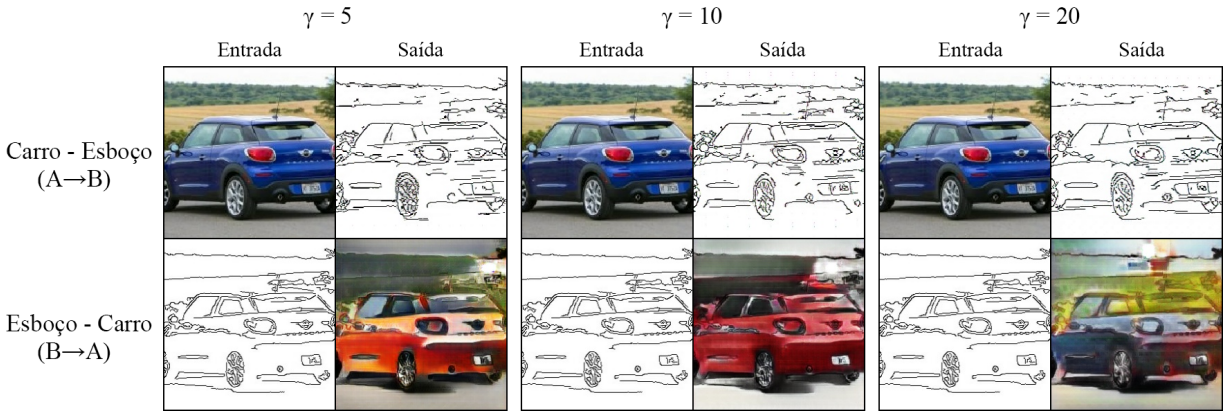

Figura 4.15: Efeito da *loss* de ciclo na arquitetura CycleGAN com a base de carros. Imagens com menores valores de γ implicam em uma menor importância da *loss* de ciclo em relação à *loss* adversária.

Através da Figura 4.15, é possível comparar esse efeito em uma mesma imagem passando por ambos os processos de transformação de domínio com os três diferentes valores de γ experimentados. Com menor importância na *loss* de ciclo, atribuída por menores valores de γ, a rede tende a produzir imagens mais gerais de carro, sem se importar muito com os detalhes finos, enquanto o experimento com a maior importância no ciclo cria um carro pouco realista, mas que preserva maior parte dos traços.

Outro efeito importante notado é a aparente assimetria entre a dificuldade das tarefas, pois a transformação $A \rightarrow B$ produz resultados satisfatórios independentemente do γ escolhido, o que pode indicar que essa é uma tarefa mais simples de ser ensinada à rede do que a transformação $B \rightarrow A$.

4.3.2 Teste de ciclo e de generalização com a base de carros

Na aplicação da CycleGAN, a métrica L1 de um ciclo foi usada para avaliar o efeito da *loss* de consistência de ciclo nas imagens sintéticas. Este conceito pode ser expandido, de forma que uma imagem pode passar por vários ciclos para verificar como o erro de reconstrução se acumula. Quanto menos erros acontecerem durante a ciclagem, mais robusta é a transformação e mais verdadeira é a afirmação de que os geradores $G_A \approx G_B^{-1}$ representam uma transformação bijetora entre os domínios.

A Figura 4.16 mostra dois exemplos de teste de ciclo usando os geradores treinados na configuração C01B. Os erros de reconstrução vão se acumulando e a imagem vai se deteriorando com mais ciclos, o que é esperado já que a transformação G_A não é a inversa da transformação G_B.

Figura 4.16: Acúmulo de erros de reconstrução em uma imagem após 10 ciclos na base de carros. Como as funções que os geradores representam não são perfeitamente inversas, a cada ciclo o erro causado pelas transformações se acumula na imagem. A imagem mais à esquerda é a original e a imagem mais à direita é o resultado da aplicação de 10 ciclos de transformações na imagem original. As imagens intermediárias mostram o resultado de cada ciclo.

Assim como feito anteriormente para a arquitetura Pix2Pix, um teste de generalização também pode ser feito com a CycleGAN. Os esboços com e sem hachura do teste da Pix2Pix foram reutilizados, mas também foram obtidas através do Google algumas imagens de carros novas, que não foram usadas no treinamento.

Os esboços passaram pelo mesmo pipeline de pré-processamento, mas as imagens de carro foram completadas com pixels brancos para ficarem quadradas e depois redimensionadas para 256×256. As imagens foram então passadas pelo seu respectivo gerador, e o resultado pode ser visualizado na Figura 4.17. Os geradores usados foram os da configuração C01B.

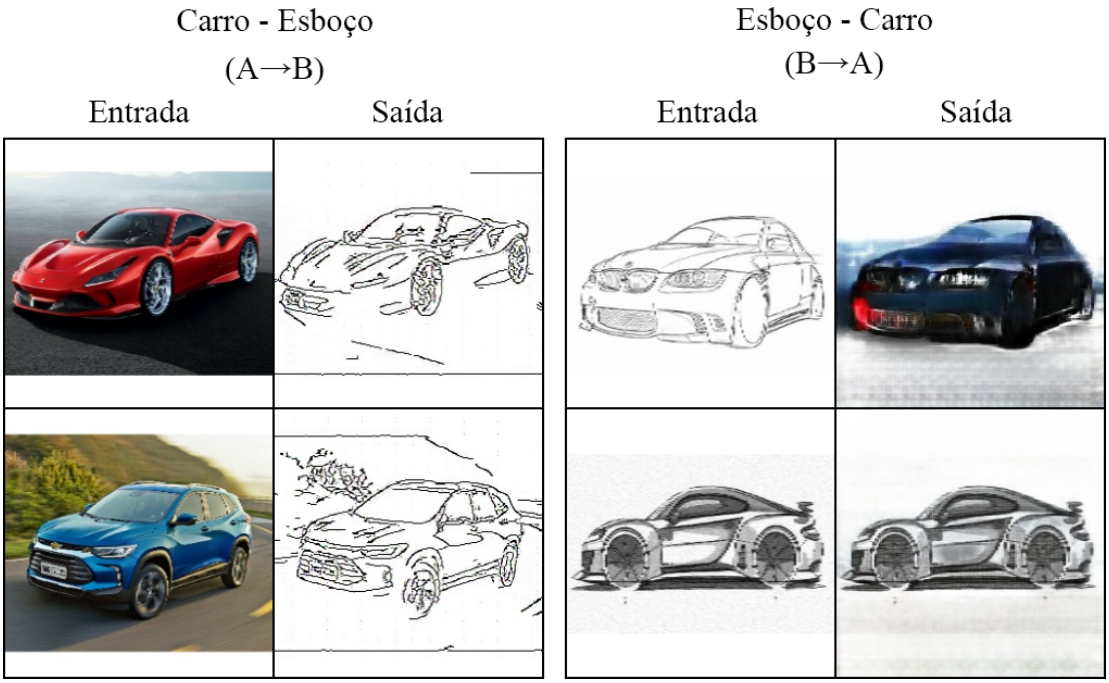

Figura 4.17: Teste de generalização da CycleGAN com imagens de carros.

A transformação carro → esboço foi feita sem maiores problemas, e o resultado ficou bastante parecido com o que seria obtido por um detector de bordas clássico, inclusive considerando a borda da imagem que foi preenchida com pixels brancos.

Já considerando a transformação esboço → carro, nos casos em que o esboço não tem hachuras e outros elementos artísticos (por exemplo o da Fig. 4.17 superior direita)

o gerador identificou os principais componentes como rodas, lanternas, lataria e vidros, e tentou preencher com cores que estivessem de acordo com o que ele viu durante o treinamento, inclusive texturizando o chão e o fundo, e colocando efeitos de iluminação. Nos casos em que o esboço é mais estilizado (por exemplo o da Fig. 4.17 inferior direita), a imagem passa por poucas alterações.

4.3.3 Efeito da assimetria nas transformações da base de carros

Nos experimentos anteriores fica evidente que as redes conseguem criar esboços visualmente muito parecidos com os originais a partir das imagens de carros, independentemente da configuração usada. Apesar das métricas FID mostrarem que há diferença entre os experimentos, os esboços sintéticos são perceptualmente muito parecidos com os originais. Além disso a métrica L1 nos experimentos (Tab. 4.4) mostra que as imagens do domínio de esboços (B) que passam pelo ciclo B-A-B acumulam menos erros do que as imagens do domínio A cicladas.

Essas observações, juntamente com os diferentes patamares das *losses* de gerador na Figura 4.13 apontam para uma aparente assimetria no aprendizado dos geradores. Para explorar se essa diferença causa algum efeito na reconstrução das imagens propomos uma regularização de assimetria, alterando a Eq. 3.5 para acrescentar um coeficiente de assimetria α, de forma que quando $\alpha > 1$ a *loss* é determinada pelas expressões

$$\mathcal{L}_{G_A} = \mathcal{L}_{CGAN}(G_A, D_B) + \gamma \mathcal{L}_{cyc}(G_A, G_B) + \frac{\gamma}{2}\mathcal{L}_{id}(G_A, G_B), \tag{4.1}$$

$$\mathcal{L}_{G_B} = \frac{1}{\alpha} \cdot \mathcal{L}_{CGAN}(G_B, D_A) + \gamma \mathcal{L}_{cyc}(G_A, G_B) + \frac{\gamma}{2}\mathcal{L}_{id}(G_A, G_B), \tag{4.2}$$

aumentando a importância da transformação A → B em relação à transformação B → A, sendo $\mathcal{L}_{G_A}$ a *loss* utilizada para atualizar o gerador G_A, $\mathcal{L}_{G_B}$ a *loss* utilizada para atualizar o gerador G_B. Similarmente, quando $\alpha < 1$ as expressões se tornam

$$\mathcal{L}_{G_A} = \alpha \cdot \mathcal{L}_{CGAN}(G_A, D_B) + \gamma \mathcal{L}_{cyc}(G_A, G_B) + \frac{\gamma}{2}\mathcal{L}_{id}(G_A, G_B), \tag{4.3}$$

$$\mathcal{L}_{G_B} = \mathcal{L}_{CGAN}(G_B, D_A) + \gamma \mathcal{L}_{cyc}(G_A, G_B) + \frac{\gamma}{2}\mathcal{L}_{id}(G_A, G_B), \tag{4.4}$$

aumentando a importância da transformação B → A em relação à transformação A → B. Quando $\alpha = 1$ as expressões são reduzidas à Eq. 3.5.

O coeficiente α foi aplicado de forma que a *loss* adversária da transformação mais importante mantenha sua proporção em relação às *losses* de ciclo e identidade, enquanto a transformação menos importante teria sua *loss* adversária reduzida. Dessa forma o parâmetro γ ajustado experimentalmente manteria sua propriedade.

Para os testes, a rede com a configuração do experimento C01B foi retreinada em três cenários com diferentes valores de α: $\frac{1}{5}$, $\frac{1}{10}$ e 5. A Figura 4.18 mostra o comportamento das *losses* de gerador para as três novas situações.

Como a *loss* do gerador G já estava superior à do gerador F, o resultado da aplicação de $\alpha = 5$ apenas agravou essa diferença. No caso em que $\alpha = \frac{1}{5}$ ambos os geradores apresentaram *losses* em um mesmo patamar, e no caso em que $\alpha = \frac{1}{10}$, a *loss* do gerador G ficou inferior à do gerador F.

Os resultados qualitativos dessa abordagem estão descritos na Tabela 4.5.

Os experimentos C02A e C02B, ambos com $\alpha < 1$, apresentaram piora na métrica FID B em relação ao experimento C01B, o que é esperado, já que eles tem menor importância na transformação A → B e consequentemente a qualidade das imagens sintetizadas do

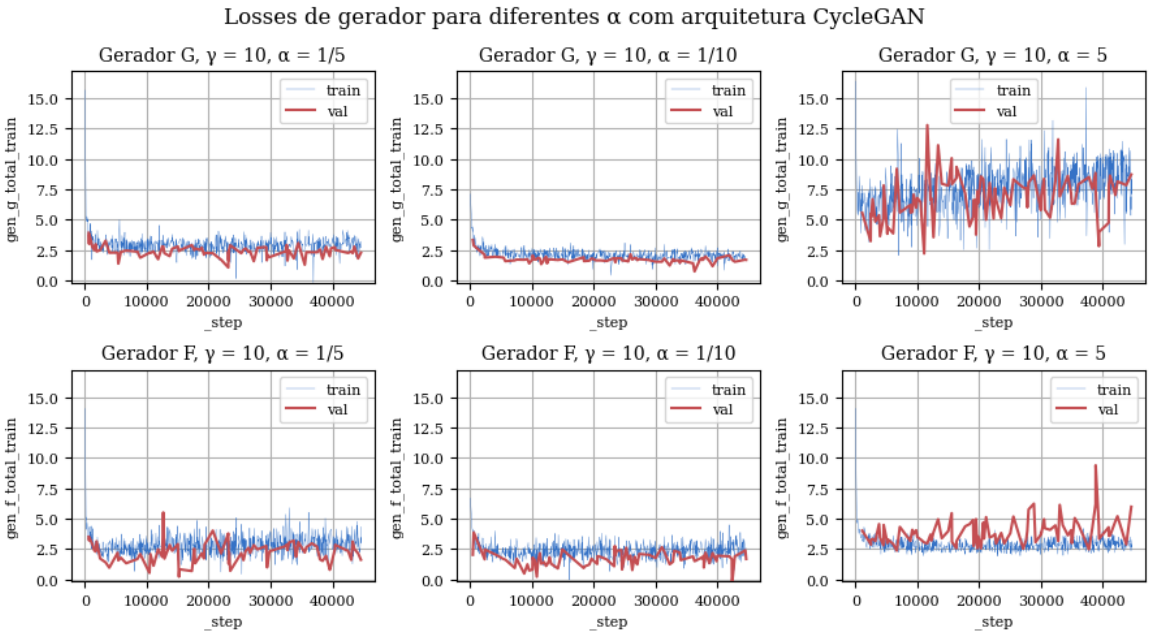

Figura 4.18: *Losses* de gerador com arquitetura CycleGAN e coeficiente de assimetria. A linha azul representa a *loss* total calculada para a base de treinamento e a linha vermelha representa a *loss* total calculada na base de validação. O gerador $G : A \rightarrow B$ realiza a transformação carro→esboço, e o gerador $F : B \rightarrow A$ realiza a transformação esboço→carro.

Experimento	Gama (γ)	Assimetria (α)	Tempo infer. A (s)	Tempo infer. B (s)	*Runtime* (h)
C01B	10	1	0,1076	0,1075	8,3
C02A	10	1/5	0,1159	0,1168	9,2
C02B	10	1/10	0,1119	0,1118	8,9
C02C	10	5	0,1088	0,1087	9,5

Experimento	FID A	FID B	L1 A-B-A	L1 B-A-B
C01B	**118,1349**	182,6977	0,2082	0,1298
C02A	160,7510	203,9337	0,3586	0,1312
C02B	145,5088	207,8608	0,3940	**0,1049**
C02C	151,2509	**174,6772**	**0,1991**	0,1301

Tabela 4.5: Comparação dos resultados dos experimentos da arquitetura CycleGAN ao se variar a regularização de assimetria.

domínio B (esboço) tende a ser pior. Apesar disso, a métrica FID A também apresentou piora, mesmo que a importância relativa para a geração de imagens do domínio A (carros) tenha sido aumentada. O experimento C02C, no entanto, apresentou melhora na métrica

FID B e piora na métrica FID A, muito em linha com o esperado, já que ele foi treinado com $\alpha > 1$.

Em relação à *loss* L1 é possível que o gerador penalizado faça uma transformação de domínio ruim e portanto o gerador reforçado teria dificuldade para trazer de volta ao domínio original, e o acúmulo de erro seria maior. Essa interpretação é suportada pelos resultados na Tabela 4.5, em que os casos C02A e C02B, que penalizam a transformação A→ B causando uma piora da métrica L1 A-B-A, enquanto o inverso ocorre no experimento C02C.

Como o principal objetivo deste estudo é a transformação de esboços em carros, o experimento C01B continua apresentando um resultado superior ao das suas versões com regularização de assimetria.

4.3.4 Discussão

A arquitetura CycleGAN permite realizar a tarefa de transformação de domínio sem a necessidade de uma base de dados pareada, o que é um benefício pela dificuldade de se obter bases com essas características. Sua abordagem não supervisionada não tem desempenho pior do que a abordagem supervisionada da Pix2Pix, inclusive alcançando valores melhores de FID na transformação esboço → carro (FID A), como é possível verificar na Tabela 4.6.

Esses benefícios, porém, vem juntos de maior custo computacional tanto de processamento quanto de memória, já que seus componentes (geradores e discriminadores) são dobrados e as *losses* tem mais termos a serem calculados. O tempo total de execução dos experimento (*runtime*) foi o dobro na arquitetura CycleGAN quando se comparando com o mesmo tipo de gerador na arquitetura Pix2Pix.

Experimento	Arquitetura	Gerador	Gama (γ)	Tempo infer. A (s)	Tempo infer. B (s)	*Runtime* (h)
C01A	CycleGAN	U-Net	5	0,1102	0,1135	7,9
C01B	CycleGAN	U-Net	10	0,1076	0,1075	8,3
C01C	CycleGAN	U-Net	20	0,1079	0,1085	8,2
P02A	Pix2Pix	U-Net	-	0,1424	-	4,1
P02B	Pix2Pix	CycleGAN	-	0,1324	-	7,9
P02C	Pix2Pix	Pix2Pix	-	0,1335	-	2,7

Experimento	FID A	FID B	L1 A-B-A	L1 B-A-B	L1 Pix2Pix
C01A	**115,6354**	166,5402	0,2387	0,1378	-
C01B	118,1349	182,6977	0,2082	0,1298	-
C01C	132,1669	166,9382	0,1758	0,1303	-
P02A	140,3958	-	-	-	0,2229
P02B	119,4360	-	-	-	0,2254
P02C	385,1325	-	-	-	-

Tabela 4.6: Comparação dos resultados dos experimentos com arquitetura CycleGAN e dos experimentos com arquitetura Pix2Pix.

Apesar do bom desempenho da CycleGAN, os ganhos de FID foram marginais, e podem não compensar o uso dessa arquitetura na situação de se ter disponível uma base de dados pareada. Isso também reforça a hipótese de que a má qualidade de síntese de imagens com ambas as arquiteturas pode estar mais ligada à qualidade e diversidade das imagens da base do que à arquitetura em si.

Partindo-se da evidência de que ambas as abordagens podem ter desempenho próximo em uma mesma base de dados, na próxima seção exploraremos o uso da CycleGAN para transformar imagens em uma aplicação parecida, a transformação de esboços de personagens humanoides em imagens de desenhos coloridos dos personagens.

4.4 Aplicação da CycleGAN na base de dados de personagens humanoides

Com o sucesso da aplicação da CycleGAN para a base de dados de imagens de carros, levanta-se um questionamento se ela consegue sustentar ou até aumentar a qualidade das imagens sintetizadas ao ser aplicada em uma base de dados de outro domínio.

O domínio escolhido para este teste é o de desenhos de figuras humanoides, mais específicamente "Os Simpsons", criados por Matt Groening. Uma GAN treinada para reconstruir desenhos a partir de seus esboços pode ser útil para a etapa de *storyboarding* da criação de uma animação. Esse domínio também nos permite estudar se elementos semânticos como como olhos, boca, nariz, roupas ou cabelo conseguem ser consistentemente reconstruídos de forma correta.

Nos experimentos desta seção, o domínio A é composto pelos desenhos finalizados de personagens e o domínio B pelas imagens de contornos (esboços). O gerador $G : A \rightarrow B$ realiza a transformação personagem→esboço, e o gerador $F : B \rightarrow A$ realiza a transformação esboço→personagem. O discriminador D_A discrimina as imagens reais e sintéticas do domínio dos personagens enquanto o discriminador D_B atua no domínio dos esboços.

Como a base de dados é menor que a do experimento anterior, o treinamento das redes nos experimentos a seguir será feito por 20 épocas.

4.4.1 Criação da base de dados de personagens e esboços

Manualmente, através do Google Imagens, foram coletadas imagens coloridas (domínio A) e de contorno (domínio B) de personagens da animação Os Simpsons. A base de dados

do domínio A ficou com 144 exemplos diferentes, enquanto o domínio B ficou com apenas 100.

As imagens então foram processadas para ficarem com dimensões 256×256 segundo a sequência a seguir. Um exemplo do pré-processamento pode ser visto na Figura 4.19.

1. A imagem foi redimensionada de forma que a aresta menor correspondesse a 256 e são feitos três cortes:
   a) um corte central de 256×256;
   b) um corte a partir da borda esquerda, se a imagem tem largura maior do que altura, ou da borda superior, caso contrário, com dimensão final 256×256 (corte esquerdo);
   c) um corte semelhante ao corte esquerdo, mas iniciando da borda direita ou da borda inferior (corte direito);
2. A imagem foi redimensionada de forma que a aresta maior correspondesse a 256 e centralizada em uma imagem branca de 256×256 (imagem enquadrada).

Figura 4.19: Exemplo de pré processamento da base de dados Simpsons. Na ordem: corte central, corte esquerdo, corte direito e enquadrada.

Na sequência as imagens são separadas em uma partição de teste e uma partição de treino de forma que 5% dos exemplos ficassem na partição de teste e 10% dos exemplos ficassem na partição de validação. Os quatro cortes recebem a mesma classificação entre teste, treino e validação, para garantir que imagens cortadas da mesma fonte não possam aparecer ao mesmo tempo em duas partições. A base de treino ficou com um total de 576 exemplos para a Classe A e 400 exemplos para a Classe B.

4.4.2 Efeito da *loss* de ciclo nas transformações da base de personagens com gerador U-Net

Assim como na seção anterior, o primeiro experimento com a nova base é a seleção de γ com o gerador U-Net para avaliar o efeito da *loss* de ciclo nas transformações. Os valores escolhidos para γ também foram 5, 10 e 20. A Figura 4.20 mostra os gráficos das *losses* dos geradores ao longo de 20 épocas de treinamento nas três configurações.

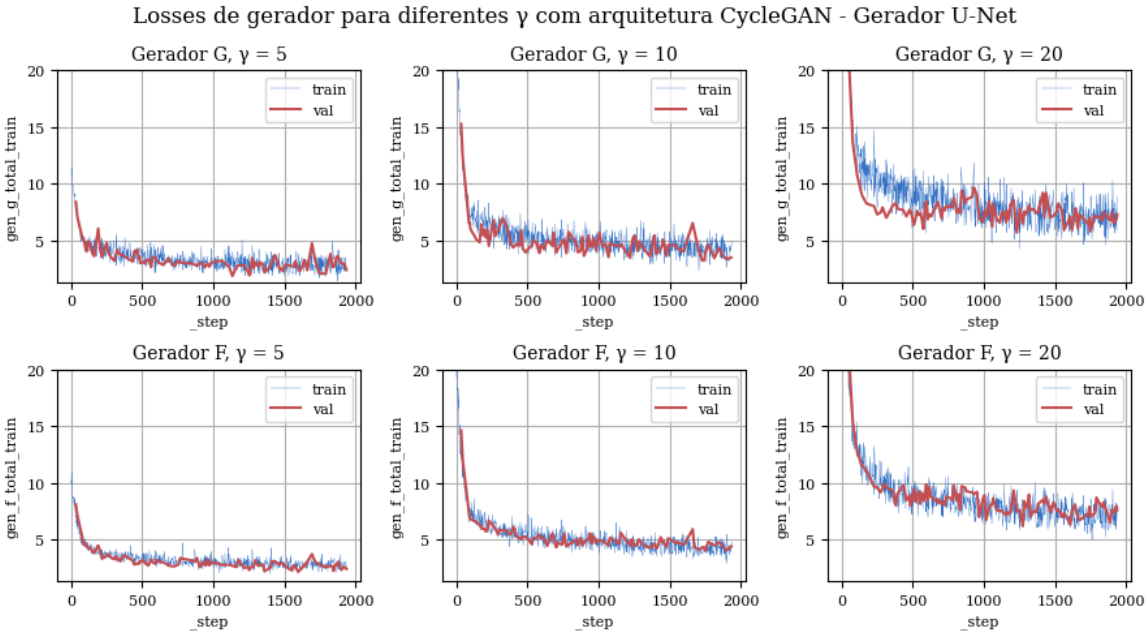

Figura 4.20: *Losses* de gerador na aplicação da base de personagens com gerador U-Net. A linha azul representa a *loss* total calculada para a base de treinamento e a linha vermelha representa a *loss* total calculada na base de validação. O gerador $G : A \rightarrow B$ realiza a transformação personagem $\rightarrow$esboço, e o gerador $F : B \rightarrow A$ realiza a transformação esboço$\rightarrow$personagem.

Em todos os casos, o treinamento parece ter alcançado um estado estável, em que mais épocas de treinamento provavelmente trariam pouco ganho para o sistema. Por conta disso, e para evitar sobreajuste, limitamos o treinamento em 20 épocas.

Outro efeito interessante é que, diferentemente da aplicação anterior, as *losses* dos geradores F e G de um mesmo experimento estão em patamares mais próximos. Por outro lado, as métricas de qualidade, apresentadas nos gráficos da Figura 4.21, tiveram um

comportamento menos estável, além de não conseguirem alcançar os mesmos patamares da aplicação anterior.

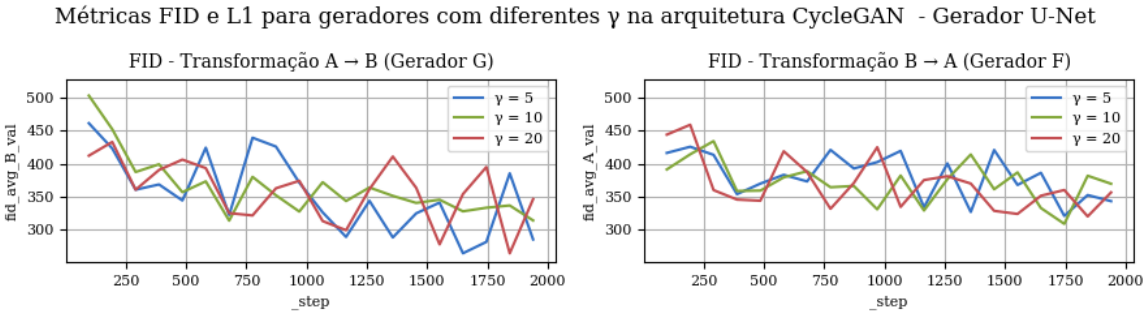

Figura 4.21: Métricas de qualidade na aplicação da base de personagens com gerador U-Net. A métrica FID A representa a diferença entre imagens reais e sintéticas do domínio de personagens e a métrica FID B representa a diferença entre imagens reais e sintéticas do domínio de esboços.

Ao se comparar as métricas de qualidade avaliadas na base de teste (Tabela 4.7) é possível verificar um desempenho claramente melhor do experimento C03A tanto na síntese de imagens do domínio A quanto de imagens do domínio B, através das métricas FID. Esse comportamento é esperado pelo fato do menor valor de γ permitir uma melhor reconstrução da imagem mesmo ao custo do seu poder de reconstrução no ciclo. Apesar disso, ele também tem a menor métrica L1 no ciclo A-B-A, o que é inesperado.

Experimento	Gama (γ)	Tempo infer. A (s)	Tempo infer. B (s)	*Runtime* (h)
C03A	5	0,0065	0,0043	0,4
C03B	10	0,0063	0,0042	0,4
C03C	20	0,0063	0,0041	0,4

Experimento	FID A	FID B	L1 A-B-A	L1 B-A-B
C03A	**318,7890**	**286,6939**	**0,3363**	0,1320
C03B	343,1006	314,0793	0,4253	0,1279
C03C	339,5002	334,4511	0,3890	**0,0999**

Tabela 4.7: Comparação dos resultados dos experimentos com a base de personagens e gerador U-Net ao se variar o parâmetro γ.

A Figura 4.22 mostra exemplos de imagens geradas pelas três configurações. Assim como com a base de dados de carros, a tarefa de criação de esboços teve um desempenho visualmente similar nos três casos.

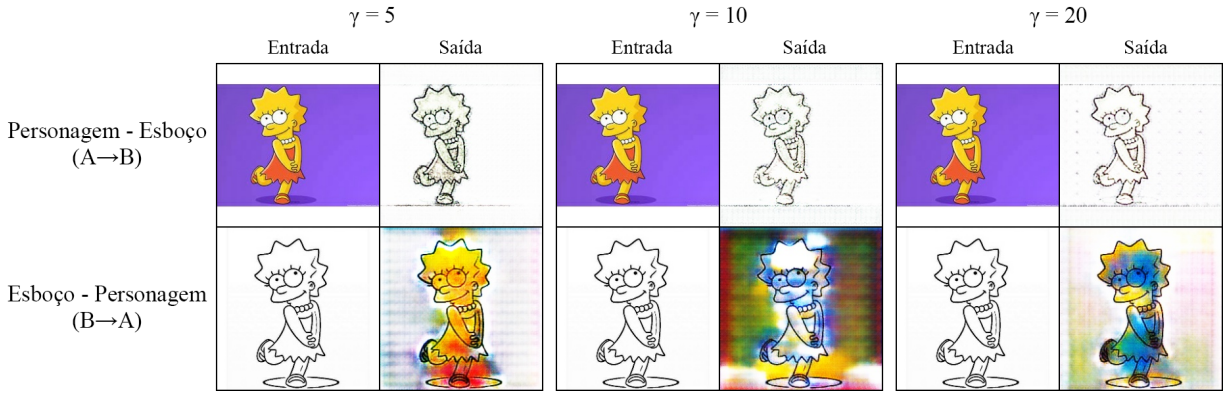

Figura 4.22: Efeito da *loss* de ciclo com gerador CycleGAN com a base de personagens. Imagens com menores valores de γ implicam em uma menor importância da *loss* de ciclo em relação à *loss* adversária.

Na base de dados de carros a rede conseguiu perceber e reconstruir corretamente elementos com sentido semântico, como rodas, lanternas, janelas e carroceria, mas aparentemente nenhuma das configurações com a base de dados de personagens conseguiu identificar componentes como olhos, cabeça, roupas e outros. Isso pode ser uma limitação do gerador em perceber características mais complexas. Portanto, no próximo experimento, utilizaremos o gerador proposto no artigo original da CycleGAN.

4.4.3 Efeito da arquitetura do gerador na configuração CycleGAN

Uma vantagem de redes convolucionais com blocos residuais, como a ResNet, é que elas conseguem ser mais profundas e com isso capturar mais características das imagens dentro da sua estrutura [23]. Zhu et al. [80] propuseram em seu artigo sobre a CycleGAN um gerador residual com 6 ou 9 blocos residuais, que pode ser superior aos existentes na época na tarefa de transformação de domínio.

Nos experimentos com a base de dados de carros, não foi possível usar esse gerador por limitações de memória e tempo de processamento no ambiente de experimentação. Porém, como a base de dados de personagens é menor, foi possível usar o gerador CycleGAN sem

consumir mais memória do que a GPU teria disponível e dentro de um tempo de execução razoável.

Comparando os gráficos da Figura 4.23 com os da Figura 4.20, as duas arquiteturas de gerador alcançaram os mesmos patamares de *loss*, mas o gerador CycleGAN convergiu mais rapidamente, apesar de ter variações de maior amplitude.

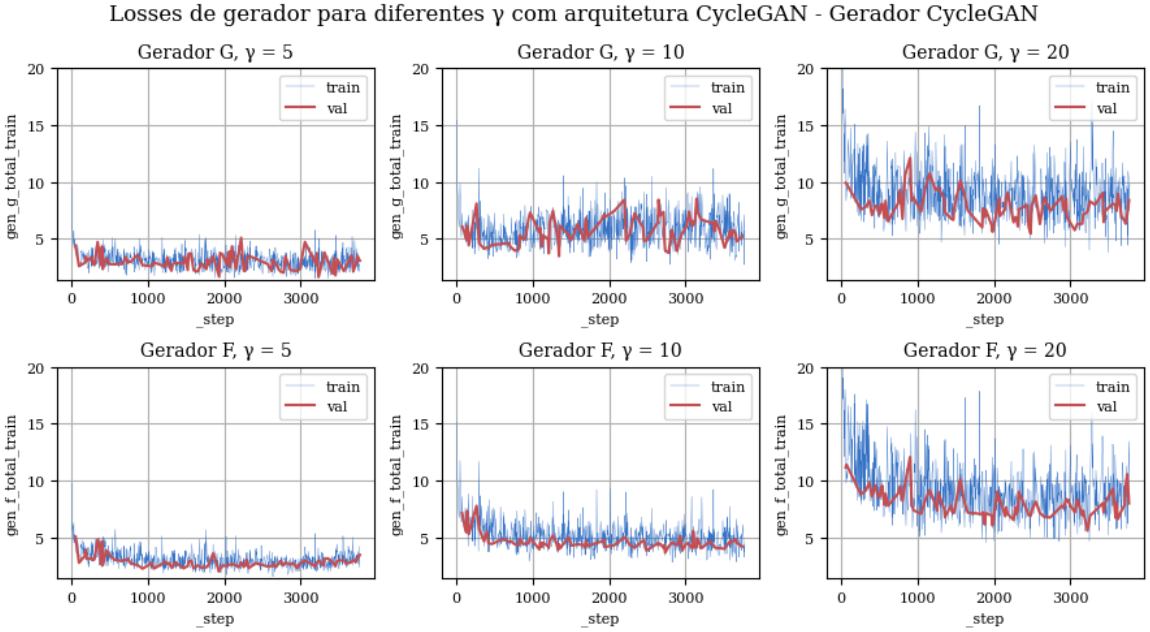

Figura 4.23: *Losses* de gerador na aplicação da base de personagens com gerador CycleGAN. A linha azul representa a *loss* total calculada para a base de treinamento e a linha vermelha representa a *loss* total calculada na base de validação. O gerador $G : A \rightarrow B$ realiza a transformação personagem →esboço, e o gerador $F : B \rightarrow A$ realiza a transformação esboço→personagem.

Em relação às métricas de qualidade, comparando os gráficos da Figura 4.24 com os da Figura 4.21, as duas abordagens tiveram o mesmo comportamento instável, e também alcançaram patamares muito próximos de FID.

Comparando-se os resultados dos seis experimentos através da Tabela 4.8, os melhores valores de FID A e FID B são com o uso dos geradores CycleGAN, mas os melhores valores de L1 são com o uso de geradores U-Net. Como o objetivo principal é a geração de imagens do domínio A (personagens), o gerador com o melhor desempenho é o C03F, que inclusive apresentou um FID B melhor do que os três experimentos com gerador U-Net.

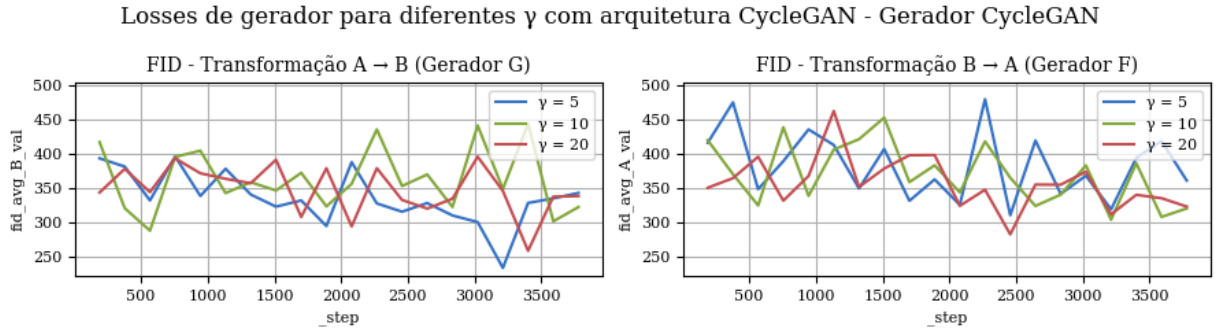

Figura 4.24: Métricas de qualidade com o uso do gerador CycleGAN na base de dados de personagens. Imagens com menores valores de γ implicam em uma menor importância da *loss* de ciclo em relação à *loss* adversária.

Experimento	Gerador	Gama (γ)	Tempo infer. A (s)	Tempo infer. B (s)	*Runtime* (h)
C03A	U-Net	5	0,0065	0,0043	0,4
C03B	U-Net	10	0,0063	0,0042	0,4
C03C	U-Net	20	0,0063	0,0041	0,4
C03D	CycleGAN	5	0,0069	0,0056	0,8
C03E	CycleGAN	10	0,0066	0,0053	0,8
C03F	CycleGAN	20	0,0067	0,0053	0,9

Experimento	FID A	FID B	L1 A-B-A	L1 B-A-B
C03A	318,7890	286,6939	**0,3363**	0,1320
C03B	343,1006	314,0793	0,4253	0,1279
C03C	339,5002	334,4511	0,3890	**0,0999**
C03D	346,2907	**270,1144**	0,5759	0,1383
C03E	344,6463	273,3681	0,5722	0,1806
C03F	**314,1982**	283,7176	0,3497	0,1421

Tabela 4.8: Comparação dos resultados dos experimentos com a base de personagens com geradores U-Net e CycleGAN ao se variar o parâmetro γ.

Entretanto, ao se visualizar as imagens geradas na Figura 4.25, percebe-se que as imagens geradas não são melhores do que as dos experimentos anteriores, e que os geradores também não conseguiram identificar semanticamente os elementos que constituem os personagens.

O teste de ciclo dos geradores da configuração C03F, ilustrado pela Figura 4.26, mostra como o erro de transformação se acumula em uma imagem ao longo de vários ciclos. Assim como no caso da base de carros, a imagem vai se deteriorando conforme vai passando pela

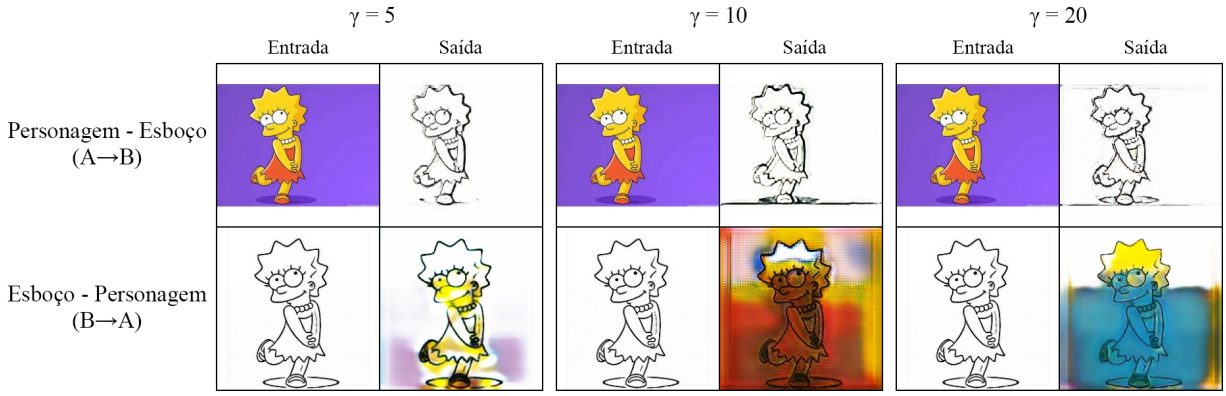

Figura 4.25: Efeito da *loss* de ciclo com gerador CycleGAN com a base de personagens e gerador residual. Imagens com menores valores de γ implicam em uma menor importância da *loss* de ciclo em relação à *loss* adversária.

sequência de geradores. Após 10 ciclos, nem o personagem e nem o esboço mantém uma boa qualidade.

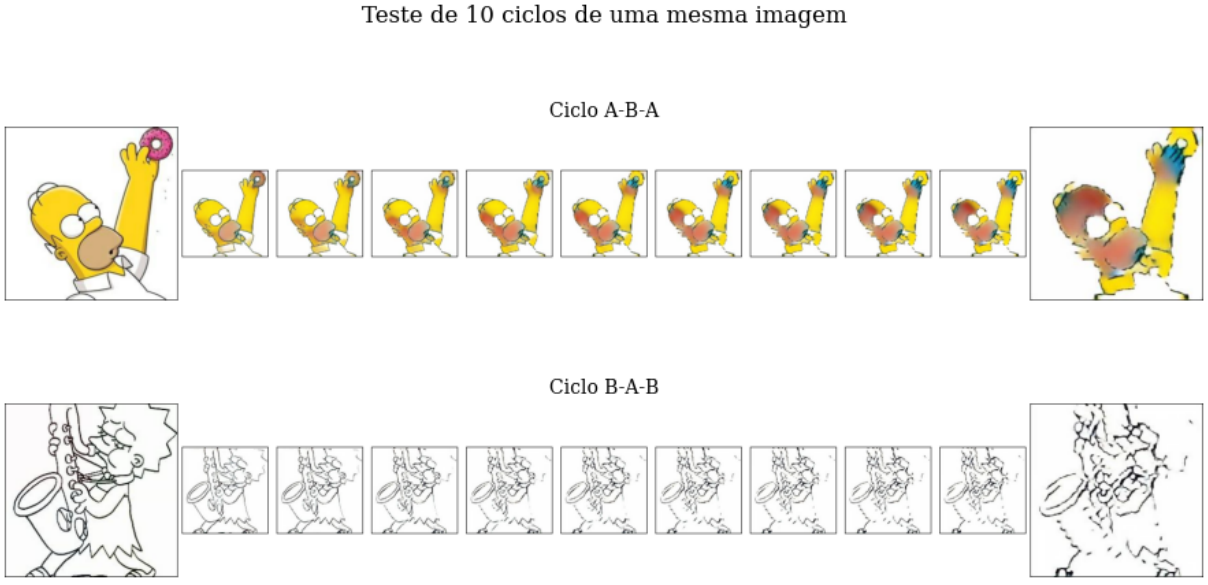

Figura 4.26: Acúmulo de erros de reconstrução em uma imagem após 10 ciclos na base de personagens. Como as funções que os geradores representam não são perfeitamente inversas, a cada ciclo o erro causado pelas transformações se acumula na imagem. A imagem mais à esquerda é a original e a imagem mais à direita é o resultado da aplicação de 10 ciclos de transformações na imagem original. As imagens intermediárias mostram o resultado de cada ciclo.

4.4.4 Discussão

Como os resultados com a base de carros foi superior, tanto visualmente quanto em valores de FID, pode-se concluir que bases de dados diferentes irão ter resultados diferentes, mesmo quando falamos da mesma aplicação, como modelagem de esboços.

Além disso, as GANs não aprenderam uma separação semântica das partes que compõem as figuras, como cabeças, roupas, braços e pernas, fundo, etc., o que surpreende, já que elas foram capazes de fazer isso com a base de dados de carros, separando lanternas, vidros, rodas e inclusive acrescentando sombras aos carros.

Um dos motivos pelo qual as GANs conseguiram aprender as segmentações dos carros e não dos personagens pode ser o tamanho da base de dados, que é muito menor no exemplo dos personagens do que no exemplo dos carros. Uma tentativa válida seria aumentar a base de dados e tentar novamente, mas isso envolveria obter imagens coerentes com ambos os domínios e pode ser feito em trabalhos futuros.

4.5 Conclusões

Os experimentos executados na base de dados de carros mostra que a Pix2Pix e a CycleGAN tem desempenho limitado na síntese de imagens finalizadas a partir dos esboços criados com detector de bordas. As imagens finais tem uma forte semelhança com carros reais, e componentes semânticos importantes das imagens, como rodas, janelas e outros elementos, normalmente eram reconstruídos corretamente. Todos os métodos também são suficientemente rápidos para se fazer uma aplicação interativa, com a inferência durando frações de segundos para acontecer.

Há evidências de que a qualidade da base de dados pode influenciar fortemente o resultado, como foi possível ver após a limpeza da base de carros com um classificador binário. Entretanto, ainda resta saber se uma base de dados com contornos reais teria desempenho superior à base de dados com bordas.

A aplicação com a base de dados de personagens mostrou resultados inferiores. Ainda que os contornos dessa base sejam muito superiores aos detectados automaticamente na base de carros, a quantidade e diversidade de imagens parece ser pequena demais para que os geradores conseguissem aprender a reconstruir corretamente a imagem finalizada.

Outros experimentos do tipo estão limitados à presença de uma base de dados bem estruturada, a qual não conseguimos encontrar ou criar durante este trabalho.

Um ponto importante ao se ressaltar a respeito das GANs é que elas tem uma convergência bastante sensível, podendo ter problemas de colapso de modo, ou de dissipação de gradiente muito facilmente [75, 4, 61]. Nas GANs que usam mudanças na arquitetura e tipos diferentes de regularização nas *losses*, como a CycleGAN e a Pix2Pix, ajustar esses termos e as possíveis combinações de gerador, discriminador e *loss* é crucial ao mesmo tempo que é difícil, e cada combinação pode causar resultados extremamente diferentes, como mostramos ao longo desta seção.

Há também um compromisso entre qualidade de imagem e diversidade de domínio que se observa nas técnicas atuais de GANs [75]. Isso parece indicar que para se gerar imagens de maior qualidade o domínio delas deve estar muito bem definido.

As métricas de avaliação existentes também não se mostram adequadas para avaliar a percepção humana a respeito das imagens geradas. Nesta seção há experimentos com valores superiores de FID que não são visualmente superiores ao seus pares, e uma possível explicação para isso está em Kynkäänniemi et al [36], com a dependência da FID nas classes da rede InceptionV3, cujas ativações são usadas para o seu cálculo. O problema da avaliação objetiva das imagens geradas ainda está em estudo e discussão na comunidade acadêmica.

4.6 Considerações finais

Por conta das limitações dos equipamentos do ambiente de experimentação, o presente trabalho estudou apenas duas arquiteturas relativamente antigas para a transformação de domínio. Existem outras abordagens que podem trazer benefícios para essa aplicação.

Uma dessas abordagens é a utilizada pela Pix2PixHD [72], e parte do seu sucesso veio do uso de mapas semânticos para construir imagens de alta resolução de forma supervisionada e condicional. Uma das contribuições mais interessantes dessa abordagem é a possibilidade de editar a imagem a ser construída a partir da manipulação do mapa semântico, como na Fig. 4.27, obtida da página do projeto Pix2Pix HD [3].

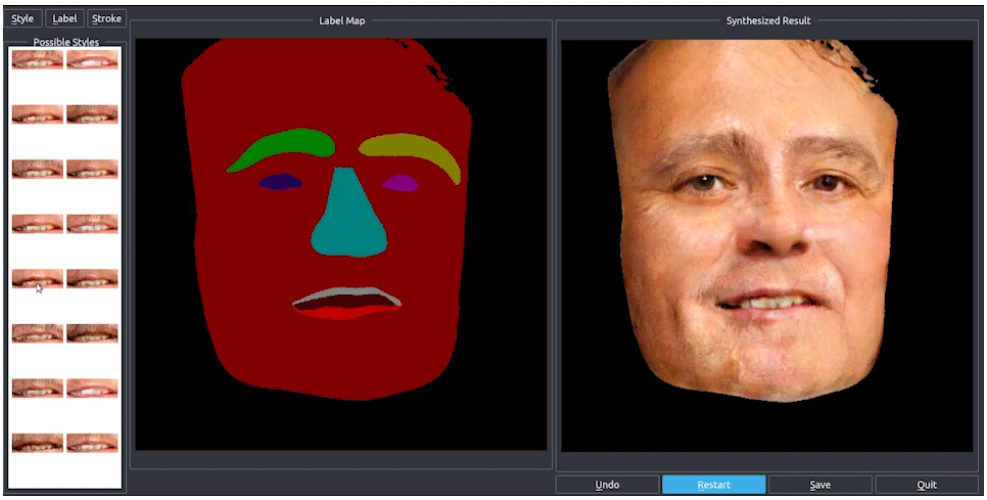

Figura 4.27: Manipulação da síntese com mapas semânticos na Pix2Pix HD. É possível controlar a construção da face a partir da manipulação do mapa semântico.

Uma desvantagem dessa abordagem é que ela é extremamente dependente da existência de um mapa semântico que identifique cada objeto a nível de pixel para cada imagem real da base de dados usada no treinamento da rede. O mapa semântico também precisa ser consistente de forma que dois mapas de imagens diferentes mantenham a mesma classificação para o mesmo objeto, ou seja, se em uma imagem uma pessoa é classificada como "Classe

3 https://tcwang0509.github.io/pix2pixHD/

1", então todas as pessoas de todas as imagens também precisam ser classificadas como "Classe 1".

Outra abordagem interessante é a do *Swapping Autoencoder* [46], que codifica as imagens em duas componentes: uma matriz com componentes espaciais e geométricos e um vetor com informações de estilo, como textura e cores. De acordo com o artigo original, essa abordagem parece funcionar bem em transferência de estilo. Talvez essa abordagem possa ser adaptada para receber o esboço como componente espacial e geométrica e um exemplo de imagem finalizada como componente de estilo.

Essas e outras abordagens podem ser estudadas em trabalhos futuros.

5 SÍNTESE DE IMAGENS COM GANS CONDICIONAIS

Em uma GAN de imagens treinada, o gerador aprende a transformar uma distribuição latente em uma distribuição sintética que corresponda às características dos dados do domínio em que foi treinado [19]. Como já discutido na Seção 3.2.2, em alguns casos específicos em que o espaço latente aprendido pela GAN é desemaranhado, é possível alterar características específicas de uma imagem sintetizada ao se manipular o vetor latente que a gera.

Geralmente, GANs não condicionais [20, 49] usam um vetor de ruído amostrado de uma distribuição (geralmente gaussiana), e o treinamento adversário induz o gerador a gerar imagens verossímeis, enquanto o discriminador se especializa em diferenciar imagens reais de sintéticas, como ilustrado pela Figura 5.1. Neste caso, a menos que o vetor latente seja previamente conhecido, não há controle da geração da imagem final.

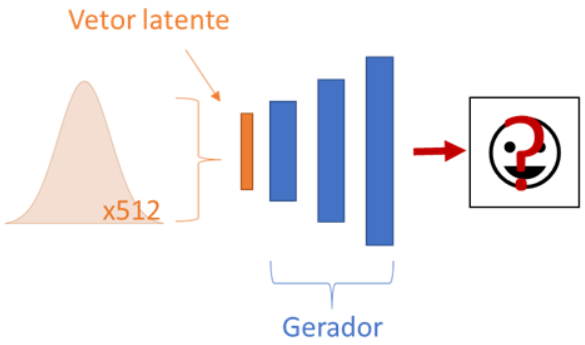

Figura 5.1: Manipulação por GAN não condicional. A figura mostra um vetor latente aleatório composto de 512 elementos amostrados independentemente de uma distribuição normal. O vetor é usado como entrada do gerador que cria uma imagem como saída. Não há controle na geração da imagem a menos que o vetor seja previamente conhecido.

Por não ser possível controlar a entrada ou a saída do gerador, o treinamento dele é não supervisionado, enquanto o treinamento do discriminador é supervisionado (mostrando imagens reais e sintéticas, que sabemos de onde vem). Esse método é usado na ProGAN [30] e na StyleGAN [33], que não tem controle de qual imagem será gerada, por não ter conhecimento do efeito dos vetores utilizados como entrada.

É possível, no entanto, mapear o espaço latente [33] ou utilizar uma técnicas para fazer a engenharia reversa do gerador e, a partir de uma imagem qualquer, descobrir qual é o vetor que geraria a mesma imagem ao ser aplicado ao gerador [79, 2, 3, 54].

Em uma abordagem supervisionada, condicionando o treinamento da GAN a utilizar uma imagem simultaneamente como entrada e como objetivo do gerador, ele acaba se tornando um *autoencoder*. Se no decorrer dessa transformação o gerador gerar um vetor, ele será equivalente à representação codificada da imagem no espaço latente desse gerador. Nesse caso, a parte do gerador à esquerda do vetor é chamada de *encoder* e a parte à direita é chamada de *decoder*, como representado na Figura 5.2.

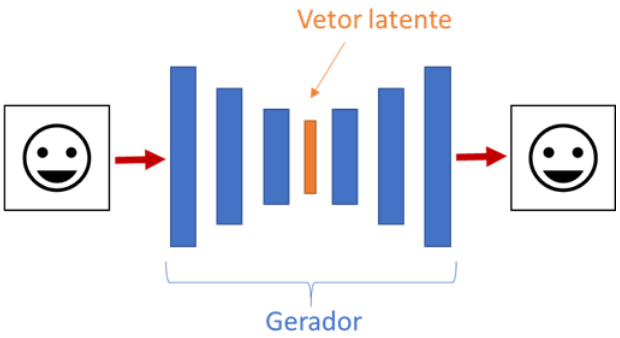

Figura 5.2: Manipulação por GAN condicional. A figura mostra um gerador treinado como um *autoencoder*, em que a parte azul à esquerda é o *encoder* e a parte azul à direita é chamada de *decoder*. O vetor latente no centro guarda uma representação comprimida da imagem que foi utilizada como entrada, e o domínio de todos os vetores latentes é o espaço latente do gerador.

Neste capítulo propusemos o uso de arquiteturas de GANs condicionais, como a da Pix2Pix, para se criar um gerador que aprenda a reconstruir a mesma imagem que recebeu como entrada e simultaneamente gerar um vetor latente que permita a manipulação da síntese. Nossa contribuição é um estudo de diversas abordagens com diferentes configurações

da rede, e como cada elemento influencia no resultado final da imagem reconstruída. Alguns exemplos de imagens geradas pelos melhores experimentos estão presentes no Apêndice A.

5.1 Metodologia

Nosso ponto de partida é a arquitetura Pix2Pix, mas com a intenção de entender o impacto de cada elemento na síntese da imagem final, realizamos alterações tanto nas configurações quanto na própria arquitetura da rede.

Inicialmente, na Seção 5.2, utilizamos o gerador U-Net, já empregado originalmente na Pix2Pix, como um *autoencoder* para reconstruir a imagem que é apresentada ao gerador. Essa arquitetura gera um vetor latente, o qual avaliamos se pode ser usado para manipular a síntese das imagens.

Em seguida, na Seção 5.3, substituímos o gerador U-Net pelo gerador residual da CycleGAN. Como o gerador residual não cria vetores latentes na sua estrutura, nós propusemos três adaptações desse gerador original que passam por uma etapa de criação do vetor entre o *encoder* e o *decoder*, assim como na Figura 5.2. Chamamos essas três adaptações de "residual adaptado", "*full residual*" e "*simple decoder*" e explicamos sua arquitetura na Seção 5.1.1.

Com o gerador residual original e suas adaptações, realizamos diversos experimentos alterando configurações da rede ou até seus elementos, como geradores e discriminadores, com o objetivo de criar imagens sintéticas realistas, e como consequência medir a capacidade dos vetores latentes, comparando os resultados de cada abordagem.

Todas as redes serão treinadas como *autoencoders*, o que significa que a mesma imagem é apresentada à rede como entrada e como objetivo. Isso incentiva a rede a recriar a imagem

da entrada de forma que o *encoder* realize uma compressão da informação e o *decoder* a descompressão para criar a imagem final mais próxima possível do objetivo.

5.1.1 Arquitetura e parâmetros

O primeiro gerador utilizado é o U-Net, ilustrado na Figura 5.3-(esquerda). Entre sua metade *encoder* e sua metade *decoder*, ele passa pela geração de um vetor, que deve carregar parte da informação que o *decoder* usa para reconstruir a imagem. A outra parte da informação vem das conexões de atalho (setas laranja) entre as metades do gerador. O gerador residual é o mesmo utilizado pela CycleGAN e também em experimentos do Capítulo 4, ilustrado na Figura 5.3. Os dois geradores foram adaptados das suas versões originais para que pudessem trabalhar com imagens de dimensões $128 \times 128 \times 3$.

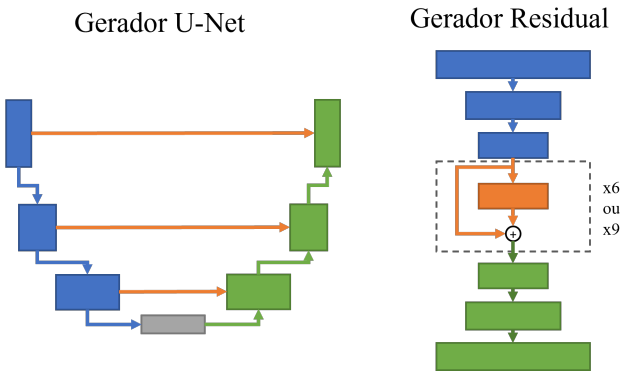

Figura 5.3: Geradores utilizados na transferência intradomínio. O gerador U-Net [56] é usado nos artigos originais tanto da Pix2Pix quanto da CycleGAN. O gerador residual, é o usado pela CycleGAN [80], com 6 ou 9 blocos residuais (em laranja) na sua construção. Os blocos azuis indicam as camadas do *encoder* e os blocos verdes indicam as camadas do *decoder*.

Uma limitação do gerador residual, é que ele não gera um vetor latente como o U-Net. A Figura 5.4 detalha a arquitetura dele e mostra que ao se inserir uma imagem de dimensões $128 \times 128 \times 3$ na entrada do gerador, o mapa de atributos na transição entre o *encoder* e o *decoder* é um tensor de dimensões $31 \times 31 \times 256$. Na imagem os blocos azuis representam o *encoder*, os blocos verdes o *decoder*, e o conjunto laranja são

os blocos residuais, que não alteram as dimensões dos mapas de atributos. Blocos `Conv2d` são convoluções bidimensionais, e blocos `Conv2dT` são camadas de convoluções transpostas, detalhadas na Seção 2.3.2.

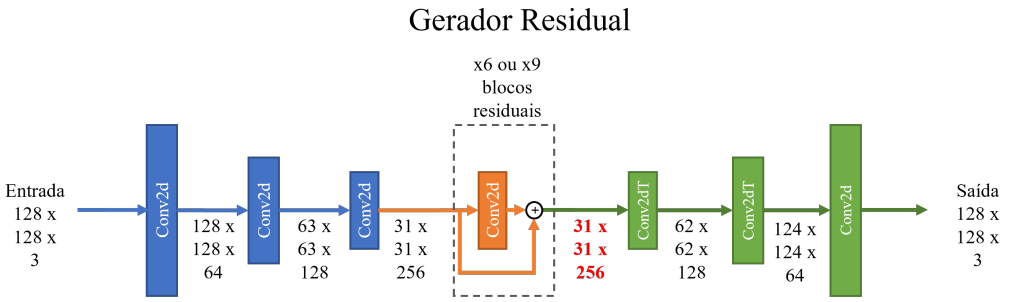

Figura 5.4: Detalhamento do gerador residual. Os blocos azuis indicam as camadas do *encoder* e os blocos verdes indicam as camadas do *decoder*. A imagem de entrada tem dimensões $128 \times 128 \times 3$ e vai se modificando a cada camada da rede, com as dimensões de cada etapa escritas abaixo da seta que indica sua transição. O mapa de atributos na transição entre o *encoder* e o *decoder* é um tensor de dimensões $31 \times 31 \times 256$ (escrito em vermelho).

A primeira adaptação proposta a essa arquitetura é o gerador "residual adaptado", detalhado na Figura 5.5, que conta com uma convolução extra para reduzir a quantidade de mapas de atributos de 256 para 1, seguida por uma camada `Dense` completamente conectada para se criar um vetor de 512 elementos. Na sequência mais convoluções transpostas com *stride* 2 para se ampliar as dimensões desse vetor novamente para $128 \times 128 \times 3$ e reconstruir a imagem em RGB.

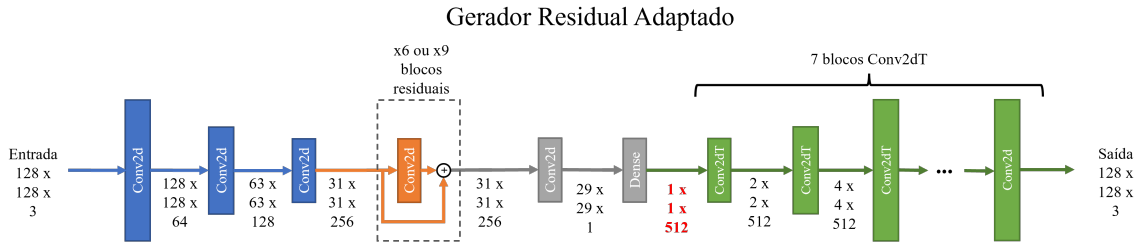

Figura 5.5: Detalhamento do gerador residual adaptado. Em relação ao residual original, foram adicionadas uma convolução e uma camada `Dense` (em cinza) para gerar o vetor latente, e novas camadas de convoluções transpostas no *decoder*.

Uma segunda adaptação busca usar a capacidade de redes residuais em capturar mais características das imagens[23] também no *decoder*. Por receber conjuntos de blocos

residuais em suas duas metades, chamamos este gerador de "*full residual*", e sua arquitetura está representada na Figura 5.6.

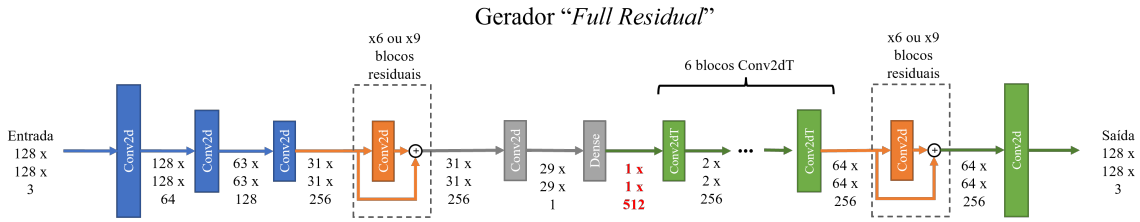

Figura 5.6: Detalhamento do gerador *full residual*. Além da geração do vetor latente, esta arquitetura conta também com blocos residuais no *decoder*.

Finalmente, como forma de evitar o problema dos artefatos quadriculados, causados pela convolução transposta e explicados na Seção 2.3.2, propusemos também o gerador *simple decoder*, inspirado pelo artigo de Odena et al. [44], e que não utiliza convoluções transpostas na etapa de ampliação da imagem final que ocorre no *decoder*. Como detalhado na Figura 5.7, os blocos de convolução transposta foram substituídos por blocos "*simple upsample*", que consistem em se dobrar as dimensões da imagem utilizando redimensionamento por interpolação bilinear, seguida de dois blocos convolucionais com normalização e ativação ReLU. O bloco "*simple upsample*"está ilustrado na Figura 5.8.

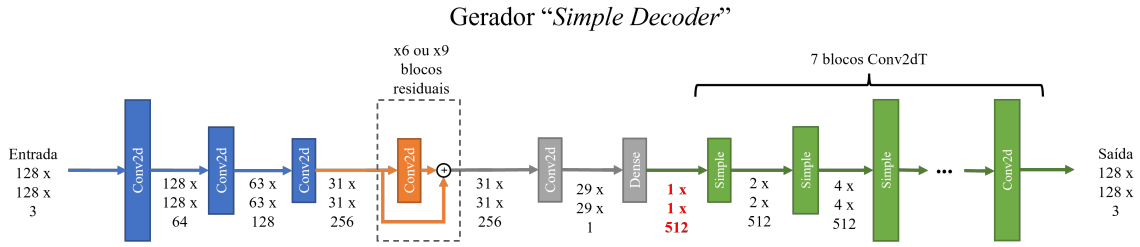

Figura 5.7: Detalhamento do gerador *simple decoder*. A diferença deste gerador para o "residual adaptado"é o uso de blocos "*simple upsample*"(Fig. 5.8) no *decoder*, como forma de evitar os artefatos quadriculados.

Sempre que possível utilizaremos o gerador residual como base de comparação dos geradores residual adaptado, *full residual* e *simple decoder*, e o gerador utilizado será explicitado em cada experimento. Ademais, as configurações utilizadas normalmente serão:

- Base de dados utilizada: CelebA-HQ [30]
- Dimensões das imagens de entrada: $128 \times 128 \times 3$

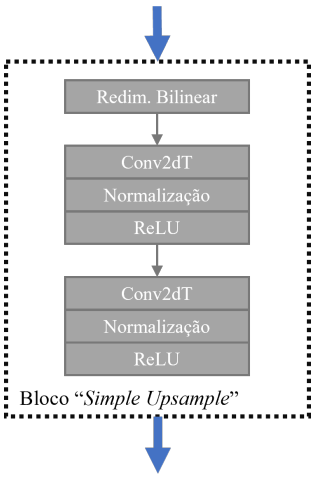

Figura 5.8: Detalhamento do bloco *simple upsample*. Esse bloco foi inspirado em Odena et al [44] como forma de evitar a geração de artefatos quadriculados, causada pelas convoluções transpostas.

- Quantidade de épocas de treinamento: 10
- Discriminadores PatchGAN [28] ou ProGAN [30], explicitados em cada experimento
- *Losses*:
  - Pix2Pix / PatchGAN: Loss adversária + L1 [28]
  - WGAN-GP + L1
  - O parâmetro $\lambda = 100$ controla o efeito da *loss* L1
- Otimizador Adam com *Learning Rate* $\alpha = 1e-5$ e
  - $\beta_1 = 0,5$ quando usando a *loss* PatchGAN [28]
  - $\beta_1 = 0,9$ quando usando a *loss* adversária WGAN-GP [22]
- Random jitter [28] não foi usado durante o treinamento

Os *batch sizes* e quaisquer alterações nas configurações acima serão descritas nos experimentos correspondentes.

A base de dados utilizada é a CelebA-HQ [30] que tem 29.100 imagens nas dimensões 1024×1024 de faces de homens (35% das imagens) e mulheres (65 % das imagens). Os exemplos de treinamento, teste e validação foram separados conforme a Tabela 5.1.

	Treinamento	**Teste**	**Validação**	**Total**	%
Female	15380	1000	2563	18943	65,1%
Male	8620	100	1437	10157	34,9%
Total	24000	1100	4000	29100	100,0%
%	82,5%	3,8%	13,7%	100,0%	-

Tabela 5.1: Divisão usada na base de dados CelebA-HQ para os experimentos

Para otimizar o treinamento em termos de tempo de processamento e memória utilizada, as imagens são redimensionadas para 128×128 no momento em que são carregadas para uso na rede. Por limitações do tempo e processamento disponíveis para este trabalho, optamos por manter um máximo de 10 épocas de treinamento exceto quando especificado um outro valor.

5.1.2 Métricas avaliadas

Durante o treinamento avaliaremos as *losses* da arquitetura para a base de treinamento a cada iteração e para a base de validação a cada 10 iterações.

Ao final de cada época, para uma amostra da base de dados de validação a métrica FID [24] será avaliada para ambas as arquiteturas, e a distância L1 [28] entre a imagem gerada e o objetivo.

Ao final do treinamento, todas as imagens da base de dados de teste serão geradas e o tempo médio de inferência (de geração) das imagens será medido. Além disso, as métricas de qualidade (FID e L1) serão avaliadas para todas as imagens da base de dados de teste.

5.1.3 Ambiente de experimentação

O ambiente de experimentação é o mesmo usado no Capítulo 4. Todos os experimentos foram executados em um computador com Windows 10, processador Intel Core i7 9700K, 32 GB de memória RAM, e uma GPU Nvidia RTX 2070 Super com 8GB de VRAM.

O ambiente utilizado foi o Python v3.8.8 com *framework* Tensorflow [1] v2.7.0, e Nvidia CUDA 11.4.

Os códigos correspondentes aos experimentos estão disponíveis no repositório do projeto no GitHub[1].

5.2 Treinamento do *autoencoder* com gerador U-Net

Com base nos experimentos de transformação de domínio, a U-Net mostrou um bom desempenho como gerador na tarefa de síntese de imagens, e por esse motivo o escolhemos para os primeiros experimentos de síntese supervisionada.

O gerador é treinado de forma supervisionada, usando a mesma imagem como entrada e objetivo. Após o treinamento espera-se que o gerador consiga aprender a representação do espaço latente e mapear cada imagem como um ponto desse espaço, na forma de um vetor latente.

Usando a base de dados CelebA-HQ [30], a rede com gerador U-Net foi treinada e por 10 épocas usando o discriminador e a *loss* da arquitetura Pix2Pix (PatchGAN), e com 10 imagens em cada *batch*. A evolução das *losses* do gerador está discriminada na Figura 5.9.

1 https://github.com/vinyluis/Autoencoders

Verifica-se que ao final das 10 épocas as *losses* começam a aumentar, o que sugere que um mecanismo de parada antecipada poderia ser usado para impedir o sobretreino da rede.

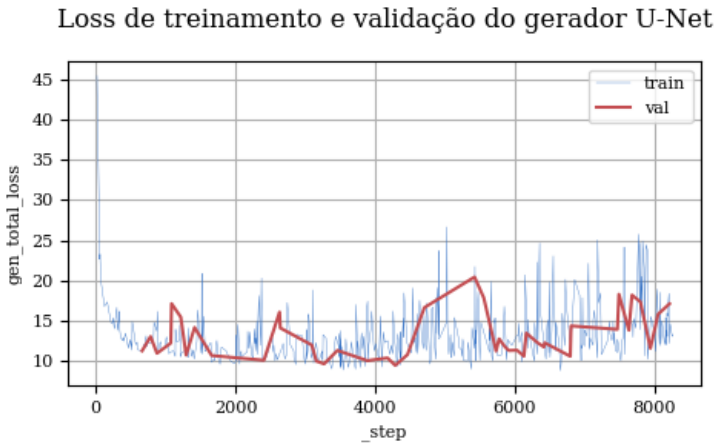

Figura 5.9: *Losses* de gerador com *autoencoder* U-Net. A linha azul representa a *loss* total calculada para a base de treinamento e a linha vermelha representa a *loss* total calculada na base de validação.

Apesar disso, as métricas de qualidade da Figura 5.10, calculadas ao fim de cada época para a base de dados de teste, mostram uma melhora na síntese conforme as épocas passam, sendo que a FID melhora até a última época, enquanto a L1 acompanha o comportamento da *loss* de gerador e piora levemente nas últimas épocas. Isso é coerente com o fato de que a L1 é calculada na base de treinamento como termo de regularização na *loss* do gerador. Os resultados completos estão descritos na Tabela 5.2.

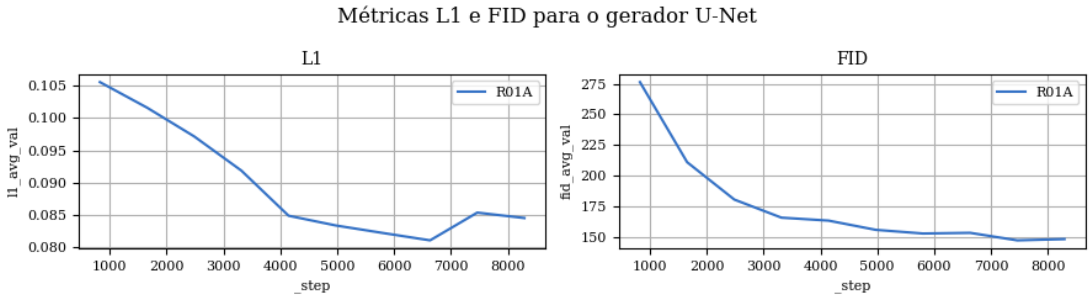

Figura 5.10: Métricas de qualidade com *autoencoder* U-Net. A métrica FID representa a diferença entre a distribuição das imagens reais e sintéticas, e a métrica L1 representa a diferença média pixel a pixel de cada imagem sintética com seu respectivo objetivo.

Visualmente, a reconstrução das faces foi satisfatória, embora seja possível perceber através da Figura 5.11 que as imagens geradas apresentam maior desfoque e ruído. Ainda

EXP	Gerador	Discriminador	Loss	Normalização
U01A	U-Net	PatchGAN	PatchGAN	BatchNorm

EXP	*Batch Size*	FID Teste	L1 Teste	Tempo infer. (s)	*Runtime* (h)
U01A	10	137,3725	0,0781	0,1910	1,2

Tabela 5.2: Resultados do treinamento do *autoencoder* U-Net.

assim, o gerador conseguiu representar faces masculinas e femininas em diferentes poses e inclusive com adereços como chapéus.

Figura 5.11: Resultados do treinamento do *autoencoder* U-Net. As colunas "entrada"representam o que o gerador recebeu como informação de entrada e as colunas "saída"representam o que o gerador sintetizou.

5.2.1 Avaliação do espaço latente aprendido

Na U-Net, a informação passa por dois caminhos. Um deles é a sequência de camadas da rede que passa pelo *encoder*, pela geração do vetor latente, e depois pelo *decoder*, até a geração da imagem final. O segundo caminho são as conexões de atalho, que transmitem informações de diferentes etapas do processamento do *encoder* diretamente para o *decoder*. Isso implica que possivelmente nem toda a informação necessária para a reconstrução da imagem seja contida no vetor latente.

Para experimentar essa hipótese, podemos fazer o experimento exemplificado pela Figura 5.12, em que inicialmente treinamos uma rede U-Net por tempo o suficiente até que ela aprenda a reconstruir corretamente a imagem, como fizemos na seção anterior, e em seguida separamos o decoder e o deixamos reconstruir a imagem apenas recebendo o vetor latente como entrada. Duas versões desse experimento podem ser executadas, uma em que o *decoder* utiliza os mesmos parâmetros treinados da rede anterior, e uma em que o *decoder* é retreinado partindo de uma inicialização aleatória.

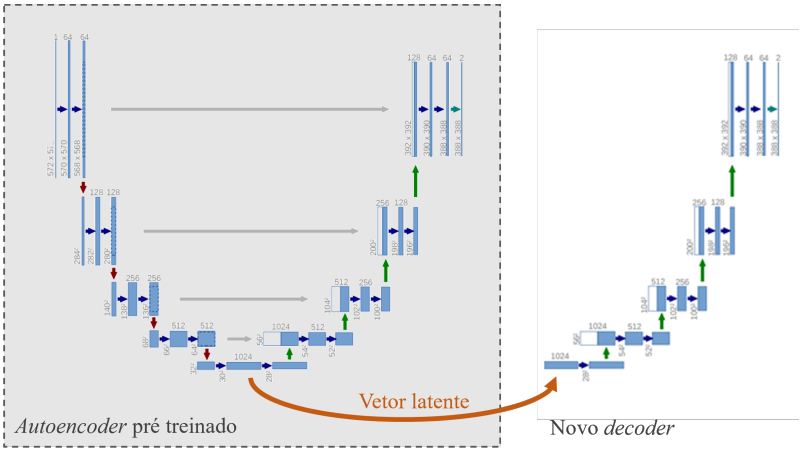

Figura 5.12: Reconstrução da imagem a partir do vetor latente na U-Net. A rede à esquerda é o *autoencoder* U-Net pré treinado. Ao se passar uma imagem por esse *autoencoder* pode-se obter seu vetor latente, que é então passado a um *decoder* (rede direita) para reconstruir a imagem sem o auxílio das conexões de atalho.

No entanto, ao se analisar os vetores latentes gerados para todas as 2000 imagens de teste, descobrimos que todos eles eram nulos. Isso significa que *nenhuma* informação estava passando por ele, e que toda a informação que o *decoder* utilizava vinha das conexões de atalho.

5.2.2 Discussão

O objetivo destes estudos é a manipulação das imagens dentro do próprio domínio em que elas se inserem, a partir da manipulação de seus vetores latentes. No caso da U-Net,

verificamos que nessa aplicação específica, nenhuma informação é registrada no vetor latente, o que impede essa manipulação.

Dessa forma, precisamos partir para outras arquiteturas que não trafegam informação do *encoder* para o *decoder* de qualquer outra maneira que não seja o próprio vetor latente. Na próxima seção apresentamos diversos testes usando o *autoencoder* residual, e adaptações que propusemos para a criação de um vetor latente nesse gerador.

5.3 Treinamento do *autoencoder* com gerador residual

Diferentemente do U-Net, o gerador residual não tem nenhuma conexão entre o *encoder* e o *decoder* que não seja a própria sequência de camadas da rede. As únicas conexões de atalho são as conexões residuais, que transportam informações de camadas superiores da rede para camadas inferiores, mas sem que essa informação passe do *encoder* para o *decoder*.

O uso de blocos residuais que transportam informações por um caminho alternativo que ignora algumas camadas, evita que a informação se degrade rapidamente, como o problema da dissipação de gradiente, e permite que a rede tenha mais camadas, e portanto, que ela treine mais *kernels* para reconhecer as características da imagem [23].

No entanto, o gerador residual proposto por Zhu et al [80] não possui uma camada que gera um vetor latente, então propusemos três diferentes alterações no gerador para incluir essa camada: o residual adaptado, o *full residual* e o *simple decoder*.

Nesta seção utilizamos o gerador residual e suas adaptações como *autoencoders* treinados para sintetizar as faces da base de dados CelebA-HQ, em diferentes configurações, se alterando normalização, discriminador, *loss* e outros componentes de sua arquitetura.

5.3.1 Efeito da normalização e quantidade de blocos residuais na reconstrução da imagem

Apesar de o gerador residual não apresentar um vetor latente entre o *encoder* e o *decoder*, podemos usa-lo como base de comparação para os outros geradores. Além disso, é necessário entender o quanto as camadas de normalização afetam o resultado final, portanto treinamos o gerador residual com discriminador e *loss* PatchGAN, conforme a arquitetura Pix2Pix, por 10 épocas.

Os autores propuseram duas versões do gerador residual: uma com seis e uma com nove blocos residuais. Em cada experimento, variamos a quantidade de blocos bem como a normalização usada entre `BatchNorm`, `InstanceNorm` e `PixelNorm`. Essas normalizações foram escolhidas por representarem algumas das mais comuns dentre as usadas para síntese de imagens nas referências deste trabalho. A evolução das *losses* de gerador de cada combinação está nos gráficos da Figura 5.13.

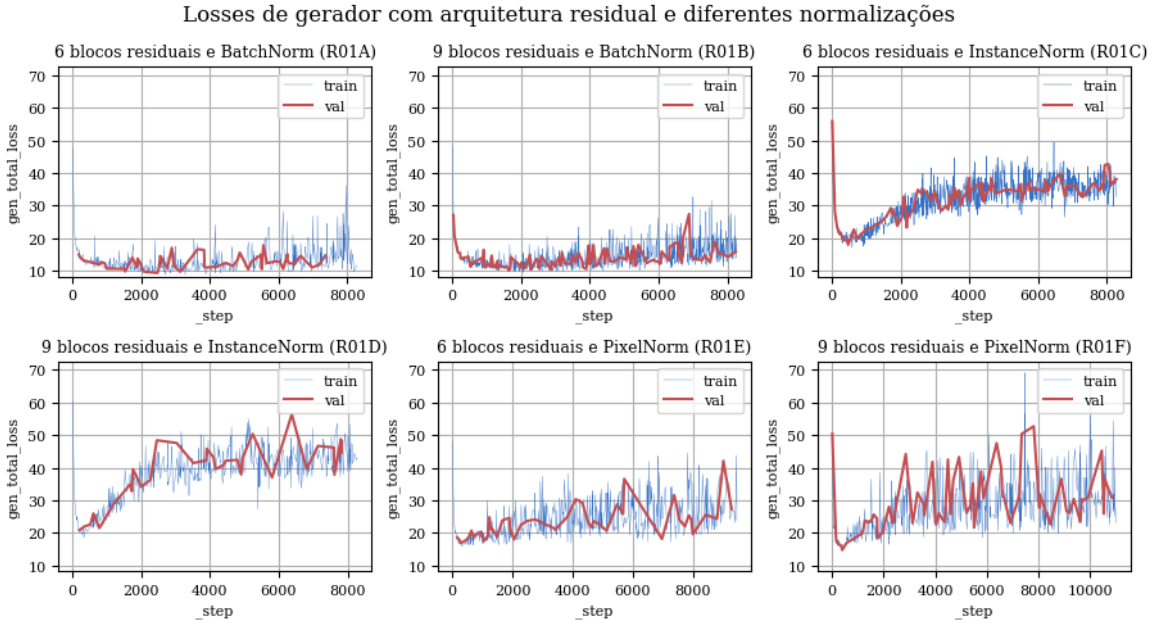

Figura 5.13: *Losses* de gerador com *autoencoder* residual ao se variar a normalização e quantidade de blocos residuais. A linha azul representa a *loss* total calculada para a base de treinamento e a linha vermelha representa a *loss* total calculada na base de validação.

Ao se comparar os gráficos, percebemos que não há grande diferença entre as versões de 6 e 9 blocos residuais, mas que há grande diferença no resultado de acordo com o tipo de normalização usado. Os exemplos com `BatchNorm` apresentaram o patamar de *loss* mais baixo em relação aos outros, seguido pelos exemplos com `PixelNorm` e depois por `InstanceNorm`. Apesar de não ter o pior valor de *loss*, a `PixelNorm` se mostrou mais instável, com amplitudes maiores nos momentos finais.

Os gráficos de métricas de qualidade da Figura 5.14, e os resultados finais da Tabela 5.3 também apontam nessa direção. As linhas em tons de azul representam os experimentos com `BatchNorm`, as linhas vermelhas com `InstanceNorm` e as verdes com `PixelNorm`. Assim como na análise anterior, os métodos com BatchNorm mostraram o melhor resultado, e sem muita diferença entre as implementações de 6 e 9 blocos.

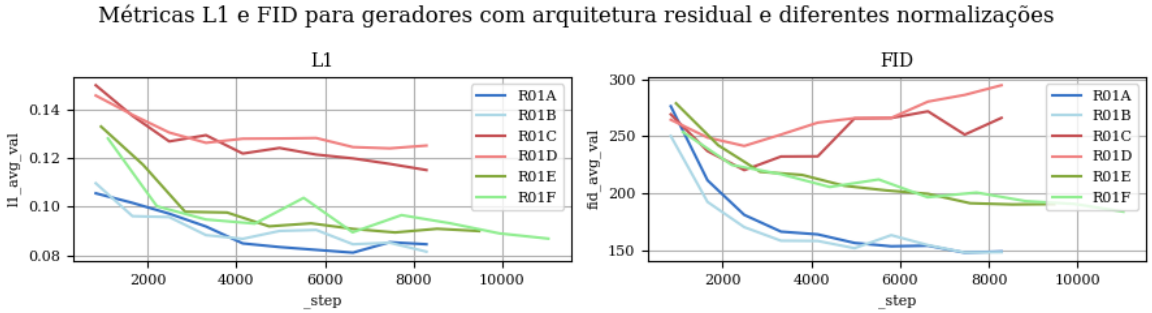

Figura 5.14: Métricas de qualidade com *autoencoder* residual ao se variar a normalização e quantidade de blocos residuais. A métrica FID representa a diferença entre a distribuição das imagens reais e sintéticas, e a métrica L1 representa a diferença média pixel a pixel de cada imagem sintética com seu respectivo objetivo.

A Figura 5.15 exibe uma comparação entre os diversos métodos para uma imagem em comum da base de dados de teste. Visualmente, as imagens sintetizadas com `BatchNorm` também tiveram uma nitidez maior do que as outras, embora ainda haja ruído (veja o detalhamento do experimento R01A, na Figura). Seguindo o comportamento das métricas de qualidade, as imagens geradas com normalização `PixelNorm` ficaram pouco abaixo das geradas com `BatchNorm`, e as imagens geradas com `InstanceNorm` ficaram com a pior qualidade dentre os três métodos.

EXP	Gerador	Blocos Residuais	Discriminador	*Loss*	Normalização
R01A	Residual	6	PatchGAN	PatchGAN	BatchNorm
R01B	Residual	9	PatchGAN	PatchGAN	BatchNorm
R01C	Residual	6	PatchGAN	PatchGAN	InstanceNorm
R01D	Residual	9	PatchGAN	PatchGAN	InstanceNorm
R01E	Residual	6	PatchGAN	PatchGAN	PixelNorm
R01F	Residual	9	PatchGAN	PatchGAN	PixelNorm

EXP	*Batch size*	FID Teste	L1 Teste	Tempo infer. (s)	*Runtime* (h)
R01A	32	151,5434	0,0907	0,1896	1,4
R01B	32	**148,0442**	0,0930	0,1831	1,5
R01C	32	259,2141	0,1131	0,1950	1,6
R01D	32	287,0606	0,1239	0,1955	1,7
R01E	28	186,7754	0,0892	0,1714	1,5
R01F	24	180,1986	**0,0866**	0,1980	1,8

Tabela 5.3: Resultados do treinamento do *autoencoder* residual com diferentes normalizações e quantidade de blocos residuais.

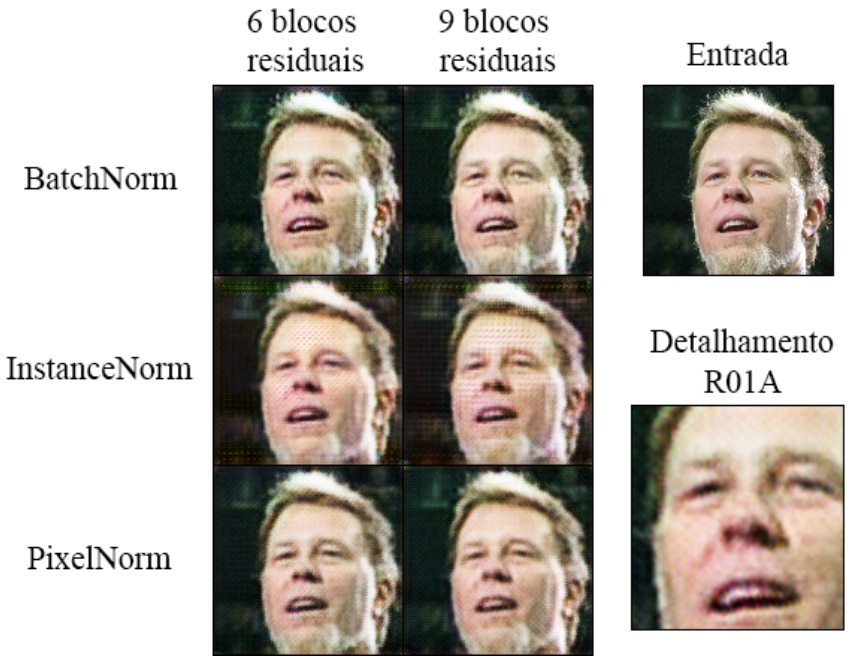

Figura 5.15: Resultados do treinamento do *autoencoder* residual com diferentes normalizações e quantidade de blocos residuais. A imagem à direita foi apresentada a todas as redes como ”entrada”. As duas colunas representam as redes com 6 e 9 blocos residuais, respectivamente, e cada linha corresponde a um tipo de normalização usada. O ”detalhamento R01A”aplica um zoom em uma região da imagem do experimento R01A (6 blocos, com normalização `BatchNorm`) para facilitar a visualização do ruído.

Como as métricas e as imagens mostraram, não há grandes diferenças perceptíveis entre as redes com 6 ou com 9 blocos residuais, portanto, nos próximos experimentos apenas a versão de 6 blocos residuais foi usada, já que ela possui menos parâmetros treináveis e ocupa menos espaço em memória. O experimento R01A (6 blocos, com normalização `BatchNorm`) é usada como base de comparação com os próximos experimentos.

5.3.2 Adaptações no gerador residual para obtenção de vetores latentes

Com uma referência de onde o *autoencoder* residual pode chegar sozinho, pudemos experimentar com os geradores alterados e comparar com a base de referência do experimento anterior. Todas as redes deste experimento foram treinadas com as mesmas configurações e parâmetros do experimento anterior, inclusive alterando-se a normalização. Diferentemente do caso anterior, não experimentaremos com 9 blocos residuais, que não trouxeram diferença perceptível na qualidade.

Os gráficos com as *losses* de gerador e as métricas de qualidade para as nove combinações de gerador e normalização estão respectivamente nas Figuras 5.16 e 5.17. Em contraste com os resultados anteriores, as implementações que usaram `InstanceNorm` como camada de normalização obtiveram um comportamento igual ou até melhor do que as que usaram `BatchNorm`. As redes com `PixelNorm`, entretanto, apresentaram um comportamento crescente da *loss* de gerador, sem indícios de que ela estabilizaria.

Na Figura 5.17, as linhas azuis representam as métricas do gerador residual adaptado, as linhas vermelhas do gerador *full residual*, e as linhas verdes do *simple decoder*. As numerações dos experimentos correspondem às apresentadas nos gráficos da Fig. 5.16. Na métrica L1 há claramente um distanciamento entre os métodos que utilizaram `PixelNorm` dos outros, enquanto a métrica FID apresenta um comportamento mais próximo da maioria das redes, com a R02D alcançando um resultado final melhor do que as demais.

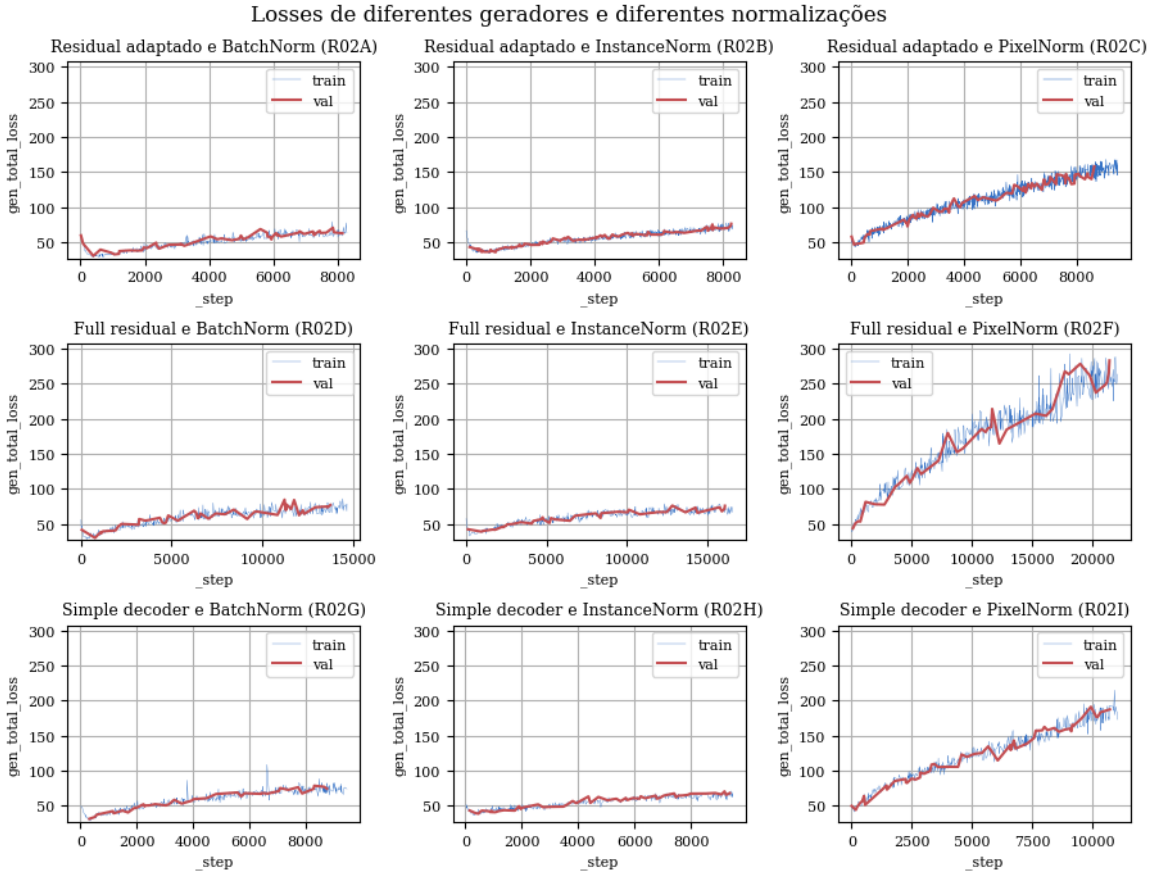

Figura 5.16: *Losses* de gerador com os *autoencoders* adaptados ao se variar a normalização. A linha azul representa a *loss* total calculada para a base de treinamento e a linha vermelha representa a *loss* total calculada na base de validação.

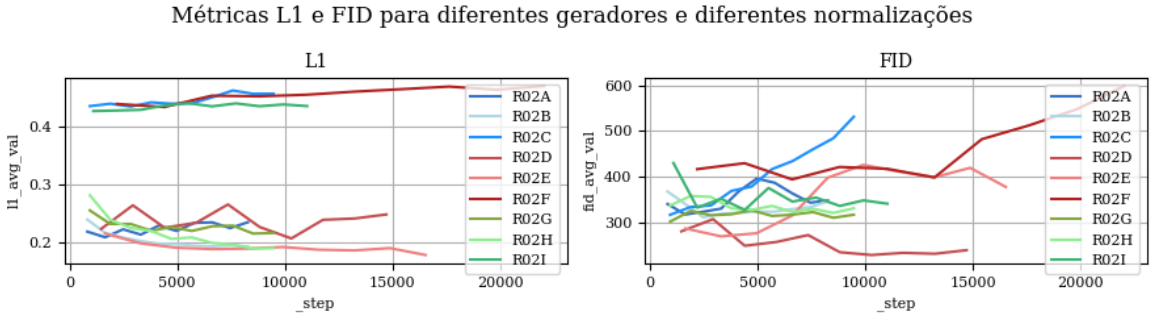

Figura 5.17: Métricas de qualidade com os *autoencoders* adaptados ao se variar a normalização. A métrica FID representa a diferença entre a distribuição das imagens reais e sintéticas, e a métrica L1 representa a diferença média pixel a pixel de cada imagem sintética com seu respectivo objetivo.

Ao se comparar os resultados finais dos experimentos com a Tabela 5.4, verificamos que os experimentos com `PixelNorm` (R02C, F e I) realmente apresentaram os piores resultados,

e os experimentos com o gerador *full residual* apresentaram os melhores resultados em FID (R02D) e L1 (R02E). Isso pode ser devido à capacidade dos blocos residuais de captarem mais características da imagem também na fase de reconstrução (*decoder*). Os experimentos com os outros geradores apresentaram resultados próximos ao do *full residual*, mas apesar dos bons resultados, nenhum método se aproximou das métricas obtidas no experimento R01A.

EXP	Gerador	Discriminador	Loss	Normalização
R01A	Residual	PatchGAN	PatchGAN	BatchNorm
R02A	Residual adaptado	PatchGAN	PatchGAN	BatchNorm
R02B	Residual adaptado	PatchGAN	PatchGAN	InstanceNorm
R02C	Residual adaptado	PatchGAN	PatchGAN	PixelNorm
R02D	*Full residual*	PatchGAN	PatchGAN	BatchNorm
R02E	*Full residual*	PatchGAN	PatchGAN	InstanceNorm
R02F	*Full residual*	PatchGAN	PatchGAN	PixelNorm
R02G	*Simple decoder*	PatchGAN	PatchGAN	BatchNorm
R02H	*Simple decoder*	PatchGAN	PatchGAN	InstanceNorm
R02I	*Simple decoder*	PatchGAN	PatchGAN	PixelNorm

EXP	*Batch size*	FID Teste	L1 Teste	Tempo infer. (s)	*Runtime* (h)
R01A	32	151,5434	0,0907	0,1896	1,4
R02A	32	352,0017	0,3058	0,1851	1,9
R02B	32	345,0529	0,1906	0,1909	1,7
R02C	28	536,6622	0,4573	0,1847	1,6
R02D	18	**245,0432**	0,3188	0,2018	2,9
R02E	16	381,1892	**0,1743**	0,2292	3,1
R02F	12	605,2670	0,4676	0,1953	3,1
R02G	28	322,5575	0,3041	0,1926	1,5
R02H	28	332,6525	0,1856	0,2059	1,8
R02I	24	341,8348	0,4374	0,1913	1,6

Tabela 5.4: Resultados do treinamento com os *autoencoders* adaptados ao se variar a normalização. O Experimento R01A aparece na primeira linha como base de comparação.

Comparando-se a imagem sintetizada por cada gerador a partir de uma mesma imagem da base de dados de teste (Fig. 5.18), percebe-se imediatamente que a qualidade é inferior à das imagens do experimento anterior. Em nenhum caso os geradores conseguiram reconstruir corretamente o rosto apresentado como entrada, mas nos casos com normalização `BatchNorm` e `InstanceNorm`, os geradores foram capazes de inferir e reconstruir a pose, tons de cabelo, e inclusive o formato dos elementos do rosto, mesmo que de forma grosseira. Os exemplos

com `PixelNorm` sofreram do problema do colapso de modo, em que os discriminadores prendem o gerador em um vale com um mínimo local na função de *loss* [61, 4].

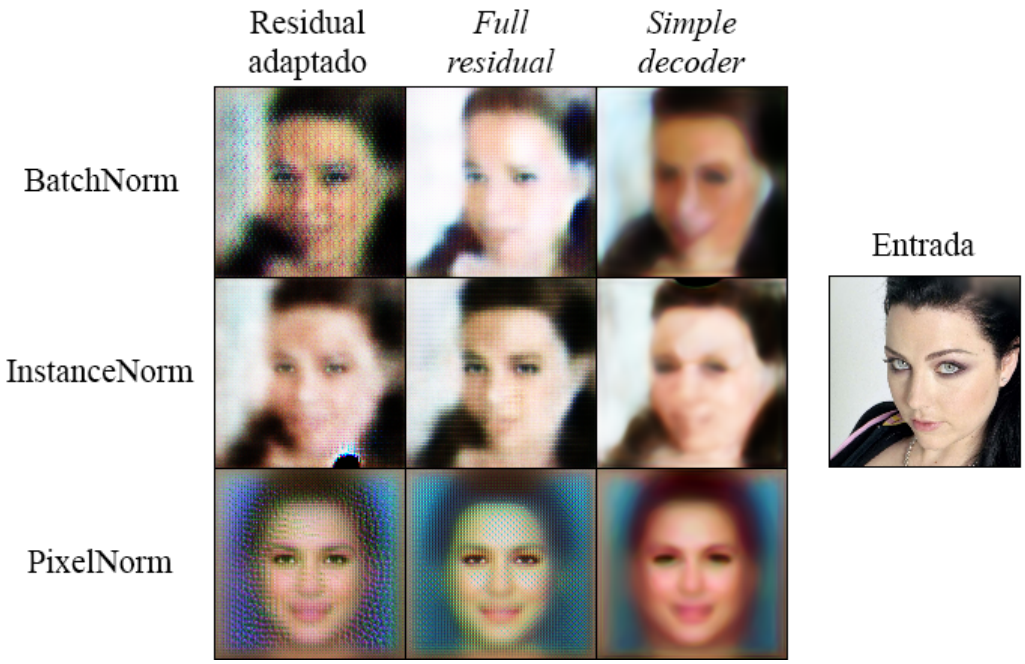

Figura 5.18: Resultados do treinamento com os *autoencoders* adaptados ao se variar a normalização. A imagem à direita foi apresentada a todas as redes como "entrada". As colunas representam um tipo de gerador, e cada linha corresponde a um tipo de normalização usada.

Outra comparação interessante e bem visível nas imagens é a presença dos artefatos quadriculados nos casos em que o *decoder* é baseado em convolução transposta (residual adaptado e *full residual*) mas que não aparecem quando usamos um *decoder* que faz um redimensionamento mais simples da imagem e usa convoluções comuns (*simple decoder*). Um lado negativo dessa abordagem é que elar deixa as imagens com maior desfoque.

Os geradores das GANs deste experimento são treinados de acordo com a percepção do discriminador em relação às imagens sintéticas, em comparação com as imagens reais (loss adversária) e de acordo com o quão longe o próprio gerador está de reconstruir exatamente a imagem objetivo (loss L1). Por conta disso, um discriminador diferente pode ser capaz de direcionar melhor o aprendizado do gerador, e essa abordagem foi testada no próximo experimento.

5.3.3 Substituição do discriminador

Uma alteração que pode ser explorada é substituir o discriminador PatchGAN por outro, mas garantindo que possa usar a mesma *loss*. Para isso, foi adaptada a arquitetura do discriminador da ProGAN [30] de forma que ele possa receber como entrada a imagem a ser discriminada juntamente com a imagem condicionante, no modelo de uma GAN condicional. Adicionalmente, a saída do discriminador foi adaptada para que ele também gere a matriz de discriminação $(30 \times 30 \times 1)$.

A escolha desse discriminador em específico é pelo fato dele ser usado na base de dados CelebA-HQ para o treinamento da ProGAN, onde comprovadamente obtém sucesso na reconstrução das imagens naquela arquitetura.

Com essa alteração, o comportamento da *loss* de gerador durante o treinamento é bem mais instável, inclusive apresentando tendências de aumento da *loss* em alguns casos, como pode ser observado nos gráficos da 5.19.

A respeito das métricas de qualidade (Fig. 5.20), em todos os casos há uma tendência lateral, com pouca variação desde o início até o final do treinamento. Em comparação com os casos anteriores, que há clara redução em ambas as métricas conforme o treinamento avança, o que indicaria uma melhora na qualidade, neste caso os indicadores seguem o comportamento instável das *losses* de gerador. A exceção é a R03A, que usa o gerador residual original e que apresentou melhora consistente em ambos os gráficos.

Os resultados, na Tabela 5.5, mostram que apesar da instabilidade no treinamento a R03A foi inclusive superior à R01A, que tinha sido a melhor rede até então. Apesar disso, as redes que usam os geradores adaptados não conseguem superar a distância que os separam dos bons resultados dos geradores residuais originais.

Observando-se as imagens geradas através da Figura 5.21, as conclusões são similares. O residual original nessa configuração conseguiu gerar uma imagem muito próxima da

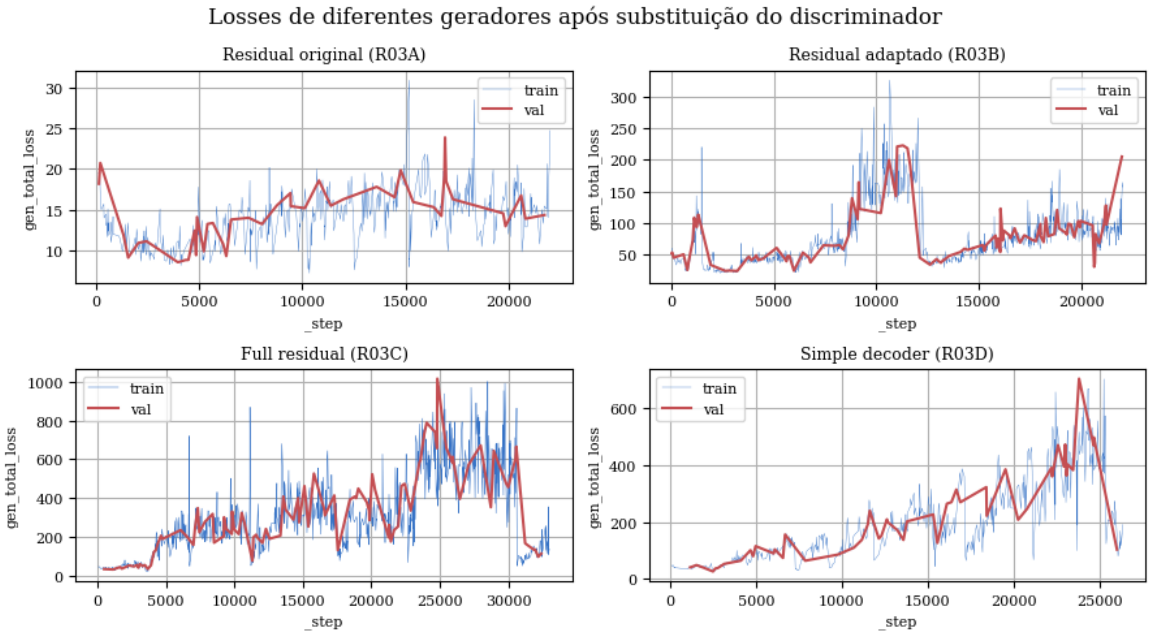

Figura 5.19: *Losses* de gerador utilizando discriminador ProGAN adaptado e *loss* PatchGAN. A linha azul representa a *loss* total calculada para a base de treinamento e a linha vermelha representa a *loss* total calculada na base de validação.

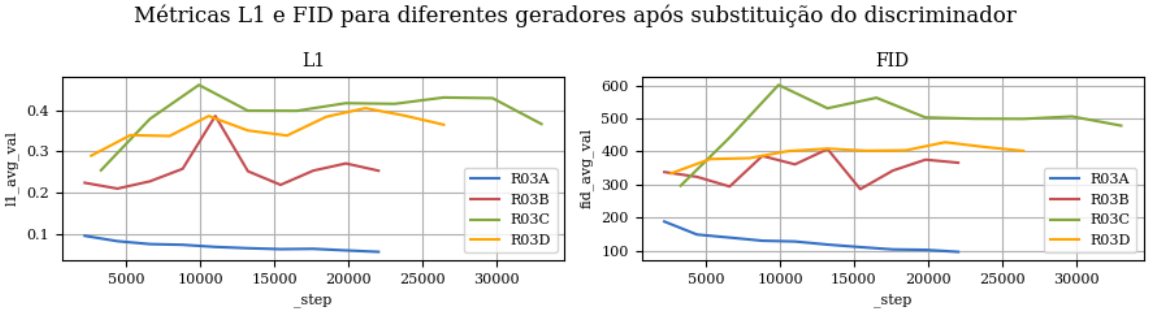

Figura 5.20: Métricas de qualidade com discriminador ProGAN adaptado e *loss* PatchGAN. A métrica FID representa a diferença entre a distribuição das imagens reais e sintéticas, e a métrica L1 representa a diferença média pixel a pixel de cada imagem sintética com seu respectivo objetivo.

original, com um pouco mais de desfoque principalmente no contorno, e com pequena diferença de cor. Os outros geradores, no entanto, tiveram desempenho claramente ruim, mesmo que tenham conseguido identificar e reproduzir diversas características presentes na imagem original.

EXP	Gerador	Discriminador	Loss	Normalização
R01A	Residual	PatchGAN	PatchGAN	BatchNorm
R03A	Residual	ProGAN	PatchGAN	BatchNorm
R03B	Residual adaptado	ProGAN	PatchGAN	InstanceNorm
R03C	*Full residual*	ProGAN	PatchGAN	InstanceNorm
R03D	*Simple decoder*	ProGAN	PatchGAN	InstanceNorm

EXP	*Batch size*	FID Teste	L1 Teste	Tempo infer. (s)	*Runtime* (h)
R01A	32	151,5434	0,0907	0,1896	1,4
R03A	12	**102,9289**	**0,0698**	0,1823	2,8
R03B	12	352,7598	0,2964	0,2020	2,9
R03C	8	484,5522	0,3653	0,2508	5,6
R03D	10	397,2816	0,3618	0,2136	3,7

Tabela 5.5: Resultados do treinamento utilizando discriminador ProGAN adaptado e *loss* PatchGAN. O Experimento R01A aparece na primeira linha como base de comparação.

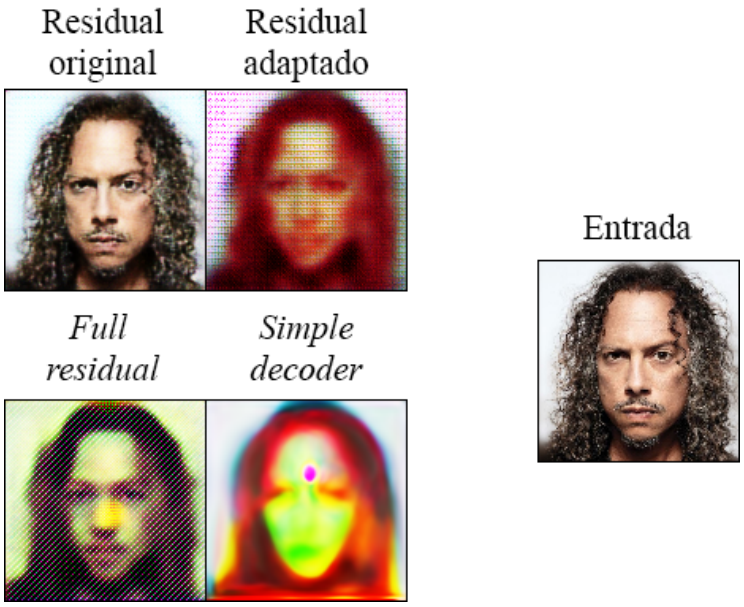

Figura 5.21: Resultados do treinamento utilizando discriminador ProGAN adaptado e *loss* PatchGAN. A imagem à direita foi apresentada a todas as redes como ”entrada”, e a saída da rede usando cada gerador é apresentada à esquerda.

Esse resultado sugere que o discriminador ProGAN pode trazer resultados melhores, apesar da instabilidade no treinamento. Arjovsky et al. [4] demonstram que essa instabilidade é intrínseca da *loss* adversária original baseada na divergência KL, usada na PatchGAN/Pix2Pix, e sugere uma alteração nessa *loss* para que seja utilizada a distância de

Wasserstein em seu lugar [5]. Em seguida experimentamos a combinação do discriminador ProGAN com a *loss* de Wasserstein.

5.3.4 Uso da *loss* da WGAN-GP

Como demonstrado por Arjovsky et al. [4], a *loss* adversária original é uma fonte de instabilidade no treinamento adversário das GANs, e propõe uma *loss* baseada na distância de Wasserstein para resolver esse problema [5, 22], chamando essa técnica de WGAN.

A versão dessa *loss* que usa um termo de penalidade de gradiente (ou *Wasserstein GAN with Gradient Penalty*, ou WGAN-GP) é a utilizada pela ProGAN em seu treinamento. Dessa forma, a loss do gerador (Eq. 3.1) e do discriminador (Eq. 3.2) Pix2Pix podem ser reescritas, respectivamente, como

$$\mathcal{L}_G = \mathcal{L}_{WGAN-GP}(G, D) + \lambda \|x_B - G(x_A)\|_1, \tag{5.1}$$

e

$$\mathcal{L}_D = \mathcal{L}_{WGAN-GP}(G, D). \tag{5.2}$$

Os experimentos desta seção tem as mesmas configurações dos experimentos da seção anterior, inclusive mantendo-se o discriminador ProGAN. A única alteração é a substituição da *loss* PatchGAN pela *loss* WGAN-GP, adaptada para o caso condicional. É importante ressaltar que, por se tratarem de métricas diferentes, o resultado numérico da *loss* WGAN não pode ser diretamente comparado com o da *loss* PatchGAN. Em vez disso, para se comparar os experimentos com diferentes *losses*, deve-se levar em consideração apenas as métricas de qualidade.

Os gráficos da Figura 5.22 mostram que o treinamento inicia com um pico muito alto da loss e rapidamente decresce até um patamar mais baixo, reduzindo levemente com o

tempo. Em comparação com o experimento anterior, nota-se que as curvas apresentam um comportamento mais estável. Comparando-se os quatro exemplos deste experimento, todos alcançam um patamar de *loss* muito próximo, entre 100 e 200, exceto o que usa o gerador residual original (R04A) que chega em um patamar inferior a 50.

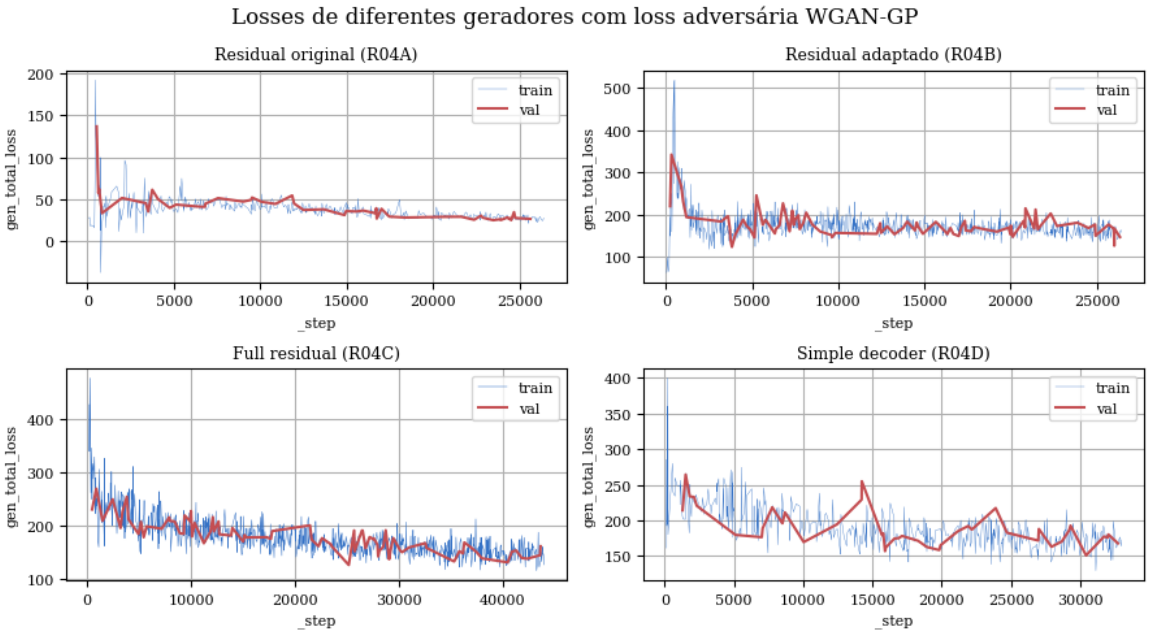

Figura 5.22: *Losses* de gerador utilizando discriminador ProGAN adaptado e *loss* WGAN-GP. A linha azul representa a *loss* total calculada para a base de treinamento e a linha vermelha representa a *loss* total calculada na base de validação.

As métricas de qualidade também apresentam melhora com o tempo, conforme gráficos da Figura 5.23. No entanto, assim como nos casos anteriores, há uma diferença intransponível entre as métricas de qualidade aferidas nas redes com geradores adaptados em relação à rede com gerador residual original.

No experimento anterior, a rede R03A apresentou os melhores resultados usando o discriminador ProGAN, apesar do treinamento instável. Agora, corrigida a instabilidade, a rede R04A superou esse resultado, alcançando pela primeira vez neste trabalho um FID abaixo de 100 e uma L1 abaixo de 0,05, como pode ser observado com a Tabela tab:R04-Results. As outras redes (R04B, R04C e R04D) também apresentaram os melhores resultados dentre as redes com geradores adaptados, em comparação com os experimentos anteriores.

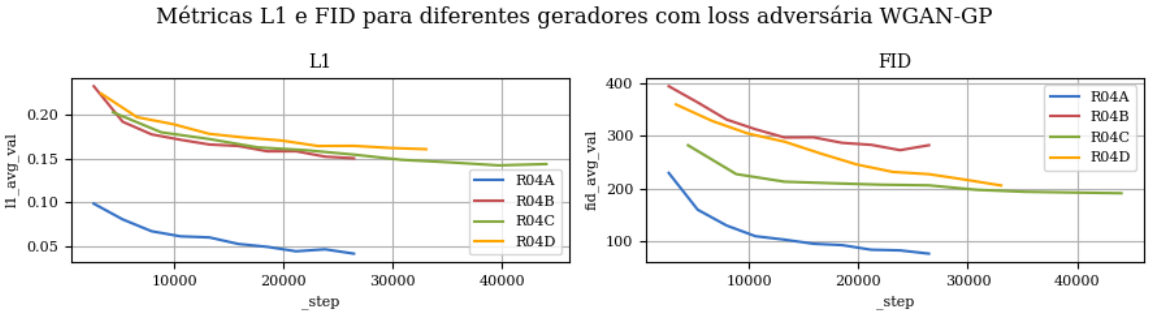

Figura 5.23: Métricas de qualidade com discriminador ProGAN adaptado e *loss* WGAN-GP. A métrica FID representa a diferença entre a distribuição das imagens reais e sintéticas, e a métrica L1 representa a diferença média pixel a pixel de cada imagem sintética com seu respectivo objetivo.

Um lado negativo dessa implementação é que ela usa mais memória no treinamento, e portanto os *batch sizes* tiveram que ser reduzidos, ocasionando em treinamentos mais lentos.

EXP	Gerador	Discriminador	Loss	Normalização
R03A	Residual	ProGAN	PatchGAN	BatchNorm
R04A	Residual	ProGAN	WGAN-GP	BatchNorm
R04B	Residual adaptado	ProGAN	WGAN-GP	InstanceNorm
R04C	*Full residual*	ProGAN	WGAN-GP	InstanceNorm
R04D	*Simple decoder*	ProGAN	WGAN-GP	InstanceNorm

EXP	*Batch size*	FID Teste	L1 Teste	Tempo infer. (s)	*Runtime* (h)
R03A	12	102,9289	0,0698	0,1823	2,8
R04A	10	**79,8176**	**0,0456**	0,1962	4,8
R04B	10	280,1921	0,1509	0,2202	5,7
R04C	6	187,0715	0,1386	0,2418	8,0
R04D	8	204,0791	0,1555	0,2081	5,5

Tabela 5.6: Resultados do treinamento utilizando discriminador ProGAN adaptado e *loss* WGAN-GP. O Experimento R03A aparece na primeira linha como base de comparação.

Assim como nos casos anteriores, as imagens sintéticas (Fig. 5.24) também apresentam características relevantes da imagem original. A imagem sintetizada pelo gerador residual original mostra maior nitidez que as anteriores, embora ainda tenha pequenas diferenças em relação à imagem de entrada, principalmente na cor. A imagem criada pelo residual adaptado apresenta um artefato no formato de uma mancha escura no topo, e artefatos

como este também aparecem em outras imagens geradas por essa configuração. Novamente, o gerador *full residual* demonstrou uma capacidade de síntese melhor que os demais, como comprovado visualmente e pelas métricas da Tabela 5.6.

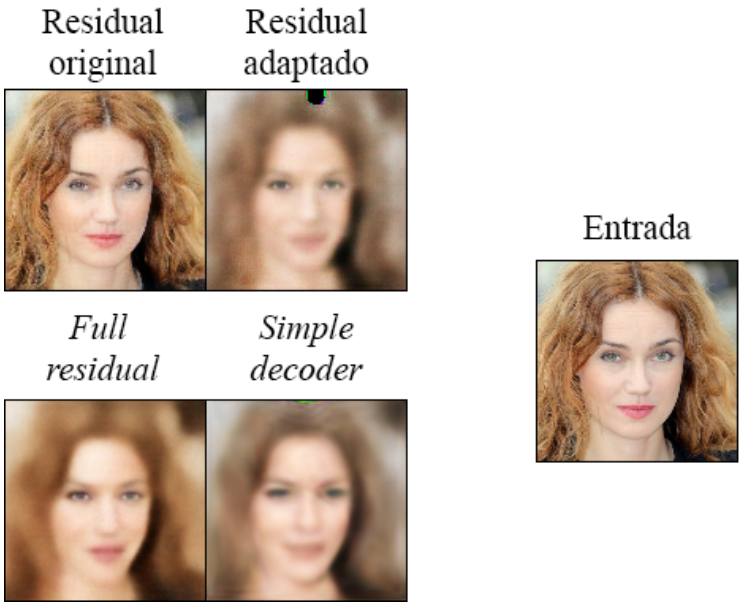

Figura 5.24: Resultados do treinamento utilizando discriminador ProGAN adaptado e *loss* WGAN-GP. A imagem à direita foi apresentada a todas as redes como "entrada", e a saída da rede usando cada gerador é apresentada à esquerda.

O fato de que características da face, inclusive a pose da cabeça, estão presentes nas imagens sintetizadas, é um indicativo de que o *encoder* está conseguindo mandar informações relevantes para o *decoder* através do vetor latente. Aproveitando essa ocorrência, no próximo experimento exploramos uma arquitetura diferente na geração do vetor latente.

5.3.5 Inclusão da *Mapping Network* para desemaranhamento

Os autores da StyleGAN [33] propuseram uma técnica chamada "*mapping network*", indicada na Fig. 3.8, que consiste em utilizar 8 camadas MLP "*fully connecetd*" (FC ou `Dense`) que fazem o mapeamento do espaço latente original Z para um espaço latente

intermediário W. De acordo com os autores, essa técnica permite que a rede aprenda a representar as características em um espaço latente desemaranhado.

Um possível problema na arquitetura dos geradores propostos neste trabalho é que a transição da última camada de convolução para a criação do vetor latente é feita em um único passo, a partir de uma única camada `Dense`, como ilustrado na Figura 5.25 à esquerda. Isso pode fazer com que o gerador tenha poucos parâmetros para ajustar na criação desse vetor, e a representação não seja a melhor possível.

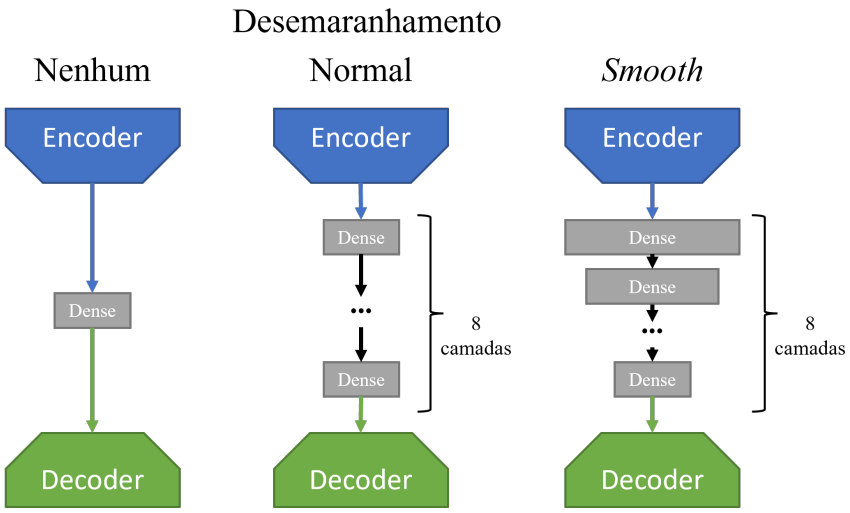

Figura 5.25: Alterações nos geradores para melhoria da geração do vetor latente. A rede à esquerda representa a concepção inicial dos geradores, em que a transição entre o *encoder* e o *decoder* utiliza apenas uma camada `Dense`. Ao centro, a rede com desemaranhamento "normal" utiliza oito camadas `Dense` com a mesma quantidade de elementos do vetor gerado, e à direita, a rede "*Smooth*" utiliza também as oito camadas, mas com a quantidade de elementos decrescendo gradualmente.

Como uma tentativa de induzir o gerador a desenvolver uma melhor representação do espaço latente, podemos incluir um módulo de desemaranhamento nos *autoencoders*, que assim como a *mapping network* da StyleGAN, também consiste em oito camadas `Dense` antes do *decoder*. Propusemos duas formas de fazer isso, o desemaranhamento *Normal* que usa as oito camadas com a mesma quantidade de elementos que o vetor latente, e o *Smooth*, em que a quantidade de elementos das camadas vão gradualmente reduzindo, como detalhado na Figura 5.25.

A hipótese nesse caso é que, além de criar um espaço desemaranhado, as camadas adicionais podem também aprender melhor a registrar no vetor latente as informações extraídas pelo *encoder*, e dessa forma permitir que o *decoder* tenha toda a informação necessária para a reconstrução da imagem.

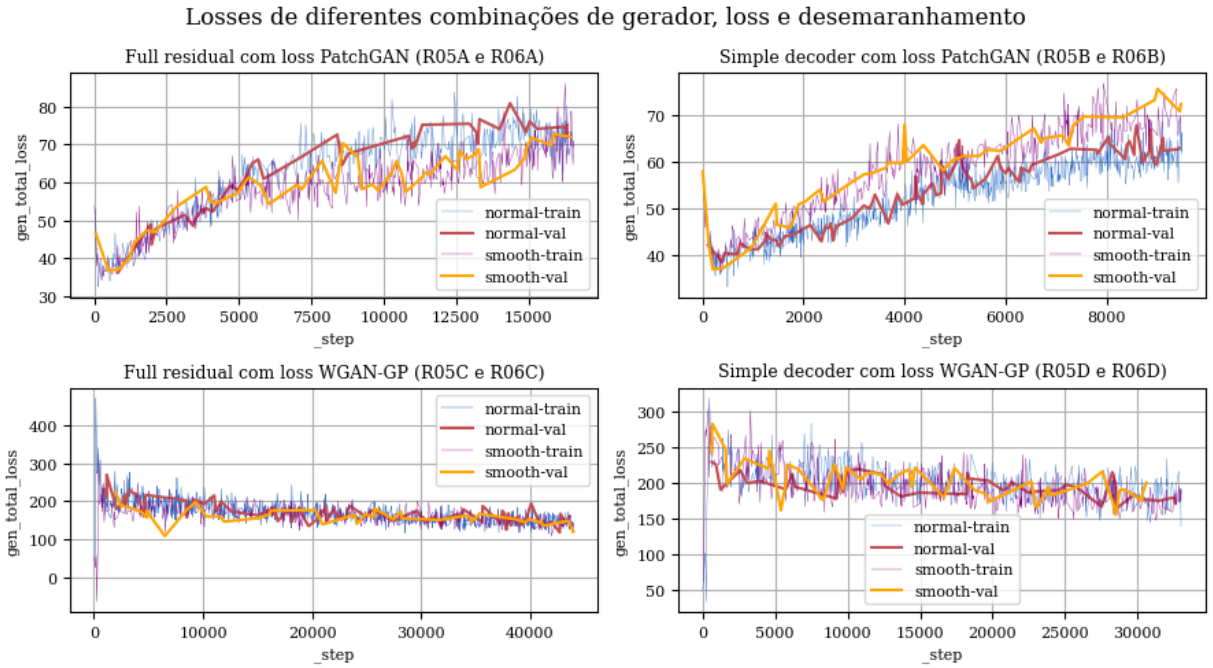

Figura 5.26: *Losses* de gerador utilizando desemaranhamento. O tipo de gerador e de loss utilizados estão descritos nos títulos de cada gráfico. Em cada combinação foi testado o uso de desemmaranhamento *normal* e *smooth*, representados em cada gráfico por linhas de cor diferente.

Os gráficos com as losses de gerador ao longo do treinamento estão representados na Figura 5.26. A proximidade das linhas das *losses* sugere que não há diferença entre usar qualquer um dos dois tipos de desemaranhamento com a *loss* WGAN-GP. No caso em que a *loss* PatchGAN foi utilizada com o gerador *full residual*, o desemaranhamento *smooth* apresentou uma *loss* menor ao longo de parte do treinamento, mas terminou a última época com *loss* muito próxima à apresentada pelo desemaranhamento normal. Por fim, na combinação com gerador simple decoder e loss PatchGAN, o desemaranhamento normal apresentou *loss* menor do que o desemaranhamento *smooth* ao longo de todo o treinamento. Na Figura 5.27 verificamos que os experimentos R05C e R06C apresentaram a maior redução na distância L1, e também os melhores resultados de FID.

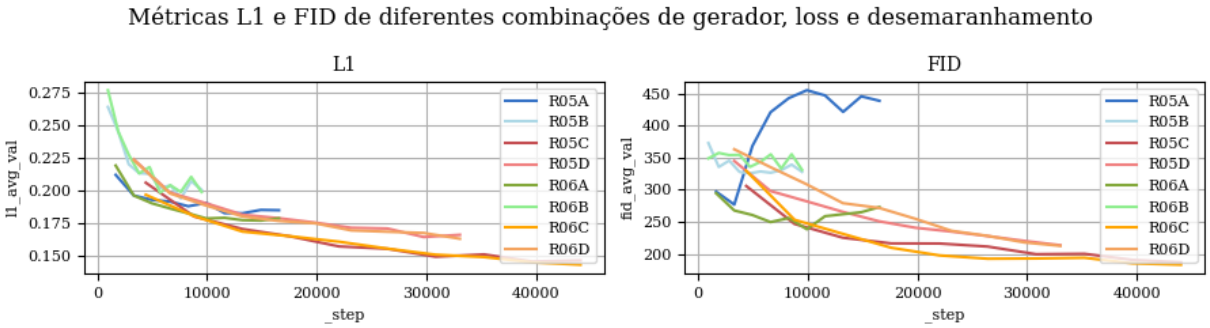

Figura 5.27: Métricas de qualidade utilizando desemaranhamento. Cada sigla da legenda equivale a um experimento, e suas configurações estão descritas na Tabela 5.7. A métrica FID representa a diferença entre a distribuição das imagens reais e sintéticas, e a métrica L1 representa a diferença média pixel a pixel de cada imagem sintética com seu respectivo objetivo.

Ao se comparar os resultados apresentados na Tabela 5.7, fica claro que algumas dessas estratégias pioraram consideravelmente as métricas de qualidade que avaliamos, com o pior caso ultrapassando os 400 pontos de FID. Dentre os métodos com desemaranhamento, o melhor foi o R06C, que utilizou o gerador *full residual* com *loss* WGAN-GP e desemaranhamento *smooth*, e que apresentou melhores resultados do que os experimentos R04B, R04C e R04D, sendo então a melhor arquitetura dentre as que geram vetores latentes. Ainda assim, nenhum desses métodos ultrapassou nossa base de comparação do experimento R04A.

Observa-se também que os métodos com desemaranhamento *smooth* apresentaram melhores resultados em relação aos seus equivalentes com desemaranhamento normal. Isso pode ser devido à maior quantidade de parâmetros nas primeiras camadas que realiza uma transição mais suave no método *smooth*.

Alguns exemplos gerados pelos experimentos desta seção estão na Figura 5.28. Nos casos com gerador *full residual* e *loss* PatchGAN há a presença de artefatos quadriculados. Nos casos com gerador *simple decoder* e *loss* PatchGAN, há também a presença de artefatos circulares e predominância da cor rosa. A combinação *full residual* + *loss* WGAN-GP foi a que teve melhores resultados, mas não conseguiram reconstruir corretamente a imagem de entrada.

EXP	Gerador	Discriminador	Loss	Normalização	Desemaranhamento
R04A	Residual	ProGAN	WGAN-GP	BatchNorm	-
R05A	Full residual	PatchGAN	PatchGAN	InstanceNorm	Normal
R05B	Simple decoder	PatchGAN	PatchGAN	InstanceNorm	Normal
R05C	Full residual	ProGAN	WGAN-GP	InstanceNorm	Normal
R05D	Simple decoder	ProGAN	WGAN-GP	InstanceNorm	Normal
R06A	Full residual	PatchGAN	PatchGAN	InstanceNorm	Smooth
R06B	Simple decoder	PatchGAN	PatchGAN	InstanceNorm	Smooth
R06C	Full residual	ProGAN	WGAN-GP	InstanceNorm	Smooth
R06D	Simple decoder	ProGAN	WGAN-GP	InstanceNorm	Smooth

EXP	*Batch size*	FID Teste	L1 Teste	Tempo infer. (s)	*Runtime* (h)
R04A	10	79,8176	0,0456	0,1962	4,8
R05A	16	434,8664	0,1817	0,2423	3,2
R05B	28	327,8972	0,1953	0,2422	1,9
R05C	6	181,0428	0,1426	0,2360	7,8
R05D	8	210,9448	0,1619	0,2086	5,5
R06A	16	270,7976	0,1748	0,2475	3,1
R06B	28	329,7352	0,1931	0,2167	1,7
R06C	6	**174,4460**	**0,1385**	0,2410	7,5
R06D	8	205,2359	0,1603	0,2313	5,7

Tabela 5.7: Resultados do treinamento utilizando desemaranhamento. O Experimento R04A aparece na primeira linha como base de comparação.

5.3.6 *Transfer learning*

Até o momento, o experimento que apresentou os melhores resultados de reconstrução foi o R04A, que utilizou o gerador residual original com a *loss* WGAN-GP. Infelizmente esse gerador não tem em si a criação de um vetor latente, o que dificulta a tarefa de manipulação.

Podemos utilizar a técnica de transfer learning, utilizando o *autoencoder* R04A original já treinado como base e incluir um trecho novo no centro, que corresponde às camadas de geração do vetor latente e às primeiras camadas de reconstrução da imagem, como exemplificado na Figura 5.29. Os parâmetros do *encoder* e *decoder* da rede pré treinada se mantém fixos e apenas os parâmetros do trecho central passa pelo treinamento. Após a convergência, pode-se liberar todos os parâmetros para o treinamento da rede completa de forma a se realizar um ajuste fino no modelo.

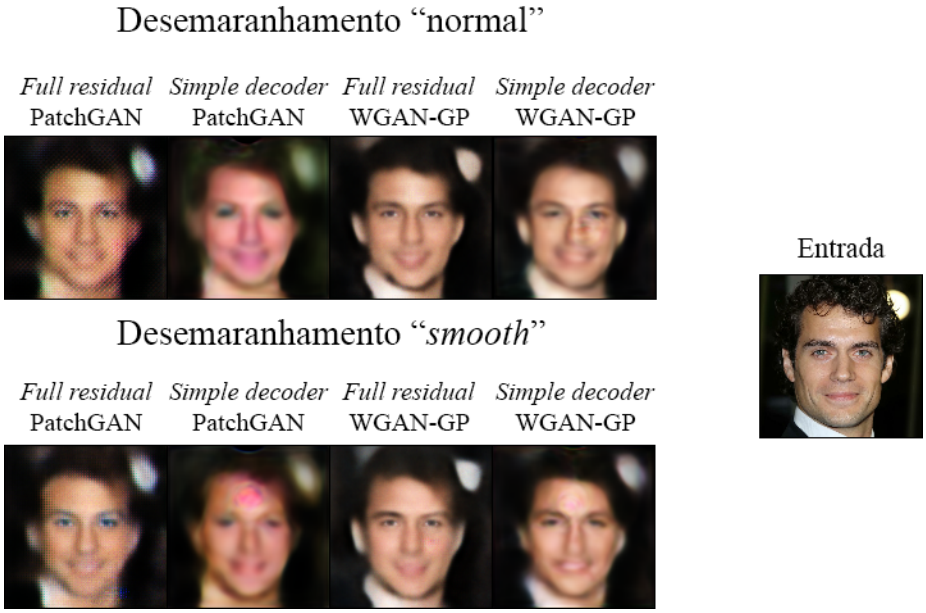

Figura 5.28: Resultados do treinamento utilizando desemaranhamento. A imagem à direita foi apresentada a todas as redes como "entrada", e a saída da rede usando cada configuração é apresentada à esquerda.

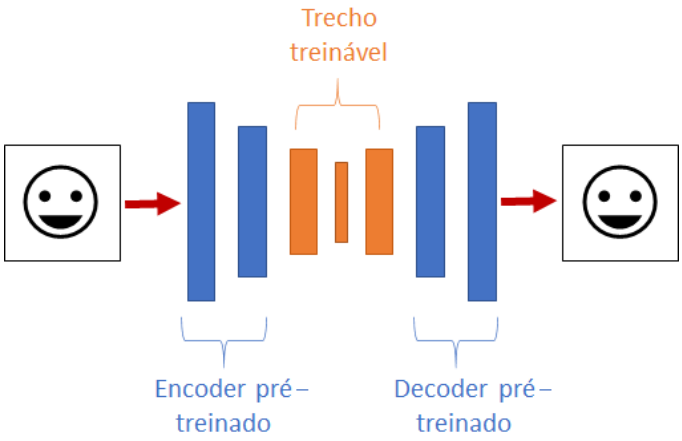

Figura 5.29: Uso de *Transfer Learning* na adaptação do *autoencoder* ResNet. O *autoencoder* original, sem adaptações, é pré-treinado e usado como base (azul) na construção do novo gerador. Um trecho treinável (laranja) é então acrescentado para transformar a saída do *encoder* original em um vetor latente e criar as primeiras camadas de reconstrução do *decoder*. No primeiro momento apenas o trecho central é treinado, mas após a convergência a rede toda pode ser treinada em conjunto para ajuste fino.

Como detalhado na Seção 2.3.2, há duas formas principais de se fazer a ampliação da imagem entre as camadas: diretamente por meio de convolução transposta (`Conv2DTranspose`), ou realizando um redimensionamento clássico (Bilinear) seguido de

uma convolução. Ambos os métodos foram avaliados na criação das camadas do *decoder* do novo trecho central. Todos os experimentos desta seção foram feitos com desemaranhamento *smooth*, e as configurações deles estão na Tabela 5.8.

O experimento R07E foi realizado utilizando-se as mesmas configurações do R07B, porém com todas as camadas liberadas para treinamento desde o início, e não apenas as novas. O R07B foi escolhido como base por ter apresentado os melhores resultados dentre os que utilizaram *transfer learning*.

A evolução das *losses* de gerador ao longo do treinamento está nos gráficos da Figura 5.30. Nos casos com a *loss* WGAN-GP houve pouca diferença entre as duas estratégias de redimensionamento no *decoder*. No entanto, a convolução transposta apresentou um patamar de *loss* mais baixo nos casos com a *loss* PatchGAN. Ao se comparar os experimentos R07B e R07E (Fig. 5.30 inferior), percebe-se que ao se liberar o treinamento de todas as camadas como foi feito no R07E, a *loss* pode ser reduzida em comparação com o caso anterior.

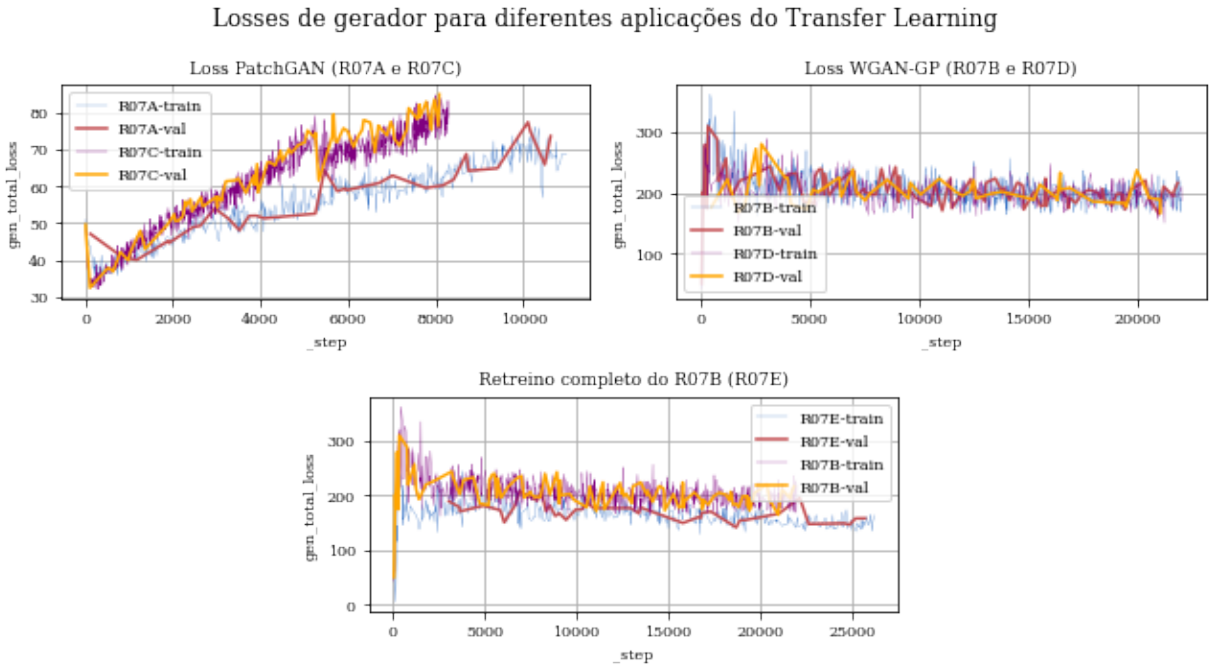

Figura 5.30: *Losses* de gerador utilizando *transfer learning*. As configurações de cada experimento estão detalhadas na Tabela 5.8

Os gráficos da Figura 5.30 indicam que o experimento R07E conseguiu reduzir ambas as métricas a um nível melhor do que os outros experimentos, sugerindo que treinar todas as camadas desde o começo, inclusive as pré treinadas, pode trazer benefícios. Apesar disso, observando-se os resultados descritos na Tabela 5.8, verificamos que os resultados obtidos por esse método ainda são inferiores aos obtidos anteriormente.

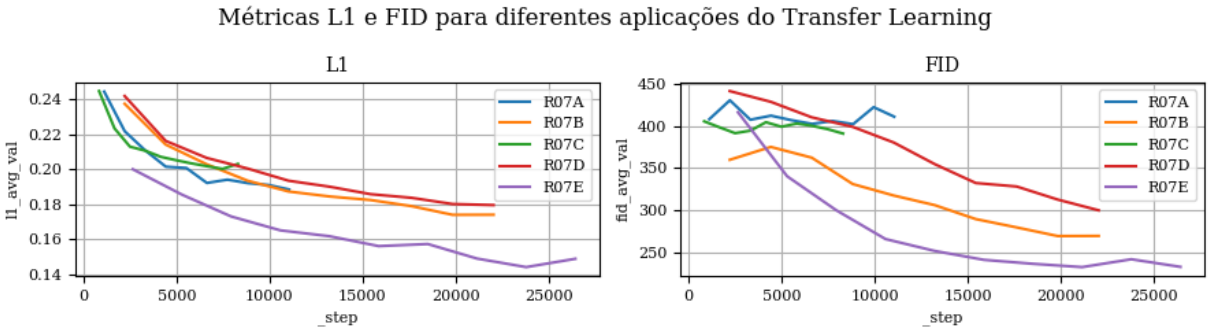

Figura 5.31: Métricas de qualidade utilizando *transfer learning*. Cada sigla da legenda equivale a um experimento, e suas configurações estão descritas na Tabela 5.8. A métrica FID representa a diferença entre a distribuição das imagens reais e sintéticas, e a métrica L1 representa a diferença média pixel a pixel de cada imagem sintética com seu respectivo objetivo.

EXP	Gerador	Discriminador	Loss	Normalização	Upsampling
R04A	Residual	ProGAN	WGAN-GP	BatchNorm	-
R07A	Transfer R04A	PatchGAN	PatchGAN	InstanceNorm	`Conv2DTranspose`
R07B	Transfer R04A	ProGAN	WGAN-GP	InstanceNorm	`Conv2DTranspose`
R07C	Transfer R04A	PatchGAN	PatchGAN	InstanceNorm	Bilinear
R07D	Transfer R04A	ProGAN	WGAN-GP	InstanceNorm	Bilinear
R07E	Transfer R04A	ProGAN	WGAN-GP	InstanceNorm	`Conv2DTranspose`

EXP	*Batch size*	FID Teste	L1 Teste	Tempo infer. (s)	*Runtime* (h)
R04A	10	79,8176	0,0456	0,1962	4,8
R07A	24	416,2222	0,1863	0,1872	1,3
R07B	12	273,3008	0,1712	0,1931	4,0
R07C	32	405,9852	0,2004	0,1979	1,4
R07D	12	306,5646	0,1759	0,1980	4,2
R07E	10	229,5790	0,1468	0,1796	5,1

Tabela 5.8: Resultados do treinamento utilizando *transfer learning*. O Experimento R04A aparece na primeira linha como base de comparação.

A Figura 5.32 apresenta algumas imagens sintéticas criadas utilizando o *transfer learning*. A baixa qualidade nas imagens e os erros de reconstrução reforçam os resultados observados nas métricas de qualidade.

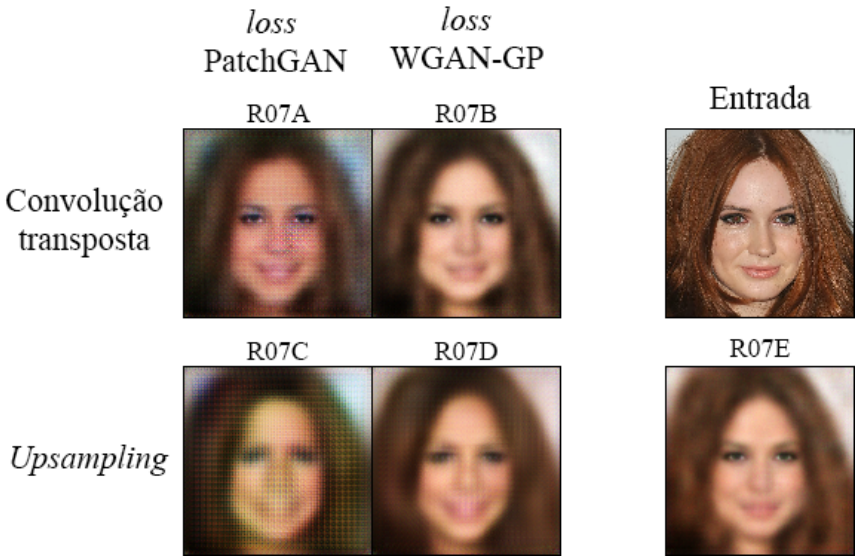

Figura 5.32: Resultados do treinamento utilizando *transfer learning*. A imagem do canto superior direito foi apresentada a todas as redes como "entrada", e a saída da rede usando cada configuração é apresentada com seu respectivo código de experimento acima.

5.3.7 Comparação do método adversário com não adversário

Sabe-se que o treinamento de uma rede generativa utilizando-se apenas a *loss* L1 gera imagens menos nítidas, e que incluindo-se o treinamento adversário a qualidade da imagem é melhorada [28]. Apesar disso, podemos testar o impacto dessa abordagem na nossa arquitetura.

Para esse experimento, removemos o discriminador e treinamos os geradores apenas com a *loss* L1, sem a componente adversária. Os geradores que geram vetores latentes tiveram o desemaranhamento *smooth* incluído e as configurações de cada experimento estão detalhadas na Tabela 5.9

Os gráficos da Figura 5.33, indicam uma clara e rápida redução nas *losses* de gerador, convergindo rapidamente para valores próximos do valor final do experimento. As métricas de qualidade, na Figura 5.34 também seguem esse comportamento, indicando que maior parte do ganho de qualidade com o treinamento do gerador ocorre já nas primeiras iterações.

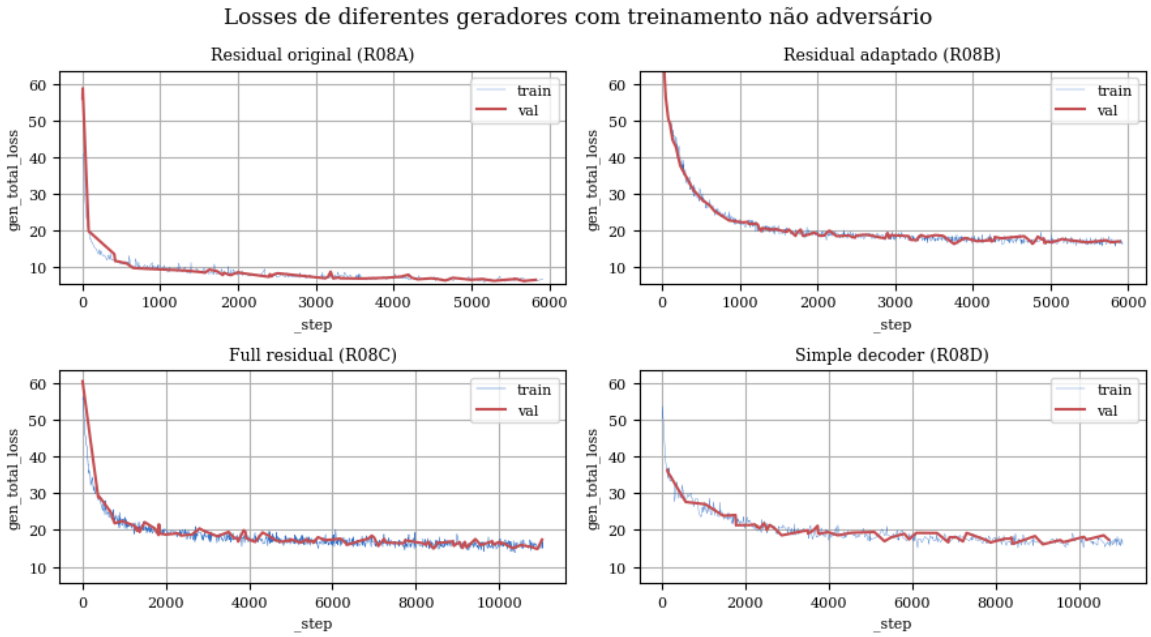

Figura 5.33: *Losses* de gerador com treinamento não adversário. As configurações de cada experimento estão detalhadas na Tabela 5.9

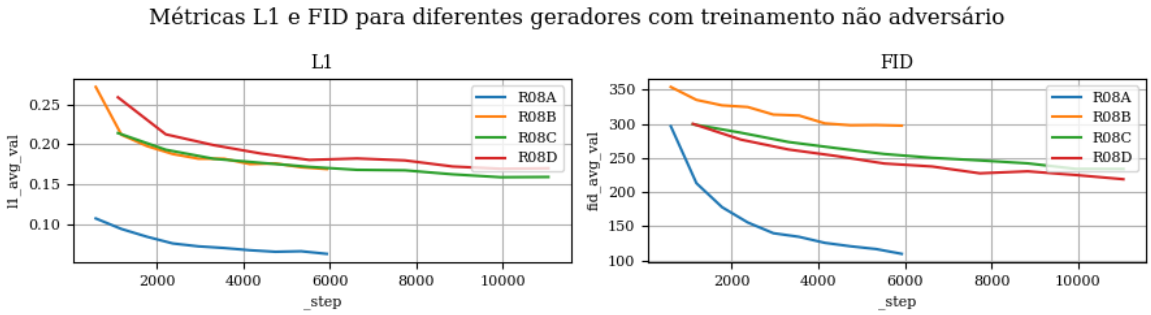

Figura 5.34: Métricas de qualidade com treinamento não adversário. Cada sigla da legenda equivale a um experimento, e suas configurações estão descritas na Tabela 5.9. A métrica FID representa a diferença entre a distribuição das imagens reais e sintéticas, e a métrica L1 representa a diferença média pixel a pixel de cada imagem sintética com seu respectivo objetivo.

Como esperado, os resultados expressos na Tabela 5.9 indicam que o resultado do experimento não adversário com gerador que não cria um vetor latente (R08A) é inferior ao

mesmo experimento com treinamento adversário (R04A). Os resultados dos outros geradores também são inferiores aos melhores casos obtidos anteriormente (R06C, Tabela 5.7). Isso reforça que a estratégia adversária auxilia na criação de imagens de melhor qualidade segundo as métricas de avaliação. No entanto, um benefício da estratégia não adversária é o tempo de treinamento (*runtime*) mais rápido, por ter uma rede a menos a ser treinada e os cálculos da *loss* serem mais simples.

EXP	Gerador	Discriminador	Loss	Normalização	Desemaranhamento
R04A	Residual	ProGAN	WGAN-GP	BatchNorm	-
R08A	Residual	-	L1	BatchNorm	-
R08B	Residual adaptado	-	L1	InstanceNorm	Smooth
R08C	Full residual	-	L1	InstanceNorm	Smooth
R08D	Simple decoder	-	L1	InstanceNorm	Smooth

EXP	*Batch size*	FID Teste	L1 Teste	Tempo infer. (s)	*Runtime* (h)
R04A	10	79,8176	0,0456	0,1962	4,8
R08A	45	120,5322	0,0860	0,1777	0,8
R08B	45	298,1232	0,1671	0,2613	1,2
R08C	24	228,0921	0,1547	0,2396	2,3
R08D	24	217,5620	0,1667	0,2084	1,0

Tabela 5.9: Resultados do treinamento não adversário. O Experimento R04A aparece na primeira linha como base de comparação.

Observando algumas imagens geradas (Fig. 5.35), verificamos que a criada pelo gerador residual original (que não tem um vetor latente) é boa, mas carece de nitidez. As imagens sintetizadas pelos outros geradores estão com problemas claros de reconstrução, mesmo que consigam registrar claramente o formato e posição da cabeça.

5.3.8 Taxas de aprendizado diferentes

Arjovsky e Bottou demonstraram que quando o discriminador se especializa muito rapidamente, o gradiente do gerador é reduzido, ocasionando no problema de dissipação do

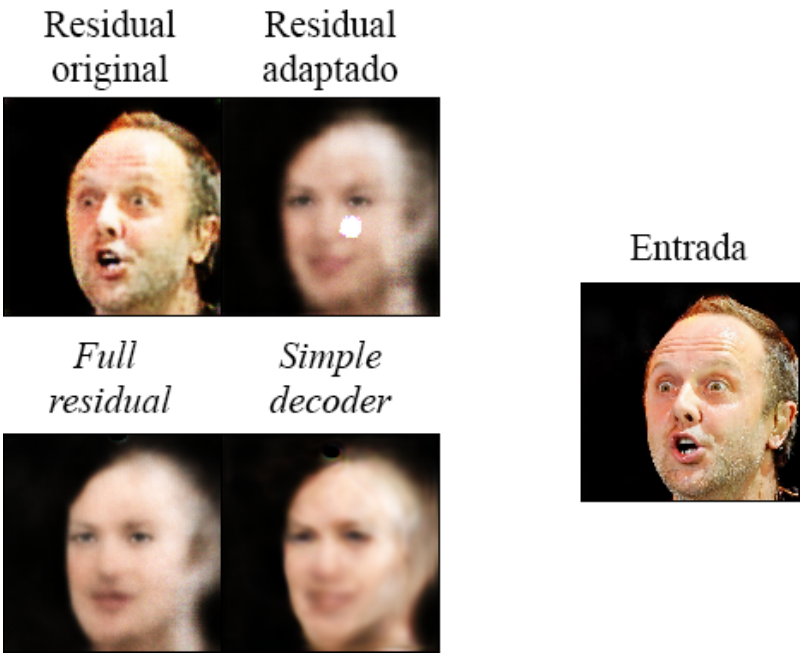

Figura 5.35: Resultados do treinamento não adversário. A imagem à direita foi apresentada a todas as redes como "entrada", e a saída da rede usando cada configuração é apresentada à esquerda.

gradiente e em uma ineficácia de se treinar o gerador [4]. Em resumo, um discriminador ótimo não fornece informação suficiente para que o gerador aprenda a melhorar.

Como uma forma de avaliar esse efeito em nosso *autoencoder*, realizamos um experimento que consistia em defasar em uma ordem de grandeza a taxa de aprendizado do discriminador (α_D) da taxa de aprendizado do gerador (α_G), de forma que o discriminador aprenda mais lentamente do que o gerador.

Para efeitos de comparação utilizamos apenas duas arquiteturas: o gerador residual original com *loss* WGAN-GP e o gerador *full residual* com *loss* WGAN-GP e desemaranhamento *smooth*. A Tabela 5.10 explicita as configurações de cada experimento.

A Figura 5.36 exibe os gráficos das *losses* de gerador alinhados com os gráficos das *losses* de discriminador.

O experimento R09A tem a mesma taxa de aprendizado do gerador que o R04A, nosso controle, mas uma taxa de aprendizado dez vezes menor no discriminador. A *loss* do

discriminador se mantém estável até por volta da iteração 20.000, onde apresenta um pico que afeta também a *loss* do gerador. Isso se deve, provavelmente, ao atraso do discriminador em aprender a diferenciar as imagens reais das sintéticas.

Comportamento semelhante ocorre no experimento R09B, que tem uma taxa de aprendizado de discriminador igual à do R04A, mas uma taxa de aprendizado de gerador dez vezes maior. Nesse caso o pico ocorre por volta da iteração 5.000.

Nos experimentos com gerador *full residual*, todos mantiveram o mesmo comportamento ao longo de todo o treinamento, exceto o experimento R09E que apresenta uma tendência de aumento lento e gradual da *loss* de gerador e simultaneamente de diminuição da *loss* de discriminador. Essa configuração foi a que apresentou os piores resultados, conforme a Tabela 5.10.

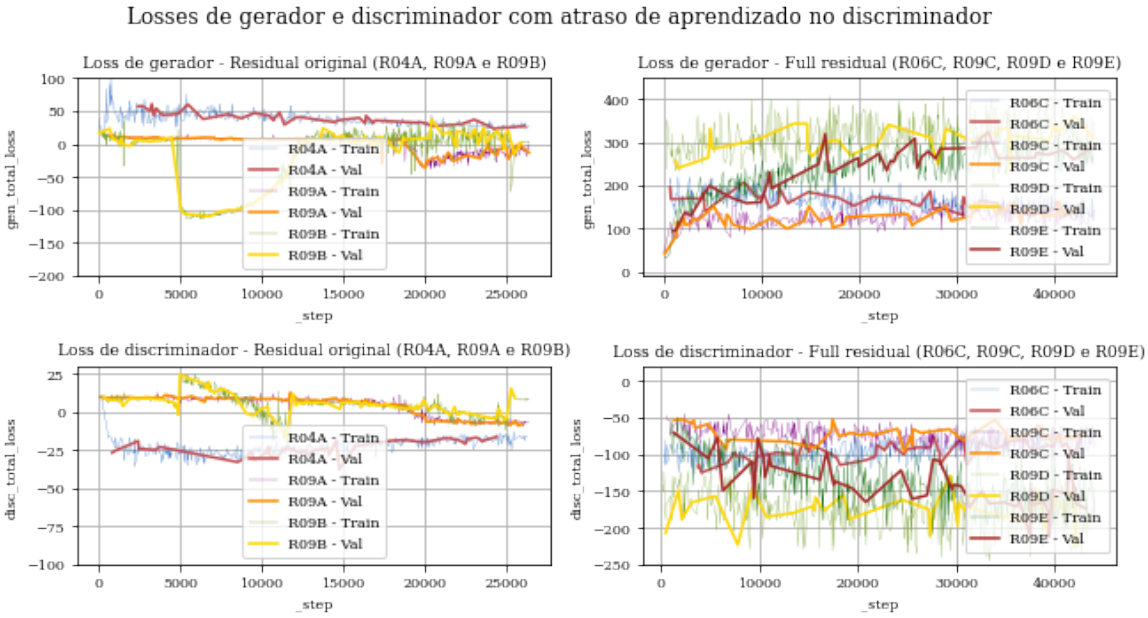

Figura 5.36: *Losses* de gerador variando-se as taxas de aprendizado. As configurações de cada experimento estão detalhadas na Tabela 5.10

A evolução das métricas de qualidade, expressas nos gráficos da Figura 5.37, apresentaram um comportamento muito similar entre si com exceção do experimento R09B que teve um pico por volta da iteração 5.000, condizendo com o pico nas *losses*. A partir do momento em que o discriminador começa a atuar mais fortemente as métricas são reduzidas.

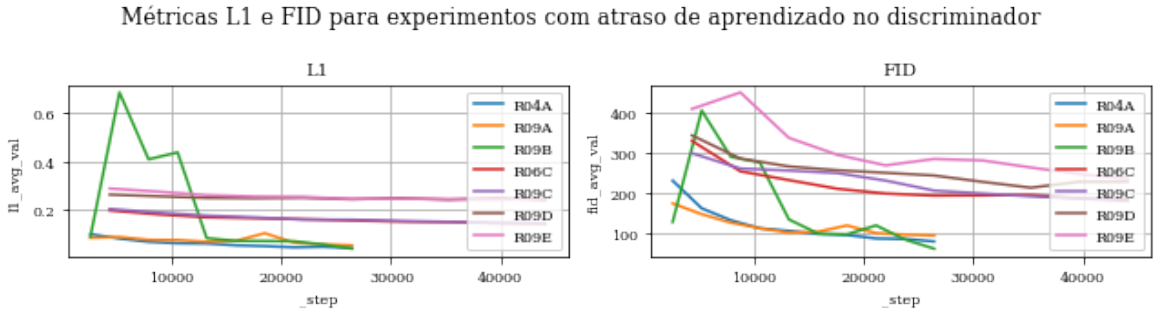

Figura 5.37: Métricas de qualidade variando-se as taxas de aprendizado. Cada sigla da legenda equivale a um experimento, e suas configurações estão descritas na Tabela 5.10. A métrica FID representa a diferença entre a distribuição das imagens reais e sintéticas, e a métrica L1 representa a diferença média pixel a pixel de cada imagem sintética com seu respectivo objetivo.

De acordo com os resultados da Tabela 5.10, o experimento R09B conseguiu melhorar o FID do experimento R09A, mantendo o L1 em um valor próximo. No grupo de experimentos com o gerador *full residual*, o R09C foi o que chegou mais próximo do resultado do R06C, nosso controle.

EXP	Gerador	Discriminador	Loss	Normalização	Desemaranhamento
R04A	Residual	ProGAN	WGAN-GP	BatchNorm	-
R06C	Full residual	ProGAN	WGAN-GP	InstanceNorm	*Smooth*
R09A	Residual	ProGAN	WGAN-GP	BatchNorm	-
R09B	Residual	ProGAN	WGAN-GP	BatchNorm	-
R09C	Full residual	ProGAN	WGAN-GP	InstanceNorm	*Smooth*
R09D	Full residual	ProGAN	WGAN-GP	InstanceNorm	*Smooth*
R09E	Full residual	ProGAN	WGAN-GP	InstanceNorm	*Smooth*

EXP	*Batch size*	FID Teste	L1 Teste	Tempo infer. (s)	*Runtime* (h)	α_G	α_D
R04A	10	79,8176	0,0456	0,1962	4,8	1×10^{-5}	1×10^{-5}
R06C	6	174,4460	0,1385	0,2410	7,5	1×10^{-5}	1×10^{-5}
R09A	10	91,9607	0,0534	0,1835	4,4	1×10^{-5}	1×10^{-6}
R09B	10	**68,3303**	0,0476	0,1898	4,4	1×10^{-4}	1×10^{-5}
R09C	6	175,1687	0,1405	0,2339	7,4	1×10^{-5}	1×10^{-6}
R09D	6	226,4603	0,2364	0,2313	7,4	1×10^{-4}	1×10^{-5}
R09E	6	234,2565	0,2368	0,2383	7,5	1×10^{-4}	1×10^{-6}

Tabela 5.10: Resultados do treinamento variando-se as taxas de aprendizado. Os Experimentos R04A e R06C aparecem nas primeiras linhas como base de comparação.

Visualmente, as imagens geradas pelo gerador residual original (Fig. 5.38, esquerda) são muito similares entre si e em comparação com a imagem de entrada. Por outro lado, as imagens geradas pelo gerador *full residual* (Fig. 5.38, centro) tem muitas diferenças em relação à entrada, e apenas uma é similar ao nosso controle.

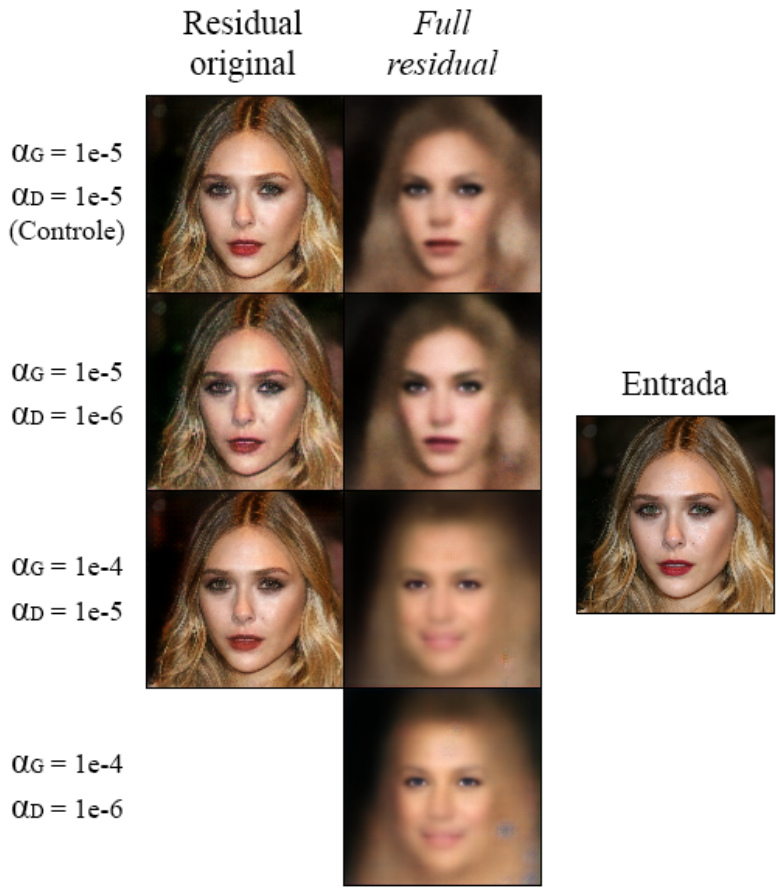

Figura 5.38: Resultados do treinamento variando-se as taxas de aprendizado. A imagem à direita foi apresentada a todas as redes como "entrada", e a saída da rede usando cada configuração é apresentada à esquerda.

5.3.9 Treinamento longo com o melhor *autoencoder*

Optamos por limitar em 10 épocas o treinamento dos experimentos deste capítulo, por conta de restrições de tempo e processamento. No entanto, pode-se argumentar que

esse tempo de treinamento seja insuficiente, e que melhores resultados podem ser obtidos com treinamentos mais longos.

Dentre os *autoencoders* que geram vetores latentes, o que teve o melhor desempenho foi o do experimento R06C, que utilizou gerador *full residual* com desemaranhamento *smooth* e loss WGAN-GP. Ao se observar a *loss* do gerador desse experimento na Fig. 5.26, nota-se que há uma leve tendência de queda e que provavelmente ela ainda não convergiu para seu valor final.

Podemos testar essa hipótese de uma forma simples: repetir o treinamento desse melhor experimento, aumentando o limite de épocas. Conforme a Tabela 5.11, foi usado o experimento R06C como base, mas os novos experimentos, R11B e R11C, foram treinados com 25 e 100 épocas de treinamento, respectivamente, em vez das 10 épocas originais.

Entretanto, por conta das limitações do equipamento utilizado até então, o experimento R11C foi feito num computador diferente, equipado com uma GPU RTX 3060 de 12GB de VRAM. Todos os aspectos de *software*, como versão do Python ou do Tensorflow, foram mantidos iguais.

Os gráficos da Figura 5.39 mostram que as *losses* de gerador do treinamento dos três geradores seguem o mesmo comportamento, como esperado. Ao final das 25 épocas de treinamento, o gerador do treinamento longo (R11B) aparenta estar com uma tendência mais horizontal do que a clara tendência de queda que apresentava antes. Apesar de treinar a rede por quatro vezes mais tempo, o experimento R11C não conseguiu uma redução tão expressiva da *loss*, em relação ao experimento R11B.

O mesmo comportamento se repete nas métricas L1 e FID (Fig. 5.40). Após seguir com a redução de ambas as métricas, essa tendência vai ficando menos perceptível conforme o treinamento avança, ao ponto que próximo do final do treinamento a tendência é quase horizontal.

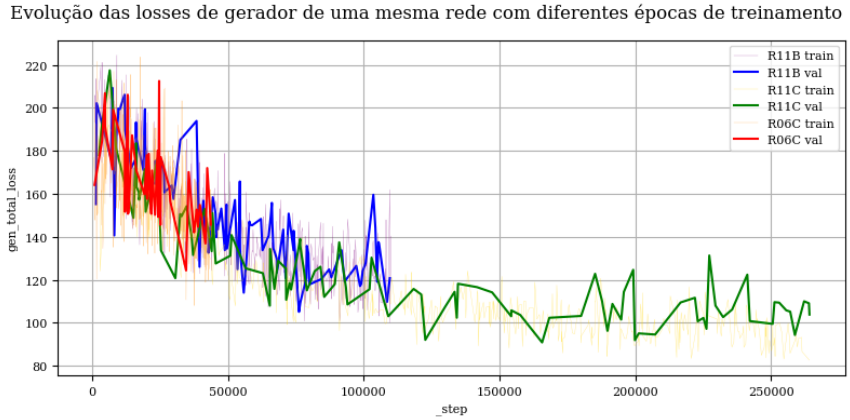

Figura 5.39: *Losses* de gerador do treinamento longo do melhor *autoencoder*. As configurações de cada experimento estão detalhadas na Tabela 5.11

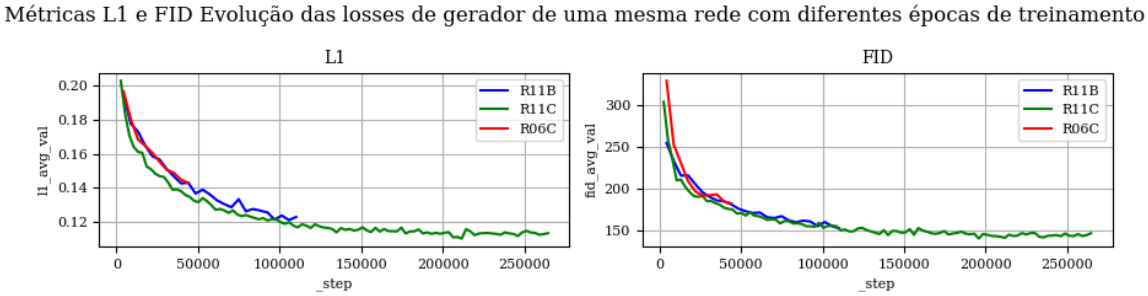

Figura 5.40: Métricas de qualidade do treinamento longo do melhor *autoencoder*. Cada sigla da legenda equivale a um experimento, e suas configurações estão descritas na Tabela 5.11. A métrica FID representa a diferença entre a distribuição das imagens reais e sintéticas, e a métrica L1 representa a diferença média pixel a pixel de cada imagem sintética com seu respectivo objetivo.

O resultado final, apresentado na Tabela 5.11, corrobora com essas observações: o experimento R11B conseguiu reduzir sensivelmente tanto o FID e o L1 calculados na base de dados de teste, em relação ao experimento R06C, ao passo que o R11C reduziu mais discretamente as métricas em relação ao experimento anterior.

Ao se comparar o resultado da aplicação de ambos os geradores em diversas imagens de entrada (Fig. 5.41), a imagem gerada pelo gerador treinado por mais tempo é mais nítida, detalhada e parecida com a imagem original, inclusive no caso em que a pessoa usa maquiagem. A perceptível melhora entre os experimentos R11B e R11C não reflete tão claramente nas métricas de qualidade da Tabela 5.11.

EXP	Gerador	Discriminador	Loss	Normalização	Desemaranhamento
R06C	Full residual	ProGAN	WGAN-GP	`InstanceNorm`	*Smooth*
R11B	Full residual	ProGAN	WGAN-GP	`InstanceNorm`	*Smooth*
R11C	Full residual	ProGAN	WGAN-GP	`InstanceNorm`	*Smooth*

EXP	Épocas	*Batch size*	FID Teste	L1 Teste	Tempo infer. (s)	*Runtime* (h)
R06C	10	6	174,4460	0,1385	0,2410	7,5
R11B	25	6	147,7250	0,1209	0,2312	18,3
R11C	100	10	**136,2515**	**0,1109**	0,3138	67,8

Tabela 5.11: Resultados do treinamento longo do melhor *autoencoder*. Os experimentos R11B e R11C usaram o R06C como base, mas foram treinados por 25 e 100 épocas, respectivamente.

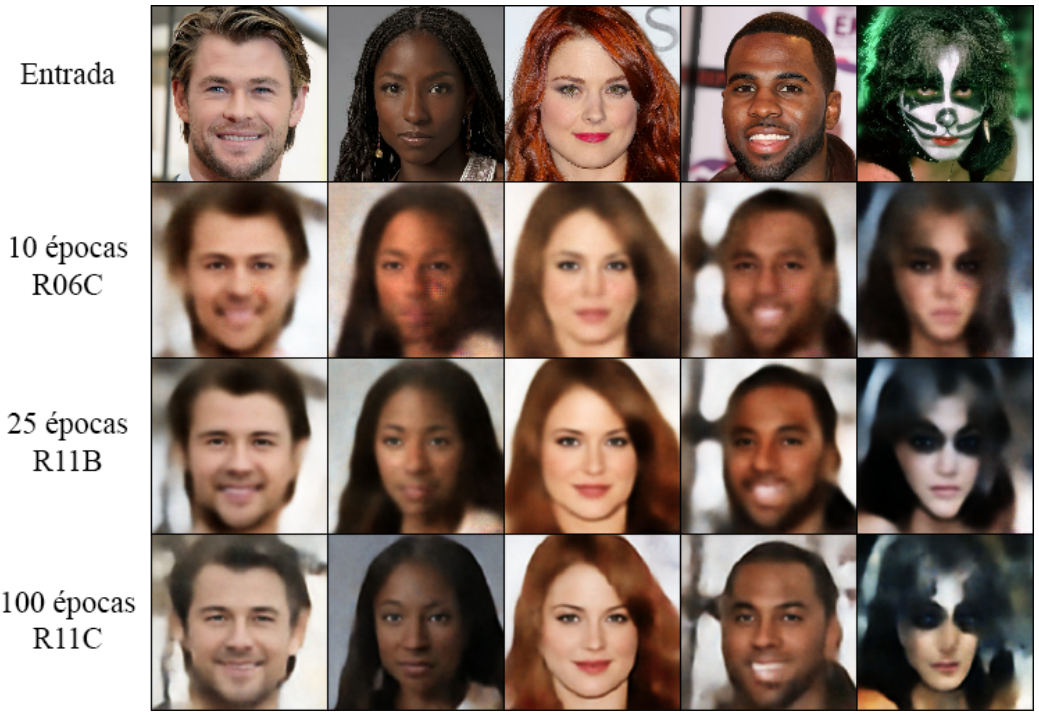

Figura 5.41: Resultados do treinamento longo do melhor *autoencoder*. As imagem da linha superior foram apresentada a ambas as redes como "entrada", as linhas abaixo representam o resultado da imagem sintetizada por cada um dos geradores testados.

Apesar dos melhores resultados, ainda não é possível assumir que a nova rede não consegue se beneficiar de um treinamento ainda mais longo. Isso pode ser facilmente testado no futuro.

5.3.10 Discussão

Observamos que o treinamento adversário somado à loss L1 permite a criação de imagens com um alto grau de similaridade com a imagem original nos *autoencoders* que não geram um vetor latente. No entanto, a mesma qualidade não foi obtida nos outros *autoencoders*.

Um motivo que pode ter causado a falha do método é que há uma restrição muito grande na GAN: ela deve aprender simultaneamente a codificar a imagem em um espaço muito reduzido, representar nesse espaço as características semânticas de uma face (tais como olhos, tipo de cabelo, gênero, e tudo aquilo que se pode descrever) e, a partir dessa representação codificada, aprender a reconstruir *a mesma* imagem com uma precisão a nível de pixels.

Sabemos que essas tarefas são possíveis individualmente. A Pix2Pix [28, 72] é capaz de reconstruir uma imagem com precisão a nível de pixels se ela não tiver que codificar a imagem num espaço latente muito reduzido. As StyleGANs [33, 34, 31, 32] conseguem aprender a sintetizar faces realistas a partir de um espaço latente reduzido que mantém a representação semântica dessas faces. O problema da nossa abordagem está em juntar ambas as tarefas em um único modelo.

Mesmo explorando alternativas de como utilizar combinações de discriminadores, *losses* e taxas de aprendizado diferentes, ou técnicas avançadas como *transfer learning*, o método não foi capaz de se especializar nessa tarefa tão complexa.

5.4 Conclusão

A tarefa de se utilizar GANs condicionais como a Pix2Pix para aprender a codificar a imagem em um vetor latente e simultaneamente reconstruir a mesma imagem a partir desse vetor, se mostrou mais complexa do que parecia a princípio.

Mesmo após testar dezenas de diferentes configurações, propor novas arquiteturas e combinar elementos de técnicas consolidadas, nossa melhor rede ainda não foi capaz de gerar imagens nítidas e de alta qualidade enquanto codifica a informação dessa imagem em um vetor latente ao mesmo tempo.

Apesar desse resultado, pudemos contribuir com um estudo extensivo que mostra o efeito isolado de cada elemento dessas combinações no resultado final, e as próximas gerações de pesquisadores podem utilizar desses resultados para compreender melhor o funcionamento das GANs.

5.5 Considerações Finais

O método proposto era uma forma de permitir a manipulação de uma imagem diretamente pelo seu vetor latente, sem usar uma rede adicional para descobrir qual vetor deve ser usado.

Uma abordagem de *autoencoder* adversário alternativa é exemplificada na técnica *Swapping Autoencoder* [46], que codifica a imagem em dois espaços latentes distintos: um geométrico e um de estilo. Um estudo posterior interessante seria verificar se esses espaços latentes são desemaranhados, se permitem a manipulação das características da imagem individualmente, e que tipos de manipulações são possíveis.

Outra alternativa que pode melhorar os resultados do nosso método é reduzir a complexidade do treinamento através do crescimento progressivo [30], que confere maior estabilidade à rede e maior qualidade às imagens geradas, mas que pode necessitar de maior esforço computacional e tempo de treinamento.

5.5.1 Métodos não condicionais

GANs não condicionais como a BigGAN [11], ou as StyleGANs [33, 34, 31, 32] alcançam resultados impressionantes na síntese de imagens. A forma mais comum de se realizar a manipulação dessas imagens pelo seu vetor latente é através de métodos que descobrem qual é o vetor que equivale a uma imagem específica no espaço da GAN em estudo.

Diversos métodos foram criados para resolver esse problema [54, 79, 2, 3], e podem ser objeto de estudo de trabalhos futuros.

6 DISPOSIÇÕES FINAIS

O tema de síntese de imagens usando *Deep Learning* está em rápida evolução, o que a torna uma área desafiadora de se acompanhar [75]. Desde o início deste trabalho, dezenas de novos métodos foram lançados, e por limitações de tempo não pudemos testar todos eles.

Recentemente, o estado-da-arte em geração de imagens sintéticas de qualidade não é mais através de GANs, mas sim por meio de modelos de difusão (*denoising diffusion probabilistic models*) [15]. Esses modelos, ilustrados pela Figura 6.1, criam cadeias de Markov que aumentam o grau de ruído de uma imagem até que ela apresente apenas ruído, e treinam redes que aprendem o caminho reverso, efetivamente gerando imagens a partir de ruído [67, 25]. Desses métodos pode-se ressaltar o Dall-E 2 [50], o Imagen [59] e o Stable Diffusion [55].

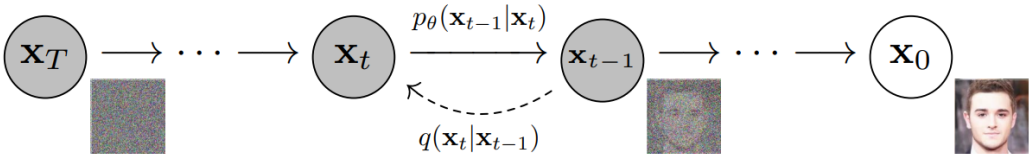

Figura 6.1: Funcionamento de um modelo de difusão. Uma imagem x_0 passa por um processo q que aumenta gradativamente a quantidade de ruído presente nela. O modelo de difusão p_θ é treinado para aprender a fazer o caminho inverso e gerar uma imagem a partir de uma entrada de ruído x_T. Fonte [25].

Apesar desses resultados, esse tipo de modelo ainda é muito lento no momento da inferência [77], fazendo com que GANs sejam preferíveis para aplicações interativas ou de tempo real. Xiao *et al.* [77] enunciaram o "*generative learning trilemma*"(Fig. 6.2),

argumentando que modelos generativos buscam três características: gerar imagens de alta qualidade, gerar imagens rapidamente, e gerar imagens com diversidade de modo.

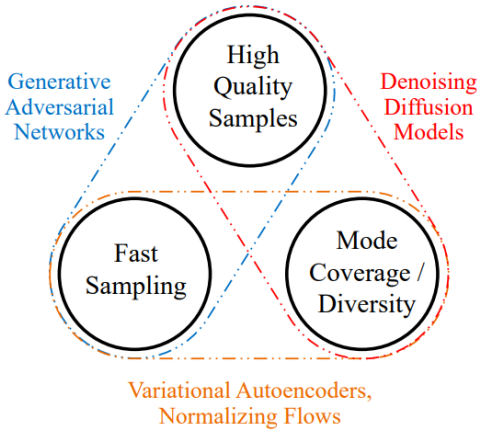

Figura 6.2: O *generative learning trilemma*. Normalmente, os métodos generativos atuais possuem apenas duas das três características. Fonte [77].

As GANs funcionam bem com as duas primeiras características, mas ainda tem dificuldades com a última. Tendo em mente que a motivação deste trabalho era o uso em aplicações artísticas interativas, a escolha das GANs segue justificada. Ainda assim, há trabalhos que apontam na direção de resolver simultaneamente o *trilemma*, como o Denoising Diffusion GAN [77] e a StyleGAN-XL [63], e podemos imaginar que no futuro esse problema será resolvido.

Outro ponto importante a se ressaltar, é a complexidade computacional envolvida no treinamento dessas redes. Ficamos limitados pelos nossos recursos computacionais e de tempo, a usar arquiteturas mais antigas e testar menos configurações por poucas épocas de treinamento. A StyleGAN3, por exemplo, usou em seu desenvolvimento o equivalente a 91,77 anos de processamento de uma única GPU Volta, acelerando esse tempo por meio de um ou mais *clusters* Nvidia DGX-1, e consumindo 225 MWh de eletricidade [32]. Neste trabalho usamos um sistema com uma GPU Nvidia RTX2070 Super, obtida por recursos próprios, e que tem um desempenho inferior a um desses *clusters*.

6.1 Conclusão

Em nossos estudos, o uso da arquitetura supervisionada Pix2Pix para a transformação de esboços sintéticos em fotos de carros teve um bom desempenho com figuras da mesma base de dados, porém não mostrou uma boa capacidade de generalização quando aplicada a esboços criados por artistas. Entretanto, a arquitetura não supervisionada CycleGAN mostrou resultados melhores do que a Pix2Pix, inclusive com maior capacidade de generalização. Infelizmente essa abordagem não se mostrou eficiente na base de personagens humanoides.

Na tarefa de síntese avaliamos o uso de *autoencoders* como geradores para gerar uma representação vetorial da imagem no espaço latente, ao mesmo tempo em que a rede aprende a criar imagens sintéticas de alta qualidade. Esse resultado não foi alcançado, pois as imagens geradas por geradores que também criavam vetores latentes não tiveram qualidade, e as imagens que apresentavam maior qualidade vinham de geradores que não criavam os vetores latentes.

Em ambos os casos, os experimentos cobriram diversos pontos de falha dos métodos, e permitem uma compreensão mais aprofundada dos mecanismos que fazem com que as GANs aprendam a representar corretamente as distribuições originais dos dados. Esperamos que este trabalho sirva como guia para novos pesquisadores que buscam melhor entendimento das GANs.

6.2 Trabalhos Futuros

Futuramente, é possível tentar melhorar a qualidade da transformação de contornos em imagens, através do uso de técnicas mais recentes, como a *Swapping Autoencoder* [46], ou a Pix2PixHD [72].

Outro trabalho também pode ir na direção de melhorar as bases de dados, transformando os contornos em algo mais parecido com esboços criados por um artista e usar esses esboços no treinamento. Pode-se também tentar aumentar a base de dados das animações, incluindo estilos de outros artistas e usar uma GAN condicional multi-classe como a SAGAN [78] ou a BigGAN [11] para criar desenhos de estilos mais diversos.

A respeito da aplicação de síntese, podemos tentar usar o treinamento progressivo [30] para simplificar a tarefa de se recriar uma imagem de alta qualidade, quebrando-a em passos menores de geração de imagens de menor resolução.

Outro estudo interessante é treinar uma rede já consolidada como a ProGAN [30] ou a StyleGAN [33] em uma base de dados de outro domínio, como carros ou animais, e estudar como realizar as manipulações das suas características através do espaço latente.

REFERÊNCIAS

[1] Abadi, M., et al.: Tensorflow: Large-scale machine learning on heterogeneous systems (2015). URL https://www.tensorflow.org/. Software available from tensorflow.org

[2] Abdal, R., Qin, Y., Wonka, P.: Image2stylegan: How to embed images into the stylegan latent space? In: Proceedings of the IEEE/CVF International Conference on Computer Vision, pp. 4432–4441 (2019)

[3] Abdal, R., Qin, Y., Wonka, P.: Image2stylegan++: How to edit the embedded images? In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pp. 8296–8305 (2020)

[4] Arjovsky, M., Bottou, L.: Towards principled methods for training generative adversarial networks. In: International Conference on Learning Representations (2017)

[5] Arjovsky, M., Chintala, S., Bottou, L.: Wasserstein generative adversarial networks. In: International conference on machine learning, pp. 214–223. PMLR (2017)

[6] Ba, J.L., Kiros, J.R., Hinton, G.E.: Layer normalization. arXiv preprint arXiv:1607.06450 (2016)

[7] Bengio, Y., Simard, P., Frasconi, P.: Learning long-term dependencies with gradient descent is difficult. IEEE transactions on neural networks **5**(2), 157–166 (1994)

[8] Bińkowski, M., Sutherland, D.J., Arbel, M., Gretton, A.: Demystifying MMD GANs. In: International Conference on Learning Representations (2018)

[9] Bishop, C.M.: Pattern Recognition and Machine Learning. Springer-Verlag New York (2006)

[10] Bochkovskiy, A., Wang, C.Y., Liao, H.Y.M.: Yolov4: Optimal speed and accuracy of object detection. arXiv preprint arXiv:2004.10934 (2020)

[11] Brock, A., Donahue, J., Simonyan, K.: Large scale GAN training for high fidelity natural image synthesis. In: International Conference on Learning Representations (2019)

[12] Canny, J.: A computational approach to edge detection. IEEE Transactions on pattern analysis and machine intelligence pp. 679–698 (1986)

[13] Chai, J., Zeng, H., Li, A., Ngai, E.W.: Deep learning in computer vision: A critical review of emerging techniques and application scenarios. Machine Learning with Applications **6**, 100,134 (2021)

[14] Coppin, B.: Inteligência Artificial. LTC, Rio de Janeiro (2015)

[15] Dhariwal, P., Nichol, A.: Diffusion models beat gans on image synthesis. Advances in Neural Information Processing Systems **34**, 8780–8794 (2021)

[16] Dumoulin, V., Visin, F.: A guide to convolution arithmetic for deep learning. arXiv preprint arXiv:1603.07285 (2016)

[17] Gatys, L.A., Ecker, A.S., Bethge, M.: Image style transfer using convolutional neural networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 2414–2423 (2016)

[18] Glorot, X., Bordes, A., Bengio, Y.: Deep sparse rectifier neural networks. In: Proceedings of the fourteenth international conference on artificial intelligence and statistics, pp. 315–323. JMLR Workshop and Conference Proceedings (2011)

[19] Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press (2016). http://www.deeplearningbook.org

[20] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial nets. Advances in neural information processing systems **27** (2014)

[21] Gui, J., Sun, Z., Wen, Y., Tao, D., Ye, J.: A review on generative adversarial networks: Algorithms, theory, and applications. IEEE Transactions on Knowledge and Data Engineering (2021)

[22] Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., Courville, A.C.: Improved training of wasserstein gans. Advances in neural information processing systems **30** (2017)

[23] He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 770–778 (2016)

[24] Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., Hochreiter, S.: Gans trained by a two time-scale update rule converge to a local nash equilibrium. Advances in neural information processing systems **30** (2017)

[25] Ho, J., Jain, A., Abbeel, P.: Denoising diffusion probabilistic models. Advances in Neural Information Processing Systems **33**, 6840–6851 (2020)

[26] Huang, X., Belongie, S.: Arbitrary style transfer in real-time with adaptive instance normalization. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 1501–1510 (2017)

[27] Ioffe, S., Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: International conference on machine learning, pp. 448–456. PMLR (2015)

[28] Isola, P., Zhu, J.Y., Zhou, T., Efros, A.A.: Image-to-image translation with conditional adversarial networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 1125–1134 (2017)

[29] Johnson, J., Alahi, A., Fei-Fei, L.: Perceptual losses for real-time style transfer and super-resolution. In: European conference on computer vision, pp. 694–711. Springer (2016)

[30] Karras, T., Aila, T., Laine, S., Lehtinen, J.: Progressive growing of gans for improved quality,

stability, and variation. In: International Conference on Learning Representations (2018)

[31] Karras, T., Aittala, M., Hellsten, J., Laine, S., Lehtinen, J., Aila, T.: Training generative adversarial networks with limited data. In: IEEE Conference on Neural Information Processing Systems; (2020)

[32] Karras, T., Aittala, M., Laine, S., Härkönen, E., Hellsten, J., Lehtinen, J., Aila, T.: Alias-free generative adversarial networks. In: Proc. NeurIPS (2021)

[33] Karras, T., Laine, S., Aila, T.: A style-based generator architecture for generative adversarial networks. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 4401–4410 (2019)

[34] Karras, T., Laine, S., Aittala, M., Hellsten, J., Lehtinen, J., Aila, T.: Analyzing and improving the image quality of stylegan. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 8110–8119 (2020)

[35] Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. In: International Conference on Learning Representations (2015)

[36] Kynkäänniemi, T., Karras, T., Aittala, M., Aila, T., Lehtinen, J.: The role of imagenet classes in fr\'echet inception distance. arXiv preprint arXiv:2203.06026 (2022)

[37] LeCun, Y.: Generalization and network design strategies. Technical Report CRG-TR-89-4, University of Toronto (1989)

[38] Ledig, C., Theis, L., Huszár, F., Caballero, J., Cunningham, A., Acosta, A., Aitken, A., Tejani, A., Totz, J., Wang, Z., et al.: Photo-realistic single image super-resolution using a generative adversarial network. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 4681–4690 (2017)

[39] Lira, W., Merz, J., Ritchie, D., Cohen-Or, D., Zhang, H.: Ganhopper: Multi-hop gan for unsupervised image-to-image translation. In: European Conference on Computer Vision, pp. 363–379. Springer (2020)

[40] Lun, Z., Gadelha, M., Kalogerakis, E., Maji, S., Wang, R.: 3d shape reconstruction from sketches via multi-view convolutional networks. In: 2017 International Conference on 3D Vision (3DV), pp. 67–77. IEEE (2017)

[41] dazza mate: Bmw e92 sketch. URL https://www.deviantart.com/dazza-mate/art/BMW-e92-Sketch-122643179. Last access: Nov 11 2022

[42] McCulloch, W.S., Pitts, W.: A logical calculus of the ideas immanent in nervous activity. The bulletin of mathematical biophysics **5**(4), 115–133 (1943)

[43] Mirza, M., Osindero, S.: Conditional generative adversarial nets. arXiv preprint arXiv:1411.1784 (2014)

[44] Odena, A., Dumoulin, V., Olah, C.: Deconvolution and checkerboard artifacts. Distill (2016). DOI 10.23915/distill.00003. URL http://distill.pub/2016/deconv-checkerboard

[45] Park, T., Liu, M.Y., Wang, T.C., Zhu, J.Y.: Semantic image synthesis with spatially-adaptive normalization. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pp. 2337–2346 (2019)

[46] Park, T., Zhu, J.Y., Wang, O., Lu, J., Shechtman, E., Efros, A., Zhang, R.: Swapping autoencoder for deep image manipulation. Advances in Neural Information Processing Systems **33**, 7198–7211 (2020)

[47] Paszke, A., et al: Pytorch: An imperative style, high-performance deep learning library. In: H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, R. Garnett (eds.) Advances in Neural Information Processing Systems 32, pp. 8024–8035. Curran Associates, Inc. (2019). URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[48] Pro, C.D.: Car sketches. URL https://cardesignpro.artstation.com/projects/3XYJB. Last access: Nov 11 2022

[49] Radford, A., Metz, L., Chintala, S.: Unsupervised representation learning with deep convolutio-

nal generative adversarial networks. In: International Conference on Learning Representations (2016)

[50] Ramesh, A., Dhariwal, P., Nichol, A., Chu, C., Chen, M.: Hierarchical text-conditional image generation with clip latents. arXiv preprint arXiv:2204.06125 (2022)

[51] Ramos, S., Trevisan, D.F., Batagelo, H.C., Sousa, M.C., Gois, J.P.: Contour-aware 3d reconstruction of side-view sketches. Computers & Graphics **77**, 97–107 (2018)

[52] Redmon, J., Divvala, S., Girshick, R., Farhadi, A.: You only look once: Unified, real-time object detection. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 779–788 (2016)

[53] Redmon, J., Farhadi, A.: Yolov3: An incremental improvement. arXiv preprint arXiv:1804.02767 (2018)

[54] Roich, D., Mokady, R., Bermano, A.H., Cohen-Or, D.: Pivotal tuning for latent-based editing of real images. arXiv preprint arXiv:2106.05744 (2021)

[55] Rombach, R., Blattmann, A., Lorenz, D., Esser, P., Ommer, B.: High-resolution image synthesis with latent diffusion models. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 10,684–10,695 (2022)

[56] Ronneberger, O., Fischer, P., Brox, T.: U-net: Convolutional networks for biomedical image segmentation. In: International Conference on Medical image computing and computer-assisted intervention, pp. 234–241. Springer (2015)

[57] Rosenblatt, F.: The perceptron: a probabilistic model for information storage and organization in the brain. Psychological review **65**(6), 386 (1958)

[58] Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Learning internal representations by error propagation. Tech. rep., California Univ San Diego La Jolla Inst for Cognitive Science (1985)

[59] Saharia, C., Chan, W., Saxena, S., Li, L., Whang, J., Denton, E., Ghasemipour, S.K.S., Ayan, B.K., Mahdavi, S.S., Lopes, R.G., et al.: Photorealistic text-to-image diffusion models with

deep language understanding. arXiv preprint arXiv:2205.11487 (2022)

[60] Saharia, C., Ho, J., Chan, W., Salimans, T., Fleet, D.J., Norouzi, M.: Image super-resolution via iterative refinement. arXiv preprint arXiv:2104.07636 (2021)

[61] Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., Chen, X.: Improved techniques for training gans. Advances in neural information processing systems **29**, 2234–2242 (2016)

[62] Santurkar, S., Tsipras, D., Ilyas, A., Mądry, A.: How does batch normalization help optimization? In: Proceedings of the 32nd international conference on neural information processing systems, pp. 2488–2498 (2018)

[63] Sauer, A., Schwarz, K., Geiger, A.: Stylegan-xl: Scaling stylegan to large diverse datasets. In: ACM SIGGRAPH 2022 Conference Proceedings, SIGGRAPH '22. Association for Computing Machinery, New York, NY, USA (2022). DOI 10.1145/3528233.3530738

[64] Scherer, D., Müller, A., Behnke, S.: Evaluation of pooling operations in convolutional architectures for object recognition. In: International conference on artificial neural networks, pp. 92–101. Springer (2010)

[65] Schroff, F., Kalenichenko, D., Philbin, J.: Facenet: A unified embedding for face recognition and clustering. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 815–823 (2015)

[66] Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. In: International Conference on Learning Representations (2015)

[67] Sohl-Dickstein, J., Weiss, E., Maheswaranathan, N., Ganguli, S.: Deep unsupervised learning using nonequilibrium thermodynamics. In: International Conference on Machine Learning, pp. 2256–2265. PMLR (2015)

[68] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A.: Going deeper with convolutions. In: Proceedings of the IEEE conference

on computer vision and pattern recognition, pp. 1–9 (2015)

[69] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., Wojna, Z.: Rethinking the inception architecture for computer vision. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 2818–2826 (2016)

[70] Ulyanov, D., Lebedev, V., Vedaldi, A., Lempitsky, V.S.: Texture networks: Feed-forward synthesis of textures and stylized images. In: ICML, vol. 1, p. 4 (2016)

[71] Ulyanov, D., Vedaldi, A., Lempitsky, V.: Instance normalization: The missing ingredient for fast stylization. arXiv preprint arXiv:1607.08022 (2016)

[72] Wang, T.C., Liu, M.Y., Zhu, J.Y., Tao, A., Kautz, J., Catanzaro, B.: High-resolution image synthesis and semantic manipulation with conditional gans. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 8798–8807 (2018)

[73] Wang, X., Li, Y., Zhang, H., Shan, Y.: Towards real-world blind face restoration with generative facial prior. In: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2021)

[74] Wang, Z., Bovik, A.C., Sheikh, H.R., Simoncelli, E.P.: Image quality assessment: from error visibility to structural similarity. IEEE transactions on image processing **13**(4), 600–612 (2004)

[75] Wang, Z., She, Q., Ward, T.E.: Generative adversarial networks in computer vision: A survey and taxonomy. ACM Computing Surveys (CSUR) **54**(2), 1–38 (2021)

[76] Wu, Y., He, K.: Group normalization. In: Proceedings of the European conference on computer vision (ECCV), pp. 3–19 (2018)

[77] Xiao, Z., Kreis, K., Vahdat, A.: Tackling the Generative Learning Trilemma with Denoising Diffusion GANs. In: International Conference on Learning Representations (ICLR) (2022)

[78] Zhang, H., Goodfellow, I., Metaxas, D., Odena, A.: Self-attention generative adversarial networks. In: International conference on machine learning, pp. 7354–7363. PMLR (2019)

[79] Zhu, J., Shen, Y., Zhao, D., Zhou, B.: In-domain gan inversion for real image editing. In: European conference on computer vision, pp. 592–608. Springer (2020)

[80] Zhu, J.Y., Park, T., Isola, P., Efros, A.A.: Unpaired image-to-image translation using cycle-consistent adversarial networks. In: Proceedings of the IEEE international conference on computer vision, pp. 2223–2232 (2017)

A IMAGENS GERADAS PELOS MELHORES EXPERIMENTOS

Neste apêndice apresentamos diversos exemplos de imagens geradas pelos geradores dos nossos melhores experimentos, usando imagens da base de dados de teste (ou seja, nunca antes vistas pela rede) como base.

Imagens da base de dados de teste geradas com a melhor Pix2Pix na base de dados de carros (Experimento P02B):

Figura A.1: Exemplos de imagens geradas a partir do gerador na base de dados de carros.

Imagens da base de dados de teste geradas com a melhor CycleGAN na base de dados de carros (Experimento C01B):

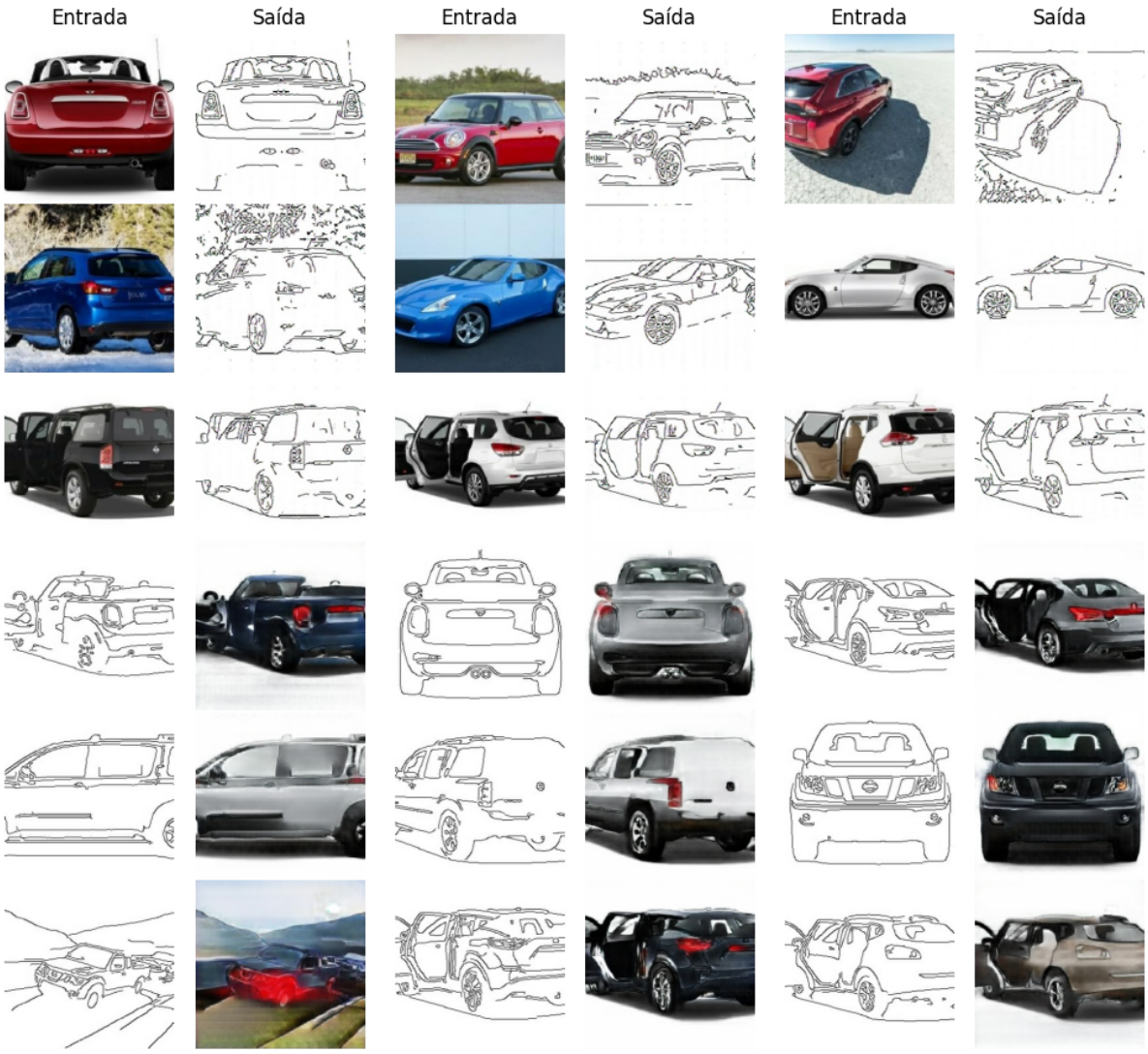

Figura A.2: Exemplos de imagens geradas a partir do gerador C01B na base de dados de carros.

Imagens da base de dados de teste geradas com a melhor CycleGAN na base de dados de personagens (Experimento C03F):

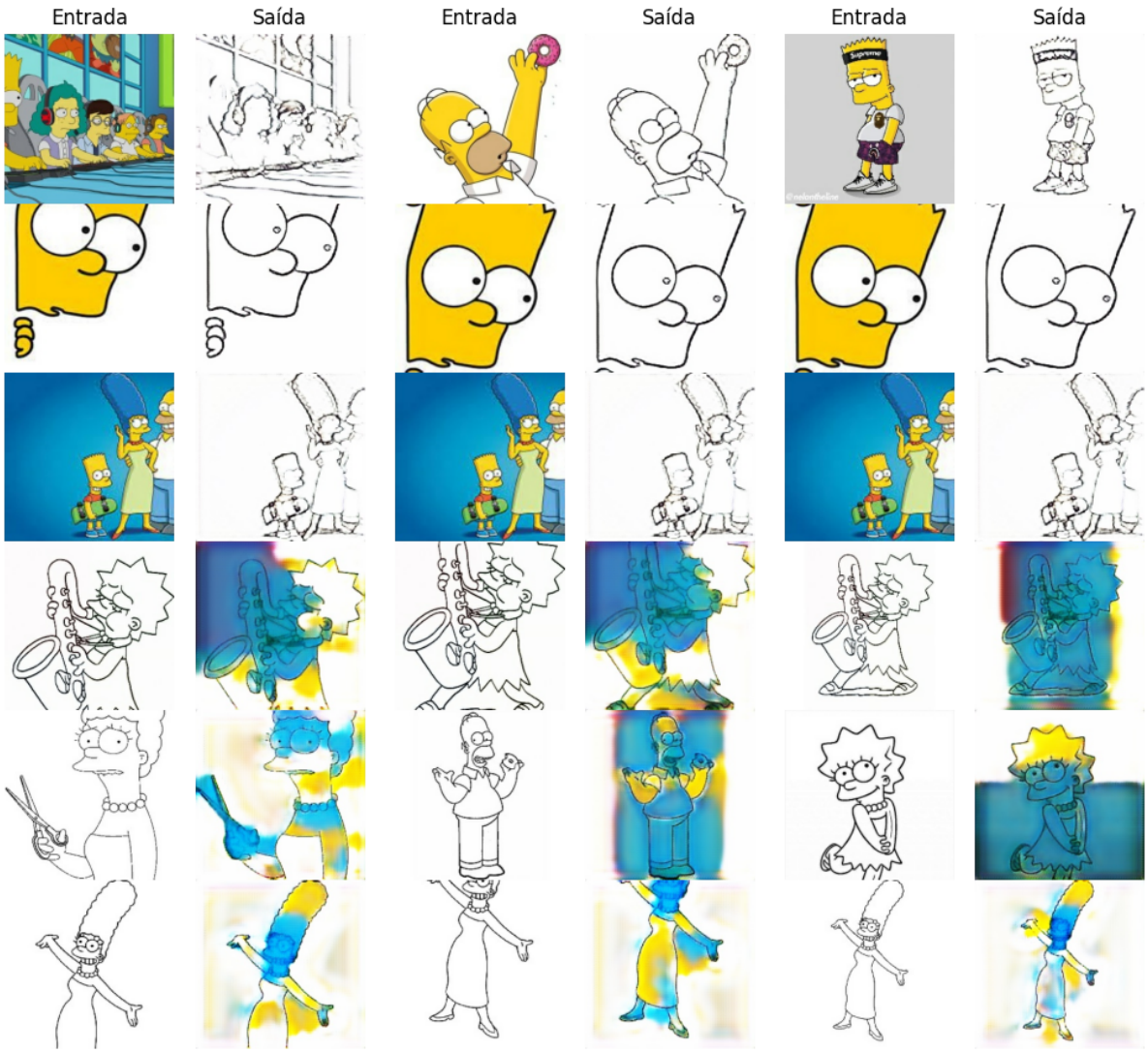

Figura A.3: Exemplos de imagens geradas a partir do gerador C03F na base de dados de personagens.

Imagens da base de dados de teste geradas com o melhor autoencoder na base de dados CelebaHQ (Experimento R11B):

Figura A.4: Exemplos de imagens geradas a partir do gerador R11B na base de dados CelebaHQ.