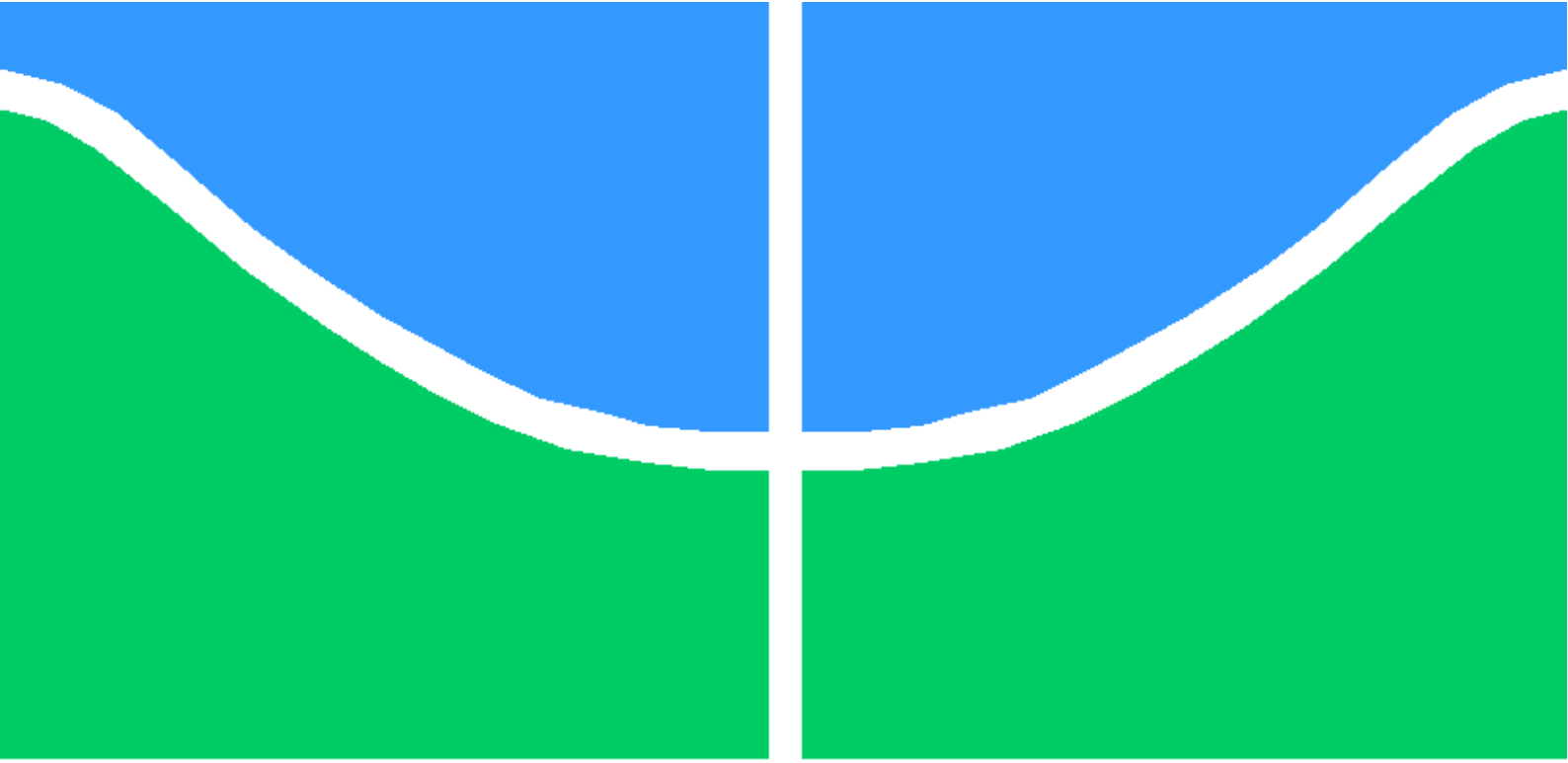


TRABALHO DE CONCLUSÃO DE CURSO

REDES GERADORAS ADVERSÁRIAS EM GERAÇÃO DE IMAGENS

Lucas de Freitas Mariz

Brasília, dezembro de 2018

UNIVERSIDADE DE BRASÍLIA

FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASILIA
Faculdade de Tecnologia

TRABALHO DE CONCLUSÃO DE CURSO

REDES GERADORAS ADVERSÁRIAS EM GERAÇÃO DE IMAGENS

Lucas de Freitas Mariz

Relatório submetido ao Departamento de Engenharia Elétrica como requisito parcial para obtenção do grau de Engenheiro Eletricista

Banca Examinadora

Prof. Alexandre Romariz, ENE/UnB
Orientador

Prof. Alexandre Zaghetto, CIC/UnB
Examinador interno

Prof. Vinícius Silva, ENE/UnB
Examinador interno

Dedicatória

Dedicado ao meu irmão Luiz Augusto.

Lucas de Freitas Mariz

Agradecimentos

A todos que me ajudaram durante estes longos anos de jornada, muito obrigado.

Lucas de Freitas Mariz

RESUMO

Conhecer bem o universo e seus fenômenos físicos é motivação fundamental para a criação de novas tecnologias. Modelos geradores permitem dotar máquinas desta compreensão, por meio de algoritmos que analisem e padronizem parte da imensa quantidade de dados presentes no mundo. Este trabalho busca a concepção de uma rede adversária geradora (GAN) capaz de estudar uma distribuição de probabilidade complexa, por meio da produção de amostras que imitem exemplos da distribuição original. São introduzidos os conceitos necessários para entender o funcionamento destas redes, iniciando com os fundamentos de aprendizado de máquina e conseguinte apresentação da arquitetura de redes adversárias. Aplicando o modelo ao conjunto de imagens CIFAR10, o qual contém 60 mil imagens de 10 diferentes classes, foi possível gerar imagens que se assemelhem às originais, evidenciando o poder representacional do modelo gerador estudado.

Palavras-chave: aprendizado de máquina, redes neurais, modelagem geradora, redes geradoras adversárias, DCGAN, Keras.

ABSTRACT

Understand the universe and its physical phenomena is a fundamental motivation for the creation of new technologies. Generative models endow machines with this ability, through algorithms that analyze and standardize part of the immense amount of data present in the world. This project aims at designing a generative adversarial network (GAN) capable of studying a complex probability distribution by producing samples that mimic examples of the original distribution. The necessary concepts to understand the operation of these networks are first introduced, starting with the fundamentals of machine learning and consequently presentation of the architecture of adversarial networks. By applying the model to the dataset CIFAR10, which contains 60 thousand images of 10 different classes, it was possible to generate images that strongly resemble the originals, evidencing the representational power of the generator model studied.

Keywords: machine learning, neural networks, generative modeling generative adversarial networks, GAN, DCGAN, Keras.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SÍMBOLOS

Notação

$\mathbb{X}$	Conjunto
x	Vetor de dados
$\hat{x}$	Vetor de predição
θ	Parâmetros ou pesos da rede
$f_\theta(.)$	Função parametrizada por θ
p	Distribuição de probabilidade
$x \sim p$	Vetor de dados x amostrado da distribuição p
$p(;\theta)$	Distribuição de probabilidade parametrizada por θ
$p(.,.)$	Distribuição de probabilidade conjunta
$p(.\|.)$	Distribuição de probabilidade condicional
$\mathbb{E}_z[X]$	Valor esperado de X com respeito a z

Siglas

IA	Inteligência Artificial
ANN	Rede Neural Artificial (*Artificial Neural Network)*
MLP	*Perceptron* Multicamadas (*Multilayer Perceptron)*
CNN	Rede Neural Convolucional (*Convolutional Neural Network)*
GAN	Rede Geradora Adversária (*Generative Adversarial Network)*
DCGAN	Rede Convolucional Geradora Adversária Profunda (*Deep Convolutional Generative Adversarial Networks)*
RGB	Vermelho Verde Azul (*Red Green Blue)*

Capítulo 1

Introdução

O sonho da máquina humana permeia o imaginário social há milhares de anos. Já na mitologia grega, Talos era um gigante *automaton*, máquina automática que segue uma sequência predeterminada de operações, feito de bronze e guardador da Ilha de Creta. Mais de dois mil anos depois, Asimov escrevia as 3 leis da robótica, princípios idealizados para controlar o comportamento dos robôs em seu universo de ficção científica. Em 2015, Elon Musk, Stephen Hawking e dúzias de especialistas na área de inteligência artificial (IA) assinaram a "Carta Aberta sobre Inteligência Artificial", destacando os efeitos positivos e negativos desta tecnologia na sociedade moderna, em busca de um desenvolvimento benéfico e realçando preocupações que surgem com o avanço dessa ciência.

Neste contexto, o aprendizado de máquina desponta como ramo da IA com a visão de que programas de computador podem aprender a se desenvolver e evoluir automaticamente se obtiverem acesso a dados de treinamento. A aprendizagem profunda, de maneira mais específica, faz uso de estruturas que se baseiam nas sinapses do cérebro humano - as redes neurais - para processar os dados através de diversas camadas, a fim de encontrar parâmetros que possam representar a informação de entrada da maneira mais eficiente possível.

Este projeto faz uso de redes neurais em um modelo gerador: redes geradoras tratam de criação. É mais fácil identificar pinturas clássicas do que pintar uma, mas a busca por formas de reproduzir o mundo ao redor nos leva também a melhor entendê-lo. Dessa forma, o desenvolvimento de tal modelo gerador permite analisar padrões que compõem um conjunto de dados e aprender características sobre sua origem, explícita ou implicitamente.

Criar um modelo gerador que trabalhe com imagens é um meio ideal de provar as capacidades dessa rede. Da mesma forma que pontos retirados de uma curva normal representam uma coleção de amostras, imagens podem ser interpretadas como amostras de uma distribuição de probabilidade de enorme complexidade e dimensionalidade, que é a nossa realidade - grandes demais para serem conhecidas. É possivel, no entanto, usar modelos geradores em busca de estimar estas distribuições e produzir amostras similares.

Usando a arquitetura de redes geradoras adversárias (GANs - *generative adversarial networks*), deseja-se imitar amostras do conjunto de imagens CIFAR10, o qual apresenta 60 mil fotos de baixa

resolução divididas em 10 classes distintas: cachorros, aviões, aves, etc. GANs fazem uso de redes neurais profundas para estabelecer o modelo gerador, auxiliado por um modelo secundário também composto por redes profundas, o discriminador, que fornece *feedback* ao primeiro em busca de avaliar a qualidade das amostras geradas.

Uma vez que são utilizadas fotografias como amostras, usa-se a arquitetura específica de redes convolucionais geradoras adversárias profundas (DCGANs - *Deep Convolutional Generative Adversarial Networks*), que são GANs com determinadas diretrizes que as tornam especializadas no processamento de imagens.

Esta introdução representa o primeiro capítulo do trabalho. No segundo, são explicados os conceitos básicos de aprendizado de máquinas necessários para o desenvolvimento de uma GAN, ao passo que as estruturas da rede geradora propriamente dita são esclarecidas no terceiro capítulo. O quarto capítulo diz respeito à aplicação da rede no conjunto de imagens CIFAR10, com os resultados do experimento sendo apresentados no quinto capítulo. O sexto e último capítulo apresenta as conclusões do projeto.

Capítulo 2

Aprendizado de Máquina

Sendo um campo de estudo da inteligência artificial, o aprendizado de máquina representa uma forma de estatística aplicada à construção de algoritmos que permitam a uma máquina aprender, isto é, melhorar seu desempenho em determinada tarefa por meio da experiência [1]. O processo de aprender sem ser explicitamente programado possibilita que o programa se desenvolva automaticamente, por meio da observação de um conjunto de dados e elaboração de padrões que motivem a máquina a tomar decisões no futuro com base nos dados previamente fornecidos, após o devido processo de treino [2].

Quando o conjunto de dados provido na entrada é acompanhado de seus respectivos rótulos de saída, tem-se um algoritmo supervisionado, o qual produz uma função de aproximação para prever valores futuros. Ao comparar os resultados obtidos com os valores corretos, o programa pode modificar seus parâmetros, por meio de um processo de treino, para melhor representar os dados escolhidos . Busca-se assim aprender uma função $f : \mathrm{X} \rightarrow \mathrm{Y}$ dado um conjunto rotulado de treinamento $\{< x_i, y_i >\}$ de entradas x_i e saídas $y_\mathrm{i} = f(x)$ [2].

Em contrapartida, algoritmos não supervisionados abordam problemas cuja estrutura dos dados de entrada não é conhecida. Não há rótulos, de forma que cabe à máquina identificar os padrões que melhor representem relações entre as variáveis fornecidas, identificando arquétipos ocultos nestas estruturas.

Outras categorias estão presentes ao tratar de aprendizado de máquina, como aprendizado semi-supervisionado e por reforço, porém não são relevantes ao desenvolvimento deste projeto.

2.1 Redes Neurais Artificiais

Redes Neurais Artificiais (ANN) são algoritmos capazes de usar o aprendizado de máquina em uma estrutura inspirada nos sistemas nervosos biológicos para a aproximação de funções. Tais redes são constituídas por camadas de elementos interconectados - os nêuronios - através de parâmetros ajustáveis, os quais fazem paralelo às sinapses cerebrais [3].

De acordo com Haykin [4], uma rede neural se assemelha ao cérebro em dois aspectos:

1. Conhecimento é adquirido pela rede a partir de seu ambiente através de um processo de aprendizagem.

2. Forças de conexão entre neurônios, chamadas de pesos sinápticos, são usadas para armazenar este conhecimento.

ANNs podem, teoricamente, aproximar qualquer função em um domínio compacto se fizerem uso de pelo menos uma camada invisível, atuando assim como aproximador universal [5]. Tal propriedade confere à rede um imenso poder representacional, uma vez que, quanto mais profundo é o modelo, maior é a sua capacidade de aproximar funções complexas. Com a adição de camadas ocultas ao sistema, cada nível transforma seus dados de entrada em uma representação mais abstrata do que o nível anterior, aprendendo assim as características ótimas [6]. Essas redes profundas, ou *deep networks*, são o estado da arte em diversos problemas de aprendizado de máquina e servem como estímulo para uma gama de diferentes arquiteturas.

2.1.1 Perceptron

Em sua forma mais simples, uma rede neural age como um único neurônio artificial, por meio de uma operação de classificação binária dada pelo *perceptron*. Atuando como um classificador linear, este neurônio mapeia um vetor de entrada $x = [x_1, x_2, ..., x_m]$ em uma função binária y, submetendo inicialmente o vetor x a uma soma ponderada com os pesos de sinapse $\theta = [\theta_1, \theta_2, ..., \theta_m]$, acrescida de um valor de *bias* b que apenas aumenta seu grau de liberdade, fornecendo a função soma s [7].

$$s = \sum_i x_i \theta_i + b_i \tag{2.1}$$

O resultado desta operação é submetido a uma função de ativação - a função degrau - mapeando s em uma saída binária conforme equação 2.2. O *perceptron* atua, dessa forma, como um classificador linear, categorizando os dados de entrada em duas classes através de um hiperplano delimitado por $s = 0$. Caso os pesos θ associados aos respectivos valores de entrada sejam grandes o suficiente para alcançar o valor de ativação da função degrau, a sinapse é disparada e $y = 1$.

$$y = \begin{cases} 0, & para \quad s < 0 \\ 1, & para \quad s \geq 0 \end{cases} \tag{2.2}$$

Quando dados não podem ser linearmente separados, é necessário o uso de modelos mais poderosos para realizar tais operações. Nesta situação é possível combinar *perceptrons* em uma estrutura em camadas, cada uma contendo uma quantidade diferente de neurônios. Forma-se assim uma rede neural denominada *perceptrons* multicamadas (MLP), usada no contexto deste projeto como um dos componentes que constituem uma rede geradora adversária [4]. Em um MLP, o vetor x é inserido na camada inicial cujos valores de saída servem como entradas da camada seguinte, e assim por diante, em uma estrutura totalmente conectada, na qual todos os neurônios de uma camada são ligados aos neurônios da seguinte, conforme Figura 2.2.

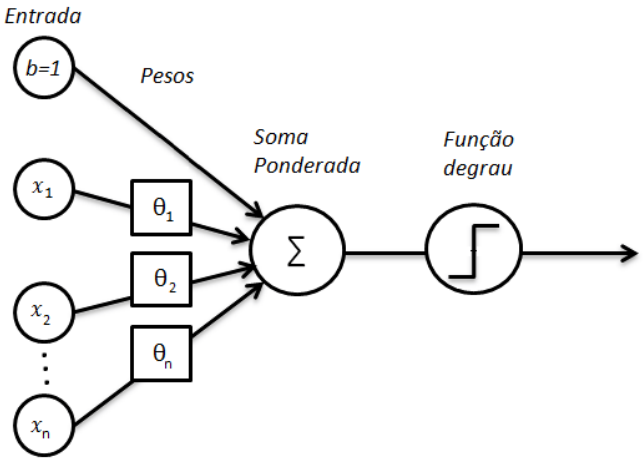


Figura 2.1: Arquitetura de um *perceptron*. Adaptado de: Ahmet Taspinar

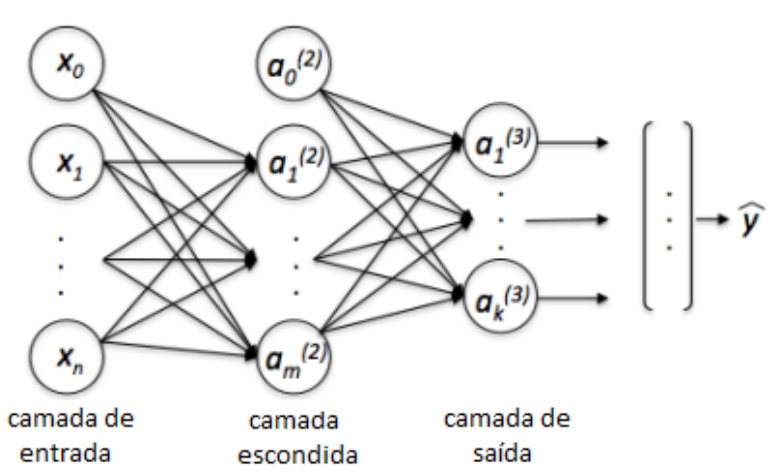


Figura 2.2: Exemplo de arquitetura para *perceptrons* multicamadas com uma camada escondida. Fonte: Erin LeDell.

2.1.2 Funções de ativação

As funções de ativação agem na rede neural definindo o comportamento de saída de cada neurônio dado um vetor x, limitando estes valores de entrada em um intervalo definido. A função de ativação utilizada no *perceptron* é a função degrau, que fornece uma saída binária e permite apenas uma divisão linear de pontos. Em situações mais complexas e com várias classes possíveis de saída, no entanto, faz-se necessária uma função que introduza não-linearidades ao sistema, permitindo uma maior capacidade representativa do modelo [4].

As funções logística (σ), tangente hiperbólica ($tanH$), ativação linear retificada ($ReLU$) e ativação linear retificada com vazamento ($Leaky\ ReLU$) são algumas das principais escolhas em se tratando de funções de ativação.

No escopo deste projeto, tais funções desempenham papéis específicos nas arquiteturas de redes

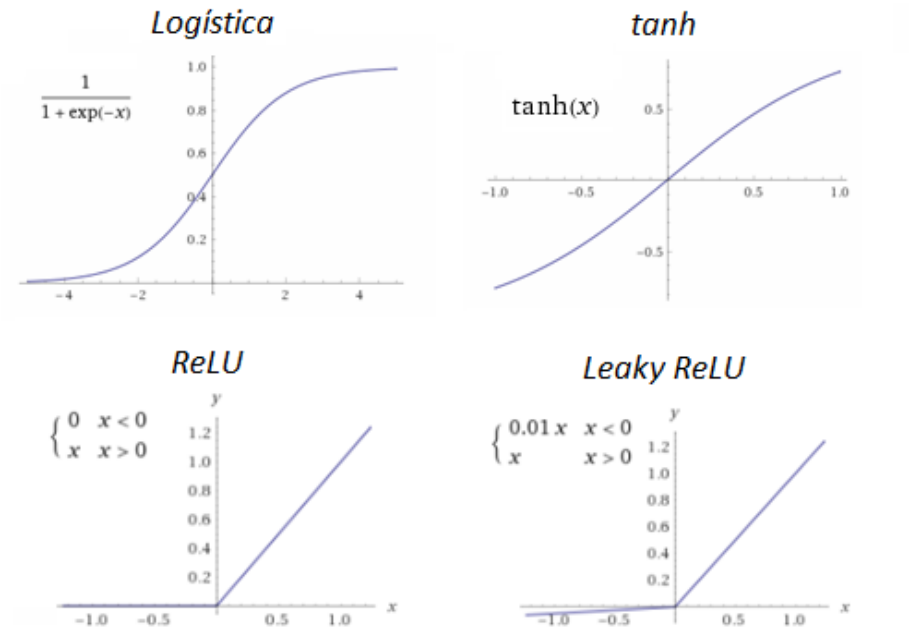


Figura 2.3: Funções de ativação.

neurais utilizadas. A função logística é usada na última camada da rede, uma vez que comprime valores reais no intervalo [0, 1], o que pode ser representado como a probababilidade de escolha de determinada classe, sendo assim fundamental em questões de classificação. Já a função *tanh* força os valores no intervalo [-1, 1] de forma que seu centro em torno de 0 facilita o treinamento de redes em determinados casos, devido ao gradiente mais acentuado [8].

A função $ReLU$ apresenta a grande vantagem de acelerar o aprendizado por apresentar gradientes facilmente calculáveis (0 ou 1, dependendo do intervalo considerado). Em se tratando do processamento de imagens, tal propriedade é relevante devido ao intenso uso computacional. *Leaky* $ReLU$ desponta como uma variedade de tal função, permitindo a passagem de pequenos valores negativos dada por uma constante muito pequena, geralmente igual a 0.01 [9].

2.1.3 Treinamento por *Backpropagation*

Algoritmos de aprendizado de máquina supervisionados, sejam eles de regressão logística, Naive-Bayes, árvore de decisão ou redes neurais, agem com os mesmos dois princípios básicos: treino e predição [10]. No primeiro, os parâmetros θ do modelo são aprendidos a partir dos dados de treinamento x e y. Usando estes parâmetros é possível fazer predições durante a segunda etapa, emitindo resultados $\hat{y}$ que devem se aproximar da saída original.

Em redes neurais, a etapa de treino é chamada de *backpropagation*, ou propagação retroativa de erro. Nesta etapa tenta minimizar-se este erro, uma função de perda H parametrizada por θ que mede a diferença entre y e $\hat{y}$, por meio de um método iterativo denominado descida de gradiente.

Ao iniciar-se o processo de predição de uma rede com seus parâmetros distribuídos de forma

aleatória, o vetor de entrada x levará a uma saída $\hat{y}$ que difere do vetor y desejado, indicando assim um valor elevado para a função de perda, que pode assumir diversas formas, dependendo do tipo de rede trabalhada.

Redes supervisionadas de classificação usualmente têm como função de perda a função de entropia-cruzada [11], dada por

$$H(y, \hat{y}) = -\sum_i y_i \log(\hat{y}_i) \tag{2.3}$$

em que y e $\hat{y}$ indicam as distribuições "real"e "estimada", respectivamente. Mede-se assim a distância entre esses dois vetores discretos de probabilidades, estimando o quanto a distribuição estimada se aproxima da real em um universo com i classes distintas.

Como a função de perda depende dos parâmetros da rede, o método do gradiente pode ser aplicado para minimizar H de forma iterativa, em que, a cada passo, θ é atualizado na direção negativa do gradiente, correspondente à direção de máximo declive:

$$\nabla H = (\frac{\partial H}{\partial \theta_1}, ..., \frac{\partial H}{\partial \theta_i}) \tag{2.4}$$

$$\theta_i := \theta_i - \alpha \frac{\partial H}{\partial \theta_i} \tag{2.5}$$

O termo α recebe o nome de taxa de aprendizado e define o tamanho de cada passo tomado no processo. Para uma rede neural profunda, os pesos das camadas ocultas devem ser atualizados de acordo com os valores dos pesos provenientes de seu neurônio mais à direita, o que pode ser visto ao tomar-se a regra da cadeia na equação 2.4, de modo a propagar o erro inicial desde a saída até os nós iniciais da rede.

Uma vez que os parâmetros θ tenham sido atualizados, entra-se novamente na etapa de predição em busca de satisfazer $\hat{y} = y$.

2.2 Redes Neurais Convolucionais

Um tipo específico de rede profunda recebe o nome de rede neural convolucional (CNN), desenvolvida especialmente para reconhecer formas bidimensionais com um elevado grau de invariância à translação, redimensionamento, inclinação e outras formas de distorção [12]. CNNs têm estruturas muito semelhantes às ANNs ordinárias, compostas por neurônios ponderados por parâmetros modificáveis que recebem um vetor de entrada e realizam operações não-lineares em busca de classificá-lo corretamente. No entanto, CNNs fazem uso da convolução, operação matemática entre duas funções - no âmbito do processamento de imagens, estas equivalem a uma imagem e a um filtro - que altera a intensidade de um pixel para refletir as intensidades dos pixels que o circundam.

Várias arquiteturas de redes convolucionais foram propostas ao longo dos anos, porém neste projeto faz-se uso da *All Convolutional Network* - Rede Totalmente Convolucional [13], cujo es-

queleto consiste em repetidas camadas de convolução sobrepostas, seguidas por algumas camadas totalmente conectadas.

2.2.1 Camada de Convolução

Imagens coloridas podem ser decompostas em três canais de cores: vermelho, verde e azul. Interpretando tal imagem como um vetor tridimensional de altura (A) X largura (L) X canal de cor (C_{in}), pode-se entender cada elemento deste vetor por um subpixel cujo valor equivale à sua intensidade para aquele canal, que varia entre 0 e 255.

A principal proposta da convolução é a de extrair características da imagem de entrada, por meio de pequenas matrizes (usualmente, não maiores que 5x5 [13]) denominadas filtros ou *kernels*, os quais "varrem"toda a imagem em um processo de convolução, uma soma ponderada que expressa como as características da figura são modificadas pelo filtro. Para isso, cada pixel da imagem original é pareado com o filtro, incluindo também os pixels de fronteira, como vê-se na Figura 2.4. Cada elemento do *kernel* é então multiplicado pelo par correspondente, somando-se os produtos para gerar o novo valor de saída.

Considerando uma imagem 3D, a convolução entre o pixels p e o filtro f pode ser vista na equação 2.6, a qual se assemelha muito à fórmula 2.1 para o cálculo dos parâmetros θ de uma rede neural. Tal propriedade não é por acaso, uma vez que o filtro da CNN é composto por parâmetros de sinapse treináveis da mesma forma que para um MLP, porém agora em uma disposição tridimensional, organizada em uma estrutura de pesos compartilhados e não mais totalmente conectados, já que os filtros são replicados ao longo de toda a imagem de entrada.

$$conv(p * f)[i, j] = \sum_{k_1=1}^{K} \sum_{k_2=1}^{K} \sum_{c=1}^{C} p[i - k_1, j - k_2, c] f[c, k_1, k_2] \tag{2.6}$$

As dimensões da imagem de saída após uma camada de convolução dependem de alguns parâmetros específicos:

- **número de filtros utilizados**: para um caso com n filtros, a imagem C_{out} apresenta n canais de cores.

- ***zero-padding***: representa a adição de zeros (pixels nulos) às bordas da imagem de entrada, normalmente para permitir que C_{out} tenha as mesmas dimensões espacias que C_{in}.

- ***stride***: controla quantos pixels são pulados a cada produto escalar no processo de convolução, regulando também as dimensões espaciais de C_{out}. Para stride = 2, por exemplo, são pulados 2 pixels na equação 2.6.

CNNs clássicas fazem uso de camadas de *pooling* intercaladas com as camadas de convolução descritas, uma técnica que busca redimensionar C_{out1} para uma imagem C_{out2} com resolução mais baixa, buscando achar uma representação ideal com uma quantidade menor de parâmetros. Na Rede Totalmente Convolucional, essa abordagem é substituída pelo simples uso de *strides* > 1.

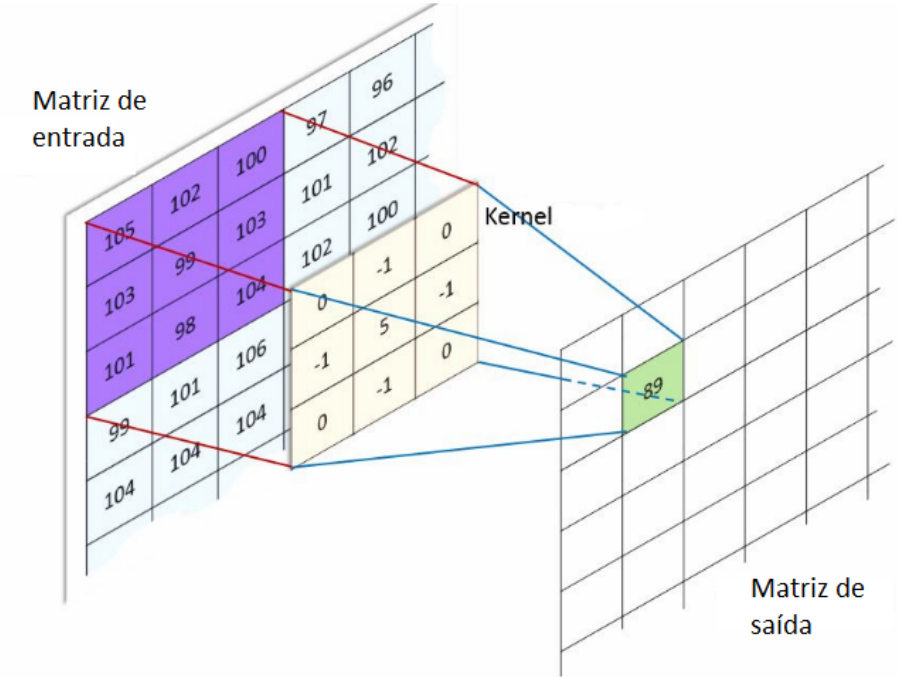


Figura 2.4: Exemplo de convolução para pixel (de intensidade 99) em imagem com 1 canal de cor. Este pixel e os demais ao seu arredor são multiplicados pelos elementos correspondetes do filtro e então somados, gerando um valor de saída. O processo se repete para todos os pixels da imagem. Adaptado de: Machine Learning Guru.

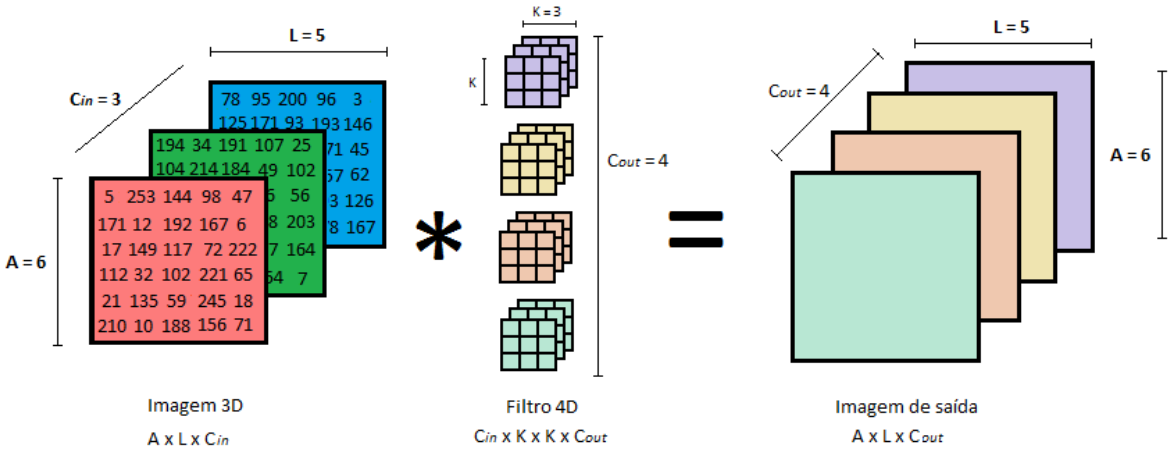


Figura 2.5: Exemplo de camada de convolução. Uma imagem de entrada 3D, com 6 pixels de altura, 5 de largura e 3 canais de cores, sofre convolução com um filtro 4D, de altura e largura K = 3, gerando uma imagem de saída com o mesmo número de canais C_{out} que a quantidade de filtros utilizada, 4. Este processo se repete várias vezes ao longo da rede convolucional.

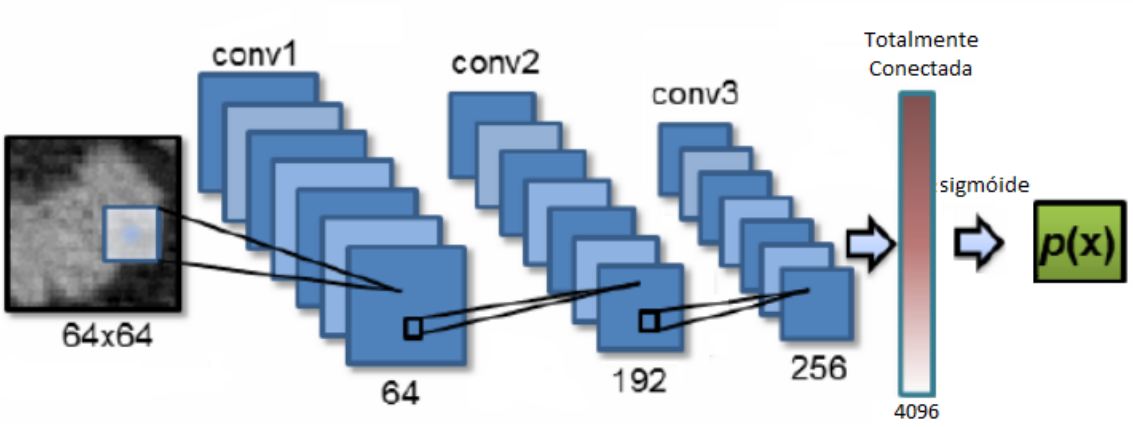


Figura 2.6: Esqueleto completo de uma CNN da forma Rede Totalmente Convolucional. Adaptado de: Le Lu.

2.3 Observações

Os conceitos explicados neste capítulo revelam a base do conhecimento de aprendizado de máquina necessária para a criação de uma rede geradora adversária, objetivo final do projeto. Não foram tratados dos aspectos técnicos e pormenores das diversas arquiteturas possíveis, como as várias topologias de redes, inicialização correta de pesos e termos de regularização que diferenciam cada escolha.

Tal abordagem foge ao escopo do trabalho em questão, já que a criação de um modelo gerador não depende, de fato, das redes neurais, escolhidas somente como ferramenta para este fim, devido à possibilidade de treinar os parâmetros usando apenas o conceituado método de *backpropagation* [14] e ao seu elevado poder representacional, capaz de aproximar funções com grande destreza.

Capítulo 3

Redes Geradoras Adversárias

3.1 Modelagem Geradora

A aprendizagem profunda ou *deep learning* constitui área de pesquisa da inteligência artificial que faz uso de diversas camadas não-lineares para descobrir padrões que modelem corretamente uma distribuição de probabilidade, usando de diferentes níveis de abstração como visto no capítulo anterior.

Os principais estudos na aprendizagem profunda têm seu foco em modelos discriminadores, os quais buscam mapear dados de entrada multidimensionais x em uma classe de saída y por meio da probabilidade condicional p(y|x;θ). Um *perceptron*, por exemplo, modela $p(y|x;\theta)$ como $\hat{y}_\theta$(x) = g(θ^{T}x) em que g é a função degrau. As ANNs e CNNs vistas até aqui também se encaixam nessa categoria, de forma a encontrar apenas uma fronteira que separe corretamente os dados de entrada em classes [15].

Considere um problema de classificação no qual deseja-se diferenciar entre gatos ($y = 0$) e cães ($y = 1$), com base em alguma característica dos animais. Dado os pares de treinamento (x, y), o algoritmo tenta encontrar uma linha ou curva (a fronteira de decisão) que separe os cães e gatos, como na Figura 3.1. Em seguida, averigua em qual lado da fronteira o próximo exemplo está localizado e o classifica de acordo. Tais algoritmos tentam assim responder à pergunta "dado estas entradas qual a probabilidade de se ter esta classe como a mais provável?".

Modelos geradores, por outro lado, tentam aprender a probabilidade conjunta $p(x, y)$, a qual pode ser usada em modelos de classificação através da conversão para $p(x|y)$. Usando do mesmo exemplo em que y denota cães (1) e gatos (0), $p(x|y = 1)$ modela a distribuição das características dos cães $p(x|y = 0)$ modela a dos gatos. O modelo gerador pode, em seguida, usar estes valores para encontrar $p(y|x)$ indiretamente pelo teorema de Bayes (equação 3.1), servindo como ferramenta para problemas de classificação.

$$p(y|x) = \frac{p(x|y)p(y)}{p(x)} \tag{3.1}$$

O verdadeiro potencial de modelos geradores, no entanto, está em sua capacidade de entender estruturas dos dados de entrada e criar novos pares prováveis (x, y) em um processo de geração, à medida em que foca nas classes em busca de assimilar suas características, e não apenas a fronteira que as separa. Este processo pode ocorrer de duas maneiras, ambos seguindo uma estimativa por máxima verossimilhança: modelos geradores explícitos, os quais definem uma distribuição p_{model} que se aproxima da distribuição de amostras p_{data} e modelos geradores implícitos, que aprendem uma função que faça amostragens de p_{model} sem defini-la diretamente. Redes geradoras adversárias, o foco deste trabalho, se encaixam no segundo grupo.

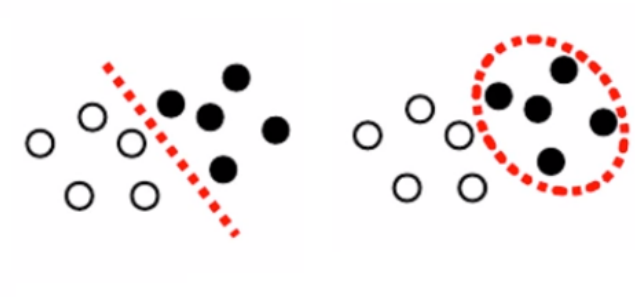


Figura 3.1: Modelos discriminadores e modelos geradores. Fonte: Nigel Goddard.

3.2 Modelo para Redes Geradoras Adversárias

Para estimar modelos geradores, foi criada uma arquitetura que treina duas redes simultaneamente, em processo adversário entre um modelo discriminador D e um gerador G chamado de redes geradoras adversárias (GAN). Nesta estrutura, o objetivo do gerador é criar amostras que sejam idealmente indistinguíveis de amostras reais, com o nível de semelhança entre elas sendo medido pelo discriminador. No caso específico em que os modelos são representados por redes neurais de *perceptrons* multicamadas (MLP), ambos podem ser treinados com o uso do algoritmo *backpropagation* [14].

Para tal, define-se uma distribuição conhecida p_{z} da qual são amostradas variáveis latentes de ruído z, representado por $z \sim p_{\text{z}}$. Usando uma rede neural $G(z,\ \theta_g)$ parametrizada por θ_{g} que tem como entrada o vetor z, é possível treiná-la para criar amostras de sua distribuição geradora implícita p_{g}. A rede discriminadora $D(x,\ \theta_d)$ tem θ_{d} como parâmetros e recebe como entrada imagens x_{i} amostradas de uma distribuição desconhecida $p_{\text{data}}(x)$ ou geradas por $G(z)$, retornando a probabilidade de que a imagem em questão seja proveniente de p_{data}. $D(x)$ dessa forma equivale a valores próximos a 1 quando a imagem vem de p_{data} e próximos a 0 quando a imagem é falsa, vinda de p_{g}, arquitetura vista na imagem 3.2. No caso ideal correspondente ao Equilíbrio de Nash do sistema, tem-se $p_{\text{g}} = p_{\text{data}}$.

Vê-se assim que a rede discriminadora atua de forma supervisionada (com rótulos) para auxiliar a rede geradora, em um processo não-supervisionado. Um ponto interessante que surge como consequência desta configuração é que $G(z)$ têm como ferramenta para aprimorar suas amostras apenas o *feedback* obtido por $D(x)$, sem nunca ter acesso direto às amostras reais.

Uma analogia usada por Goodfellow [14] para esclarecer o funcionamento de GANs é comparar a geradora a um falsificador, que gera notas falsas enquanto que a discriminadora tem papel análogo à polícia, tentando diferenciar as falsificações das originais. A competição entre os dois leva ambos a melhorarem seus métodos de falsificação e identificação, respectivamente, repetindo o processo até que as notas reais e falsas sejam indistinguíveis com $D(x) = 1/2$.

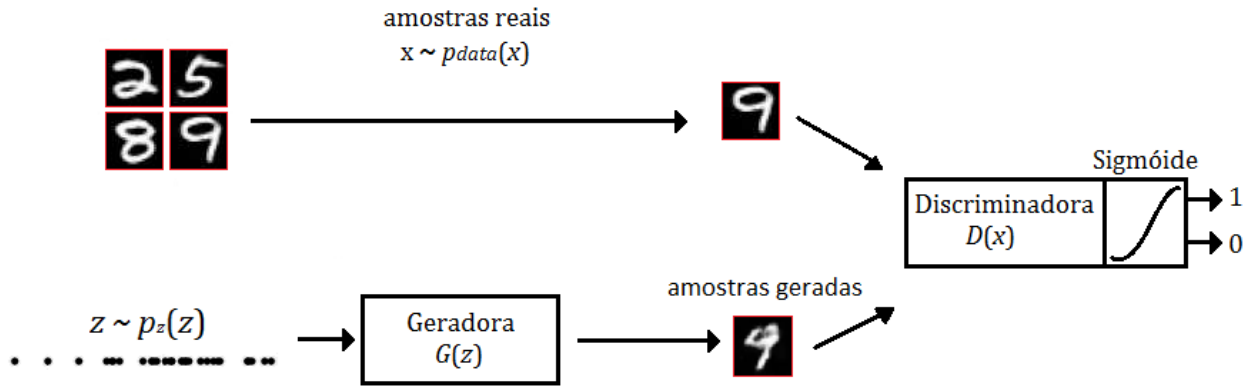


Figura 3.2: Modelo original da GAN.

3.3 Funções de Perda

3.3.1 Rede Discriminadora

Redes supervisionadas de classificação comumente fazem uso da função de entropia-cruzada dada na equação 2.1 como função de perda, considerando um sistema com i diferentes classes. A rede discriminadora atua com apenas 2 classes (imagens reais e geradas) de forma a otimizar uma função treinável $D : \mathbb{R}^n \rightarrow [0, 1]$ em que D indica a probabilidade de algum ponto $xi \in \mathbb{R}^n$ pertencer à primeira classe. Neste caso,a função de perda do discriminador pode ser simplificada pela entropia cruzada binária. Tomando $(x_i, y_i) \in (\mathbb{R}^n, \{0, 1\})$ como os pares de dados de entrada em D e considerando um *dataset* de N imagens, tal função pode ser representada por

$$H((x_i, y_i)_{i=1}^N, D) = -\sum_{i=1}^{N} y_i \log D(x_i) - \sum_{i=1}^{N} (1 - y_i) \log(1 - D(x_i)) \tag{3.2}$$

A equação acima apresenta dois termos distintos, e um deles sempre será reduzido a 0 de acordo com o valor de y_i. $D(x_i)$ indica a probabilidade que uma imagem x_i seja real, já a probabilidade que a imagem tenha sido gerada é dada por (1-$D(x_i)$). Os valores de D, dessa forma, devem estar contidos no intervalo [0,1], o que é possível com o uso da função logística na ultima camada do discriminador.

Ao trabalhar com GANs é costumeiro estabelecer algumas particularidades sobre os dados

utilizados. Sabe-se que as imagens x_{i} provêm necessariamente de duas fontes possíveis: $x_{\text{i}} \sim p_{\text{data}}$ para a distrubuição real ou $x_{\text{i}} = G(z)$ com $z \sim p_{\text{gerador}}$, para a distribuição do gerador. Considerando o caso ideal no qual cada fonte provê metade dos dados e substituindo os somatórios por expectativas para uma conotação probabilistíca a equaçaõ 3.2 passa a representar a função de perda J^D do discriminador [16], dada por

$$J^D(\theta d, \theta g) = H((x_i, y_i)_{i=1}^{\infty}, D) = -\frac{1}{2}\mathbb{E}_{x\sim p_{data}}[logD(x)] - \frac{1}{2}\mathbb{E}_z[\log(1 - D(G(z)))] \tag{3.3}$$

Ao minimizar a entropia cruzada binária em respeito aos parâmetros de rede θd e θg, o discriminador tem como missão diferenciar corretamente as imagens reais das geradas. Este processo é dado pelo método do gradiente, descrito no capítulo anterior.

3.3.2 Rede Geradora

A rede geradora deve maximizar sua habilidade em enganar a Discriminadora, cenário no qual sua função de perda pode ser dada, teoricamente, por

$$J^G = -J^D \tag{3.4}$$

Nesta situação tem-se um caso ideal descrito na teoria dos jogos como *minimax* ou jogo de soma-zero, o qual, como o nome já diz, ocorre quando a soma dos custos de todos os jogadores é sempre zero. Na prática porém, a escolha desta J^G não resulta em bons resultados [16] uma vez que inicialmente as amostras geradas por $G(z)$ são facilmente distinguíveis por $D(x)$ das amostras originais, situação em que $D(G(z))$ se aproxima de 0 e tem-se o problema de desaparecimento dos gradientes. Nesse caso, as derivadas da função de perda, calculadas pelo método de descida de gradiente para otimizá-la, antingem valores próximos de zero e os parâmetros de rede não são atualizados [16], caso do gráfico 3.3.

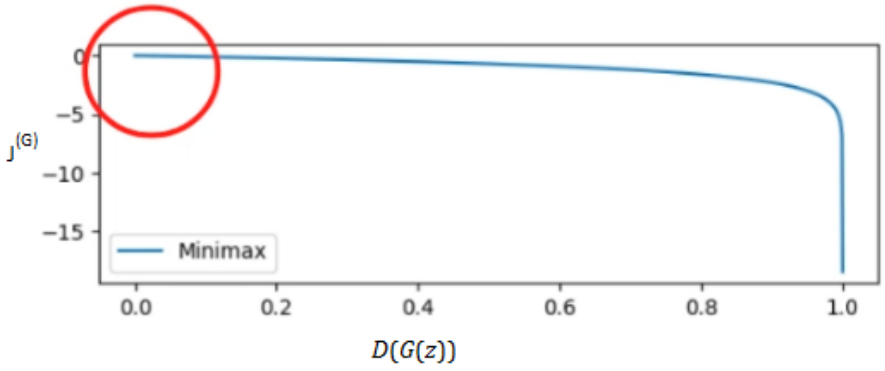


Figura 3.3: Gráfico da função de erro J^G ao se adotar $J^G = J^D$. Na porção destacada, o gradiente da curva atinge valores muito pequenos, levando ao problema de desaparecimento de gradientes e impossibilidade de atualização de parâmetros via *backpropagation* [3]. Fonte: Lazy Programmer.

Para solucionar tal problema, adota-se uma função heurística para a perda do gerador, que não mais tenta maximizar o erro de entropia cruzada do discriminador, mas sim diminuir sua própria entropia por meio da inversão de rótulos. $G(z)$ neste caso interpreta as imagens reais como 0 e as imagens geradas como 1, levando à nova função de perda J^G, vista na equação 3.5.

$$J^G = -\frac{1}{2}\mathbb{E}_z[logD(G(z))] \tag{3.5}$$

Da mesma forma que o discriminador, o gerador também apresenta função de perda diferenciável e pode ser treinado por meio de *backpropagation*. J^G e J^D no cenário proposto deixam de representar um jogo de soma-zero.

3.4 Redes Convolucionais Geradoras Adversárias Profundas

GANs convolucionais profundas (DCGAN) foram inicialmente sugeridas como modelo alternativo para melhor estabilizar o treinamento de redes adversárias [17]. Embora GANs já apresentassem camadas profudas e de convolução antes das DCGANs, seu nome é usado para referir a um tipo específico de arquitetura, que faz uso de algumas diretrizes especiais:

1. Uso de *Batch Normalization* - normalização em lote, técnica que consiste no redimensionamento dos dados na entrada de todas as camadas da rede, permitindo o uso de taxas de aprendizado maiores.

2. Uso da rede totalmente convolucional como ferramenta para simplificar o processo computacional, acelerando o processo de treinamento. Para aumentar a resolução da imagem, é usado um tipo especial de convolução denominado convolução transposta.

3. Utilização do algoritmo especial para método do gradiente denominado *Adam* [18].

4. Eliminação de camadas totalmente conectadas no gerador.

5. Aplicar a função de ativação $ReLU$ em todas as camadas do gerador, com exceção da última a qual faz uso da função tangente hiperbólica *tanh*.

6. Aplicar a função de ativação $LeakyReLU$ em todas as camadas do discriminador, com exceção da última a qual faz uso da função logística.

Essas instruções são aplicadas à rede discriminadora para formar uma rede convolucional semelhante à da figura 2.6., recebendo uma imagem x e forcenendo a probabilidade dela vir de p_{data}.

A rede geradora efetua pequenas modificações neste processo, uma vez que sua camada inicial recebe um vetor z ao invés de uma imagem, aplicando sobre este um algoritmo de redimensionamento (*reshape*) para torná-lo tridimensional. Para aumentar a resolução da imagem são usadas camadas de convolução transposta, representadas por convoluções com *stride* < 1. Isso pode ser entendido como a adição de pixels nulos entre os pixels originais da imagem antes de sua passagem pelo filtro, dilatando-a.

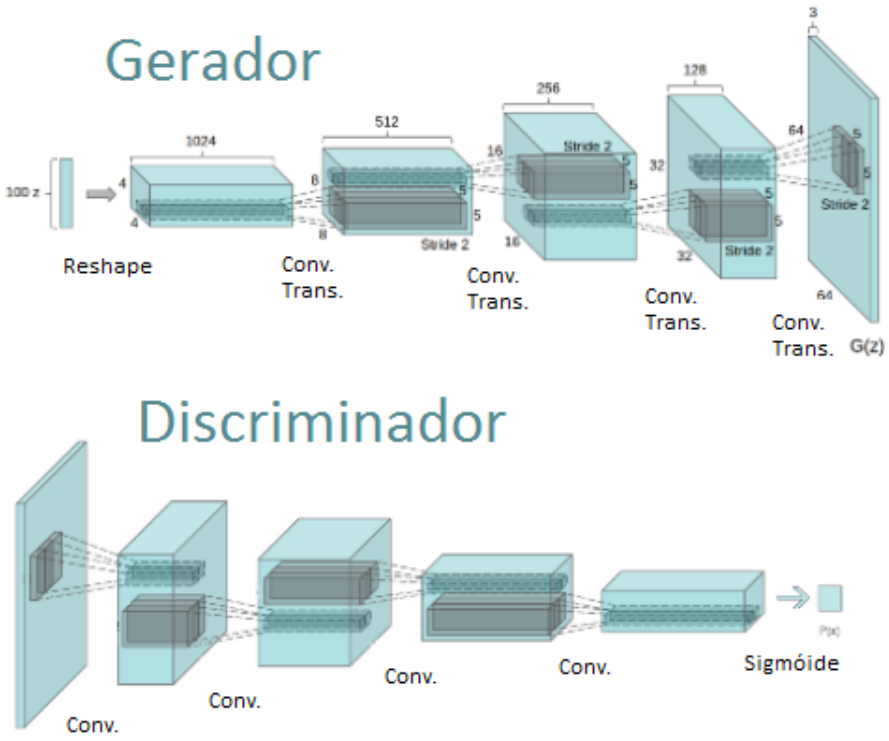


Figura 3.4: Exemplo de esqueleto para uma DCGAN. Adaptado de [17].

Capítulo 4

Aplicação

4.1 Introdução

Compreendendo a estrutura e as particularidades que compõem uma rede geradora adversária é possível aplicá-la na solução de problemas que permeiam o universo do aprendizado de máquina. Neste trabalho, deseja-se usar o elevado poder de representação destas redes para criação de imagens que se passem por amostras de um *dataset* real, ou seja, imitem a distribuição de probabilidade das figuras originais [14], como explicado no capítulo 3.

A geração de imagens naturais mostra-se um desafio de grande interesse para o ambiente de *DeepLearning*, uma vez que o sucesso na criação de imagens naturais realistas significa modelar uma distribuição de probabilidade de enorme dimensionalidade, um grande avanço para as pesquisas de aprendizado de máquina não-supervisionado [16].

O objetivo é gerar imagens que imitem a distribuição dada pelo conjunto de dados CIFAR10, o qual contém 60 mil figuras separadas em 10 classes, cada uma composta por 6 mil imagens coloridas de 32x32 pixels. Estas figuras foram escolhidas para alimentar uma Rede Convolucional Geradora Adversária Profunda(DCGAN) construída com arquitetura baseada nas pesquisas mais recentes desse tipo de rede geradora, com propriedades baseadas na experimentação que permitem uma melhor convergência dos resultados, como será visto ao longo do capítulo. Espera-se com isso reconhecer a relevância de redes adversárias em mapear dados de vasta complexidade, pavimentando o caminho dos estudos nesta área em expansão que é a modelagem geradora [14].

Para criação da GAN foi usada linguagem Python 3.0 com auxílio da interface Keras no ambiente Google Colaboratory, o qual faz uso da GPU Tesla K80 para seus cálculos.

4.2 CIFAR10

Imagens podem ser interpretadas como amostras de uma distribuição de probabilidade hiperdimensional. Ao tirar uma foto, por exemplo, pode-se amostrar desta distribuição sem conhecê-la explicitamente. Neste cenário, o conjunto de dados escolhido para representar a distribuição a ser

modelada $p(x)$ é o CIFAR10 [19], formado por 60 mil imagens separadas nas classes avião, automóvel, pássaro, gato, veado, cachorro, sapo, cavalo, navio e caminhão. Cada classe possui 6 mil imagens coloridas. Analisando algumas amostras deste conjunto de dados na Figura 4.1. pode-se perceber pontos interessantes.

As figuras apresentam baixa resolução, por estarem reduzidas à 32x32 pixels. Por serem coloridas, apresentam 3 canais de cores RGB de forma que cada imagem tem dimensionalidade 32x32x3 = 67.584. Cada uma destas dimensões corresponde à intensidade de cada pixel da figura. É evidente assim porque as imagens são reduzidas, visto que caso apresentassem uma resolução maior o número de dimensões representadas cresceria na ordem n^2, exigindo um grande poder computacional no cálculo dos parâmetros da rede.

Observando as imagens como elementos de $\mathbb{R}^{67584}$, tem-se uma noção da tamanha complexidade que a modelagem geradora toma ao tentar criar amostras que imitem este *dataset* e o porquê das imperfeições ainda presentes neste modelo, o qual não apresenta funções objetivamente definidas e faz uso da experimentação, com ciclos de tentativa e erro, para alcançar o melhor resultado [20].

Outro ponto de interesse diz respeito à grande variação presente nas imagens, nas quais os objetos representados apresentam formas, cores e ângulos muito diversos. Essa diversidade de características representa uma distribuição multimodal e é parte do que se deseja aprender a reproduzir com a rede geradora. Em contrapartida, garantir que as entradas da rede tenham caracterísitcas semelhantes facilita o seu treinamento [21]. Por isso a rede geradora neste projeto foi treinada separadamente em cada classe, garantindo que a mesma aprenda de forma mais fiel às estruturas de cada uma. Tem-se assim a mesma arquitetura em todos os casos porém pesos específicos para cada classe.

4.3 Arquitetura do Modelo

Dado um conjunto de dados de treinamento, GANs podem aprender a estimar a distribuição de probabilidade subjacente aos mesmos, por meio de um ciclo de aprendizado entre gerador e discriminador em busca do equilíbrio de Nash [22] em um cenário ideal. Ao representar o gerador e o discriminador como *perceptrons* multicamadas (MLP), ambos podem ser treinados com o uso do bem-sucedido método de *backpropagation*, e as amostras de $G(z)$ obtidas usando somente *forward propagation*, sem a necessidade de Cadeias de Markov ou inferência aproximada [14].

Tais facilidades circundam obstáculos presentes em outros modelos geradores, os quais apresentam dificuldades em aproximar distribuições de probabilidade muito complexas. Assim, o modelo de redes neurais utilizado neste projeto é de redes convolucionais geradoras adversárias profundas (DCGANs) baseado na rede adversária de Goodfellow citada acima. Este modelo, por representar o gerador e o discriminador como redes convolucionais, apresenta grande poder representacional no processamento de imagens, justificando sua escolha.

Em se tratando de redes adversárias, a escolha dos diversos parâmetros (momento, taxa de *dropout*, número épocas, etc) deve fazer uso de constante experimentação, uma vez que não se tem

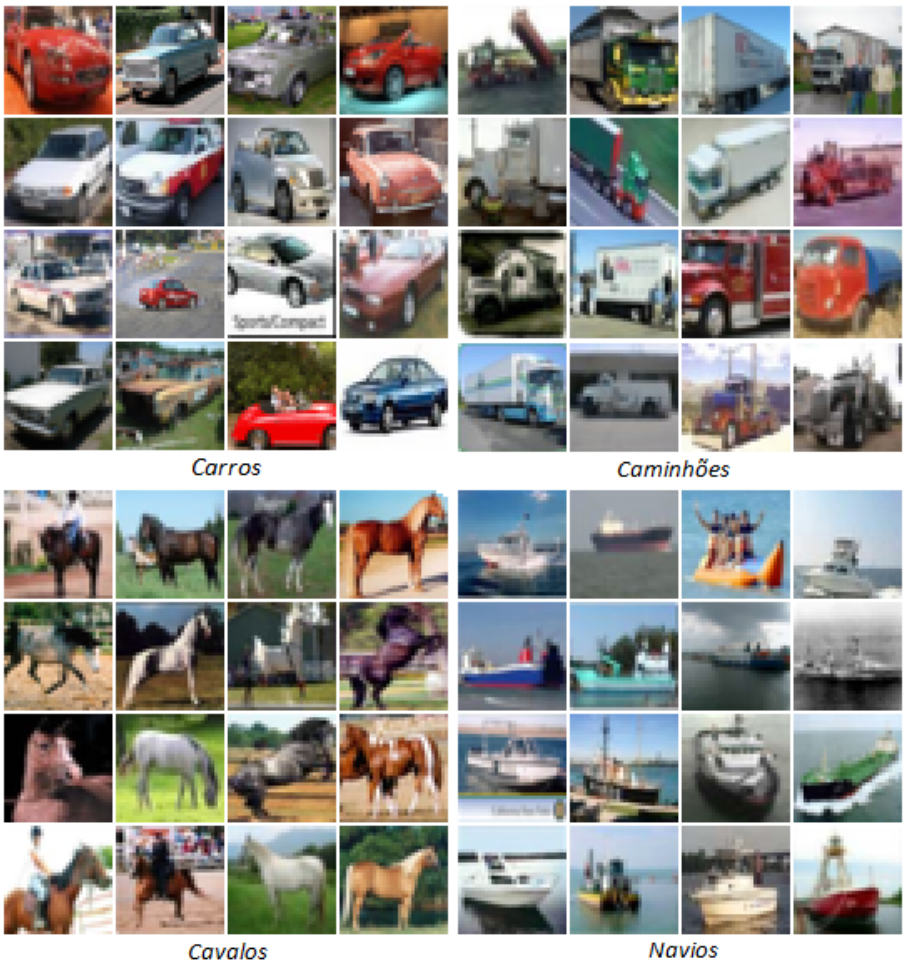


Figura 4.1: Amostras de 4 das 10 classes que compõem o dataset CIFAR10. As imagens são 32x32x3 com todas as classes sendo mutuamente exclusivas: não há sobreposição entre caminhões e carros, por exemplo.

uma função de perda objetiva, dificultando a comparação de performance entre modelos diferentes [23].

Seguindo as diretrizes mais atuais sobre o esqueleto de uma rede adversária [17, 23] e após contínua avaliação entre as amostras produzidas com a variação desses parâmetros, criou-se a arquitetura de rede vista na Figura 4.2.

4.3.1 Modelo do Discriminador

O discriminador $D(x; \theta d)$ é uma rede neural convolucional (CNN) de classificação binária, parametrizada por θd. Recebe como entrada uma imagem 32x32x3 que pode vir do conjunto CIFAR10 ou do gerador $G(z; \theta g)$ e deve discenir a sua origem, tendo como saída 1 para imagens reais e 0 para imagens geradas (falsas). Para isso, a imagem recebida passa por 4 camadas de convolução intercaladas com 4 de normalização e 4 de ativação por $LeakyReLU$. Por fim, uma camada de *Flatten* redimensiona o tensor tridimensional em um vetor unidimensional, submetendo-o a uma

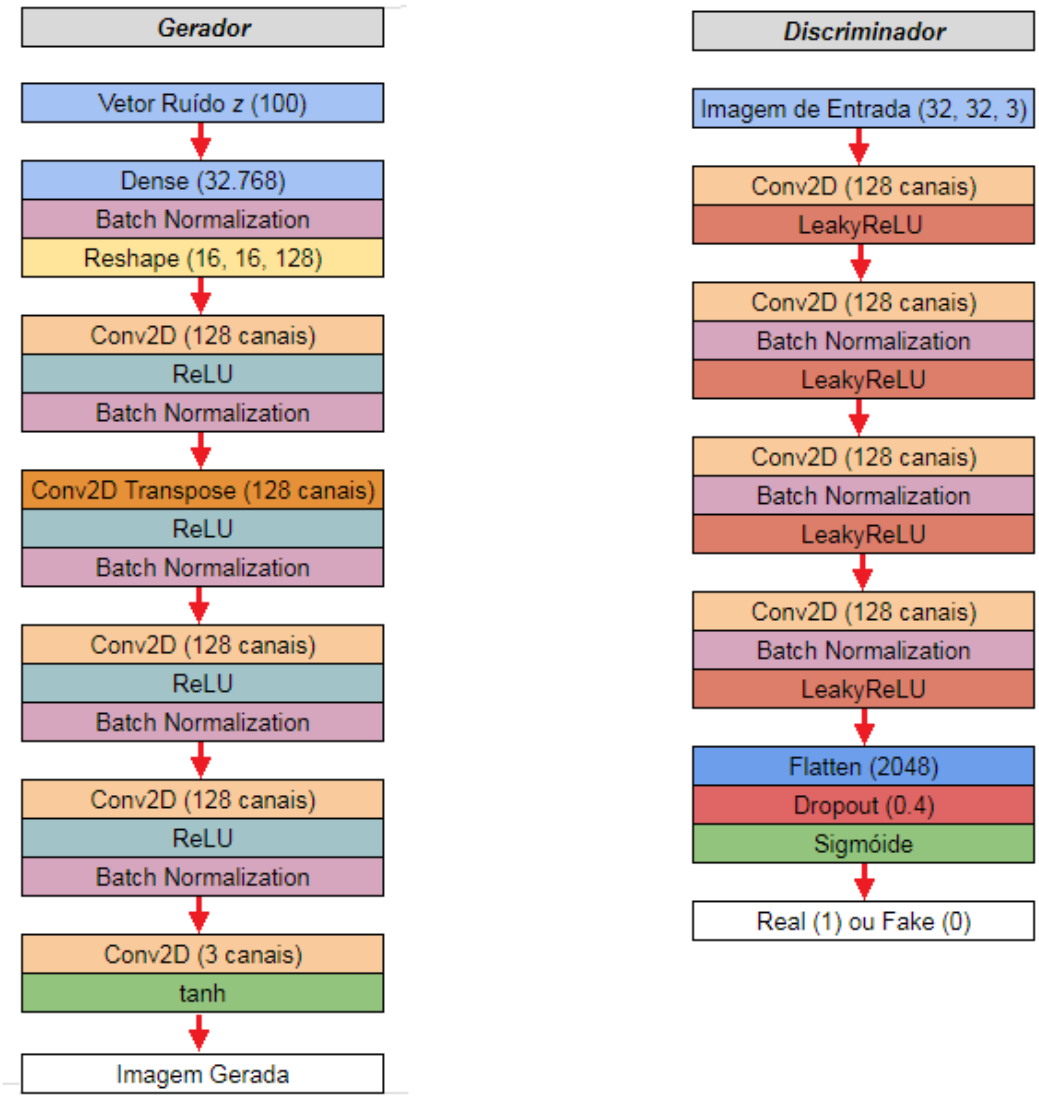


Figura 4.2: Estrutura do gerador e discriminador usados na DCGAN do projeto. As camadas de convolução são bastante eficientes para o processamento de imagens, e estão intercaladas com camadas de normalização *batches* e ativação pela função *LeakyReLU* para o caso do discriminador. As versões finais dos parâmetros foram obtidas com contínua experimentação, seguindo diretrizes recomendadas pelas pesquisas mais recentes em DCGANs.

Camada	Dimensões de Saída
Entrada	(32, 32, 3)
Convolução 2D (LeakyReLU, filtros 3x3, strides 1x1)	(32, 32, 128)
Convolução 2D (LeakyReLU, filtros 4x4, strides 2x2)	(16, 16, 128)
Convolução 2D (LeakyReLU, filtros 4x4, strides 2x2)	(8, 8, 128)
Convolução 2D (LeakyReLU, filtros 4x4, strides 2x2)	(4, 4, 128)
Flatten	(2048)
Dense (Sigmóide - Logística)	(1)

Tabela 4.1: Arquitetura DCGAN para o discriminador.

camada totalmente conectada com ativação por função logística e uso da técnica *Dropout*, arquitetura evidenciada na tabela 4.1. O número total de parâmetros treináveis ao fim da configuração é de 793,473.

A função de erro J^D do discriminador é dada, na prática, pela equação 3.2 de entropia cruzada binária, a qual equivale à divergência de Kullback-Leibler em problemas de classificação [24].

O treinamento do discriminador ocorre separadamente do gerador, em etapas alternadas. Neste processo, $D(x; \theta d)$ é alimentado com imagens reais e falsas (em *mini-batches* separados) tendo como saída um número E [0,1] representando a probabilidade de origem da figura. As atualizações dos parâmetros θd ocorrem por *backpropagation* seguindo o método do gradiente da função de erro J^D.

4.3.2 Modelo do Gerador

O gerador $G(z; \theta g)$ é uma CNN parametrizada por θg que recebe como entrada um vetor z de um espaço latente, em que z é composto por 100 números aleatórios retirados de uma distribuição gaussiana p_z. Outras distribuições podem ser usadas, como a distribuição uniforme [25].

O vetor de entrada é inserido na rede profunda e redimensionado em um tensor tridimensional (altura x largura x cores) por meio da camada *reshape* passando em seguida por camadas de convolução e convolução transposta, a qual permite o aumento da resolução da imagem. Normalização em *batches* e ativação por $ReLU$ também foram usadas de forma intercalada. A última camada faz uso da ativação pela função hiperbólica *tanh*, a qual mostrou-se superior na criação de imagens do que a função logística [26]. Ao final do processo tem-se 4.877.187 parâmetros treináveis - vê-se que a geradora apresenta uma quantidade muito superior de parâmetros do que a discriminadora, uma vez que tem um trabalho mais complexo de criação de amostras.

Para o treinamento do gerador, deve-se escolher uma função de erro apropriada. Como exposto no capítulo anterior, a forma mais simples J^G = -J^Dnão gera bons resultados ao ser aplicada na prática, de forma que foi adotada a versão heurística de J^G correspondente à equação 3.5.

Durante seu treinamento, o gerador é conectado em série com o discriminador, tendo como saída a probabilidade $D(x)$ sobre a imagem gerada, valor que alimenta a função de erro J^G, atualizando os parâmetros θg por meio do método do gradiente em *backpropagation*. Durante este processo, os

Camada	Dimensões de Saída
Entrada	(100)
Dense	(32.768)
Reshape	(16, 16, 128)
Convolução 2D (ReLU, filtros 5x5, strides 1x1)	(16, 16, 128)
Convolução 2D Transposta (ReLU, filtros 4x4, strides 2x2)	(32, 32, 128)
Convolução 2D (ReLU, filtros 5x5, strides 1x1)	(32, 32, 128)
Convolução 2D (ReLU, filtros 5x5, strides 1x1)	(32, 32, 128)
Convolução 2D (ReLU, filtros 5x5, strides 1x1)	(32, 32, 3)
Ativação (tanh)	(32, 32, 3)

Tabela 4.2: Arquitetura DCGAN para o gerador.

parâmetros da rede discriminadora estão congelados.

4.4 Algoritmo

Os modelos das rede adversárias são treinados seguindo algoritmo 4.1. A atualização dos pesos de $G(z)$ e $D(x)$ ocorre por meio do cálculos dos gradientes $\nabla\theta_{\mathrm{d}}$ e $\nabla\theta_{\mathrm{g}}$ com a técnica de otimização *Adam*.

Inicialmente são amostradas imagens reais da distribuição real $p_{\mathrm{data}}(x)$, nas quais performa-se a atualização dos parâmetros do discriminador. Em seguida, é realizado o mesmo processo de amostragem e treinamento destes parâmetros, porém com amostras da distribuição a priori $p_{\mathrm{g}}(z)$.

Por fim, ocorre o treinamento da rede geradora. Esse processo ocorre com os pesos θ_{d} congelados, de forma que imagens geradas alimentam $D(x)$ atualizando somente $G(z)$, em um processo iterativo que visa a geração de amostras cada vez mais semelhantes às originais.

for *número de iterações* **do**
  for *k iterações* **do**
    Amostrar minibatch de m amostras $\{\mathrm{z}^{(1)},...,\mathrm{z}^{(\mathrm{m})}\}$ da distribuição a priori $p_{\mathrm{g}}(z)$;
    Amostrar minibatch de m exemplos $\{\mathrm{x}^{(1)},...,\mathrm{x}^{(\mathrm{m})}\}$ da distribuição real $p_{\mathrm{data}}(x)$;
    Calcular o erro do discriminador:
    $J^D = -\frac{1}{n}\left(\sum_{i=1}^{n}\log D(x_i) + \sum_{i=1}^{n}\log(1 - D(G(z_i)))\right)$
    Atualizar os parâmetros θ do discriminador pelo método do gradiente usando a técnica *Adam*.
  end
  Amostrar minibatch de m amostras $\{\mathrm{z}^{(1)},...,\mathrm{z}^{(\mathrm{m})}\}$ da distribuição a priori $p_{\mathrm{g}}(z)$;
  Calcular o erro do gerador:
  $J^G = -\frac{1}{n}\left(\sum_{i=1}^{n}\log(D(G(z_i)))\right)$
  Atualizar os parâmetros θ do gerador pelo método do gradiente usando a técnica *Adam*.
end

Algorithm 1: Atualização dos parâmetros do discriminador e gerador.

4.4.1 Dificuldades no Treinamento

Treinar GANs não é uma tarefa trivial, de modo que certos ramos de pesquisa são dedicados inteiramente à otimização desse processo [23, 25]. Essa dificuldade fica clara quando entendemos que as redes dão passos sucessivos em busca de minimizar objetivos não-convexos e acabam em um processo oscilante, em vez de diminuir um objetivo único como ocorre com as ANNs discriminadoras simples.

Várias instabilidades decorrem desta oscilação, o que é visto quando D ou G acabam por sobreperformar a rede adversária e seu erro decai rapidamente à 0, causando a divergência do erro da outra. Ao longo do projeto, este processo de divergência se deu principalmente com o gerador, o que é intuitivo visto que sua maior complexidade exige um treinamento mais longo para que a rede atualize seus parâmetros. Algumas técnicas podem ser usadas para combater o desequilíbrio, como treinar mais vezes a rede perdedora ou otimizar sua arquitetura para que a mesma atualize os parâmetros de forma mais eficiente. A segunda opção, na forma de tentativa e erro, foi a tomada no projeto.

Outro cenário de falha comum em GANs é chamado *mode collapse* - colapso de modos [27]. Uma vez que as distribuições de probabilidades trabalhadas pelas redes geradoras são de grande complexidade, tais distribuições apresentam diversos "picos"em que os diferentes subgrupos de amostras estão concentrados. Por vezes o gerador atualiza seus parâmetros de modo a produzir imagens de apenas um destes sub-grupos, criando amostras com baixa entropia em busca da imagem ótima $x* = argmaxD(x)$. No caso mais extremo, $x*$ independe de z e o modo colide para um único ponto. Assim, por mais que o gerador consiga enganar o discriminador, suas amostras apresentam pouquíssima (ou nenhuma!) variedade. Tal fenômeno é visto na Figura 5.3, e pode ser mitigado com o uso de discriminação de *mini-batch* [28]. o algoritmo original, $D(x)$ processa cada exemplo de forma independente, não havendo coordenação entre seus gradientes e, portanto, sem mecanismos para incentivar as saídas do gerador à tornarem-se mais dissimilares. [23]. Uma estratégia óbvia para evitar esse tipo de falha é permitir que o discriminador observe vários exemplos do conjunto de dados em combinação, evitando assim o colapso do gerador.

4.4.2 Outras Técnicas de Otimização

Desde sua concepção original, GANs apresentaram uma série de modificações para permitir melhor convergência de seus parâmetros e maior qualidade das amostras geradas. A própria arquitetura DCGAN faz uso de estratégias diversas para permitir tais melhorias. Como realçado no capítulo 3, mudanças como a substituição de camadas de *pooling* por camadas de convolução ($D(x)$) e convolução transposta ($G(z)$), uso de normalização em *batches*, remoção de camadas totalmente conectadas e uso dos filtros $ReLU$ e $LeakyReLU$ já permitem uma grande melhoria quando comparadas às arquiteturas pioneiras de redes adversárias [17].

Além de tais técnicas, bem como as mencionadas no tópico 4.4.1., outras estratégias foram utilizadas para melhorar o desempenho da rede adversárias. São elas:

- Modificação da função de erro. Em acordo com as noções mais atuais de pesquisa, foi adotada

a versão heurística da função de perda do gerador vista na equação 3.5, permitindo uma maior eficiência na convergência dos resultados.

- Uso de uma distribuição gaussiana (e não uniforme) para amostragem da variável latente z, permitindo produção de amostras mais nítidas [29].

- Uso de *Label Smoothing* (suavização), redefinindo os rótulos das amostras que ingressam na rede discriminadora para valores atenuados. Assim, os valores 0 e 1 representativos de amostras reais e falsas, respectivamente, passam a valer números próximos à 0.1 e 0.9. Tais medidas reduzem a vulnerabilidade de redes neurais em exemplos adversários [30].

- Inserção de ruído nos rótulos de treinamento das amostras, de modo que para uma pequena porcentagem (no projeto, 5%) das imagens que foram alimentadas ao discriminador, os rótulos foram invertidos aleatoriamente, isto é, imagens reais foram rotuladas como geradas e imagens geradas rotuladas como reais [31].

- Uso de da técnica de *Dropout* que desconsidera alguns nêurons da rede aleatoriamente e servindo como forma de regularização [32].

Capítulo 5

Resultados

5.1 Qualidade das Amostras Geradas

Para avaliar a qualidade da GAN projetada, deve-se ter alguma forma de comparar as imagens criadas às fotos originais. Infelizmente ainda não há uma maneira clara de avaliar quantitativamente modelos geradores [33]. Alguns métodos foram idealizados, como ajustar uma janela de Parzen às amostras geradas e referir a função de verossimilhança sob essa distribuição, porém apresentam alta variância e não têm boa performance em espaços hiper-dimensionais [14].

A função de erro da GAN se compora de forma diferente de modelos discriminadores: em um típico cenário de aprendizado de máquina, tem-se uma função de erro que decai suavemente com o avanço do treino. isso não é o caso para a GAN, como evindeciado nos gráficos de erro J^D e J^G, uma vez que as duas redes estão constantemente trabalhando para minimizar seu própio erro e acabam por aumentar o do adversário. O erro de uma GAN, dessa forma, não mede objetivamente a qualidade das amostras como nos casos de outras redes simples, mas sim representa uma forma de medida interna, indicando somente o quão bem $D(x)$ e $G(z)$ conseguem enganar seu oponente. O problema da não-convergência é, com efeito, um dos maiores obstáculos enfrentados atualmente no desenvolvimento de redes adversárias [16].

Uma forma intuitiva de se medir a capacidade de geração de GANs é pela avalição subjetiva de observadores [23, 34], usando das funções de erro como auxiliares em avaliar a robustez das redes. Este foi o método utilizado no projeto, parando o treinamento da rede quando as funções J^D e J^G aproximam-se de um equilíbrio e as amostras geradas assemelham-se subjetivamente às figuras do conjunto de dados CIFAR-10.

5.2 Exemplos de Amostras Geradas

Pode-se ver na Figura 5.1 algumas imagens selecionadas geradas pelo gerador a partir de pontos no espaço latente amostrado de $p(z)$. De modo geral, é possivel ver como a GAN conseguiu representar corretamente diversas propriedades primárias das figuras, com escolha correta de cores,

formas e ângulos dos objetos para boa parte dos casos.

As figuras se aproximam das classes desejadas em graus diferentes, de forma que para classes como barcos e cavalos as amostras geradas apresentam um grau de verossimilhança bastante elevado com o *dataset* original ao passo em que amostras de caminhões apresentam qualidade inferior. O mesmo comportamento é observado nas amostras das demais classes, vistas na Figura 5.3.

Focando na classe de cavalos, por exemplo, ficam evidentes as características aprendidas pela rede geradora, com a presença de crinas, caudas, pelagens variadas e planos de fundo bem definidos. É notório pelas amostras de barcos como a DCGAN conseguiu assimilar propriedades diversas de uma distribuição multimodal, evidente nos variados moldes de carcaças: veêm-se amostras que recordam grandes embarcações, pequenos barcos de pesca e *jet-skis*.

O método ainda apresenta imperfeições visíveis, com algumas classes tendo gerado amostras pouco reconhecíveis. Tais defeitos se dão por uma variedade de fatores:

-A configuração da rede criada não é ótima para cada caso, visto que a mesma arquitetura da DCGAN foi utilizada para todas as classes. Os resultados nesta configuração se mostram experimentalmente superiores ao caso em que a rede foi treinada juntamente em todas as classes, mas se abre assim um leque de opções para explorar cada classe individualmente.

- O conjunto de dados CIFAR10 tem poucas amostras, com 6 mil imagens por classe. Se os dados existentes são esparsos, a GAN não irá aprender uma distribuição ideal. Isso ocorre porque o gerador tenta memorizar os poucos modos presentes no dataset, não aprendendo a interpolar adequadamente entre as imagens de treinamento.

- As imagens apresentam grande variação (o que é mais acentuado em algumas classes), com ângulos e formas muito distintas, dificultando à rede a estudar todos seus detalhes.

5.2.1 Exemplos Adicionais e Comparações

Submentendo as demais classes do conjunto CIFAR10 à rede DCGAN, é possível obter resultados que legitimizam a flexibilidade do modelo gerador em representar diferentes dados. A figura 5.2 apresenta exemplos do conjunto original para suas 6 classes restantes, enquanto a Figura 5.3 exemplifica amostras geradas por $G(z)$ para tais classes, com graus de sucesso variados, porém com tendências positivas para todas as classes. As amostras correspondentes à classe de veados apresentaram o maior grau de verossimilhança com as originais. A Figura 5.4 revela situação de *mode collapse* parcial, em que amostras geradas apresentam a mesma textura, cor e formas, indicando que $G(z)$ mapeia diferentes valores de entrada z para o mesmo ponto de saída.

Resultados interessantes podem ser obtidos ao alimentar a rede geradora com pontos fixos do espaço latente, mostrando a evolução da qualidade das amostras ao longo das iterações como visto na Figura 5.5.

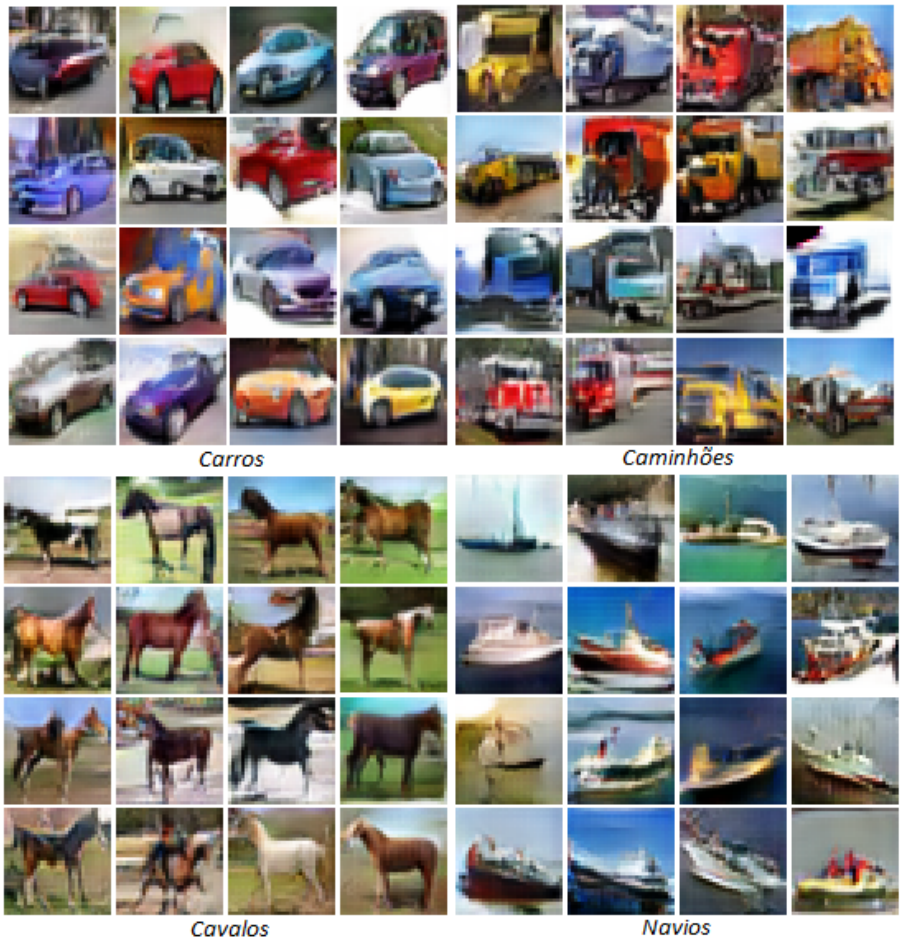


Figura 5.1: Amostras geradas por $G(z)$ após aproximadamente 60 épocas para 4 das 10 classes do dataset CIFAR10. Vê-se que a qualidade das imagens é muito boa no geral, com detalhes mais presentes em determinadas classes. Atenção para particularidades primárias como os formatos e contornos complexos dos cavalos e detalhes secundários como cores metálicas nos carros e planos de fundo relevantes nos barcos e cavalos.

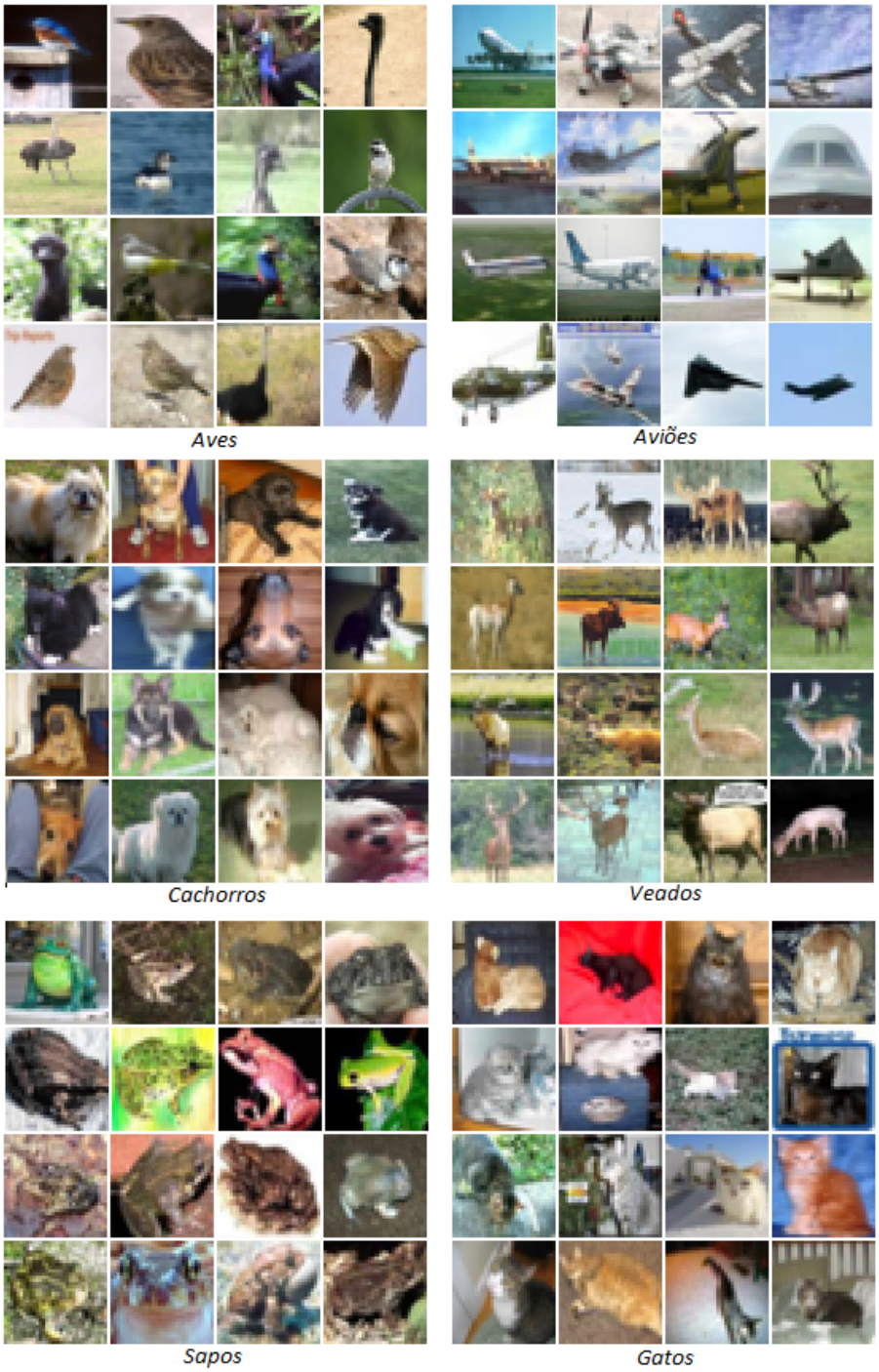


Figura 5.2: Amostras de imagens para as demais classes do conjunto de dados CIFAR10.

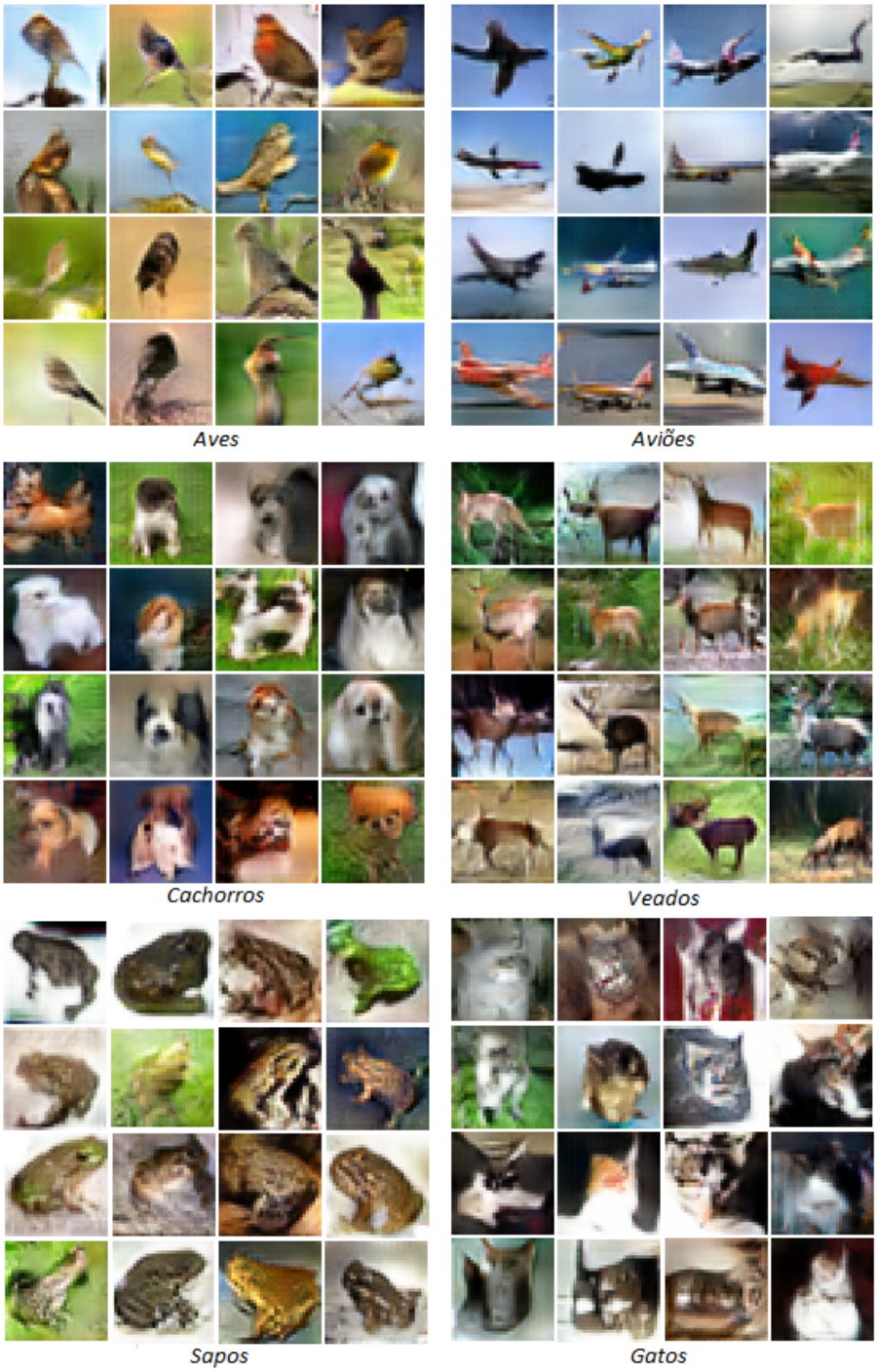


Figura 5.3: Amostras de imagens geradas por $G(z)$ para as demais classes do conjunto de dados CIFAR10, após aproximadamente 60 épocas. É notória a qualidade das figuras ao compará-las com amostras originais vistas na Figura 5.2, evidenciando o poder de representação da DCGAN.

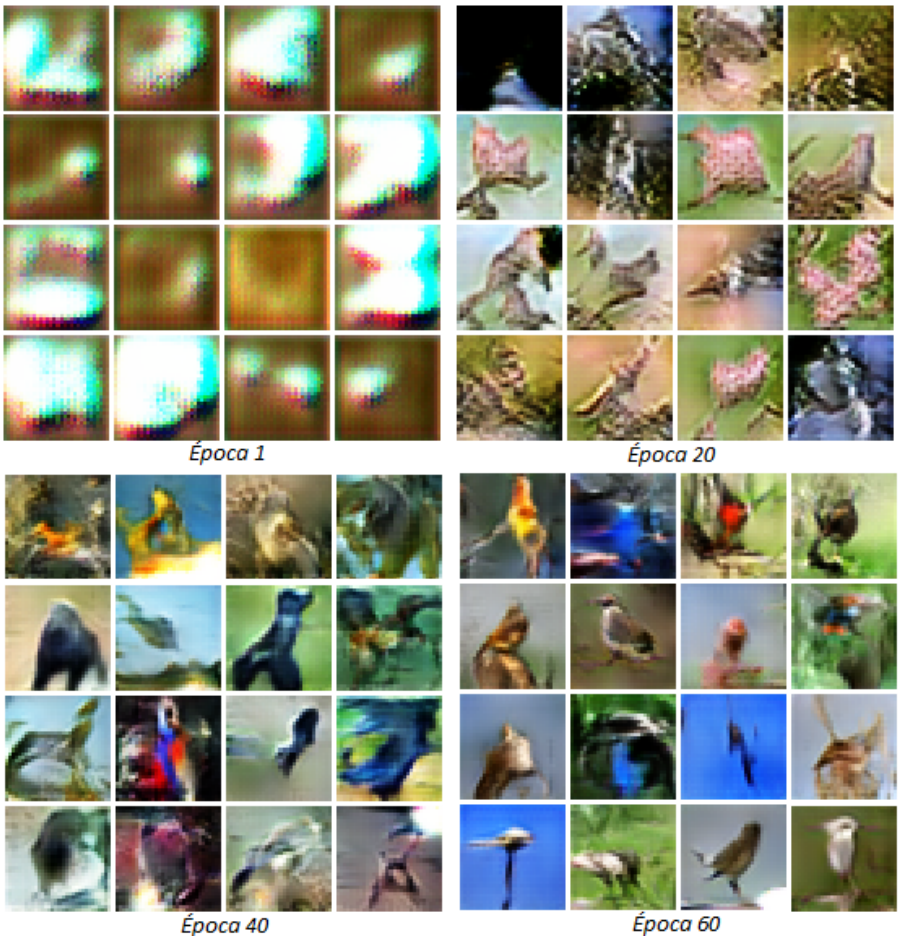


Figura 5.4: Evolução das amostras geradas por $G(z)$ para a classe de aves ao longo de 60 épocas. Amostras muito parecidas na época 20 revelam o problema de *mode collapse*.

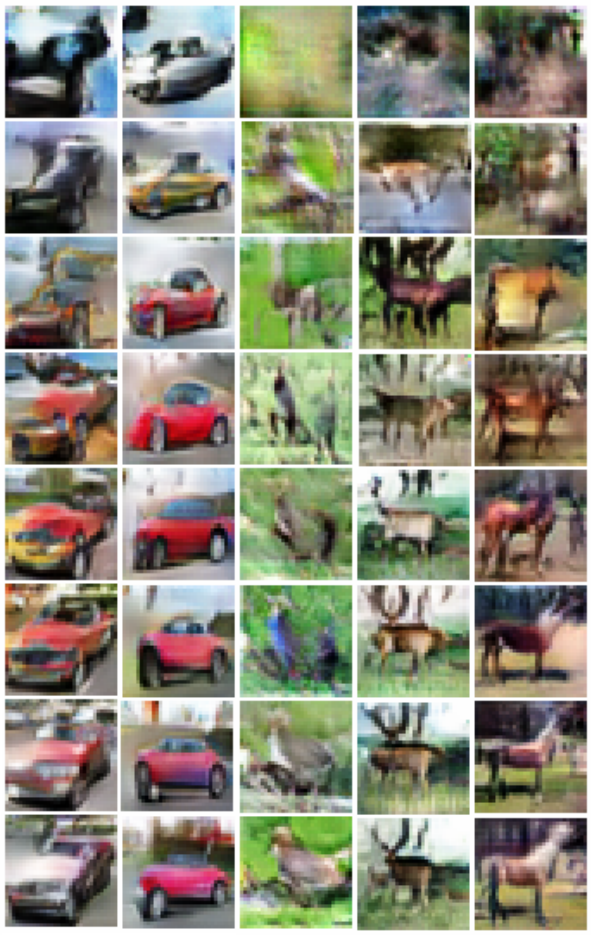


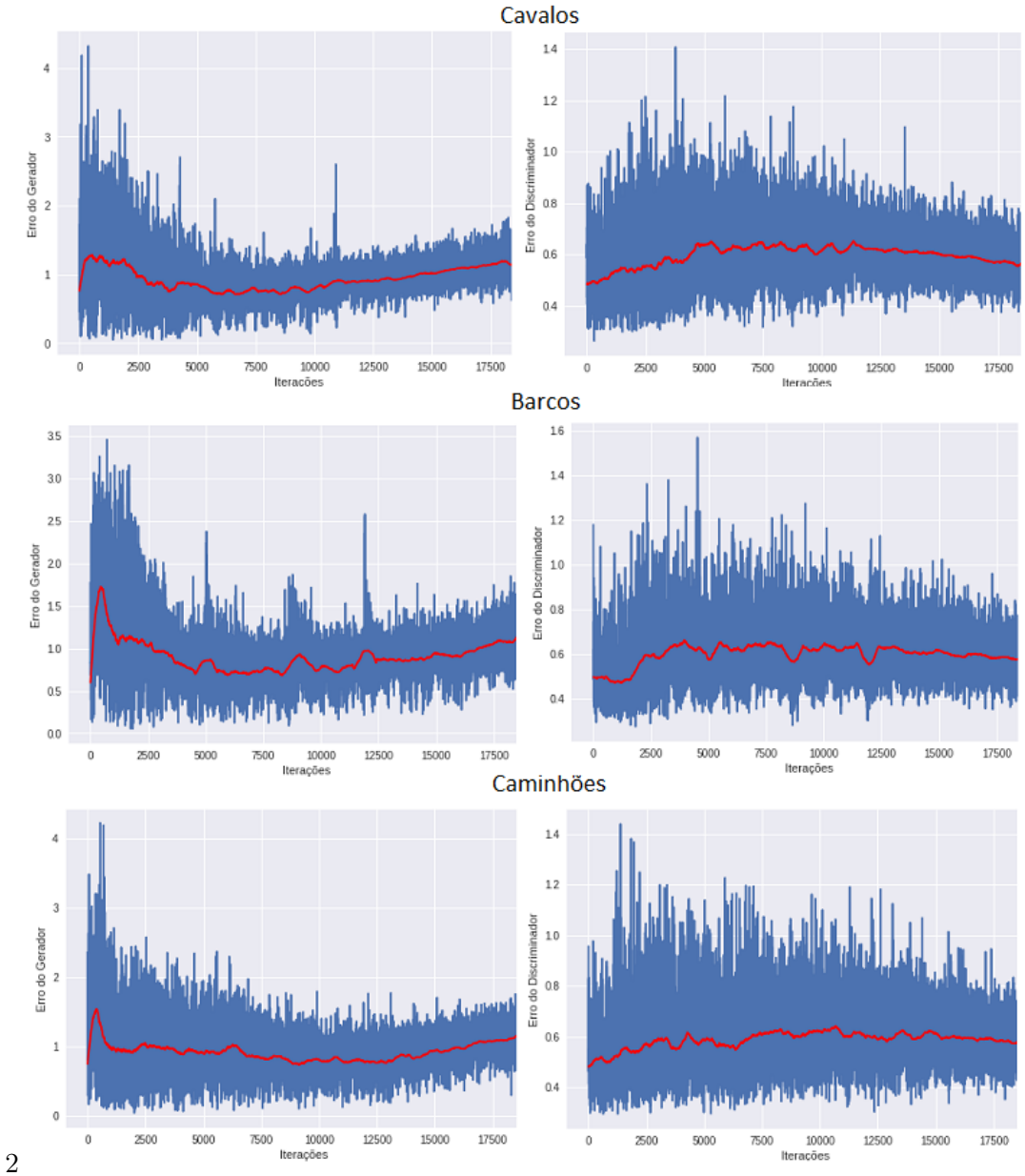
Figura 5.5: Amostras geradas para as classes carro, ave, veado e cavalo por várias iterações considerando um vetor fixo z do espaço latente.

5.2.2 Evolução das Funções de Erro

Os gráficos expostos na Figura 5.6 revelam os custos J^G e J^D para algumas das classes de CIFAR10 ao longo das iterações. Como já exposto, estas funções de erro não conseguem fornecer uma apreciação objetiva das imagens criadas por $G(z)$, mas permitem avaliar de formas indiretas o desempenho das redes adversárias.

O ponto mais interessante a ser notado é o comportamento oscilante das curvas de erro, antagônicos ao caso das redes neurais discriminativas de classificação, as quais apresentam uma suave curva descendente. As curvas grosseiras surgem uma vez que o discriminador e o gerador, em busca de otimizar seu próprio objetivo, acabam por aumentar o erro de seu adversário. Esse cenário, baseado num jogo de *minimax*, exibe apenas o quão bem um dos jogadores se sai em relação ao outro, caso oposto à situação do custo único, no qual o erro diminui gradativamente com a atualização dos gradientes. Apesar de não trabalhar com o caso ideal em que J^G e J^D representam funções opostas em um jogo de soma-zero, resquícios deste comportamento podem ser vistos nos gráficos, que apresentam curvaturas contrárias: à medida que um dos erros aumenta o outro decresce e vice-versa. J^G e J^D podem também fornecer dados importantes sobre o comportamento da GAN

e como melhorar seus parâmetros: em casos de desequilíbrio entre o gerador e o discriminador, um dos erros converge rapidamente à 0, evidenciando a superioridade do jogador em sobreperformar a rede adversária.



2

Figura 5.6: Gráficos das funções de custo J^G e J^D por iterações, para as classes de cavalos, barcos e caminhões. As linhas em vermelho representam os erros suavizados por um filtro de Savitzky–Golay.

Capítulo 6

Conclusão

Yann LeCun, pioneiro nos estudos de redes neurais e diretor da área de pesquisa de IA do Facebook, fez referência às redes adversárias como "a ideia mais interessante em aprendizado de máquina nos últimos 10 anos"[35].

É visível, no atual universo da inteligência artifical, crescente interesse em programar redes que façam mais do que estabelecer as fronteiras entre diferentes classes, mas que também mapeiem as distribuições das mesmas. Tal abordagem, tomada por modelos geradores, mostra grande valor quando usada em arquiteturas de redes não-supervisionadas.

Redes adversárias geradoras, estudadas neste projeto, atuam como modelos geradores ao aprender de forma implícita uma aproximação de determinada distribuição p_{data} por meio de amostras deste conjunto, em processo de geração de novas amostras [14].

A motivação para estudo de modelos geradores pode parecer nebulosa, principalmente em se tratando de modelos implícitos que apenas geram amostras, e não diretamente uma estimativa da função densidade de probabilidade. Tal estudo, porém, mostra-se essencial ao expandir os conhecimentos atuais em manipular espaços hiperdimensionais e de grande complexidade, conceitos fundamentais em diversos campos da engenharia e da matemátia aplicada [16].

Assim, o objetivo do atual projeto foi o estudo de GANs como instrumento em modelar as distribuições de imagens do conjunto de dados CIFAR10, gerando novas amostras que se aproximem deste *dataset* e evidenciem o poder representacional destas redes. A arquitetura específica DCGAN foi escolhida por ser altamente eficiente em transformações de imagens, capaz de escaloná-las com facilidade e gerar amostras de grande verossimilhança com as figuras originais.

Usando o ambiente Google Colaboratory com a interface Keras, foi possível implementar uma arquitetura de redes adversárias que utilizam suas diretrizes mais atuais, com a escolha de vários aspectos da rede sendo escolhidos após extensa triagem, analisando subjetivamente a qualidade das amostras criadas com as diferentes configurações. Para melhor representação de distribuições multimodais, optou-se por treinar a GAN separadamente para cada classe do *dataset*, de forma que cada caso apresente o mesmo esqueleto de rede, porém com diferentes parâmetros que se adequem aos dados de entrada.

Os resultados obtidos mostram imagens de qualidade muito satisfatória, semelhantes às originais em relação às formas, cores e caracterísicas secundárias. Ocasionalmente foram geradas figuras irreconhecíveis ou pouco relevantes, porém os traços e particularidades desejados foram, em geral, bem representados pela rede.

Desse modo, treinar e amostrar a partir de modelos geradores de forma bem-sucedida aumenta a nossa capacidade de manipular espaços e distribuições de muitas dimensões, ampliando o conhecimento em áreas essenciais da ciência. Em se tratando do contexto de imagens, tais técnicas podem ser aplicadas à previsão de *frames* de vídeo, preenchimento de imagens incompletas, redimensionamento por *super-resolution* [36] e outras aplicações que justificam o investimento de recursos no estudo de redes geradoras.

Nas palavras de Richard Feynman, "O que eu não posso criar, eu não compreendo".

6.1 Trabalhos Futuros

DCGANs representam um grande avanço na arquitetura de redes adversárias. Não são, porém, as únicas categorias de GANs que merecem atenção. Recentemente, novas arquiteturas de redes geradoras vêm apresentando grande potencial na criação de imagens, abrindo um leque de opções no mapeamento de conjuntos de dados.

Dessa forma, sugere-se, para trabalhos futuros, explorar tais arquiteturas a fim de obter amostras de maior qualidade, além de buscar formas objetivas de se comparar as imagens geradas com maior precisão.

Redes geradoras adversárias condicionais (CGANs) [21], por exemplo, fazem uso de rótulos adicionais y que alimentam o discriminador e o gerador, os quais passam a estar condicionados na forma $D(x|y)$ e $G(z|y)$. Neste modelo, a adição dos rótulos cria uma nova camada de informação ao sistema, gerando assim amostras com caracterísitcas específicas e com propriedades mais fiéis às originais.

Já Wasserstein GANs [37] tentam resolver o problema clássico de convergência, no qual a função de erro não tem relação direta com a qualidade das imagens geradas. GANs podem ser interpretadas como maneiras de minizar a divergência de Jensen-Shannon, que vale 0 caso as distribuições real e falsa não se sobreponham, o que geralmente ocorre. Ao invés de minimizar esta divergência, os autores usam a distância de Wassertein, que descreve a distância entre os pontos de uma distribuição à outra, trabalhando assim com uma função de erro que se correlaciona com a qualidade das amostras e permite convergência.

CycleGANs [38] realizam transferência de imagem de domínio cruzado, tomando uma figura de um domínio de entrada Di e transformando-a em outra do domínio de saída Dt. Esta técnica pode ser usada para lidar com uma variedade de problemas à medida que variam os domínios de entrada e saída: desde adicionar efeitos às fotografias até traçar mapas de contorno a partir de imagens de satélites.

Implementar tais arquiteturas no contexto do processamento de imagens pavimenta o caminho

da evolução de redes geradoras, expandindo a fronteira do aprendizado de máquina.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] T. M. Mitchell, *Machine Learning.* McGraw-Hill, 1997.

[2] ——, *The discipline of machine learning.* Carnegie Mellon University, School of Computer Science, Machine Learning Department Pittsburgh, PA, 2006, vol. 9.

[3] S. B. Maind, P. Wankar *et al.*, "Research paper on basic of artificial neural network," *International Journal on Recent and Innovation Trends in Computing and Communication*, vol. 2, no. 1, pp. 96–100, 2014.

[4] S. S. Haykin, S. S. Haykin, S. S. Haykin, and S. S. Haykin, *Neural networks and learning machines.* Pearson Upper Saddle River, 2009, vol. 3.

[5] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of control, signals and systems*, vol. 2, no. 4, pp. 303–314, 1989.

[6] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, p. 436, 2015.

[7] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain." *Psychological review*, vol. 65, no. 6, p. 386, 1958.

[8] Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, "Efficient backprop," in *Neural networks: Tricks of the trade.* Springer, 2012, pp. 9–48.

[9] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *Proc. icml*, vol. 30, no. 1, 2013, p. 3.

[10] S. B. Kotsiantis, I. Zaharakis, and P. Pintelas, "Supervised machine learning: A review of classification techniques," *Emerging artificial intelligence applications in computer engineering*, vol. 160, pp. 3–24, 2007.

[11] K. Janocha and W. M. Czarnecki, "On loss functions for deep neural networks in classification," *arXiv preprint arXiv:1702.05659*, 2017.

[12] Y. LeCun, Y. Bengio *et al.*, "Convolutional networks for images, speech, and time series," *The handbook of brain theory and neural networks*, vol. 3361, no. 10, p. 1995, 1995.

[13] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller, "Striving for simplicity: The all convolutional net," *arXiv preprint arXiv:1412.6806*, 2014.

[14] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, 2014, pp. 2672–2680.

[15] A. Ng, "Generative learning algorithms," 2008.

[16] I. Goodfellow, "Nips 2016 tutorial: Generative adversarial networks," *arXiv preprint arXiv:1701.00160*, 2016.

[17] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," *arXiv preprint arXiv:1511.06434*, 2015.

[18] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[19] A. Krizhevsky, V. Nair, and G. Hinton, "The cifar-10 dataset," *online: http://www. cs. toronto. edu/kriz/cifar. html*, 2014.

[20] M. Lucic, K. Kurach, M. Michalski, S. Gelly, and O. Bousquet, "Are gans created equal? a large-scale study," *arXiv preprint arXiv:1711.10337*, 2017.

[21] M. Mirza and S. Osindero, "Conditional generative adversarial nets," *arXiv preprint arXiv:1411.1784*, 2014.

[22] P. Grnarova, K. Y. Levy, A. Lucchi, T. Hofmann, and A. Krause, "An online learning approach to generative adversarial networks," *arXiv preprint arXiv:1706.03269*, 2017.

[23] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved techniques for training gans," in *Advances in Neural Information Processing Systems*, 2016, pp. 2234–2242.

[24] T. T. Georgiou and A. Lindquist, "Kullback-leibler approximation of spectral density functions," *IEEE Transactions on Information Theory*, vol. 49, no. 11, pp. 2910–2917, 2003.

[25] A. Creswell and A. A. Bharath, "Inverting the generator of a generative adversarial network (ii)," *arXiv preprint arXiv:1802.05701*, 2018.

[26] X. Mao, Q. Li, H. Xie, R. Y. K. Lau, Z. Wang, and S. P. Smolley, "On the effectiveness of least squares generative adversarial networks," *IEEE transactions on pattern analysis and machine intelligence*, 2018.

[27] D. Bang and H. Shim, "Mggan: Solving mode collapse using manifold guided training," *arXiv preprint arXiv:1804.04391*, 2018.

[28] H. Thanh-Tung, T. Tran, and S. Venkatesh, "On catastrophic forgetting and mode collapse in generative adversarial networks," *arXiv preprint arXiv:1807.04015*, 2018.

[29] T. White, "Sampling generative networks," *arXiv preprint arXiv:1609.04468*, 2016.

[30] T. Hazan, G. Papandreou, and D. Tarlow, *Perturbations, Optimization, and Statistics*. MIT Press, 2016.

[31] N. Natarajan, I. S. Dhillon, P. K. Ravikumar, and A. Tewari, "Learning with noisy labels," in *Advances in neural information processing systems*, 2013, pp. 1196–1204.

[32] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[33] D. J. Im, H. Ma, G. Taylor, and K. Branson, "Quantitatively evaluating gans with divergences proposed for training," *arXiv preprint arXiv:1803.01045*, 2018.

[34] E. L. Denton, S. Chintala, R. Fergus *et al.*, "Deep generative image models using a laplacian pyramid of adversarial networks," in *Advances in neural information processing systems*, 2015, pp. 1486–1494.

[35] Y. LeCun, "Yann lecun's answer to "what are some recent and potentially upcoming breakthroughs in deep learning?"," Jul. 2016. [Online]. Available: https://www.quora.com/What-are-some-recent-and-potentially-upcoming-breakthroughs-in-deep-learning/answer/Yann-LeCun

[36] C. Ledig, L. Theis, F. Huszár, J. Caballero, A. Cunningham, A. Acosta, A. P. Aitken, A. Tejani, J. Totz, Z. Wang *et al.*, "Photo-realistic single image super-resolution using a generative adversarial network." in *CVPR*, vol. 2, no. 3, 2017, p. 4.

[37] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein gan," *arXiv preprint arXiv:1701.07875*, 2017.

[38] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," *arXiv preprint*, 2017.