UNIVERSIDADE FEDERAL DE SANTA CATARINA
CAMPUS FLORIANÓPOLIS

Vinícius Domingos Alves

Otimização de Hiperparâmetros Arquiteturais em Redes Adversárias Generativas

FLORIANÓPOLIS
2021

VINÍCIUS DOMINGOS ALVES

Otimização de Hiperparâmetros Arquiteturais em Redes Adversárias Generativas

Trabalho de Conclusão de Curso submetido à Universidade Federal de Santa Catarina, como requisito necessário para obtenção do grau de Bacharel em Engenharia Elétrica

Florianópolis, Fevereiro de 2022

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Domingos Alves, Vinícius
Otimização de Hiperparâmetros Arquiteturais em Redes Adversárias Generativas / Vinícius Domingos Alves ; orientador, Danilo Silva, 2022.
83 p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Centro Tecnológico, Graduação em Engenharia Elétrica, Florianópolis, 2022.

Inclui referências.

1. Engenharia Elétrica. 2. Aprendizado de Máquina. 3. Modelos Geradores. 4. Redes Adversárias Generativas. I. Silva, Danilo. II. Universidade Federal de Santa Catarina. Graduação em Engenharia Elétrica. III. Título.

UNIVERSIDADE FEDERAL DE SANTA CATARINA

VINÍCIUS DOMINGOS ALVES

Esta Monografia foi julgada adequada para a obtenção do título de Bacharel em Engenharia Elétrica, sendo aprovada em sua forma final pela banca examinadora.

Documento assinado digitalmente
Jean Vianei Leite
Data: 17/03/2022 11:54:47-0300
CPF: 003.474.909-80
Verifique as assinaturas em https://v.ufsc.br

Coordenador do Curso de Graduação: Prof. Dr. Jean Vianei Leite
Universidade Federal de Santa Catarina - UFSC

Banca Examinadora:

Documento assinado digitalmente
Danilo Silva
Data: 14/03/2022 11:14:31-0300
CPF: 031.557.924-26
Verifique as assinaturas em https://v.ufsc.br

Orientador(a): Prof. Dr. Danilo Silva
Universidade Federal de Santa Catarina - UFSC

Documento assinado digitalmente
Marcio Holsbach Costa
Data: 14/03/2022 11:36:21-0300
CPF: 572.170.680-53
Verifique as assinaturas em https://v.ufsc.br

Prof. Dr. Márcio Holsbach Costa
Universidade Federal de Santa Catarina - UFSC

Documento assinado digitalmente
Richard Demo Souza
Data: 14/03/2022 11:24:25-0300
CPF: 004.267.379-89
Verifique as assinaturas em https://v.ufsc.br

Prof. Dr. Richard Demo Souza
Universidade Federal de Santa Catarina - UFSC

Florianópolis, 11 de Março de 2022

Agradecimentos

Em primeiro lugar, gostaria de agradecer principalmente aos meus pais, mas também aos meus avós e tias, que, por meio do exemplo, me incentivarem aos estudos, a diligência e a lutar por meus objetivos desde cedo, assim como, por proverem todos os recursos necessários, emocionais e materiais, que me permitiram chegar até aqui.

Gostaria de agradecer também aos meus amigos Josué Silveira e Victor Grecco, que me acompanharam por essa jornada acadêmica, dividindo o fardo da graduação e compartilhando bons momentos.

E, por fim, mas não menos importante, gostaria de agradecer ao meu orientador, Prof. Danilo Silva, pela paciência e dedicação na orientação desse trabalho, assim como, em todas as diversas matérias em que tive o prazer de ser seu aluno.

Resumo

A recente evolução notória no campo de *Machine Learning* tem demonstrado a relevância na capacidade de modelos em extrair características e internalizar eficientemente representações de espaços complexos e de alta dimensionalidade como imagens, áudio e linguagem natural. Nesse âmbito, a pesquisa em torno de modelos profundos não-supervisionados constitui uma importante fronteira tecnológica, sendo os resultados relevantes para todo o espectro de aplicações do campo. Em 2014 com a introdução das Redes Adversárias Generativas (GANs), e sua posterior evolução, as Redes Profundas Convolucionais Adversárias Generativas (DCGANs), obteve-se um progresso significativo na capacidade de geração de imagens. No entanto, embora essa subclasse de modelos geradores apresente performances sem precedentes, treiná-las continua sendo um desafio árduo para pesquisadores e desenvolvedores na indústria. Parte significativa da literatura sobre GANs se dedica ao estudo de arquiteturas, técnicas de treinamento e heurísticas para o ajuste de hiperparâmetros, com o objetivo de aumentar a estabilidade do treinamento, assim como obter melhores performances. Esse trabalho se dedica a explorar o impacto do ajuste de hiperparâmetros arquiteturais na estabilidade e performance do treinamento de DCGANs. Para tanto, propõe-se uma metaparametrização da arquitetura de uma DCGAN e desenvolve-se um estudo empírico do impacto na performance dos modelos treinados com diferentes combinações de características arquiteturais descritas pela metaparametrização. Conclui-se que a progressão das dimensões das camadas convolucionais apresenta potencial de otimização relevante; no entanto, altamente dependente do *dataset*, o que inviabiliza qualquer prescrição quantitativa. Por outro lado, a distribuição dos filtros ao longos das camadas, apesar de apresentar baixo potencial de otimização, apresentou resultados consistentes a medida em a distribuição se aproxima de uma curva linear. E, por fim, o ajuste da profundidade assim como da complexidade apresentam os maiores potenciais de otimização, superiores a 50% e 20%, respectivamente, e que a otimização arquitetural, embora relevante, deve ser feita em conjunto com a otimização dos hiperparâmetros de treinamento, sendo esses primordiais para a estabilização da maioria dos ajustes arquiteturais testados.

Abstract

The recent notorious evolution in the field of Machine Learning has demonstrated the relevance of feature extraction and the efficient internal representation of complex high dimensional spaces like images, audio and natural language. On this context, the research of deep non-supervised models are an important technological frontier to achieve these capabilities. With the introduction of GANs in 2014, and afterwards DCGANs, a significative improvement was obtained in the field of image generation. Nevertheless, despite this new subclass presented an unprecedented performance, it's training continues to be a challenging task for researchers and practitioners in the industry. A significative part of the GANs related literature is dedicated to the study of architectures, training techniques and heuristics to adjust hyperparameters in order to increase training stability and the model's performance. The goal of this work is to explore the impact of the adjustment of architectural hyperparameters on the stability of DCGANs training as well as on their performance. It is, therefore, proposed a metaparametrization of the DCGAN architecture which encodes and gives control over notable aspects of the DCGAN architecture, enabling an empirical study of the performance impact of models trained with different architectural characteristics described by the metaparametrization. The study concluded that the progression of the convolutional layers dimensions present a relevant potential of optimization although heavily dependent on the dataset, which impossibilitates any quantitative prescription. On the other hand, the progression of the number of filters throughout the convolutional layers, despite the low optimization potential, has presented consistent results for distributions closer to a linear curve. At last, the depth and complexity adjustments presented the highest optimization potentials, over 50% and 20%, respectively. It was also noted that the optimization of training hyperparameters should always be done in conjunction to the architectural optimization, as most architectural combinations tested were initially unstable.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

GAN	*Generative Adversarial Network*
RBM	*Restricted Boltzmann Machine*
DBN	*Deep Belief Network*
GSN	*Generative Stochastic Network*
DCGAN	*Deep Convolutional Generative Adversarial Network*
FVBN	*Fully Visible Belief Network*
VAE	*Variational Autoencoders*
SGD	*Stochastic Gradient Descent*
ReLU	*Rectified Linear Unit*
FID	*Frechet Inception Distance*
FPECB	Função de Perda de Entropia Cruzada Binária
TPD	Taxa de Progressão da Dimensão
TPNF	Taxa de Progressão do Número de Filtros

Sumário

1 Introdução

Desenvolvimentos recentes no campo de *Machine Learning*, em especial a subclasse *Deep Learning*, demonstram como a extração de características essenciais e a capacidade de internalizar eficientemente representações de espaços complexos de alta dimensionalidade, como imagens, áudio e linguagem natural, permitem a obtenção de resultados ímpares no processamento de sinais. Nesse âmbito, modelos não supervisionados, como os modelos geradores, se apresentam como uma das mais promissoras alternativas na busca por técnicas capazes de representar eficientemente espaços de alta dimensionalidade e extrair características naturais desses, que podem ser utilizadas tanto para o aperfeiçoamento de técnicas supervisionadas quanto para solução de problemas que requerem a modelagem de uma distribuição a priori [4] [5].

Recentemente, Modelos Geradores Profundos, como *Restricted Boltzmann Machines* (RBMs), *Deep Belief Networks* (DBNs), *Generative Stochastic Networks* (GSNs) atraíram bastante atenção por conseguirem capturar tais distribuições de alta dimensionalidade e sintetizar amostras inéditas. No entanto, tais modelos sofrem de uma série de limitações como amostragem lenta e custosa, no caso dos modelos baseados em Cadeias de Markov, e maldição da dimensionalidade em problemas envolvendo imagens e vídeo, no caso de modelos baseado na maximização da verossimilhança. Nesse contexto, Redes Adversárias Generativas (*Generative Adversarial Networks*, GANs), uma subclasse de modelos geradores introduzidas em 2014 por Ian Goodfellow [6] rapidamente ganhou notoriedade ao contornar os problemas anteriormente mencionados, o que se refletiu na capacidade desses modelos em gerar imagens precisas e realistas, apresentar coerência semântica no espaço latente e permitir amostragem direta e computacionalmente eficiente. Tais características promoveram a aplicação de GANs em diversos problemas como a geração de imagens [7] [8], vídeo [9], super-resolução de imagens [10], reconhecimento de fala [11] e resolução de problemas inversos [12].

Embora apresentem performance inigualada por outros modelos, GANs são notoriamente difíceis de se treinar. Em contraste com a maioria dos modelos em *Machine Learning*, o treinamento de GANs não consiste apenas na otimização dos parâmetros de um modelo perante um conjunto de dados, mas da busca da solução de um jogo de soma zero entre modelos adversários, um gerador e um discriminador. Tal dinâmica promove a constante variação da superfície de otimização do ponto de vista dos modelos, o que pode resultar em modos de falha como a não convergência do treinamento e o colapso de modos. Além disso, GANs se revelam altamente sensíveis à escolha de hiperparâmetros e até mesmo à inicialização dos pesos das redes que a compõem e ao particionamento do conjunto de dados, o que torna ainda mais difícil o processo de otimização e comparação

de performance entre modelos [13].

Devido a esses desafios uma parte significativa da literatura relacionada a GANs é focada na proposição de técnicas de treinamento, arquiteturas, heurísticas para determinação de hiperparâmetros, assim como, análises teóricas a fim de estabilizar-se o treinamento e obter-se a convergência no jogo de soma zero disputado entre o gerador e discriminador. Dentre os desenvolvimentos, em 2016 os autores de [14] propuseram uma série de diretrizes arquiteturais para GANs voltadas para geração de imagens, a DCGAN, que ganharam notoriedade em aplicações envolvendo a geração de imagens por aumentar significativamente a estabilidade durante o treinamento e, consequentemente, a performance dos modelos. As DCGANs são a base para a maioria das arquiteturas de GANs propostas posteriormente, tal influência é facilmente atestada pelo fato desse artigo ser o oitavo mais citado no campo de Inteligência Artificial e Machine Learning [15].

No entanto, mesmo com as restrições arquiteturais das DCGANs há ainda uma miríade de possibilidades na configuração de uma GAN, tanto do ponto de vista arquitetural quanto da otimização do treinamento, requerendo que pesquisadores e praticantes percorram um árduo caminho de ajustes manuais até que encontrem o conjunto de hiperparâmetros que melhor se adequem ao problema que estejam resolvendo. A literatura contempla a proposição de técnicas de treinamento, como normalização espectral e *one-side label smoothing*, variações da função perda, técnicas de regularização, variações arquiteturais e sugestões e heurísticas quanto à otimização de hiperparâmetros. No escopo da otimização de hiperparâmetros percebe-se um foco nos hiperparâmetros de treinamento, como a taxa de aprendizado, os coeficientes de decaimento do otimizador Adam, a inércia do *Batch Normalization* e o grau de regularização, caso haja o emprego dessa técnica. Observou-se, por sua vez, que a otimização de hiperparâmetros arquiteturais, como a quantidade de filtros por camada, a quantidade de camadas, a progressão das dimensões das camadas, embora presente em diversos trabalhos não é explorada, para tanto quanto é do conhecimento do autor desse trabalho, de forma sistemática pela literatura, com o objetivo de propor heurísticas e diretrizes quanto a seleção de tais hiperparâmetros.

Nesse contexto, este documento tem por objetivo primário propor uma nova forma de parametrizar a arquitetura de uma DCGAN, por meio de uma metaparametrização, de forma a permitir a exploração das características arquiteturais que se mostram relevantes na literatura e avaliar quantitativamente, por meio de experimentos empíricos, o impacto da seleção de hiperparâmetros arquiteturais na performance e robustez do treinamento de DCGANs. E, dessa forma, permitir ao praticante melhor direcionar a escolha de hiperparâmetros a otimizar, destinando mais eficientemente o seu orçamento.

Este trabalho tem por objetivo secundário constituir uma introdução ao tópico de GANs que seja acessível, com uma abordagem partindo-se do conceito de modelos geradores, contextualizando-se GANs frente a esses e, por fim, introduzindo-se formalmente

GANs por meio da teoria dos jogos e deduzindo-se de forma gradual resultados chave para a comprovação da dinâmica de treinamento das GANs. Além de tais aspectos teóricos, como parte de tal introdução à GANs, objetivou-se a abordagem de temas fundamentais para a aplicação práticas de tais modelos como a explicação dos modos de falha e as técnicas e diretrizes mais comuns utilizadas para estabilizar-se o treinamento e melhorar a performance de tais modelos.

2 Fundamentação Teórica

Deep Learning é uma área do *Machine Learning* primariamente focada no aprendizado de modelos capazes de representar distribuições de probabilidades complexas em espaços de alta dimensionalidade, tais como as encontradas no processamento de áudio, imagens e linguagem natural.

Os modelos com maior proeminência no *Deep Learning*, cujos resultados já se encontram em diversas aplicações comerciais, são os chamados modelos condicionais ou discriminadores, isto é, modelos que estimam uma distribuição condicional entre duas variáveis. Geralmente a distribuição de uma variável representa uma categoria condicionada a uma variável de alta dimensionalidade que representa um sinal como uma imagem, por exemplo:

$$P(\boldsymbol{Y}|x \in \boldsymbol{X}) \tag{2.1}$$

Sendo $\boldsymbol{Y}$ um conjunto de N rótulos e $\boldsymbol{X}$ o espaço amostral do sinal, temos:

$$P(\boldsymbol{Y}|x_i) = \{p(y_1), p(y_2), ..., p(y_N)\} \text{ dado } x_i \in \boldsymbol{X} \tag{2.2}$$

Além dos modelos discriminadores, no entanto, existe uma outra classe que modela diretamente a distribuição de alta dimensionalidade, permitindo a obtenção, ou geração, de amostras inéditas de tal distribuição, tais modelos são conhecidos como modelos geradores.

$$P(y, x) \text{ ou } P(x) \text{ para qualquer } y \in \boldsymbol{Y} \text{ e } x \in \boldsymbol{X} \tag{2.3}$$

Existem diversos tipos de modelos geradores assim como diversas formas de modelar-se a distribuição subjacente de um conjunto de dados. Sendo as GANs uma subclasse de modelos geradores. Para melhor entender-se suas características é produtivo entender-se os diferentes tipos de modelos geradores existentes.

2.1 Modelos Geradores

Os modelos geradores podem ser divididos em duas principais categorias, como pode ser visto na figura 1: modelos de função densidade explícita ou implícita. Os modelos de função densidade explícita são aqueles cujo o resultado final da modelagem consiste numa função que descreve a distribuição, permitindo não apenas a amostragem mas também o cálculo da probabilidade de uma determinada região do espaço amostral. Os modelos de função densidade implícita, por outro lado, não apresentam como resultado a função densidade de probabilidade, mas uma função geradora capaz de gerar amostras tais quais fossem amostradas da função de densidade de probabilidade subjacente.

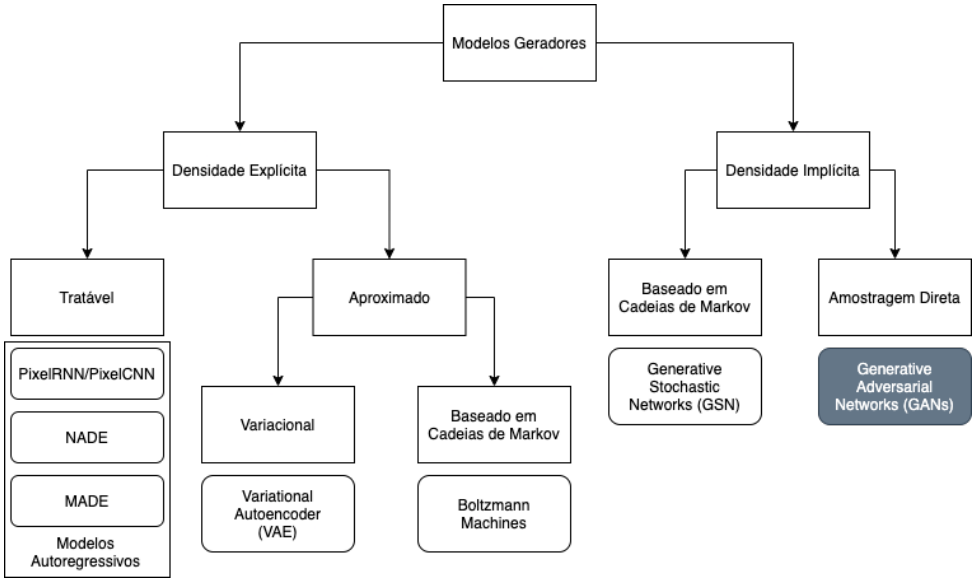


Figura 1 – Taxonomia dos Modelos Geradores. Fonte: adaptado de [1]

2.1.1 Modelos Geradores de Função Densidade de Probabilidade Explícita

Dentre os modelos de função densidade de probabilidade explícita podemos classificá-los quanto à tratabilidade de sua função densidade de probabilidade. Modelos com funções tratáveis são aqueles cuja função densidade de probabilidade é construída de tal forma a permitir a otimização direta dos seus parâmetros em prol da maximização da verossimilhança em relação a um determinado conjunto de dados.

2.1.1.1 Função de Densidade de Probabilidade Tratável

Um exemplo de tal modelo são as PixelCNNs [16], um tipo de *Fully Visible Belief Network* (FVBN). As FVBNs são uma subclasse de modelos nos quais a distribuição de probabilidade de uma imagem é modelada através da probabilidade conjunta de todos os pixels dessa:

$$P(x) = \prod_{i=1}^{N} p(x_i|x_1, ..., x_{i-1}) \tag{2.4}$$

No caso das PixelCNNs a função que expressa a probabilidade de um pixel i dado os pixels anteriores é modelada através da convolução de um kernel aprendido (F_{kernel}), que seja grande o suficiente para encobri-los, submetido a uma ativação ϕ:

$$p(x_i|x_1, ..., x_{i-1}) = \phi(F_{kernel} * [x_1, ..., x_{i-1}]) \tag{2.5}$$

Outros modelos que se utilizam da modelagem baseada na probabilidade conjunta dos pixels, como PixelRNN [17], também são classificados como modelos autoregressivos, justamente por apresentarem interdependência sequencial entre os pixels de uma imagem.

2.1.1.2 Função de Densidade de Probabilidade Intratável

Por outro lado, existem modelos que apresentam como resultado uma função distribuição de probabilidade aproximada por utilizarem uma abordagem sem solução fechada para a modelagem da distribuição. Esse é o caso dos *Variational Autoencoders* (VAEs) que introduzem um espaço latente de menor dimensão com o objetivo de codificar características semânticas e condicionar a geração de amostras do espaço de alta alta dimensionalidade. Dessa forma expressam o espaço de alta dimensionalidade utilizando a marginalização da distribuição condicional desse espaço em relação ao espaço latente da seguinte forma:

$$p_\theta(x) = \int p_\theta(z)p_\theta(x|z)dz \tag{2.6}$$

No entanto, apesar da distribuição a priori do espaço latente ser conhecida (geralmente utiliza-se uma Gaussiana), o cálculo da integral é intratável devido à distribuição condicional ser modelada por uma rede decodificadora (*decoder*) $p_\theta(x|z)$.

Para resolver esse problema, os VAEs introduzem, além da rede decodificadora p_θ (necessária para geração de amostras), uma rede codificadora q_ϕ (*encoder*) que permite

expressar a probabilidade de uma amostra dada a distribuição que se deseja modelar. Assim p_θ pode ser reescrito de forma a determinar-se um limite inferior da probabilidade de uma amostra, dado um conjunto de parâmetros da rede codificadora e decodificadora. Dessa forma é possível aplicar-se o método da maximização da verossimilhança e otimizar-se os parâmetros de tais redes.

Os VAEs, no entanto, são modelos de densidade de probabilidade explícita, portanto, as redes codificadora e decodificadora não modelam diretamente amostras da distribuição p(x), mas calculam os parâmetros de uma função densidade de probabilidade Gaussiana multivariada, cujas amostras se assemelham a amostras do conjunto de dados utilizados no processo de maximização da verossimilhança.

2.1.2 Modelos Geradores de Função Densidade de Probabilidade Implícita

Por outro lado, nos modelos de função densidade de probabilidade implícita não temos como parte do resultado do treinamento uma função densidade de probabilidade nem um meio de estimar probabilidades relacionadas a uma determinada amostra, mas apenas uma função capaz de amostrar a distribuição modelada que encontra-se implícita no modelo da função geradora. Os modelos de FDP implícita podem ser baseados em Cadeias de Markov, como no caso de *Generative Stochastic Networks*, no qual os parâmetros da função de transição são treinados de forma que amostras obtidas a partir da amostragem da cadeia de Markov eventualmente convirjam para as amostras da distribuição que se almeja modelar. Esses modelos requerem um processo iterativo de amostragem que é normalmente custoso computacionalmente e que muitas vezes falham em escalar para espaços de alta dimensionalidade.

Alternativamente aos métodos baseados em cadeia de Markov, temos os métodos diretos, isto é, que permitem a obtenção sem o processo iterativo, de amostras da distribuição modelada, como é o caso das GANs.

2.2 Redes Adversárias Generativas

Formalmente GANs são modelos probabilísticos estruturados [1] que modelam uma variável observável X condicionada a uma variável latente Z, como denota o grafo na figura 2:

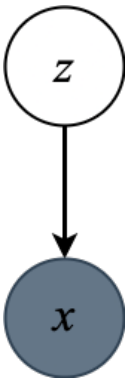


Figura 2 – Estrutura em grafo de uma GAN. Fonte: adaptado de [1]

Topologicamente as GANs são similares aos VAEs, pois são descritas pelo mesmo grafo direcionado. E, assim como os VAEs, as GANs condicionam a amostragem da distribuição modelada a uma variável latente e são constituídas por uma rede geradora (análoga à rede decodificadora do VAE) utilizada para fazer a inferência. No entanto, por ser um modelo de função de distribuição de probabilidade implícita, a rede geradora não apresenta como resultado da geração os parâmetros da função distribuição de probabilidade modelada, mas gera diretamente amostras dessa. Apesar da similaridade topológica, a diferença no resultado da inferência desses modelos, isto é, devido à GAN ser um modelo implícito, implica em formas diferentes de se treiná-los.

O treinamento de uma GAN ocorre através de um jogo no qual uma rede auxiliar, o discriminador (de certa forma análogo ao codificador dos VAEs, tanto em sua topologia quanto ao uso dessa rede somente em tempo de treinamento), é utilizado para julgar as amostras geradas em comparação às amostras reais provenientes do conjunto de treinamento. Ao discriminar as imagens geradas e imagens reais, o discriminador não só aprende por meio de aprendizado supervisionado como diferenciar as duas categorias, uma vez que as imagens são rotuladas quanto à origem, mas também gera o feedback necessário para que o gerador aprenda a gerar imagens mais similares às reais, uma vez que seu objetivo é enganar o discriminador.

2.2.1 Intuição

Intuitivamente pode-se compreender o processo de treinamento de uma GAN como análogo a um falsificador tentando produzir dinheiro falso sem ser detectado pela polícia enquanto a polícia tenta diferenciar o dinheiro original do dinheiro falso. A competição

entre a polícia e o falsificador leva ambos a aperfeiçoarem suas técnicas de detecção e falsificação, respectivamente. Se nenhum dos dois lados se sobressair ao ponto de tornar os esforços do outro fúteis, o processo converge para o estado em que as imagens geradas serão indistinguíveis das imagens reais.

2.2.2 Modelagem

Na prática o gerador consiste numa rede neural G com parâmetros θ_G que tem como entrada amostras, z, do espaço latente, e executa o up-sampling desse com o objetivo de transformá-las em amostra da distribuição sob modelagem. Enquanto que o discriminador, que também é uma rede neural D com parâmetros θ_D, recebe amostras do espaço sob modelagem, sejam elas reais ou geradas, e estima a probabilidade destas serem amostras reais.

O treinamento consiste na amostragem de um minibatch de amostras reais e de amostras do espaço latente z. O minibatch de amostras do espaço latente é submetido ao gerador resultando num minibatch de amostras geradas. Ambos minibatches, são submetidos ao discriminador que estima as probabilidades de cada amostra ser real. De posse dos rótulos de cada imagem e da predição do discriminador, calcula-se o gradiente da perda de ambos, que, por fim, é retropropagado promovendo a atualização dos parâmetros θ_G e θ_D de forma a aumentar a capacidade de gerar imagens mais realistas e discriminar com maior precisão imagens reais de imagens geradas, respectivamente.

A dinâmica de treinamento descrita anteriormente de forma analógica, pode ser formalmente modelada por meio da Teoria dos Jogos como um jogo de soma zero não-cooperativo de dois jogadores, também conhecido por jogo minimax.
Em teoria dos jogos um jogo em forma normal é descrito por:

1. Um conjunto finito de jogadores $N = \{1, 2, \ldots, n\}$
2. Uma coleção de conjuntos de estratégias $\{S_1, S_2, ..., S_n\}$
3. Um conjunto de funções *payoff* $\{v_1, v_2, ..., v_n\}$ definidas tais que:
   $v_i : S_1 \times S_2 \times ... \times S_n \to \mathbb{R} \ \forall \ i \in N$

Dessa forma pode-se definir o treinamento de uma GAN como um jogo:

1. de dois jogadores, $N = \{1, 2\}$, representando o gerador e discriminador
2. com dois conjuntos de estratégias S_1 e S_2 definidos respectivamente pelo conjunto de parâmetros do gerador e discriminador, respectivamente, tais que:
   - $S_1 = \{\theta_G \in \Theta_G\}$
   - $S_2 = \{\theta_D \in \Theta_D\}$
3. E duas funções de *payoff* que descrevem a dinâmica da competição entre o gerador e discriminador, mais precisamente, a função *payoff* do discriminador deve apresentar *payoffs* maiores à medida que esse consiga, em média, detectar com maior acurácia as amostras falsas geradas pelo gerador ao mesmo tempo que também consegue detectar com maior acurácia as amostras reais provenientes do conjunto de treinamento. Enquanto o gerador apresenta função *payoff* oposta, isto é, seu *payoff* aumenta à medida que o *payoff* do discriminador é reduzido. Dessa forma, por ser um jogo de soma zero, pode-se descrever as funções de *payoff* como:
   - $v_1 = J^{(G)} = -J$
   - $v_2 = J^{(D)} = J$

     Tal que $J^{(G)} = -J^{(D)}$

A função de *payoff* pode assumir diversas formas. No artigo que introduziu GANs [6], por exemplo, a função *payoff* proposta é a seguinte:

$$J = \boldsymbol{E}_{x \sim p_{data}(x)}\big[logD(x)\big] + \boldsymbol{E}_{z \sim p_z(z)}\big[log(1 - D(G(z))\big] \tag{2.7}$$

Sendo $D(\cdot)$ e $G(\cdot)$ o discriminador e gerador, respectivamente, e $p_{data}(x)$ e $p_z(z)$ a distribuição de dados e de variáveis do espaço latente, respectivamente. Goodfellow mostra que a solução desse jogo minimax pela perspectiva do gerador, isto é, encontrar o valor minimax do gerador, equivale à convergência da distribuição implícita do gerador, p_g, para a distribuição dos dados, p_{data}.

Basicamente a abordagem para provar esse resultado consiste em mostrar que, dada a função *payoff* acima, para um dado gerador G, temos que o discriminador ótimo (conjunto de parâmetros que maximizam o *payoff* do discriminador é dado por 2.8:

$$D^*_G(x) = \frac{p_{data}(x)}{p_{data}(x) + p_g(x)} \tag{2.8}$$

Portanto, dado um discriminador ótimo para um G arbitrário, podemos descrever o valor do jogo em função de G da seguinte forma:

$$\begin{aligned} C(G) &= \arg\max_{D} V(G,D) \\ &= \boldsymbol{E}_{x\sim p_{data}(x)}\big[logD^*_G(x)\big] + \boldsymbol{E}_{z\sim p_z(z)}\big[log(1-D^*_G(G(z))\big] \\ &= \boldsymbol{E}_{x\sim p_{data}(x)}\big[logD^*_G(x)\big] + \boldsymbol{E}_{x\sim p_g}\big[log(1-D^*_G(x)\big] \\ &= \boldsymbol{E}_{x\sim p_{data}(x)}\left[log\frac{p_{data}(x)}{p_{data}(x)+p_g(x)}\right] + \boldsymbol{E}_{x\sim p_g}\left[log\frac{p_g(x)}{p_{data}(x)+p_g(x)}\right] \end{aligned} \tag{2.9}$$

Sabendo-se que 2.10:

$$\boldsymbol{E}_{x\sim p_{data}(x)}\big[-log2\big] + \boldsymbol{E}_{x\sim p_g}\big[-log2\big] = -log4 \tag{2.10}$$

E subtraindo-se 2.10 de 2.9 é possível reescrever 2.9 da seguinte forma:

$$C(G) = -log4 + KL\left(p_{data}\middle\|\frac{p_{data}+p_g}{2}\right) + KL\left(p_g\middle\|\frac{p_{data}+p_g}{2}\right) \tag{2.11}$$

Como se sabe as divergências de Kullback-Leibler ($KL(\cdot)$) são valores maiores ou iguais à zero, temos que o mínimo global de $C(G)$ é $-log4$. Dado que se p_{data} for igual à p_g, $C(G)$ é igual a $-log4$, podemos concluir que a minimização de $C(G)$, isto é, o valor minimax do jogo para o Gerador, só é atingida quando a distribuição dos dados é perfeitamente modelada pelo Gerador.

2.2.3 Treinamento

Tanto o Gerador quanto os Discriminadores, como visto anteriormente, são geralmente modelados por redes neurais, sejam elas densas ou convolucionais, o que torna a solução do jogo descrito anteriormente um problema intratável de forma analítica, requerendo-se uma abordagem numérica para a obtenção do par de estratégias que soluciona o jogo.

Para tal, é proposto em [6] um algoritmo que consiste na utilização de SGD (*Stochastic Gradient Descent*) aplicado às funções perda associadas ao discriminador e gerador, respectivamente, de forma iterativa e alternada. Dada uma inicialização aleatória de parâmetros para ambos discriminador e gerador, iterativamente amostra-se um *minibatch* do conjunto de treinamento e amostras do espaço latente e utilizam-se essas amostras para, inicialmente, atualizar os parâmetros do discriminador mantendo-se os parâmetros do gerador fixados, tal atualização ocorre por meio da retro-propagação (*backpropagation*) dos gradientes da função perda do discriminador. Em seguida, utiliza-se o discriminador com os parâmetros atualizados para efetuar-se a atualização dos parâmetros do gerador de forma análoga ao discriminador, porém utilizando-se a função perda associada a esse.

2.2.4 Desafios associados ao treinamento de GANs

Apesar do sucesso de GANs em diversas áreas como geração de linguagem natural, síntese de séries temporais e, principalmente, geração de imagens, é notoriamente difícil treiná-las [18].

2.2.4.1 Convergência

Para compreender os desafios no treinamento de GANs é fundamental compreender-se que, diferentemente da otimização de um discriminador num determinado conjunto de dados, o processo de treinamento de uma GAN, como visto anteriormente, não apenas requer a repetida solução desse problema de otimização dado um gerador fixo, mas também a busca da solução do jogo de soma-zero entre o discriminador e o gerador, isto é, encontrar-se o conjunto de estratégias (parâmetros do gerador e discriminador) que minimizem a função *payoff* do jogo. E é justamente essa busca da solução para o jogo que torna o treinamento de uma GAN desafiante e qualitativamente diferente do treinamento de um simples discriminador, pois a atualização da estratégia de um dos jogadores em direção ao ponto de equilíbrio não necessariamente promoverá uma reação do outro jogador na mesma direção, ou seja, o outro jogador pode regredir sua estratégia de forma a combater a atualização mais recente de sua contraparte, mesmo que isso signifique se afastar do ponto de equilíbrio. Essa dinâmica possibilita que o treinamento de uma GAN não atinja convergência, apresentando comportamentos cíclicos ou oscilatórios [18]. Isso pode levar o treinamento a ficar preso num mínimo local, ou então oscilar indefinidamente entre um subconjunto de estratégias. Teoricamente provou-se que GANs em sua formulação original [6] apresentam convergência por meio do método do gradiente aplicado simultaneamente ao gerador e discriminador se as atualizações ocorrerem no espaço de funções. Na prática, no entanto, não é o que ocorre, uma vez que ambas estratégias são definidas por redes neurais e atualizadas no respectivo espaço de parâmetros dessas redes neurais. Não há prova que o treinamento converge com atualizações no espaço de parâmetros, atualmente [1].

2.2.4.2 Colapso de Modo

Além da falha em convergir para a solução ótima do jogo, uma GAN pode apresentar o que é conhecido por colapso de modos, no qual o gerador aprende a mapear o espaço latente para apenas algumas amostras do conjunto de treinamento, isto é, a função densidade de probabilidade modelada pelo gerador colapsa para apenas alguns modos da distribuição dos dados, gerando amostras de boa qualidade porém com baixíssima diversidade.

2.2.4.3 Alta Sensibilidade

Além da existência dos modos de falha citados anteriormente, a dificuldade no treinamento é acentuada devido à alta sensibilidade da GAN quanto à seleção de hiperparâmetros, variações arquiteturais e inicialização. Isso implica que o ajuste de tais hiperparâmetros é desafiador, pois pequenas variações podem ser suficientes para colapsar a GAN para um modo de falha. Por outro lado, a sensibilidade à inicialização dificulta a mensuração de performance, uma vez que para obter resultado com rigor metodológico são necessários múltiplos treinamentos com o mesmo conjunto de hiperparâmetros para obtenção de uma performance média ou máxima [13].

2.2.5 DCGANs

Em 2016, no que tange a aplicação de GANs a geração de imagens, [14] introduziram uma série de diretrizes arquiteturais para utilização de redes convolutivas em GANs que resultou em uma melhora de performance significativa na qualidade das imagens geradas, assim como, a habilidade de gerar imagens de alta dimensão em um único estágio, em contraste com as LAPGANs, que antecedem as DCGANs na capacidade de gerar imagens de alta dimensão, porém requerendo múltiplos estágios de geração. As DCGANs representam um marco na geração de imagens com GANs, sendo base da maioria das GANs utilizadas e de diversas variantes subsequentes.

2.2.5.1 Diretrizes arquiteturais das DCGANs

As DCGANs, como classe, são constituídas por 5 características arquiteturais:

- Utilização de Convoluções e Convoluções Transpostas para efetuar o *downsampling* e *upsampling* no discriminador e gerador, respectivamente.

- Utilização de *Batch Normalization* nas camadas tanto do gerador quanto do discriminador, com exceção da última camada do gerador e primeira camada do discriminador.

- Não utilização de camadas densas para arquiteturas mais profundas.

- Utilização da ativação ReLU em todas as camadas do gerador exceto na última, que deve ser Tanh.

- Utilizar a ativação *leaky ReLU* em todas as camadas do discriminador.

A seguir descreve-se o que se constitui cada uma dessas diretrizes, assim como as motivações por detrás delas.

2.2.5.1.1 Camadas Convolucionais

Redes neurais, em especial perceptrons multicamadas, utilizam por base transformações afins no qual um vetor de entrada é multiplicado por uma matriz de parâmetros e, cujo resultado, é submetido a uma transformação não-linear. Qualquer sinal pode ser vetorizado e submetido a essas redes neurais, porém, sinais como áudio ou imagens, que possuem um estrutura intrínseca expressa na forma como esses dados são ordenados nos arranjos multidimensionais que os definem, perdem tais características topológicas ao serem achatados num vetor unidimensional requerido pela entrada da rede neural. Além disso, a rede neural trata todos os elementos de entrada de forma independente, ignorando relações entre as amostras de uma série temporal, por exemplo, mesmo que essa seja naturalmente unidimensional.

No processamento de imagens, em específico, é evidente que o ordenamento espacial dos pixels de uma imagem contém informação relevante para o processamento dessa. Nesse contexto, a convolução discreta se apresenta como uma boa alternativa.

Convoluções discretas são transformações lineares que preservam o ordenamento espacial do sinal. Uma convolução multi-dimensional é definida pela expressão 2.12:

$$C = \sum_{k_1=-\infty}^{\infty} \sum_{k_2=-\infty}^{\infty} \cdots \sum_{k_M=-\infty}^{\infty} h(k_1, k_2, ..., k_M) x(n_1, n_2 - k_2, ..., n_M - k_M) \tag{2.12}$$

No qual um filtro, ou *kernel* da transformação ($h(k_1, ...k_M)$), é transladado sobre um sinal de entrada, *input feature map*, x, em cada posição ocupada pelo filtro é efetuada a soma dos produtos elemento-a-elemento entre os elementos do filtro e do sinal de entrada. O sinal de saída, *output feature map*, é constituído pelo arranjo ordenado dessas somas de produtos. Essa operação não apenas permite a manipulação do conteúdo do sinal, mas também sua forma, isto é, a dimensão do sinal de saída é dependente não apenas da dimensão do sinal de entrada mas também do *stride* (passo) da translação do filtro, do *padding* aplicado ao sinal de entrada e do tamanho do filtro em si.

Numa rede convolucional os parâmetros que definem o filtro são parâmetros treináveis da rede, portanto isso permite que o modelo aprenda filtros que extraiam as features relevantes do sinal sendo processado, seja para uma tarefa de classificação ou geração.

No contexto de GANs, a utilização de camadas convolucionais, permite que o modelo aprenda a transformação que executará, em contraste com camadas de pooling anteriormente usadas que apresentam operações pré-definidas, como a soma ou máximo dos pixels de uma região.

2.2.5.1.2 *Batch Normalization*

A atualização dos parâmetros de uma rede neural por meio do gradiente em relação à função de perda, isto é, à retropropagação da perda ou erro, é aplicada sob a assunção de que os demais parâmetros das camadas anteriores se mantenham estáticos [19]. Na prática, no entanto, todos os parâmetros são modificados conjuntamente. Do ponto de vista de um determinado parâmetro isto faz com que a superfície sendo percorrida pela descida do gradiente mude não apenas em virtude dos dados utilizados durante o treinamento, mas também da mudança dos parâmetros que antecedem o parâmetro em questão no processo de retropropagação. Isso possibilita que a atualização dos parâmetros siga um alvo móvel uma vez que os parâmetros de uma camada são atualizados assumindo-se que a saída da camada anterior seguirá uma dada distribuição, porém essa distribuição se altera a cada atualização de parâmetros [20]. Tal variação da distribuição da saída das camadas ocultas é conhecida na literatura por *internal covariate shift*.

Batch Normalization é uma técnica proposta para amenizar o impacto do *internal covariate shift* através da normalização da entrada de uma camada. Para efetuar tal normalização o algoritmo da camada de *Batch Normalization* extrai a média e a variância do mini-batch em questão:

$$\mu_{\mathcal{B}} = \frac{1}{m}\sum_{i=1}^{m} x_i \tag{2.13}$$

$$\sigma_{\mathcal{B}}^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \tag{2.14}$$

Normaliza-se a entrada da camada (x_i)

$$\hat{x}_i = \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \tag{2.15}$$

E aplica-se um escalonamento e translação no valor de entrada normalizado

$$y_i = \gamma\hat{x}_i + \beta \equiv BN_{\gamma,\beta} \tag{2.16}$$

Os parâmetros do escalonamento são parâmetros do modelo, portanto, parâmetros a serem aprendidos, de forma a permitir que o poder de representação do modelo possa ser restaurado, isto é, caso γ e β assumam valores que revertam a normalização, caso isso seja a solução ótima.

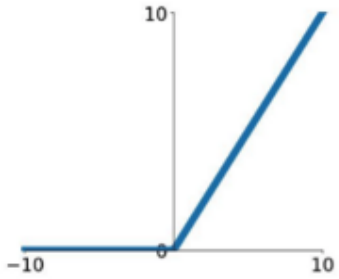
Dessa forma a média e variância da distribuição de entrada é estabilizada, reduzindo-se o impacto da atualização dos parâmetros das camadas anteriores à camada posterior a

normalização. Em modelos de redes profundas, como no caso das DCGANs, o impacto do *internal covariate shift* é ainda mais acentuado por conta do maior número de camadas ocultas, tornando a utilização da técnica de *Batch Normalization* fundamental para estabilização do treinamento. No artigo [14] constatou-se, no entanto, que a aplicação de *Batch Normalization* a todas as camadas causa instabilidade e comportamentos oscilatórios no treinamento e, por esse motivo, a técnica não é aplicada na camada de saída do gerador e na camada de entrada do discriminador.

2.2.5.1.3 Ativações Tanh, ReLU e *Leaky* ReLU

A função de ativação não-linear das redes neurais é o que confere a elas a capacidade de representar funções não-triviais, no entanto, existem diversas opções de funções não-lineares que podem ser utilizadas para fins de ativação de um nó de uma rede neural. No contexto das DCGANs três ativações são particularmente relevantes: ReLU, Tanh e *Leaky* ReLU.

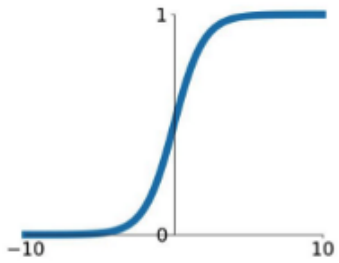
A ReLU ou Unidade Linear Retificada é definida por:



$$\begin{cases} 0, \, x \leq 0 \\ x, \, x > 0 \end{cases} \tag{2.17}$$

As vantagens da ReLU incluem uma melhor propagação de gradientes, uma vez que apresenta menor tendência à saturação (quando comparada com ativações como a Sigmóide que saturam em ambas as direções), invariância ao escalonamento e eficiência computacional, requerendo apenas operações de comparação, adição e multiplicação. Por outro lado, a ReLU apresenta desvantagens, como não ser diferenciável em zero e permitir estados no qual o nó da rede torna-se inativo devido à saturação promovida por entradas com valores negativos. No contexto das DCGANs, as ativações ReLUs são empregadas no gerador com exceção da última camada, que utiliza a ativação Tanh.

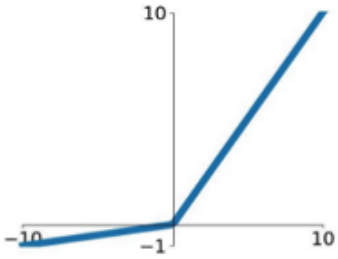
A ativação Tanh ou Tangente Hiperbólica é definida por:



$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.18}$$

Suas vantagens incluem ser centrada em zero e ser infinitamente diferenciável, no entanto, por ter a saída limitada entre -1 e 1, apresenta saturação em ambas direções. Em geral isso é problemático pois promove gradientes evanescentes. No contexto das DCGANs, no entanto, é justamente essa característica que motiva o emprego dessa ativação com o objetivo de saturar mais rapidamente a saída do gerador e cobrir o espaço de cores da distribuição dos dados de treinamento [14].

Por fim, o artigo introdutório das DCGANs sugere o uso das *Leaky* ReLUs no discriminador. A *Leaky* ReLU é uma variante da ReLU que corrige o problema de desativação do nó introduzindo uma pequena inclinação na faixa negativa da ativação, permitindo que o gradiente seja propagado mesmo para valores de entrada negativos.



$$\begin{cases} \alpha x, \; x \leq 0 \\ \alpha < 0 \\ x, \; x > 0 \end{cases} \tag{2.19}$$

2.2.6 Mensuração da Performance de GANs

Com a introdução de diferentes arquiteturas, perdas, regularizações e técnicas no geral para o aperfeiçoamento de GANs fica evidente a necessidade de um método objetivo para comparar a performance desses modelos.

Diferentemente dos modelos discriminadores, que estão no âmbito do aprendizado supervisionado, os modelos geradores não apresentam uma verdade fundamental para que se possa comparar a fim de avaliar a qualidade das amostras geradas. Embora seja possível a comparação das amostras geradas com as amostras do conjunto de dados, tal avaliação não é capaz de mensurar o nível de generalização do modelo e, portanto, sua capacidade de gerar amostras inéditas.

Dessa forma se fez necessária a implementação de uma métrica que pudesse refletir tanto o realismo das imagens geradas quanto a capacidade de gerar imagens diversas.

2.2.6.1 *Inception Score*

A primeira métrica a ganhar notoriedade para tal fim foi o Inception Score (IS). O IS é uma medida que tem por objetivo estar positivamente correlacionado com o julgamento subjetivo humano quanto ao realismo e diversidade de um conjunto de imagens. Para atingir esse objetivo é utilizado o modelo Inception V3 como forma de mimetizar a percepção humana. O Inception V3 é um classificador que atribuirá uma série de probabilidades de uma dada imagem pertencer a uma determinada categoria, ou seja, se apenas uma categoria apresentar uma probabilidade distintamente mais alta que as demais, isso significa que a imagem apresenta algo que seria reconhecível para um humano, enquanto que uma imagem que apresente probabilidades similares dentre todas as categorias provavelmente não contém elementos distintivos de uma imagem natural, reconhecível pela visão humana. Dessa forma, avaliando-se a distribuição das probabilidades em relação à categorias, é quantificado o realismo de uma imagem na composição do IS.

Para avaliar a diversidade de imagens, o IS analisa a diversidade de categorias que foram distinguidas pelo modelo Inception V3, isto é, caso, dentre o grupo de imagens geradas, note-se que a maioria das imagens é classificada sob uma mesma categoria, é possível que o processo de geração esteja apresentando colapso de modo. Isto é, embora gere imagens realistas não foi capaz de generalizar a distribuição e gerar imagens diversas. Por outro lado, caso a diversidade de categorias seja alta, isso significa que houve generalização e diversas imagens distintas e realistas estão sendo geradas.

Embora o *Inception Score* já tenha se mostrada uma boa métrica e apresente correlação com o julgamento subjetivo humano [21], uma de suas grandes limitações é não comparar de fato as imagens reais, ou suas estatísticas, com as imagens geradas, mas simplesmente avaliar as características estatísticas da classificação feita pelo modelo

Inception V3 das imagens geradas. Além disso o IS é altamente dependente das classes utilizadas para o treinamento do classificador, sendo incapaz de estimar a variabilidade intra-classe e, caso o gerador aprenda a memorizar as imagens utilizadas no treinamento sem incorrer em colapso de modo, a métrica poderá apresentar altos valores sem necessariamente o modelo ter aprendido a generalizar a geração de amostras.

2.2.6.2 *Fréchet Inception Distance*

A *Frechet Inception Distance*, ou FID, foi introduzida por [22] com a proposta de melhorar o, então amplamente utilizado, *Inception Score* [21]. A FID estima a qualidade de imagens sintéticas comparando a resposta das ativações da camada de codificação (camada anterior à camada de classificação) do modelo *Inception V3* [23] entre imagens geradas e imagens reais, o que elimina tanto a forte dependência à classes utilizadas no treinamento quanto a ausência de comparação entre imagens sintéticas e reais por parte do IS. A intuição por trás dessa métrica está na ideia que as camadas finais de um modelo discriminador, em especial um modelo estado da arte como o *Inception v3*, extraem *features* relevantes do ponto de vista da percepção subjetiva de imagens naturais por humanos, isso permite que a comparação de similaridade entre as imagens mimetize a percepção humana de similaridade.

Partindo-se da ideia que dada duas distribuições, p_g e p_r, isto é, a distribuição das imagens geradas e reais, respectivamente, a igualdade $p_g = p_r$ é válida se e somente se:

$$\int p_r(x)f(x)dx = \int p_g(x)f(x)dx \tag{2.20}$$

para todo $f(x)$ que seja função base do espaço das distribuições p_g e p_r. De acordo com [22], a igualdade acima é utilizada para descrever distribuições pelos seus momentos, no qual $f(x)$ são polinômios dos dados x, dessa forma generalizando-se tais polinômios com a camada de codificação do modelo *Inception V3*, como descrito anteriormente, e assumindo-se que sua resposta é uma gaussiana multivariada, são considerados apenas os dois primeiros momentos: média e covariância. Sabendo-se que a diferença entre duas Gaussianas é dada pela distância de Fréchet, define-se a FID:

$$d^2\big((m_g, C_g), (m_r, C_r)\big) = ||m_g - m_r||_2^2 + Tr\big(C_g + C_r - 2(C_g C_r)^{1/2}\big) \tag{2.21}$$

sendo m_g, m_r, C_g e C_r as médias e covariâncias, respectivamente, das amostras reais e sintéticas, calculadas como descrito acima, onde Tr é o traço da matriz resultante.

2.2.7 Ajuste de Hiperparâmetros e Estabilização do Treinamento

Como visto anteriormente, devido à natureza de seu treinamento, as GANs apresentam desafios quanto à convergência e estabilização desse processo. Devido a isso, grande

parte da literatura relacionada à GANs é dedicada à proposição de técnicas, arquiteturas, funções perda e heurísticas na escolha de hiperparâmetros com o propósito de aumentar as chances de sucesso no seu treinamento [24].

Além das modificações arquiteturais, a pesquisa em torno da estabilização das GANs pode ser segmentada na proposição de diferentes funções perda, das quais se destacam as perdas de Wasserstein [25], perda não saturante, customização de funções perda, como *Feature Matching*, *label flipping* e *one-side label smoothing* [21], regularizações, como *Gradient Clipping* [26], técnicas de treinamento como a normalização espectral [27].

Devido à grande diversidade de técnicas existentes, o escopo das subseções seguintes será limitado às técnicas empregadas neste trabalho.

2.2.7.1 Perda Não Saturante

A formulação da perda do gerador, tal qual proposta no artigo originário das GANs [6] é estruturada de forma a possibilitar a análise teórica e a consequente prova de que há convergência quando o gerador e discriminador são atualizados no espaço de função. No entanto, a perda:

$$J = log\big(1 - D(G(z))\big) \tag{2.22}$$

apresenta gradientes fracos nos estágios iniciais de treinamento, quando o gerador não consegue gerar amostras realistas, pois satura para valores muito próximos a zero, como pode ser visto na figura 3:

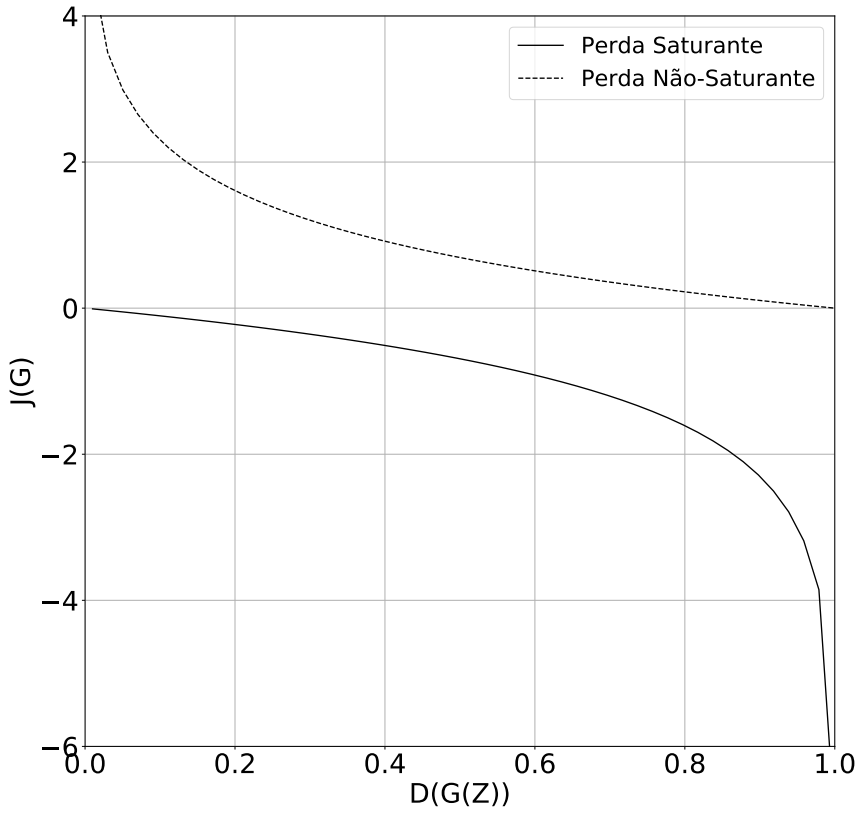


Figura 3 – Comparativo entre a Perda Saturante e Não-Saturante. Fonte: Autor

Para contornar-se esse comportamento, em [6] propõe-se uma perda heuristicamente motivada, isto é, que não possui prova de convergência, mas que apresenta comportamento não-saturante no início do treinamento, permitindo que gradientes mais fortes cheguem ao gerador. A perda não-saturante:

$$J = -log\big(D(G(z))\big) \tag{2.23}$$

apresenta uma resposta análoga a anterior, isto é, seu valor se reduz logaritmicamente à medida que o gerador é capaz de enganar o discriminador. No entanto, a sua singularidade encontra-se no extremo em que o gerador não consegue gerar boas amostras, o que implica em fortes gradientes para que os parâmetros se afastem dessa região.

Embora seja uma modificação simples em relação à versão canônica, comparada a outras perdas propostas na literatura, como a *Wassertein Loss, Least Squares loss, Hinge loss*, os autores em [13] mostram por meio de um estudo empírico de larga escala que a perda não saturante é eficiente e deve ser a escolha padrão, em conjunto com a normalização espectral, ao se aplicar GANs a um novo conjunto de dados com um orçamento computacional limitado.

Na prática, tanto a perda do gerador quanto os dois termos da perda do discriminador são implementados por meio da função de perda de entropia cruzada binária (FPECB):

$$J = -ylog(\hat{y}) + (1-y)(-log(1-\hat{y})) \tag{2.24}$$

Na perda original, motivada pela análise teórica, a classe das amostras geradas é, naturalmente, zero, o que, ao aplicarmos a FPECB, implica na redução da expressão para o segundo termo, que condiz com a formulação para perda mostrada anteriormente. No entanto, é possível observar que o primeiro termo corresponde à perda não saturante, dessa forma, a implementação da perda não saturante é feita por meio da inversão dos rótulos no cálculo da perda do gerador, de forma que a expressão da FPECB se reduza ao primeiro termo ao invés do segundo.

2.2.8 *Feature Matching*

A técnica de *Feature Matching* consiste numa modificação na perda do gerador. Em geral a perda do gerador está relacionada com a decisão tomada pelo discriminador frente uma amostra gerada. Tal comparação comprime toda a informação sobre como o discriminador classificou a amostra apenas no resultado da classificação. Alternativamente a técnica de *Feature Matching* propõe que o objetivo do gerador e, portanto, sua função perda, seja gerar amostras cujas ativações de uma camada intermediária do discriminador se assemelhem com as ativações promovidas por imagens reais. Para que o discriminador consiga classificar imagens com acurácia, esse deve capturar atributos significativos da imagens reais, tais atributos encontram-se codificados em suas camadas intermediárias, tornando natural que as imagens geradas devam buscar reproduzir as estatísticas das ativações dessas camadas de forma similar às imagens reais. Para tal, [21] propõe a seguinte perda para o gerador:

$$J = \left\| \boldsymbol{E}_{x \sim p_{data}} \boldsymbol{f}(x) - \boldsymbol{E}_{z \sim p_z} \boldsymbol{f}(G(z)) \right\|_2^2 \tag{2.25}$$

sendo $\boldsymbol{f}(\cdot)$ a ativação de uma camada intermediária do discriminador. Dessa forma, a perda proposta consiste no erro quadrático médio entre as ativações resultantes das amostras reais e geradas.

2.2.9 *One-side label smoothing*

A técnica de *one-side label smoothing* consiste em penalizar o excesso de confiança do discriminador tornando os rótulos das amostras reais inferiores a 1, geralmente utilizando-se 0.9. Dessa forma, caso o discriminador gere *logits* muito elevados esse é imediatamente penalizado, tendo que reduzir a confiança e, portanto, a probabilidade da predição sobre o dado real para valores que se aproximem do fator suavização utilizado.

Como o funcionamento de GANs se baseia na capacidade do discriminador em estimar a razão das densidades de probabilidade entre o conjunto de dados e o gerador, o desencorajamento do excesso de confiança permite que as probabilidades estimadas sejam mais suaves.

2.2.10 Hiperparâmetros de Treinamento

No artigo proponente das DCGANs também é feita a sugestão de utilização do otimizador Adam para efetuar-se o treinamento e, desde então, a sua utilização tornou-se canônica na literatura. O algoritmo de otimização Adam combina características dos otimizadores AdaGrad e RMSProp, sendo um otimizador que adapta a taxa de aprendizado com base nas médias móveis exponenciais do primeiro e segundo momento do gradiente e apresenta três hiperparâmetros:

- α: também conhecido como a taxa de aprendizagem, é a proporção pela qual os gradientes são atualizados a cada iteração do treinamento
- β_1: taxa de decaimento exponencial da média móvel do primeiro momento
- β_2: taxa de decaimento exponencial da média móvel do segundo momento

No contexto de GANs esses hiperparâmetros geralmente requerem otimização para uma determinada arquitetura e conjunto de dados, isto é, em geral os resultados reportados para uma determinada arquitetura ou técnica são posteriores à otimização dos hiperparâmetros de treinamento.

Além dos parâmetros do otimizador há também a presença do parâmetro *momentum* da técnica de *Batch Normalization*. Esse parâmetro influencia o quanto de inércia a média utilizada para normalização do lote apresenta, a medida que essa é atualizada iterativamente em diferentes lotes. Embora seja um hiperparâmetro de treinamento, isto é, que não tem influência no processo de inferência, sua otimização é raramente mencionada na literatura.

2.2.11 Hiperparâmetros de Arquiteturais

Apesar das diversas restrições arquiteturais propostas pela DCGAN, há ainda diversos graus de liberdade quanto a configuração de sua arquitetura, esses incluem:

- Número de filtros por camada
- Dimensão do kernel das camadas convolucionais
- Tamanho do *stride* das camadas convolucionais

- Dimensão das camadas convolucionais, exceto a última camada do gerador e a primeira camada do discriminador

- Número de camadas

Tanto quanto é de conhecimento do autor desse trabalho, a literatura formal não contempla recomendações quanto ao ajuste desses parâmetros e às arquiteturas disponibilizadas, embora compartilhem alguns princípios em comum, apresentam variabilidade, o que sugere que essas decisões estão sendo tomadas por meio de tentativa e erro e, carecendo de justificativa, terminam por não serem explicitadas e formalizadas [14] [26] [27] [28] [29].

3 Metaparametrização Arquitetural

É natural questionar-se sobre o impacto na performance por conta de variações nos hiperparâmetros arquiteturais, não apenas com o objetivo de otimizar problemas já explorados, mas também de determinar-se diretrizes de forma a sistematizar a escolha desses hiperparâmetros no contexto de novos problemas e conjuntos de dados.

Uma investigação exploratória com base em testes empíricos, além de eventualmente proporcionar heurísticas para uso imediato, pode resultar em intuições para futuras abordagens teóricas. Um desafio, no entanto, da aplicação de uma abordagem exploratória nesse contexto é o elevado número de variações possíveis da arquitetura de uma DCGAN e, consequentemente, um número proibitivamente alto de combinações.

Ao observar-se, no entanto, as escolhas de hiperparâmetros arquiteturais feitas por praticantes e pela literatura [14] [26] [27] [28] [29], percebe-se que, embora hajam variações, essas aparentam seguir padrões que não fazem parte da especificação original das DCGANs, mas que contribuem para redução do espaço de possibilidade das formas que uma DCGAN pode assumir. De forma concreta, os padrões apresentados são descritos a seguir:

- Distribuição do número de filtros por camada:
    - No gerador, como esse necessariamente executa o processo de upsampling e, consequentemente, a dimensão das camadas é crescente, a distribuição do número de filtros por camada geralmente segue a tendência contrária, isto é, a medida que a dimensão de uma camada aumenta o número de filtros se reduz.
    - No discriminador há um padrão simetricamente oposto, à medida que as dimensões das camadas se reduzem, devido ao *downsampling* promovido por esse, o número de filtros cresce.
- Progressão da dimensão das camadas convolucionais:
    - Embora a dimensão das camadas do gerador apresentem necessariamente crescimento entre a primeira e a última por conta do *upsampling*, observa-se que tal progressão sempre ocorre de forma monotônica, porém a taxa com que tal progressão monotônica ocorre apresenta variações.
    - No discriminador o mesmo padrão ocorre porém de forma monotonicamente decrescente.
- Número de camadas

- Em geral, o discriminador é mais profundo do que o gerador

Como pode-se observar pelos padrões descritos, há claramente uma redução no espaço de hiperparâmetros, visto que tanto a dimensão das camadas, quanto o número de filtros apresentados por essas, possuem correlações entre si, ou seja, possuem relação, o que permite um parâmetro ser determinado a partir de outros.

Por outro lado, a exploração desse espaço de hiperparâmetros, mesmo considerando a compressão imposta por tais padrões, não se torna mais simples sem uma forma estruturada de sistematizar as relações que esses representam. Isto é, mesmo sabendo que as dimensões progridem de forma monotônica ao longo das camadas, por exemplo, as dimensões de cada uma das camadas e as possíveis combinações a serem exploradas ainda requerem determinação. Além disso, a escolha de outros hiperparâmetros como, por exemplo, o número de camadas afeta diretamente o conjunto de hiperparâmetros possíveis que constitui as dimensões de cada uma das camadas, revelando a não ortogonalidade da parametrização direta, isto é, os parâmetros não podem ser livremente combinados.

A fim de permitir uma fácil exploração do espaço de hiperparâmetros, é proposta uma metaparametrização com três objetivos:

- Incorporar os padrões observados diretamente na parametrização de forma a reduzir o número de hiperparâmetros ajustáveis e, consequentemente, tornar qualquer escolha de parâmetros no espaço de metaparâmetros uma escolha válida perante tais padrões.

- Garantir a ortogonalidade entre os parâmetros do espaço de metaparâmetros, isto é, a escolha do valor de um metaparâmetro não influencia o espaço dos demais metaparâmetros.

- Normalização de parâmetros, isto é, criar, quando possível, metaparâmetros que estejam restritos ao intervalo $[-1, 1]$ de forma a simplificar o *sweep* e a segmentação.

O resultado final da metaparametrização proposta pode ser visto na tabela 2:

<table>
<tr><th>Hiperparâmetros Arquiteturais</th><th colspan="2">Metaparametrização</th></tr>
<tr><td>Dimensão da primeira camada</td><td colspan="2">Dimensão da primeira camada</td></tr>
<tr><td>Dimensão da segunda camada
⋮
Dimensão da Nésima camada</td><td colspan="2" rowspan="2">Taxa de Progressão da Dimensão
das Camadas Convolutivas</td></tr>
<tr><td>Stride, Kernel, input e output padding
da segunda camada convolutiva
⋮
Stride, Kernel, input e output padding
da Nésima camada convolutiva</td></tr>
<tr><td>Número de filtros da segunda camada
⋮
Número de filtros da Nésima camada</td><td>Taxa de Progressão
do Número de Filtros</td><td>Número Total
de Parâmetros</td></tr>
<tr><td>Profundidade</td><td colspan="2">Profundidade</td></tr>
</table>

Tabela 1 – Comparativo entre a parametrização direta (hiperparâmetros arquiteturais) e a metaparametrização

3.1 Taxa de Progressão da Dimensão das camadas convolutivas

No caso do gerador, a dimensão da primeira camada convolutiva tem valor arbitrário, isto é, pode assumir qualquer dimensão uma vez que a camada anterior, por ser uma camada densa, não impõe nenhuma restrição. A sua última camada, por outro lado, tem as dimensões determinadas pelo tamanho das imagens que se deseja gerar. Dadas as dimensões da camada inicial há diversas combinações possíveis de *padding, stride e kernel* que permitem as dimensões progredirem de forma monotônica até a dimensão da camada final. Isto é, existem combinações em que as primeiras camadas apresentarão dimensões similares e essa progredirá rapidamente para a dimensão de saída ao se aproximar da última camada, conferindo à progressão um caráter com morfologia mais similar a uma curva exponencial. Enquanto que, outras combinações apresentarão uma taxa de crescimento mais suave, ou então rapidamente crescerão para próximo da dimensão de saída, apresentando pouca variação nas camadas finais, similar a morfologia de uma curva logarítmica. As três progressões arquetípicas são ilustradas na figura 4.

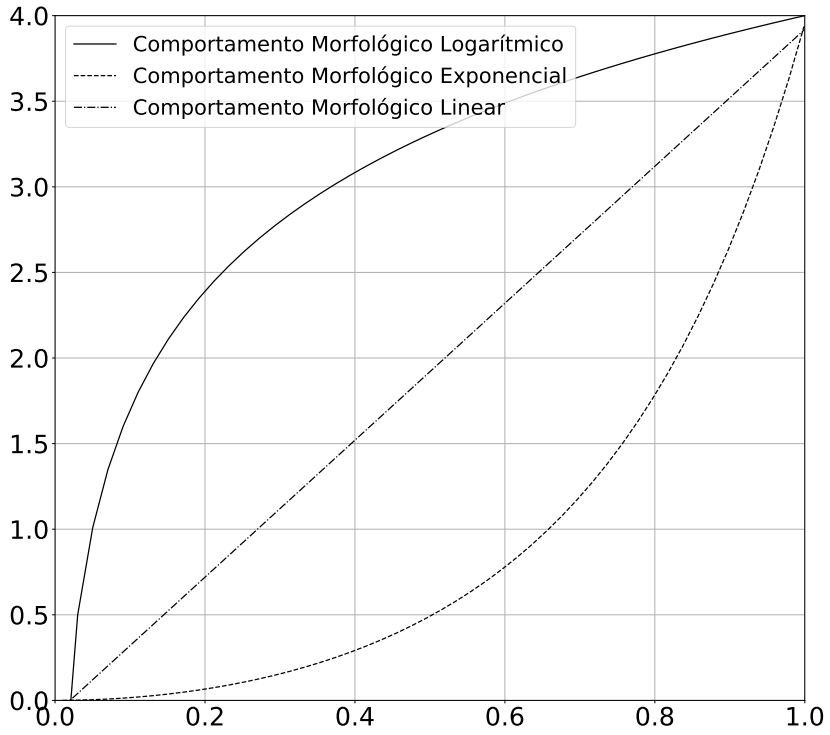


Figura 4 – Morfologias arquetípicas da progressão de dimensões. Fonte: Autor

O metaparâmetro Taxa de Progressão da Dimensão das camadas convolutivas

(TPD) permite o controle da morfologia, como as dimensões progridem ao longo das camadas. Uma TPD igual à -1 implica que a progressão é a mais exponencial possível dada a dimensão inicial, final e o número de camadas da rede, enquanto que uma TPD igual à 1 apresenta o caráter mais logarítmico possível nas condições citadas. Na figura 5 exemplificam-se as diversas progressões que um gerador de quatro camadas de dimensão inicial e e final iguais à 7 e 28, respectivamente, admite.

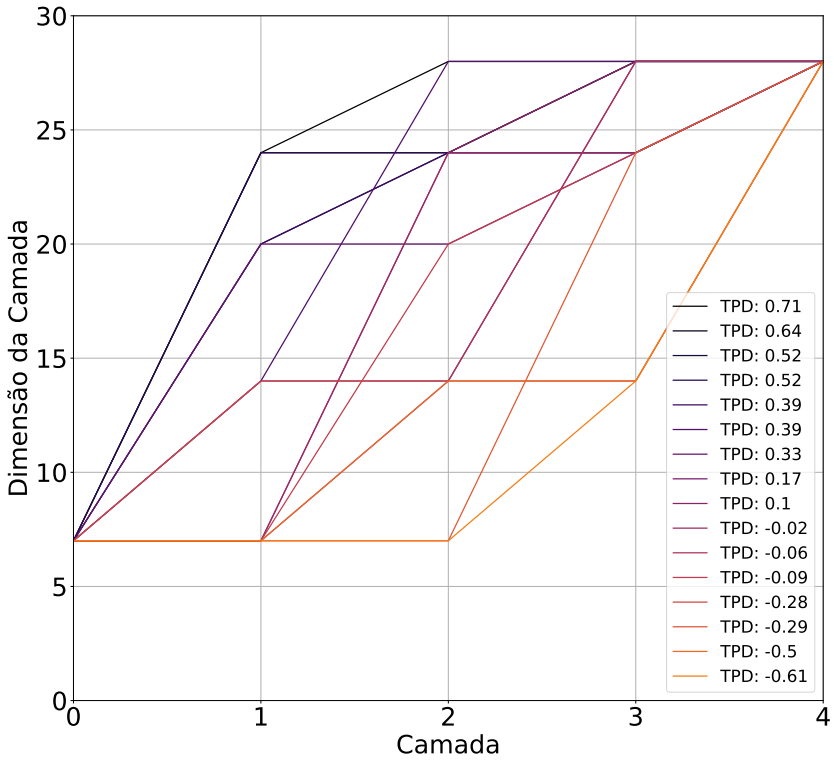


Figura 5 – TPDs possíveis para um gerador de quatro camadas de dimensão inicial igual à 7 e final igual à 28. Fonte: Autor

No caso do discriminador os mesmos princípios se aplicam, porém a camada inicial é determinada pelo tamanho da imagem enquanto a última pode ter dimensão arbitrária uma vez que se conecta com uma camada densa. Dessa forma o mesmo princípio se aplica na definição da TPD, com a diferença que o caráter morfológico da progressão está espelhado, uma vez que que o discriminador efetua o *downsampling* e as progressões são monotonicamente decrescentes.

3.2 Taxa de Progressão do Número de Filtros por camada convolutiva

A Taxa de Progressão do Número de Filtros (TPNF) por camada convolutiva é definida a partir dos mesmos princípios da TPD, porém, enquanto a última se baseia nas dimensões da primeira e última camada para definir a dimensão das demais camadas, a TPNF determina o número de filtros a partir das dimensões das camadas e o número total de parâmetros do modelo. Assim como na TPD, a TPNF é um metaparâmetro definido no intervalo $[-1, 1]$. Uma TPNF igual à -1 implica, no caso do gerador, que a progressão apresenta um decrescimento mais acentuado nas primeiras camadas ou, no caso do discriminador, um crescimento mais acentuado nas últimas camadas, conferindo à progressão um caráter morfológico semelhante a uma curva exponencial. Enquanto que em uma TPNF igual à 1 o oposto ocorre, isto é, no caso do gerador uma progressão com decrescimento mais acentuado nas últimas camadas e, no caso do discriminador, um crescimento mais acentuado nas primeiras camadas do discriminador. Na figura 6 exemplifica-se as diversas progressões de números de filtros que um gerador de imagens de três canais de quatro camadas de dimensão inicial e e final iguais à 7 e 28, respectivamente, admite.

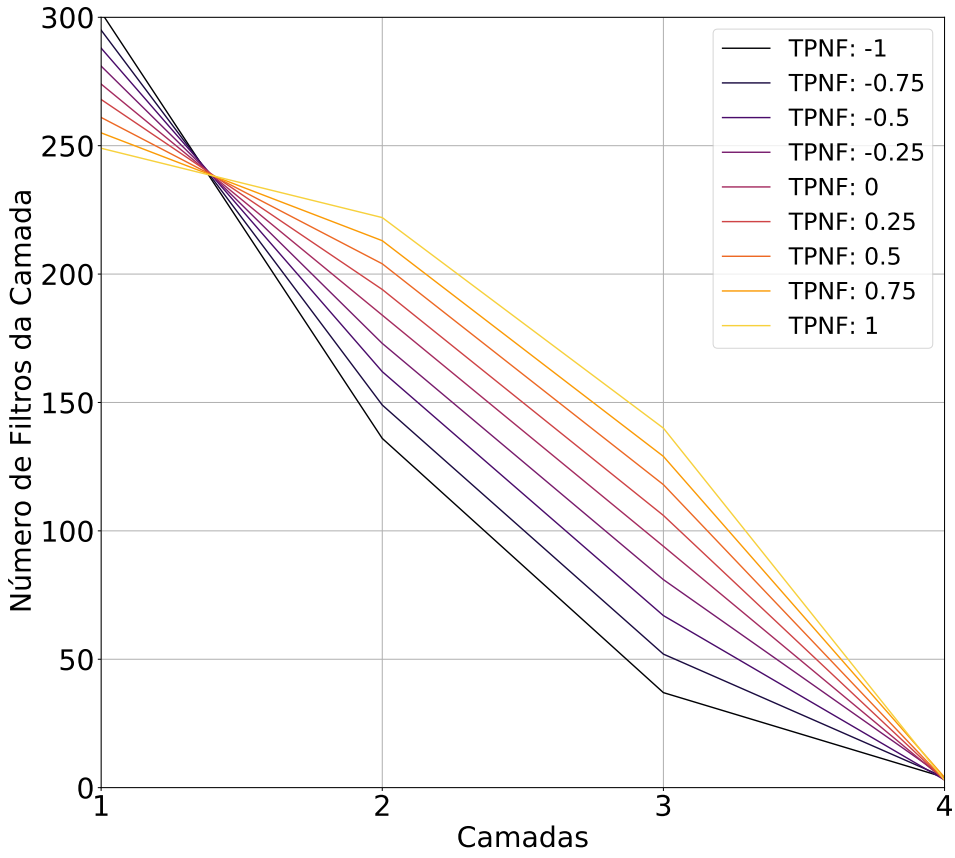


Figura 6 – Exemplos de TPNFs para um gerador de quatro camadas de dimensão inicial igual à 7, final igual à 28 e imagem de saída com 3 canais. Fonte: Autor

No discriminador, os mesmos princípios se aplicam, no entanto, o número de filtros apresenta caráter monotonicamente crescente ao longo das camadas.

3.3 Determinação dos hiperparâmetros arquiteturais a partir dos metaparâmetros

Discutiu-se anteriormente os metaparâmetros propostos neste documento e suas vantagens, no entanto, a determinação dos hiperparâmetros arquiteturais a partir dos metaparâmetros é o elemento fundamental que permite a exploração de tal técnica. A conversão dos metaparâmetros TPD e TPNF no conjunto de dimensões e número de filtros por camada ocorre em duas etapas, como será descrito abaixo, sendo o resultado da primeira etapa, o cálculo das dimensões por camada, necessário para o cálculo da segunda etapa, que resulta no número de filtros por camada.

3.3.1 Cálculo das dimensões por camada

O cálculo da dimensões por camada é dependente de quatro variáveis:

TPD: metaparâmetro fornecido pelo usuário

Dimensão inicial: dimensão de entrada da primeira camada convolutiva, valor arbitrário fornecido pelo usuário

Dimensão Final: dimensão de saída da última camada convolutiva, determinada pelo tamanho das imagens que se deseja gerar

Profundidade: hiperparâmetro arquitetural fornecido pelo usuário

O primeiro passo para o cálculo das dimensões é a geração de uma árvore contendo todas as possíveis redes com a profundidade especificada. O nó raiz da árvore representa a primeira camada convolutiva, os nós filhos, por sua vez, representam todas as possíveis camadas convolutivas seguintes. Dado um conjunto de *strides*, dimensões do *kernel, input* e *output paddings*, a dimensão dos nós filhos é determinada a partir de todas as permutações possíveis dos elementos dos conjuntos anteriormente mencionados. Novos nós são gerados a partir dos nós filhos iterativamente até atingir-se um número de camadas na árvore equivalente à profundidade especificada para a rede, como exemplificado na figura 7.

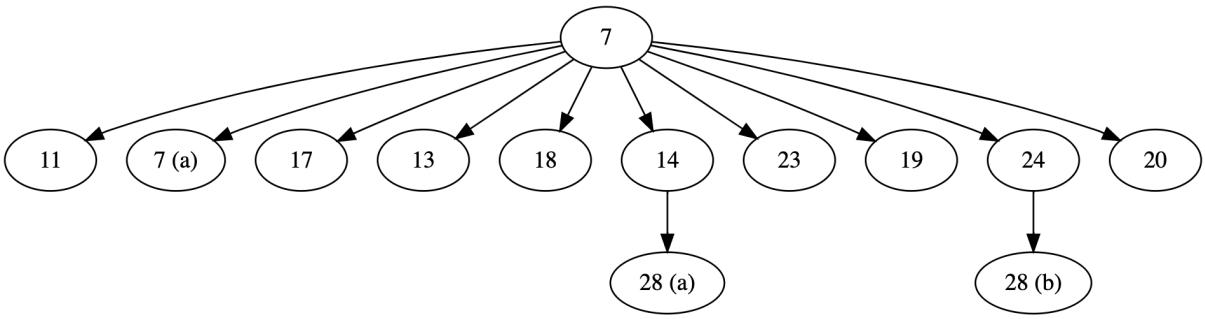


Figura 7 – Árvore representando todas as possíveis progressões de dimensões de um gerador de duas camadas. Cada nó da árvore apresenta sua respectiva dimensão de saída, sendo o nó raiz representativo da dimensão de entrada da rede convolutiva. A escolha de um gerador de apenas duas camadas de profundidade se deve ao aumento exponencial no tamanho do grafo à medida que o número de camadas cresce. Fonte: Autor

Na árvore gerada acima, cada aresta representa uma camada da rede convolutiva, o nó pai sua dimensão de entrada e o nó filho a dimensão de saída. Cada camada convolutiva

possui um conjunto de parâmetros que a definem: *strides*, dimensões do *kernel, input* e *output paddings*. A relação entre esses parâmetros e as dimensões da camada é a seguinte:

$$
\begin{aligned}
& o = (i-1)s + k - 2p + op \\
& \text{Sendo,} \\
& o \rightarrow \text{dimensão de saída} \\
& i \rightarrow \text{dimensão de entrada} \\
& s \rightarrow \textit{stride} \\
& k \rightarrow \text{dimensão do } \textit{kernel} \\
& p \rightarrow \textit{input padding} \\
& op \rightarrow \textit{output padding}
\end{aligned}
\tag{3.1}
$$

De posse da árvore gerada anteriormente, extraem-se todos os ramos que partem do nó raiz e terminam num nó da camada correspondente à profundidade da rede e que tenham dimensão igual à dimensão final especificada, como mostrado na figura 8.

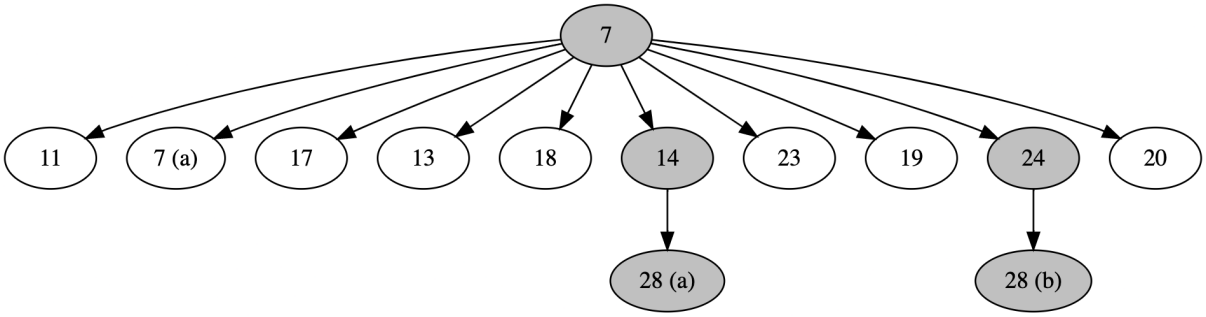


Figura 8 – Subconjunto de ramos que atendem às especificações das dimensões de entrada e saída do gerador hachurados. No caso de um gerador de apenas duas camadas de profundidade, dimensão de entrada e saída iguais a 7 e 28, respectivamente, apenas dois ramos atendem a especificação. Fonte: Autor

Calcula-se a TPD para cada um dos ramos extraídos. O cálculo da TPD consiste na normalização da área sob a curva que descreve a dimensão das camadas da rede no intervalo [-1,1]. A normalização é feita a partir do cálculo da área sob a curva de duas redes hipotéticas A e B, a rede A representa uma progressão na qual a última camada apresenta dimensão igual à dimensão final e todas as camadas anteriores dimensão igual à dimensão inicial, isto é, a rede com progressão das dimensões com caráter morfológico mais exponencial possível e, portanto, apresenta a menor área sob a curva possível, enquanto que a outra rede, B, representa a rede com progressão das dimensões com caráter morfológico mais logarítmico possível, isto é, uma rede que assume a dimensão final como dimensão de saída de sua primeira camada convolutiva e mantém essa dimensão até a camada final, apresentando, portanto, a maior área sob a curva possível. Dado que uma rede com TPD

igual à 1 deve apresentar área sob a curva igual à rede B, a rede com TPD igual à -1, área igual à rede A e a rede com TPD igual à zero, área sob a curva igual à representada pela média entre as redes A e B, temos as seguintes expressões para o cálculo da TPD para os ramos obtidos anteriormente:

$$TPD(r) = \frac{\big(A(r) - \mu\big)}{\sigma}$$

Sendo,

$$\begin{aligned} &A(\cdot) \rightarrow \text{área sob a curva de um determinado ramo/rede} \\ &\mu = \frac{A(r_A) + A(r_B)}{2} \\ &\sigma = \frac{A(r_B) - A(r_A)}{2} \end{aligned} \tag{3.2}$$

Com a TPD de todos os ramos calculadas é possível selecionar-se o ramo cuja TPD mais se aproxima da TPD fornecida pelo usuário e, dessa forma, obter-se todos os parâmetros arquiteturais (*strides*, dimensões do *kernel, input* e *output paddings*) que determinam as dimensões de cada uma das camadas.

3.3.2 Cálculo do número de filtros por camada

O cálculo do número de filtros por camada é dependente de:

TPNF: metaparâmetro fornecido pelo usuário

Dimensões de cada camada e hiperparâmetros arquiteturais associados: obtidos na etapa anterior

Número Total de Parâmetros do Modelo (TC): hiperparâmetro fornecido pelo usuário

Profundidade: hiperparâmetro arquitetural fornecido pelo usuário

O cálculo do número de filtros por camada consiste na otimização do número de filtros da primeira camada, no caso do gerador, de forma que a distribuição do número de filtros nas camadas decorrentes, dada a TPNF e o número de filtros da última camada fixado pela quantidade de canais da imagem, se ajuste de forma que o número total de parâmetros do modelo convirja para o número total de parâmetros desejado pelo usuário.

Dada uma função FP que determina a distribuição de filtros ao longo das camadas em função da TPNF, do número de filtros da primeira e da última camada, é possível definir-se uma segunda função, NC, que descreva o total de parâmetros de uma rede em função do número de filtros, dimensões e *kernel* em cada camada, assim como a dimensão

do espaço latente. Dessa forma, temos que o cálculo do número de filtros por camada consiste na solução do seguinte problema:

$$\arg\min_{\phi_0} \Big| TC - NC\big(FP(\phi_0, TPNF), \Delta, \zeta\big) \Big|$$

Sendo,

$$\begin{aligned}
&\phi_0 \rightarrow \textit{número de filtros da primeira camada} \\
&TC \rightarrow \text{Total de Parâmetros do modelo} \\
&NC(\cdot) \rightarrow \text{determina o total de parâmetros do modelo} \\
&FP(\cdot) \rightarrow \text{determina o número de filtros de cada uma das camadas } (\Phi) \\
&\Delta \rightarrow \text{dimensão de cada uma das camadas do modelo} \\
&\zeta \rightarrow \text{dimensão do espaço latente}
\end{aligned} \tag{3.3}$$

Uma vez encontrado ϕ_{0min}, calcula-se o número de filtros em cada camada através de:

$$FP(\phi_{0min}, TPNF) \tag{3.4}$$

A função FP distribui o número de filtros ao longo das camadas parabolicamente, de forma a facilitar a modelagem. Na distribuição do número de filtros a TPNF, similarmente à TPD, controla o quão rápido a distribuição de filtros se aproxima do número de filtros da camada final, enquanto que o ϕ_0 determina o ponto de partida dessa distribuição. Sendo n uma determinada camada da rede e ϕ o número de filtros nessa mesma camada, descreve-se a distribuição dos filtros, FP, como:

$$\phi = \alpha n^2 + \beta n + \gamma \tag{3.5}$$

Sabendo-se que ϕ_N é conhecido (no caso do gerador, consiste no número de canais da imagem sendo gerada, temos que:

$$\begin{aligned}
&\phi_0 = \gamma \\
&\phi_N = \alpha N^2 + \beta N + \gamma \\
&\text{O que resulta em:} \\
&\beta = \delta - \alpha N \\
&\delta = \frac{\phi_N - \phi_0}{N}
\end{aligned} \tag{3.6}$$

Assim, podemos definir o conjunto de funções parabólicas que atendem a condição acima:

$$g(n, \alpha) = \alpha n^2 + (\delta - \alpha N)n + \phi_0 \tag{3.7}$$

Como deseja-se comportamento monotônico na distribuição dos filtros e que atenda ao comportamento associado à TPNF, descrita anteriormente, podemos determinar α de forma a controlar $g(n)$ em função da TPNF da seguinte forma:

$$\alpha = f(TPNF)$$

Tendo as seguintes condições,

$$\begin{aligned} TPNF = 1 &\rightarrow \frac{dg(0)}{dn} \text{ [máximo de } g(n) \text{ na primeira camada]} \\ TPNF = -1 &\rightarrow \frac{dg(N)}{dn} \text{ [máximo de } g(n) \text{ na última camada]} \end{aligned} \tag{3.8}$$

Aplicando-se as condições e resolvendo-se o sistema, obtém-se:

$$\alpha = f(TPNF) = TPNF\frac{\delta}{N} \tag{3.9}$$

Em conclusão, temos que FP:

$$FP(\phi_0, TPNF) = \alpha n^2 + (\delta - \alpha N)n + \phi_0$$

Sendo,

$$\begin{aligned} \alpha &= \frac{\delta TPNF}{N} \\ \delta &= \frac{\phi_N - \phi_0}{N} \\ N &= \text{profundidade} - 1 \end{aligned} \tag{3.10}$$

A função NC, por sua vez, calcula o número total de parâmetros da rede, que no caso de uma DCGAN apresenta dois principais componentes:

$$NC = DLC + CLC$$

Sendo,

$$\begin{aligned} &DLC \rightarrow \text{número total de parâmetros da camada densa} \\ &CLC \rightarrow \text{número total de parâmetros das camadas convolutivas} \end{aligned} \tag{3.11}$$

O DLC é definido por:

$$DLC = \Delta(0)\,\phi_0\,\zeta$$

Sendo,

$$\begin{aligned} &\Delta(0) \rightarrow \text{dimensão da primeira camada convolutiva} \\ &\phi_0 \rightarrow \text{número de filtros da primeira camada convolutiva} \\ &\zeta \rightarrow \text{dimensão do espaço latente} \end{aligned} \tag{3.12}$$

Enquanto que o CLC é definido por:

$$CLC = \sum_{n=0}^{N} [k_n^2 \, \phi_{n-1} + 4 + 1]\phi_n$$

Sendo,

$$\begin{aligned} &k_n \rightarrow \text{dimensão do kernel da nésima camada} \\ &\phi_n \rightarrow \text{número de filtros da nésima camada} \\ &\phi_{-1} \rightarrow \text{quantidade de canais da saída} \\ &\text{redimensionada da camada densa} \end{aligned} \tag{3.13}$$

De posse das funções FP e NC, emprega-se o método Brent limitado de otimização escalar para encontrar-se o valor de ϕ_0 que minimize a expressão 3.3, ϕ_{0min}.

4 Metodologia

Com o objetivo de explorar uma lacuna na literatura relativa à otimização, estabilização e ajustes de hiperparâmetros de GANs, se propõe uma análise quantitativa do impacto na performance e estabilidade de GANs em função da seleção de hiperparâmetros arquiteturais.

Atentando-se para a disponibilidade limitada de recursos computacionais, selecionou-se dois datasets de benchmark com imagens de baixa resolução e um conjunto reduzido, porém diverso, de experimentos, a fim de obter-se uma visão ampla sobre se e como os hiperparâmetros arquiteturais impactam a performance dos modelos, por meio da comparação da Frechet Inception Distance e sua variância perante diferentes inicializações para diferentes escolhas de hiperparâmetros arquiteturais.

4.1 *Datasets*

Para que fosse possível o treinamento dos modelos num tempo razoável, dado que múltiplos treinamentos são requeridos para cada um dos experimentos, escolheu-se datasets com imagens de baixa resolução, o que reduz significativamente a complexidade dos modelos. De forma a avaliar se a diferença na natureza das imagens afetaria os resultados, optou-se pelos datasets MNIST e CIFAR-10 que são constituídos por imagens de dígitos escritos à mão e imagens naturais de diferentes classes de objetos, respectivamente.

4.1.1 MNIST

O MNIST é um conjunto de dados composto por 60 mil imagens de dígitos escritos à mão. Os dígitos foram normalizados e centrados em imagens de tamanho fixo de 28x28 pixels monocanal. Devido ao tamanho reduzido das imagens e baixa complexidade do seu conteúdo, o MNIST oferece uma forma rápida de testar modelos e hipóteses. [30]

Figura 9 – Amostras do dataset MNIST. Fonte: [2]

4.1.2 CIFAR-10

O CIFAR-10 é um conjunto de dados composto por 60 mil imagens de dez diferentes classes de animais e veículos. Diferentemente do MNIST, o CIFAR-10 apresenta imagens coloridas em três canais de tamanho 32x32 pixels, que são relativamente pequenas e ainda permitem o treinamento de modelos com uma única GPU. Uma das grandes vantagens do CIFAR-10 é permitir expor modelos de menor escala à imagens naturais, o que permitiu, no escopo deste trabalho, avaliar se o impacto da seleção de hiperparâmetros arquiteturais é modificado de forma significativa em função da mudança da natureza das imagens. [31]

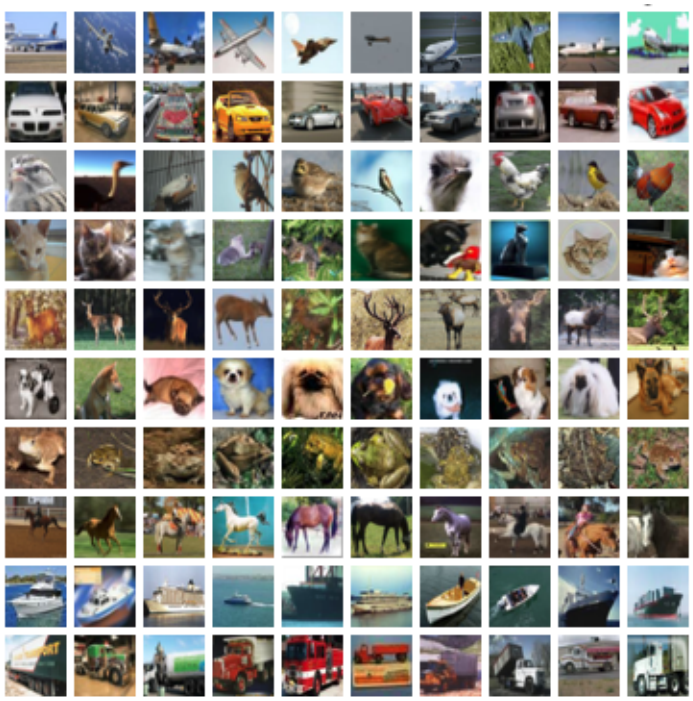


Figura 10 – Amostras do dataset CIFAR-10. Fonte: [3]

4.2 Experimentos

Os experimentos executados neste trabalho podem ser divididos em quatro grupos, cada um explorando uma dimensão de hiperparâmetros arquiteturais definidos pela meta-parametrização. Para cada experimento foram executadas múltiplas rodadas com diferentes combinações de hiperparâmetros de forma a encontrar-se a combinação ótima para uma dada escolha de hiperparâmetros arquiteturais. Por fim, para cada par de hiperparâmetros arquiteturais e seus respectivos hiperparâmetros de treinamento ótimo, foram executados de 3 a 5 rodadas de treinamentos com inicializações aleatórias para estimar-se a consistência dos resultados. A tabela abaixo lista as combinações de hiperparâmetros de treinamento exploradas de forma a otimizar cada experimento, é válido notar que essa seleção de hiperparâmetros é um subconjunto da utilizada no artigo [29]:

Taxa de Aprendizado (α)	Taxa de Decaimento 1 Momento (β_1)
0.002	0.9
0.001	0.9
0.0001	0.9
0.002	0.5
0.001	0.5
0.0001	0.5

Tabela 2 – Espaço de hiperparâmetros de treinamento explorado para cada combinação de hiperparâmetros arquiteturais

É válido mencionar-se que como encontrou-se o conjunto de hiperparâmetros arquiteturais ótimo dentro do espaço de busca proposto, os experimentos executados nesse trabalho podem ser entendidos como um estudo de *ablation* a nível de hiperparâmetros, no qual é variado apenas um dos metaparâmetros de uma escolha ótima de metaparâmetros com o objetivo de avaliar-se a relevância do metaparâmetro para constituição do resultado ótimo [32].

4.2.1 Escalamento e Combinações da Complexidade

Nesse experimento buscou-se avaliar o impacto na performance em função da complexidade, isto é, a quantidade de parâmetros do modelo gerador e discriminador. Em específico, avaliou-se a performance para diferentes complexidades, mantendo-se a razão entre a complexidade do gerador e discriminador constante, e diferentes combinações de complexidade entre o gerador e discriminador, isto é, diferentes razões entre essas complexidades. Os demais hiperparâmetros foram fixados em valores que apresentaram melhor performance nos seus respectivos grupos de experimentos. A tabela 3 e 4 descrevem as combinações de meta-parâmetros utilizadas nos datasets MNIST e CIFAR-10, respectivamente.

TPD (G)	TPD (D)	TPNF (G)	TPNF (D)	Profundidade (G)	Profundidade (D)	Dimensão Inicial (G)	Dimensão Final (D)	Profundidade Inicial dos Filtros	Total de Parâmetros (G)	Total de Parâmetros (D)
									1.5e6	0.5e6
									1.5e6	1.5e6
									1.5e6	3.0e6
									4.5e6	1.5e6
0.4	-0.4	0.4	-0.4	4	3	7	7	64	4.5e6	4.5e6
									4.5e6	9.0e6
									9.0e6	3.0e6
									9.0e6	9.0e6
									9.0e6	18.0e6

Tabela 3 – Combinação de meta-parâmetros utilizada nos experimentos de escalamento de complexidade no dataset MNIST.

TPD (G)	TPD (D)	TPNF (G)	TPNF (D)	Profundidade (G)	Profundidade (G)	Dimensão Inicial (G)	Dimensão Final (D)	Profundidade Inicial dos Filtros	Total de Parâmetros (G)	Total de Parâmetros (D)
									1.5e6	0.5e6
									1.5e6	1.5e6
									1.5e6	3.0e6
									4.5e6	1.5e6
0.0	0.4	0.0	0.0	3	4	7	7	64	4.5e6	4.5e6
									4.5e6	9.0e6
									9.0e6	3.0e6
									9.0e6	9.0e6
									9.0e6	18.0e6

Tabela 4 – Combinação de meta-parâmetros utilizada nos experimentos de escalamento de complexidade no dataset CIFAR-10.

4.2.2 Variação e Combinações de TPDs

Nesse grupo de experimentos buscou-se entender o impacto da TPD na performance avaliando-se tanto a variação dessa no gerador quanto no discriminador. Como no grupo de experimentos anterior, os demais hiperparâmetros foram fixados em valores que apresentaram melhor performance nos seus respectivos grupos de experimentos. As combinações de meta-parâmetros utilizadas encontram-se nas tabelas 5 e 6.

4.2.3 Variação e Combinação de TPNFs

Nesse grupo de experimentos buscou-se entender o impacto da TPNF na performance avaliando-se tanto a variação dessa no gerador quanto no discriminador. Como no grupo de experimentos anterior, os demais hiperparâmetros foram fixados em valores que apresentaram melhor performance nos seus respectivos grupos de experimentos. As combinações de meta-parâmetros utilizadas encontram-se nas tabelas 7 e 8.

4.2.4 Escalamento e Combinações de Profundidade

Nesse grupo de experimentos buscou-se entender o impacto da profundidade do discriminador e gerador na performance avaliando-se o escalonamento de ambos em diferentes razões fixas entre si. Como no grupo de experimentos anterior, os demais hiperparâmetros foram fixados em valores que apresentaram melhor performance nos seus respectivos grupos de experimentos. As combinações de meta-parâmetros utilizadas encontram-se nas tabelas 9 e 10.

TPD (G)	TPD (D)	TPNF (G)	TPNF (D)	Profundidade (G)	Profundidade (G)	Dimensão Inicial (G)	Dimensão Final (D)	Profundidade Inicial dos Filtros	Total de Parâmetros (G)	Total de Parâmetros (D)
-0.4	-0.4									
0.0	-0.4									
0.4	-0.4									
-0.4	0.0									
0.0	0.0	0.4	-0.4	4	3	7	7	64	1.5e6	0.5e6
0.4	0.0									
-0.4	0.4									
0.0	0.4									
0.4	0.4									

Tabela 5 – Combinação de meta-parâmetros utilizada nos experimentos de combinações de TPDs no dataset MNIST.

TPD (G)	TPD (D)	TPNF (G)	TPNF (D)	Profundidade (G)	Profundidade (G)	Dimensão Inicial (G)	Dimensão Final (D)	Profundidade Inicial dos Filtros	Total de Parâmetros (G)	Total de Parâmetros (D)
-0.4	-0.4									
0.0	-0.4									
0.4	-0.4									
-0.4	0.0									
0.0	0.0	0.0	0.0	3	4	7	7	64	1.5e6	0.5e6
0.4	0.0									
-0.4	0.4									
0.0	0.4									
0.4	0.4									

Tabela 6 – Combinação de meta-parâmetros utilizada nos experimentos de combinações de TPDs no dataset CIFAR-10.

TPD (G)	TPD (D)	TPNF (G)	TPNF (D)	Profundidade (G)	Profundidade (G)	Dimensão Inicial (G)	Dimensão Final (D)	Profundidade Inicial dos Filtros	Total de Parâmetros (G)	Total de Parâmetros (D)
		-0.4	-0.4							
		0.0	-0.4							
		0.4	-0.4							
		-0.4	0.0							
0.4	-0.4	0.0	0.0	4	3	7	7	64	1.5e6	0.5e6
		0.4	0.0							
		-0.4	0.4							
		0.0	0.4							
		0.4	0.4							

Tabela 7 – Combinação de meta-parâmetros utilizada nos experimentos de combinações de TPNFs no dataset MNIST.

TPD (G)	TPD (D)	TPNF (G)	TPNF (D)	Profundidade (G)	Profundidade (G)	Dimensão Inicial (G)	Dimensão Final (D)	Profundidade Inicial dos Filtros	Total de Parâmetros (G)	Total de Parâmetros (D)
		-0.4	-0.4							
		0.0	-0.4							
		0.4	-0.4							
		-0.4	0.0							
0.0	0.4	0.0	0.0	3	4	7	7	64	1.5e6	0.5e6
		0.4	0.0							
		-0.4	0.4							
		0.0	0.4							
		0.4	0.4							

Tabela 8 – Combinação de meta-parâmetros utilizada nos experimentos de combinações de TPNFs no dataset CIFAR-10.

TPD (G)	TPD (D)	TPNF (G)	TPNF (D)	Profundidade (G)	Profundidade (G)	Dimensão Inicial (G)	Dimensão Final (D)	Profundidade Inicial dos Filtros	Total de Parâmetros (G)	Total de Parâmetros (D)
				2	2					
				3	2					
				4	2					
				2	3					
0.4	-0.4	0.4	-0.4	3	3	7	7	64	1.5e6	0.5e6
				4	3					
				2	4					
				3	4					
				4	4					

Tabela 9 – Combinação de meta-parâmetros utilizada nos experimentos de combinações de profundidade no dataset MNIST.

TPD (G)	TPD (D)	TPNF (G)	TPNF (D)	Profundidade (G)	Profundidade (G)	Dimensão Inicial (G)	Dimensão Final (D)	Profundidade Inicial dos Filtros	Total de Parâmetros (G)	Total de Parâmetros (D)
				2	2					
				3	2					
				4	2					
				2	3					
0.0	0.4	0.0	0.0	3	3	7	7	64	1.5e6	0.5e6
				4	3					
				2	4					
				3	4					
				4	4					

Tabela 10 – Combinação de meta-parâmetros utilizada nos experimentos de combinações de profundidade no dataset CIFAR-10.

4.3 Medição de Performance

Em cada experimento mediu-se a performance do modelo utilizando-se a Frechet Inception Distance a cada 10 épocas de treinamento, sendo a extensão total do treinamento limitada a 40 épocas para todos os experimentos. Para cada cálculo da Frechet Inception Distance final é resultado da média do cálculo da FID com 2560 pares de imagens reais e geradas.

A medição da FID ao longo treinamento é feita com o propósito de avaliar-se a estabilidade e velocidade de convergência de uma determinada seleção de parâmetros.

4.4 Recursos Computacionais

Os experimentos foram implementados utilizando-se Tensorflow em Python e suas execuções foram feitas na ferramenta Google Colab Pro e numa instância dedicada no Google Cloud Platform. No Colab a GPU utilizada foi a Tesla P100, enquanto que na instância dedicada no GCP utilizou-se a Tesla V100.

5 Resultados e Discussão

Nesta seção são apresentados os resultados de forma agregada dos experimentos propostos na seção Metodologia, assim como uma análise de tais resultados sob a perspectiva da escolha e otimização de hiperparâmetros.

5.1 Escalamento e Combinações de Complexidade

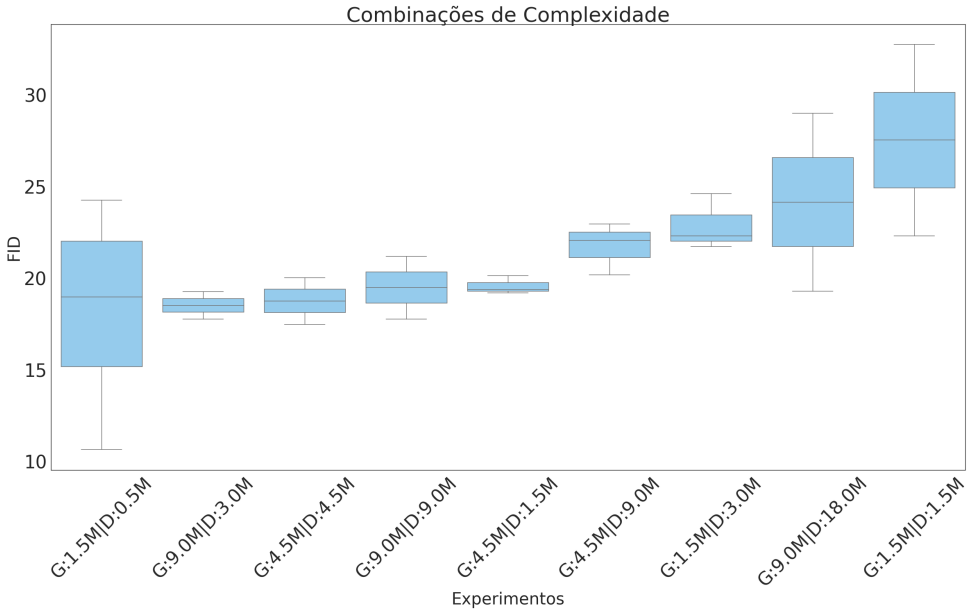


Figura 11 – Escalamento e Combinações de Complexidade no *dataset* MNIST

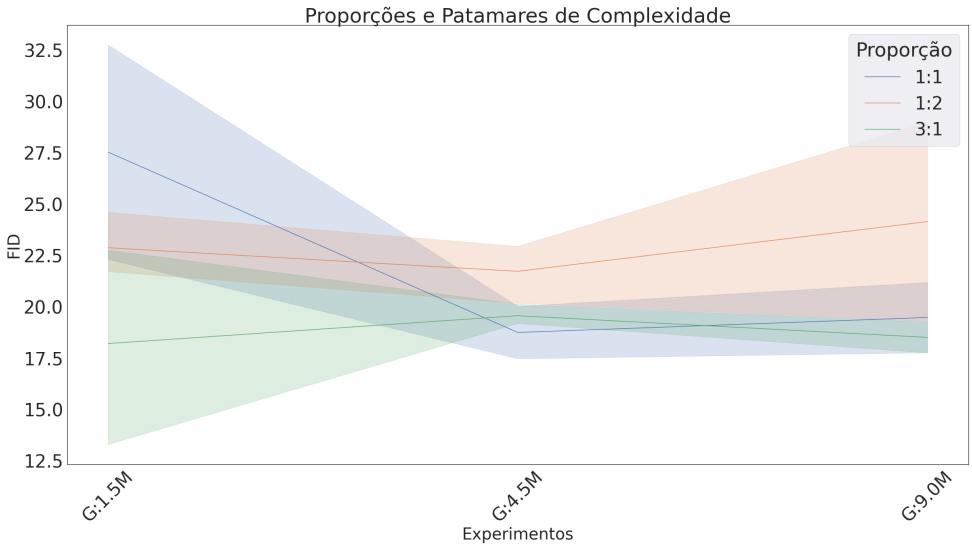


Figura 12 – Diferentes proporções e patamares de complexidade no *dataset* MNIST. As proporções estão expressas como a razão Gerador:Discriminador.

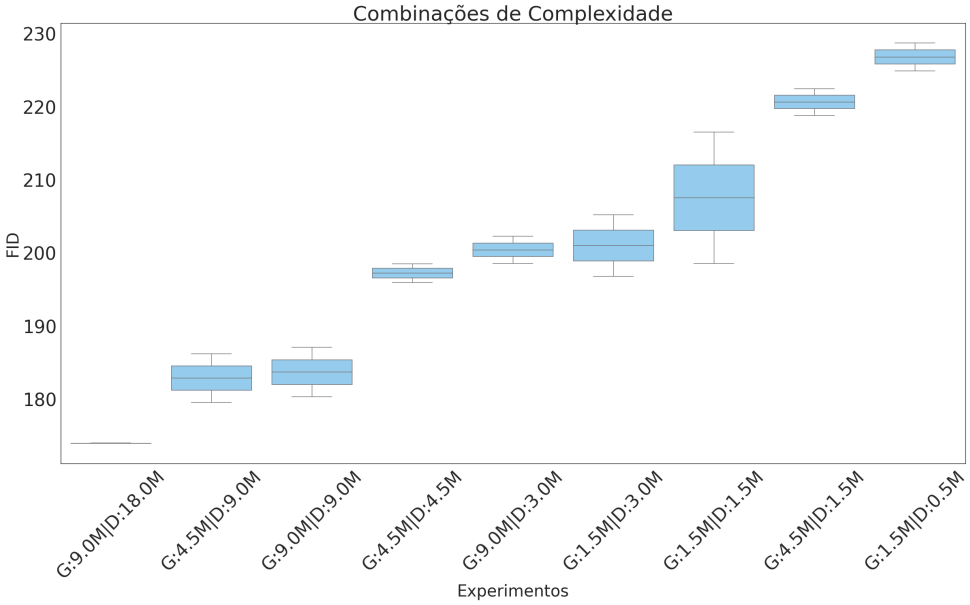


Figura 13 – Escalamento e Combinações de Complexidade no *dataset* CIFAR

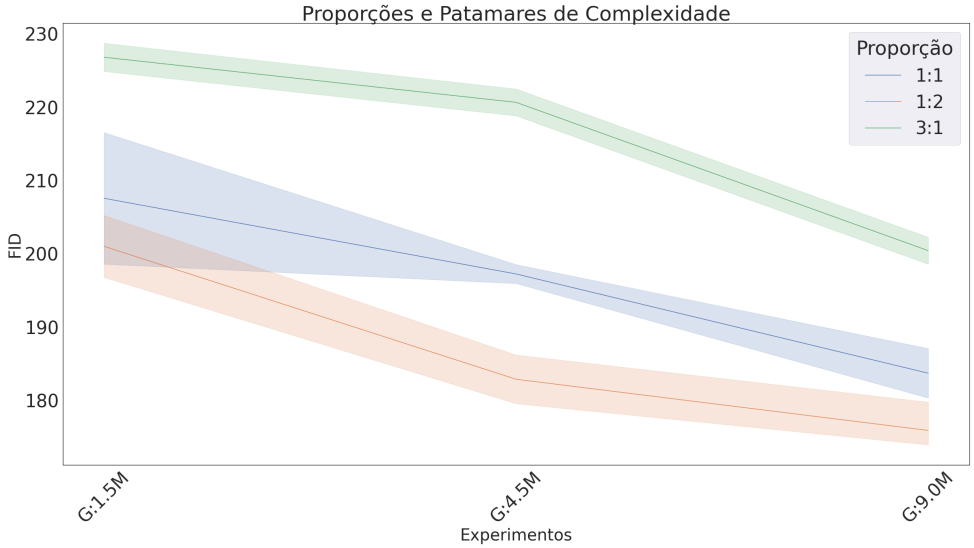


Figura 14 – Diferentes proporções e patamares de complexidade no *dataset* CIFAR. As proporções estão expressas como a razão Gerador:Discriminador.

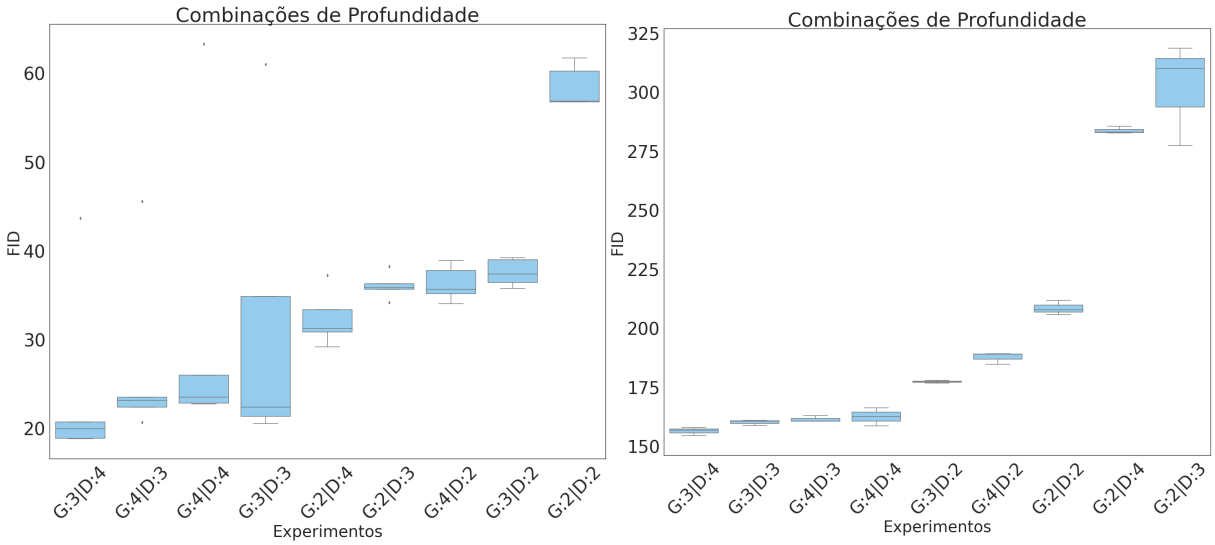
Observaram-se comportamentos distintos frente às variações de complexidade entre os *datasets* CIFAR e MNIST. Enquanto que no CIFAR observou-se uma clara correlação positiva entre performance e a complexidade dentre todas as proporções testadas de complexidade entre o Gerador e Discriminador, no MNIST foi observado um comportamento mais difuso tanto frente à complexidade total e às proporções de complexidade entre Gerador e Discriminador.

Embora ambos tenham apresentado um potencial de otimização superior a 20%, que pode ser observado nas figuras 11 e 13, é possível notar-se uma maior sobreposição entre as performances obtidas no MNIST, figura 12 do que no CIFAR, figura 14. Tal sobreposição pode ser uma consequência da proximidade desses resultados à performance máxima que pode ser obtida com tais modelos. Tal limite, funcionando como um ponto de saturação no qual as performances dos modelos tendem a se aglomerar durante o , não está presente no experimentos feitos no CIFAR, cujos resultados se apresentam a uma distância razoável do estado da arte no momento em que o treinamento é interrompido por conta da restrição de 40 épocas imposta por motivos orçamentários. Esse fenômeno também pode ser a razão pela qual os resultados perante a complexidade tenham se apresentado de forma mais difusa no MNIST do que no CIFAR.

Nota-se também que os resultados obtidos no CIFAR são consonantes com a intuição de que modelos mais complexos tendem a uma melhor performance, assim como, a expectativa da literatura de GANs de que discriminadores mais potentes que geradores tenderiam a resultar num processo de treinamento mais eficiente, uma vez que esses são responsáveis por estimar a razão entre a probabilidade de uma amostra ser falsa em relação à probabilidade dessa ser verdadeira, ou seja, uma estimativa mais precisa de tal razão resultaria em melhores gradientes para o gerador [1].

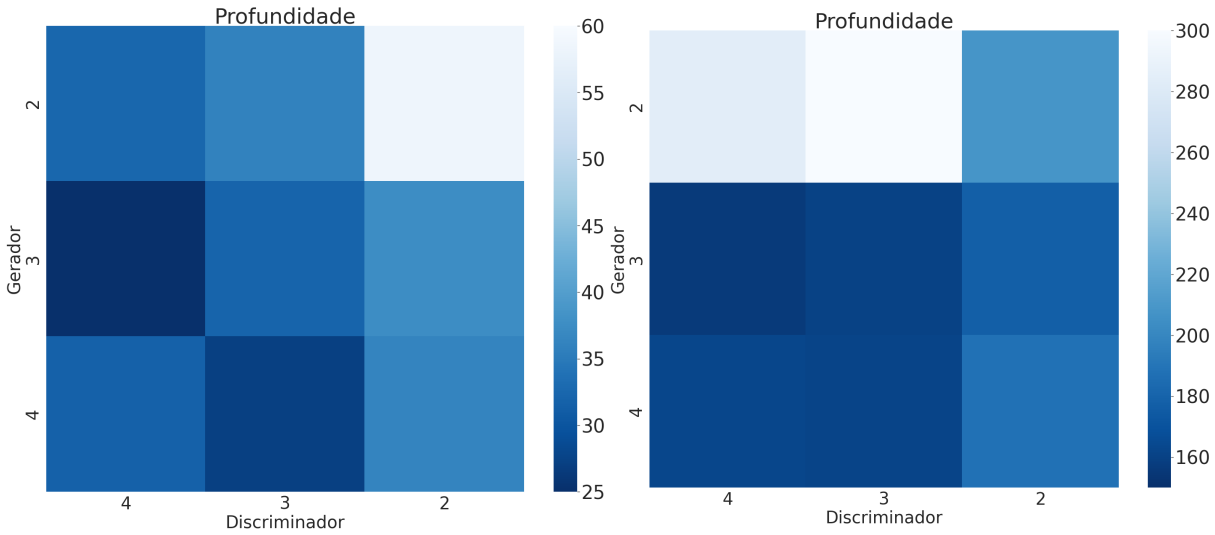
Face a otimização de hiperparâmetros, embora o potencial de otimização obtido seja significativo em ambos *datasets*, o aumento da complexidade dos modelos impacta consideravelmente no tempo de treinamento, sendo, por exemplo, os modelos com geradores de 9 milhões de parâmetros cinco vezes mais lentos para se treinar do que os modelos com gerador de 1,5 milhões de parâmetros. Portanto, embora seja uma das primeiras ações tipicamente tomadas pelos praticantes no processo de ajuste de um modelo, é necessário ter em mente tal compromisso, assim como o fato de outros hiperparâmetros apresentarem potenciais de otimização maiores, como será discutido mais a frente.

5.2 Escalamento e Combinações de Profundidade



(a) MNIST (b) CIFAR

Figura 15 – Escalamento e Combinações de Profundidade



(a) MNIST (b) CIFAR

Figura 16 – Escalamento e Combinações de Profundidade

As diferentes combinações de profundidades entre o gerador e discriminador apresentaram um significativo potencial de otimização, isto é, em relação à combinação ótima, foram obtidas melhorias de performance superiores à 50% para ambos *datasets*, como pode ser observado nas figuras 15a e 15b.

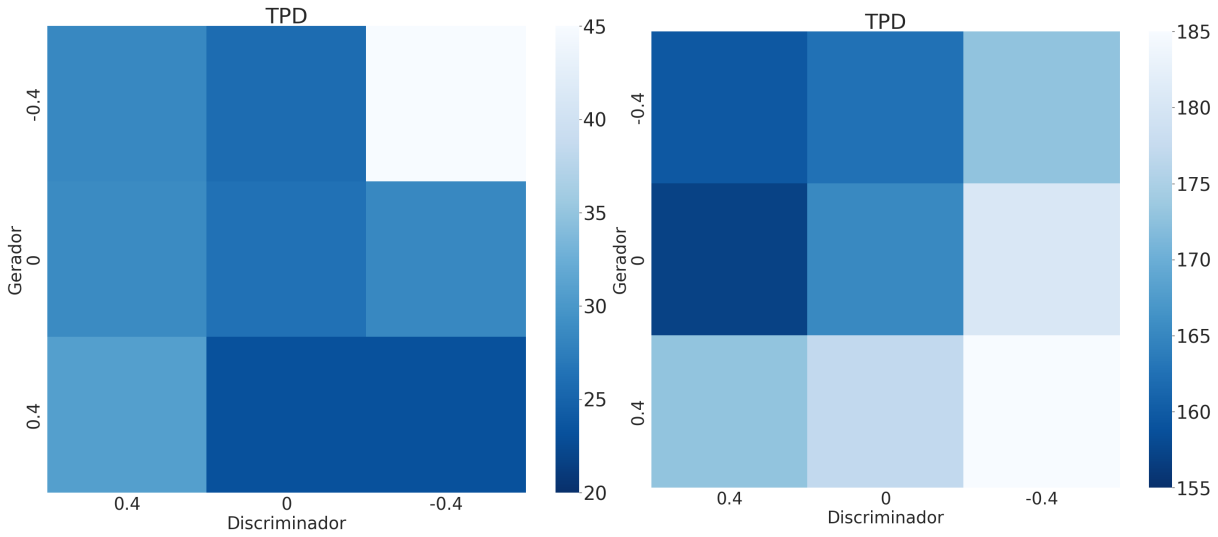
Observou-se também uma maior variabilidade na performance do modelo treinado no *dataset* MNIST acompanhando também de uma maior presença de *outliers*. Embora uma explicação precisa para esse fenômeno não seja conhecida, uma possível razão é o fato das 50

épocas de treinamento serem suficientes para o modelo atingir o seu limite de performance no MNIST mas não no CIFAR, tornando treinamentos que resultem em divergência, ou que sejam agudamente afetados por alguma instabilidade, mais proeminentes no MNIST do que no CIFAR.

Ao comparar-se os mapas de calor referentes às combinações de profundidade, figuras 16a e 16b, é possível notar uma pequena similaridade na região de distribuição das combinações com melhor performance, inclusive sendo a combinação de um gerador com três camadas de profundidade e um discriminador com quatro camadas de profundidade a que apresenta melhor performance para ambos *datasets*.

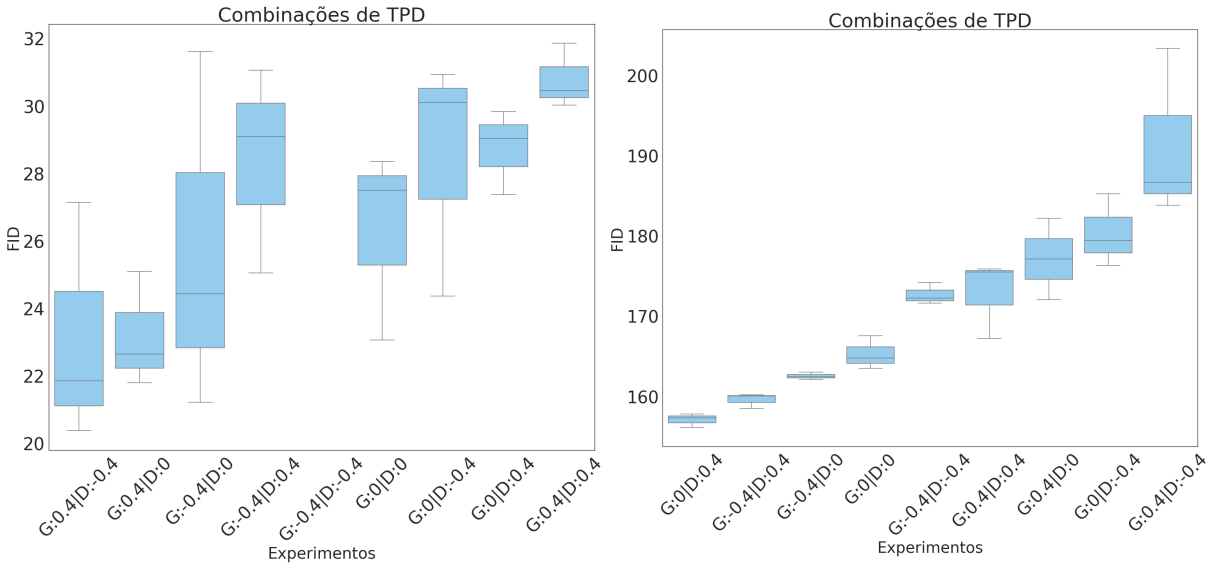
Além disso é possível notar também uma tendência de maior performance em modelos com combinações mais profundas em ambos discriminador e gerador. Tal observação combinada com a aparente consistência na performance das combinações que apresentam razão constante entre si (4/4, 3/3 e 2/2) pode ser sugestivo de que a partir de uma certa profundidade, no casos dos *datasets* apresentados, acima de duas camadas de profundidade, combinações que apresentem uma mesma razão entre a profundidade do gerador e discriminador tendam a apresentar performances médias similares.

5.3 Combinações de TPDs



(a) MNIST (b) CIFAR

Figura 17 – Combinações de TPD



(a) MNIST [1] (b) CIFAR

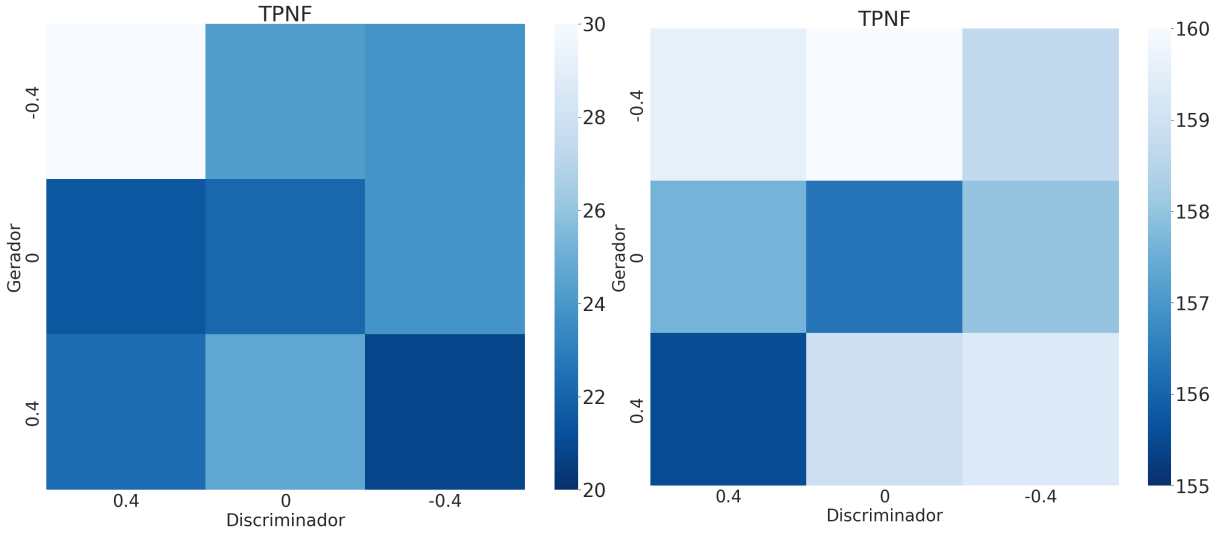
Figura 18 – Combinações de TPD

Assim como no caso das combinações de profundidade, as combinações de TPDs também apresentaram potencial de otimização significativo, atingindo melhorias na performance de 16% e 30% nos *datasets* CIFAR e MNIST, respectivamente, como pode ser constatado nas figuras 18a e 18b.

Analisando-se os mapas de calor, figuras 17a e 17a, nota-se que tanto as combinações com melhor quanto as com pior performance apresentam comportamentos bastante distintos, sugerindo que a otimização da TPD possa ser altamente dependente do *dataset* em que está se efetuando o treinamento. Mesmo assim é interessante notar a regularidade nas combinações de TPD no modelo do CIFAR, no qual geradores com TPDs menores apresentaram consistentemente melhor performance independente da escolha do discriminador, enquanto que discriminadores com TPDs maiores apresentaram consistentemente performance superior independente da escolha do gerador.

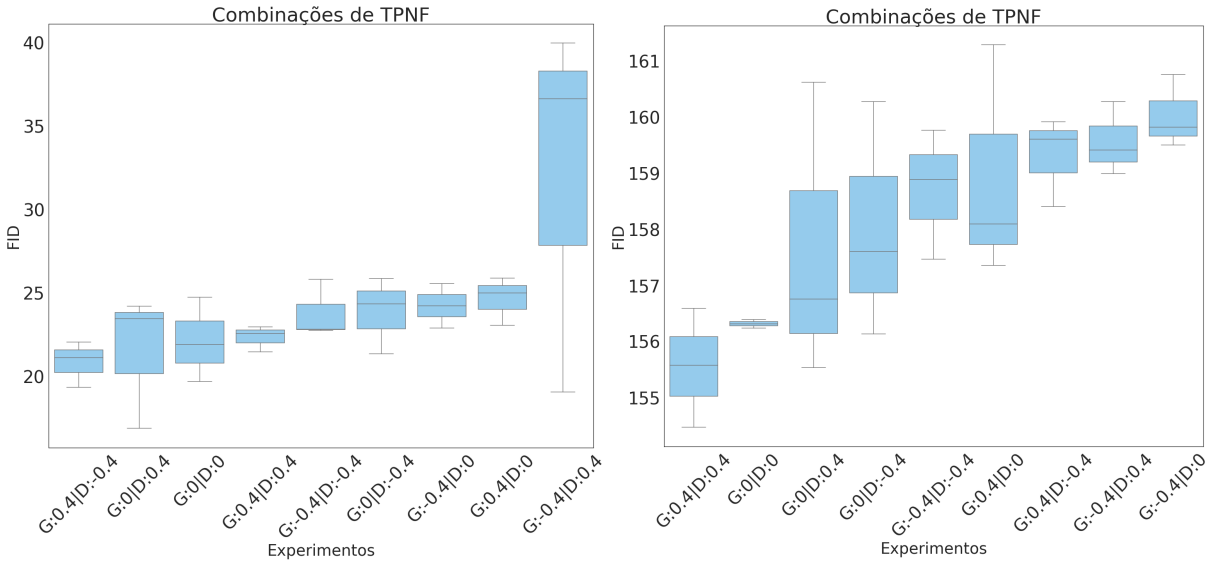
[1] A combinação -0.4/-0.4 apresentou consistentemente problemas de convergência, resultando numa FID média de 370.4 com desvio padrão de 31.1, e, portanto, não foi incluída no gráfico para que não distorcesse a escala.

5.4 Combinações de TPNFs



(a) MNIST (b) CIFAR

Figura 19 – Combinações de TPNF



(a) MNIST (b) CIFAR

Figura 20 – Combinações de TPNF

As combinações de TPNF apresentaram potencial de otimização significativamente inferior às combinações de profundidade e TPD, 2% e 12% nos modelos do CIFAR e MNIST, respectivamente, como pode ser visto nas figuras 20a e 20b. No caso do MNIST desconsiderou-se a combinação -0.4 e 0.4 por essa apresentar disparidade a todas as demais, isto é, em todas as demais combinações estipuladas o potencial de otimização é significativamente inferior em relação à combinação desconsiderada.

Por fim, a comparação dos mapas de calor, figuras 19a e 19b, revela um comportamento aparentemente divergente entre as combinações de TPNF nos *datasets* MNIST e CIFAR. Enquanto no MNIST a performance cresce a medida em que as TPNFs do gerador e discriminador divergem, com o gerador tendendo para uma TPNF positiva e o discriminador para uma negativa, no CIFAR a performance apresentou aumento a medida em que tanto o discriminador quanto o gerador convergiram para uma TPNF positiva. Tal diferença pode ser sugestiva de uma dependência do *dataset* na otimização da TPNF. É cabível notar que ambos modelos apresentaram performances relativamente altas com TPNFs iguais à zero, podendo essa combinação servir como um bom ponto de partida no processo de ajuste de hiperparâmetros arquiteturais de um modelo.

5.5 Amostras Geradas

Até o momento discutiu-se os resultados analisando-se a performance dos modelos objetivamente por meio da FID, no entanto, é interessante avaliar-se subjetivamente as imagens geradas e como a diferença obtida na performance entre os *datasets* MNIST e CIFAR impacta diretamente na capacidade de compreensão do conteúdo das imagens, como pode-se visualizar nas imagens 21 e 22, respectivamente. Isto é, os modelos treinados no MNIST, por esse ser um *dataset* mais simples, obtêm performances superiores em quarenta épocas de treinamento em comparação com o os modelos treinados no CIFAR, o que se reflete diretamente na cognoscibilidade do conteúdo das imagens.

Figura 21 – Amostras geradas com modelos treinados no *dataset* MNIST e que obtiveram performance na FID entre 20 e 30

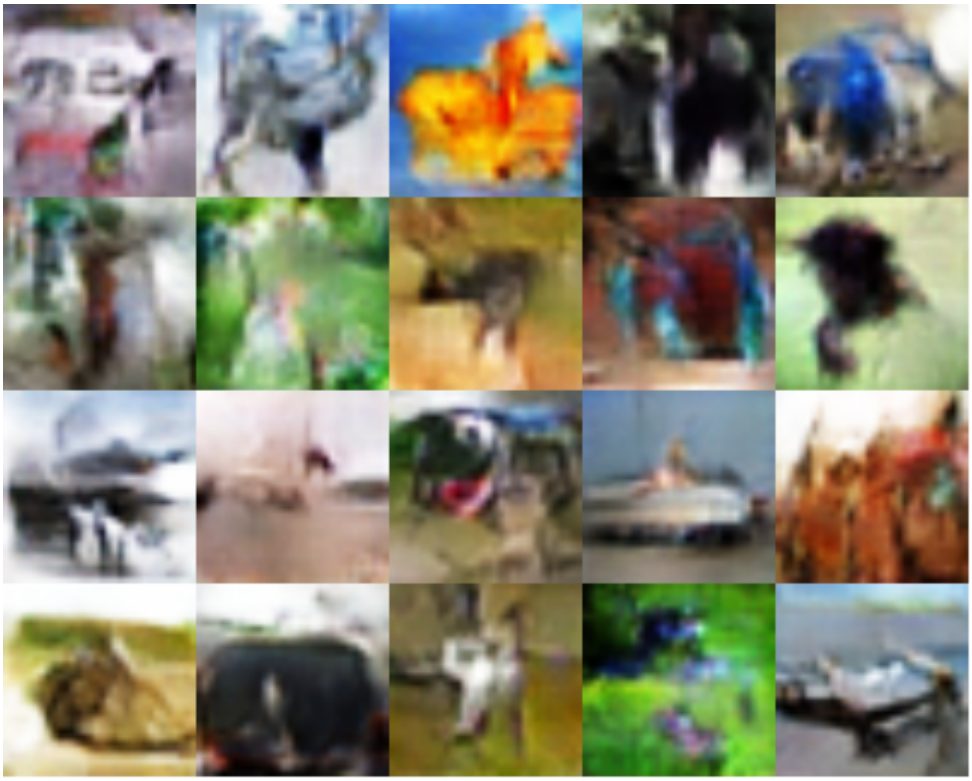


Figura 22 – Amostras geradas com modelos treinados no *dataset* CIFAR e que obtiveram performance na FID entre 150 e 170

6 Conclusão

A crescente popularidade e aplicação de técnicas de *Machine Learning* decorrente da evolução recente obtida na área, especialmente no que tange a modelos profundos e sua capacidade de internalizar e representar eficientemente espaços complexos e de alta dimensionalidade, motivam o estudo de modelos geradores não apenas por suas aplicações diretas mas também pelo aprofundamento da compreensão de técnicas e ferramentas que permitam um aumento na capacidade representativa e de aprendizado de modelos profundos. Após a introdução das GANs, intensificou-se o estudo de modelos geradores em vista dos resultados obtidos por essa subclasse. Embora as GANs tenham levado o estado da arte da geração de imagens para um novo patamar de eficiência e realismo, o treinamento desses modelos ainda se constitui como um desafio tanto para desenvolvedores da indústria quanto acadêmicos que desejem explorar esse campo, requerendo uma miríade de ajustes manuais para obter-se convergência no treinamento e eventualmente atingir-se os resultados expressos na literatura.

Tendo em vista os diversos aspectos avaliados pela literatura no que tange a estabilização do treinamento e aumento da performance de GANs, realizou-se um estudo neste trabalho quanto ao impacto de hiperparâmetros arquiteturais na performance de DCGANs por meio da introdução de um esquema de metaparametrização que possibilitasse a varredura ordenada de hiperparâmetros arquiteturais, assim como, o desacoplamento desses, isto é, permitir que a seleção de um parâmetro não modifique o espaço de possibilidades dos demais parâmetros. O objetivo primário do estudo realizado foi avaliar quais metaparâmetros, isto é, quais características arquiteturais, apresentam maior relevância quanto à otimização da performance e, portanto, devem ser priorizadas no uso do orçamento disponível para o treinamento de um modelo.

Avaliou-se cinco aspectos arquiteturais da DCGAN: complexidade total, proporção de complexidade entre o gerador e o discriminador, TPD, TPNF e profundidade. Os resultados do estudo indicam que os maiores potenciais de otimização encontram-se no ajuste da profundidade e da complexidade, que apresentaram potenciais de otimização de 50% e 20%, respectivamente, no entanto, o ajuste da complexidade vem acompanhado de uma variação no tempo requerido para treinar o modelo a medida em que a complexidade cresce. Como tal aumento no tempo de treinamento é expressivo e impacta em todo o processo de otimização subsequente, recomenda-se que a otimização da complexidade, embora tenha potencial ligeiramente maior que a otimização da TPD, deva ser feita de forma secundária a essa.

Quanto ao ajuste da profundidade, além do grande potencial constatado, observou-

se que a utilização da mesma profundidade no gerador e descriminador assim como evitar profundidades inferiores a 3 camadas constituem um bom ponto de partida para otimização e restrição das possibilidades.

A TPD, embora tenha apresentado um potencial de otimização razoavelmente elevado, apresentou comportamento altamente dependente do *dataset* impedindo qualquer forma de prescrição quanto a como otimizá-la.

E, por sua vez, a TPNF apresentou baixo potencial de otimização e uma relativa consistência na performance quando configurada para próximo de zero, isto é, uma progressão linear no número de filtros por camada, o que pode ser uma boa escolha padrão caso não haja orçamento disponível para otimizá-la.

Além dos ajustes de características arquiteturais exploradas como objetivo central do estudo, também, como parte da metodologia proposta, otimizou-se os hiperparâmetros de treinamento para as combinações de metaparâmetros exploradas. Nesse processo observou-se que a otimização dos parâmetros arquiteturais só é frutífera mediante a otimização dos hiperparâmetros de treinamento como as taxas de decaimento do Momento e, em especial, a taxa de aprendizagem, isto é, sem a seleção correta desses parâmetros a maioria dos treinamentos resulta em divergência. Concluiu-se a partir dessa observação que embora exista potencial disponível para otimização por meio de características arquiteturais, é primordial que se otimizem os hiperparâmetros de treinamento e, sendo o orçamento extremamente limitado, a otimização desses devem preceder quaisquer otimizações arquiteturais.

Observou-se também, além do impacto da otimização dos hiperparâmetros de treinamento, que as diferentes inicializações aleatórias podem, dado um conjunto de hiperparâmetros, provocar uma variabilidade significativa na performance, o que sugere a possibilidade de alocação de parte do orçamento para execução de um protocolo de treinamento que inclua diferentes reinicializações, de forma a minimizar o impacto da alta variabilidade na performance frente a essas.

Referências

1 GOODFELLOW, I. NIPS 2016 tutorial: Generative adversarial networks. Disponível em: <http://arxiv.org/abs/1701.00160>. Citado 5 vezes nas páginas 11, 24, 27, 31 e 71.

2 CAZALA, J. *MNIST Digits*. Original-date: 2015-05-03T15:56:47Z. Disponível em: <https://github.com/cazala/mnist>. Citado 2 vezes nas páginas 11 e 60.

3 KRIZHEVSKY, A. *CIFAR-10 and CIFAR-100 datasets*. Disponível em: <https://www.cs.toronto.edu/~kriz/cifar.html>. Citado 2 vezes nas páginas 11 e 60.

4 KARPATHY, A. Generative models. Disponível em: <https://openai.com/blog/generative-models/>. Citado na página 19.

5 DEEP generative models: Survey. In: . [s.n.]. Disponível em: <https://www.researchgate.net/publication/325026037_Deep_generative_models_Survey>. Citado na página 19.

6 GOODFELLOW, I. J. et al. Generative adversarial networks. Disponível em: <http://arxiv.org/abs/1406.2661>. Citado 6 vezes nas páginas 19, 29, 30, 31, 39 e 40.

7 KARRAS, T. et al. Progressive growing of GANs for improved quality, stability, and variation. Disponível em: <http://arxiv.org/abs/1710.10196>. Citado na página 19.

8 KARRAS, T.; LAINE, S.; AILA, T. A style-based generator architecture for generative adversarial networks. Disponível em: <http://arxiv.org/abs/1812.04948>. Citado na página 19.

9 SPAMPINATO, C. et al. Adversarial framework for unsupervised learning of motion dynamics in videos. Disponível em: <http://arxiv.org/abs/1803.09092>. Citado na página 19.

10 LEDIG, C. et al. Photo-realistic single image super-resolution using a generative adversarial network. Disponível em: <http://arxiv.org/abs/1609.04802>. Citado na página 19.

11 BAEVSKI, A. et al. Unsupervised speech recognition. p. 35. Disponível em: <https://proceedings.neurips.cc/paper/2021/file/ea159dc9788ffac311592613b7f71fbb-Paper.pdf>. Citado na página 19.

12 ANIRUDH, R. et al. An unsupervised approach to solving inverse problems using generative adversarial networks. Disponível em: <http://arxiv.org/abs/1805.07281>. Citado na página 19.

13 KURACH, K. et al. The GAN landscape: Losses, architectures, regularization, and normalization. p. 16. Disponível em: <https://arxiv.org/pdf/1807.04720v1.pdf>. Citado 3 vezes nas páginas 20, 32 e 40.

14 RADFORD, A.; METZ, L.; CHINTALA, S. Unsupervised representation learning with deep convolutional generative adversarial networks. Disponível em: <http://arxiv.org/abs/1511.06434>. Citado 6 vezes nas páginas 20, 32, 35, 36, 43 e 45.

15 IVANOV, S. *Top-10 Research Papers in AI*. Disponível em: <https://towardsdatascience.com/top-10-research-papers-in-ai-1f02cf844e26>. Citado na página 20.

16 OORD, A. v. d. et al. Conditional image generation with PixelCNN decoders. Disponível em: <https://arxiv.org/abs/1606.05328v2>. Citado na página 25.

17 OORD, A. v. d.; KALCHBRENNER, N.; KAVUKCUOGLU, K. Pixel recurrent neural networks. Disponível em: <http://arxiv.org/abs/1601.06759>. Citado na página 25.

18 WIATRAK, M.; ALBRECHT, S. V.; NYSTROM, A. Stabilizing generative adversarial networks: A survey. Version: 1. Disponível em: <http://arxiv.org/abs/1910.00927>. Citado na página 31.

19 GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep Learning*. [S.l.]: MIT Press, 2016. <http://www.deeplearningbook.org>. Citado na página 34.

20 IOFFE, S.; SZEGEDY, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. Disponível em: <http://arxiv.org/abs/1502.03167>. Citado na página 34.

21 SALIMANS, T. et al. Improved techniques for training GANs. Disponível em: <http://arxiv.org/abs/1606.03498>. Citado 4 vezes nas páginas 37, 38, 39 e 41.

22 HEUSEL, M. et al. GANs trained by a two time-scale update rule converge to a local nash equilibrium. In: *Advances in Neural Information Processing Systems*. Curran Associates, Inc. v. 30. Disponível em: <https://papers.nips.cc/paper/2017/hash/8a1d694707eb0fefe65871369074926d-Abstract.html>. Citado na página 38.

23 SZEGEDY, C. et al. Rethinking the inception architecture for computer vision. *CoRR*, abs/1512.00567, 2015. Disponível em: <http://arxiv.org/abs/1512.00567>. Citado na página 38.

24 SAXENA, D.; CAO, J. Generative adversarial networks (GANs): Challenges, solutions, and future directions. Disponível em: <http://arxiv.org/abs/2005.00065>. Citado na página 39.

25 ARJOVSKY, M.; CHINTALA, S.; BOTTOU, L. Wasserstein GAN. Disponível em: <http://arxiv.org/abs/1701.07875>. Citado na página 39.

26 GULRAJANI, I. et al. Improved training of wasserstein GANs. Disponível em: <http://arxiv.org/abs/1704.00028>. Citado 3 vezes nas páginas 39, 43 e 45.

27 MIYATO, T. et al. Spectral normalization for generative adversarial networks. Version: 1. Disponível em: <http://arxiv.org/abs/1802.05957>. Citado 3 vezes nas páginas 39, 43 e 45.

28 MAO, X. et al. Least squares generative adversarial networks. Disponível em: <http://arxiv.org/abs/1611.04076>. Citado 2 vezes nas páginas 43 e 45.

29 LUCIC, M. et al. Are GANs created equal? a large-scale study. Disponível em: <http://arxiv.org/abs/1711.10337>. Citado 3 vezes nas páginas 43, 45 e 61.

30 LECUN, Y.; CORTES, C. MNIST handwritten digit database. 2010. Disponível em: <http://yann.lecun.com/exdb/mnist/>. Citado na página 59.

31 KRIZHEVSKY, A. Learning multiple layers of features from tiny images. p. 60. Disponível em: <https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf>. Citado na página 60.

32 MEYES, R. et al. Ablation studies in artificial neural networks. Version: 2. Disponível em: <http://arxiv.org/abs/1901.08644>. Citado na página 61.