UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
INSTITUTO DE COMPUTAÇÃO
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

EDUARDO TAVARES LEÃO
RODRIGO CARVALHO DE FIGUEIREDO

UM ESTUDO COMPARATIVO SOBRE REDES ADVERSÁRIAS GENERATIVAS

RIO DE JANEIRO
2022

EDUARDO TAVARES LEÃO
RODRIGO CARVALHO DE FIGUEIREDO

UM ESTUDO COMPARATIVO SOBRE REDES ADVERSÁRIAS GENERATIVAS

Trabalho de conclusão de curso de graduação apresentado ao Instituto de Computação da Universidade Federal do Rio de Janeiro como parte dos requisitos para obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Prof. João Carlos Pereira da Silva

RIO DE JANEIRO
2022

Leão, Eduardo Tavares
L437e Um estudo comparativo sobre redes adversárias generativas / Eduardo Tavares Leão, Rodrigo Carvalho de Figueiredo. – 2022.

59 f.

Orientador: João Carlos Pereira da Silva.

Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) - Universidade Federal do Rio de Janeiro, Instituto de Computação, Bacharel em Ciência da Computação, 2022.

1. Inteligência artificial. 2. Aprendizado de máquina. 3. Redes neuais. 4. Modelos generativos. 5. Redes adversárias generativas. I. Figueiredo, Rodrigo Carvalho de. II. Silva, João Carlos Pereira da (Orient). III. Universidade Federal do Rio de Janeiro, Instituto de Computação. IV. Título.

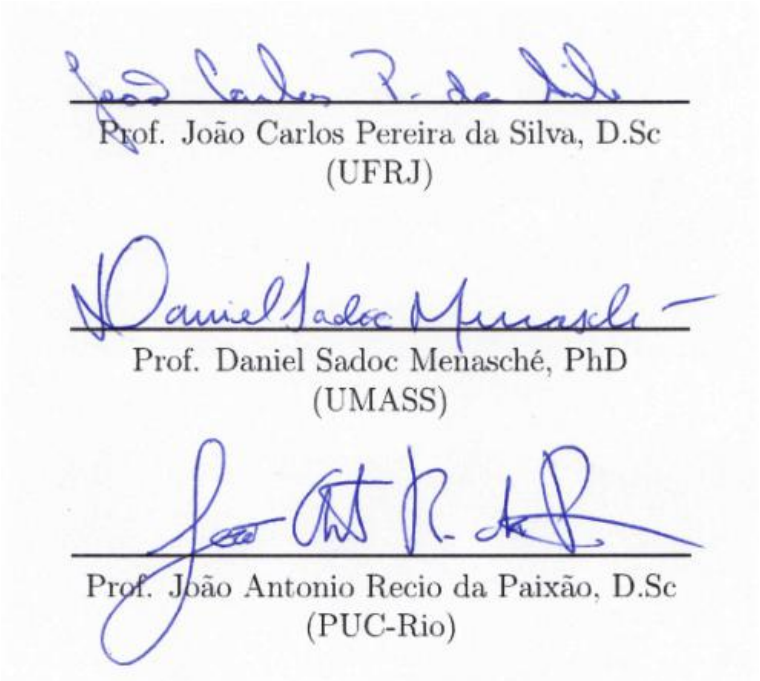
EDUARDO TAVARES LEÃO
RODRIGO CARVALHO DE FIGUEIREDO

UM ESTUDO COMPARATIVO SOBRE REDES ADVERSÁRIAS GENERATIVAS

Trabalho de conclusão de curso de graduação apresentado ao Instituto de Computação da Universidade Federal do Rio de Janeiro como parte dos requisitos para obtenção do grau de Bacharel em Ciência da Computação.

Aprovado em 29 de agosto de 2022

BANCA EXAMINADORA:



Prof. João Carlos Pereira da Silva, D.Sc
(UFRJ)

Prof. Daniel Sadoc Menasché, PhD
(UMASS)

Prof. João Antonio Recio da Paixão, D.Sc
(PUC-Rio)

AGRADECIMENTOS

Agradecemos ao professor João Carlos Pereira da Silva por aceitar o papel de ser nosso orientador durante a realização deste trabalho de pesquisa, e por fazê-lo de maneira atencionsa, paciente e motivadora. Seus conselhos foram extremamente importantes para nós e, sem eles, este trabalho não teria a mesma qualidade.

Aos professores do curso de Bacharelado em Ciência da Computação da Universidade Federal do Rio de Janeiro, pela excelência no ensino e pelas contribuições para a nossa formação profissional.

Aos nossos familiares que nos apoiaram ao longo de toda a nossa trajetória e a todos que contribuíram, direta ou indiretamente, para o nosso aprendizado e para a elaboração deste trabalho.

RESUMO

Redes Adversárias Generativas (*Generative Adversarial Networks* - GANs) são uma classe de arquiteturas de redes neurais artificiais cujo objetivo é a descoberta e aprendizado de padrões em conjuntos de dados, a fim de gerar novos exemplos únicos que sejam indistinguíveis de amostras obtidas desse conjunto. Este trabalho tem como objetivo apresentar esses tipos de redes e, particularmente, três diferentes arquiteturas: GAN, CGAN e DCGAN. É feita uma análise comparativa dos seus desempenhos a partir de um experimento base de geração de imagens com o conjunto de dados MNIST, utilizando tanto métricas quantitativas quanto qualitativas. São então propostas algumas mudanças de arquitetura e hiperparâmetros com o próposito de melhorar o desempenho dessas redes na tarefa em questão, que correspondem a: variações no tamanho do vetor latente, uso de camadas de *dropout* e adição de ruído no modelo discriminador. Por fim, são apresentados os resultados obtidos em experimentos após a adoção de cada modificação, mostrando que, em alguns casos, foram eficazes em aprimorar a qualidade das amostras geradas pelas redes, de acordo com as métricas estabelecidas.

Palavras-chave: inteligência artificial; aprendizado de máquina; redes neurais; modelos generativos; redes adversárias generativas.

ABSTRACT

Generative Adversarial Networks (GANs) are a class of artificial neural network architectures whose goal is to discover and learn patterns in data sets, in order to generate new and unique examples that are indistinguishable from samples drawn from the set. This work aims to introduce these kinds of networks and, particularly, three different architectures: GAN, CGAN and DCGAN. A comparative analysis of their performances is conducted on a basic image generation experiment with the MNIST data set, using both quantitative and qualitative metrics. Some changes are then proposed in architecture and hyperparameters with the intent of improving the performance of these networks on the task at hand, which correspond to: latent vector size variations, use of dropout layers and addition of noise to the discriminator model. Finally, the results obtained from experiments conducted after the adoption of each modification are presented, showing that, in some cases, they were effective at improving the quality of the samples generated by the networks, according to the established metrics.

Keywords: artificial intelligence; machine learning; neural networks; generative models; generative adversarial networks.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

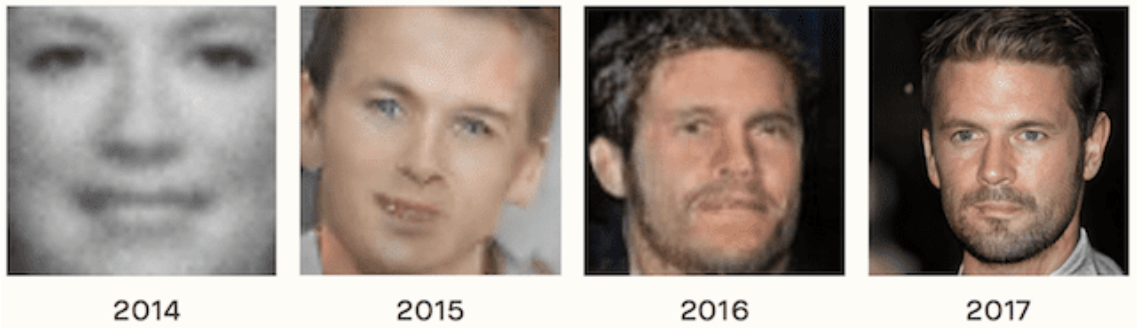
SUMÁRIO

1 INTRODUÇÃO

As redes adversárias generativas (*Generative Adversarial Networks* - GANs) foram introduzidas em 2014 por (GOODFELLOW et al., 2014). Sua arquitetura ganhou diversos adeptos e, nos últimos anos, uma grande variedade de artigos e arquiteturas foram propostas baseadas nesse projeto original. Essas redes são classificadas como modelos generativos (FOSTER, 2019) e, portanto, têm como objetivo descobrir e aprender automaticamente padrões de um determinado conjunto de dados, sendo capazes de gerar novos exemplos únicos que sejam indistinguíveis de amostras obtidas desse conjunto. Nesse tipo de tarefa, as GANs demonstraram rapidamente seu grande potencial.

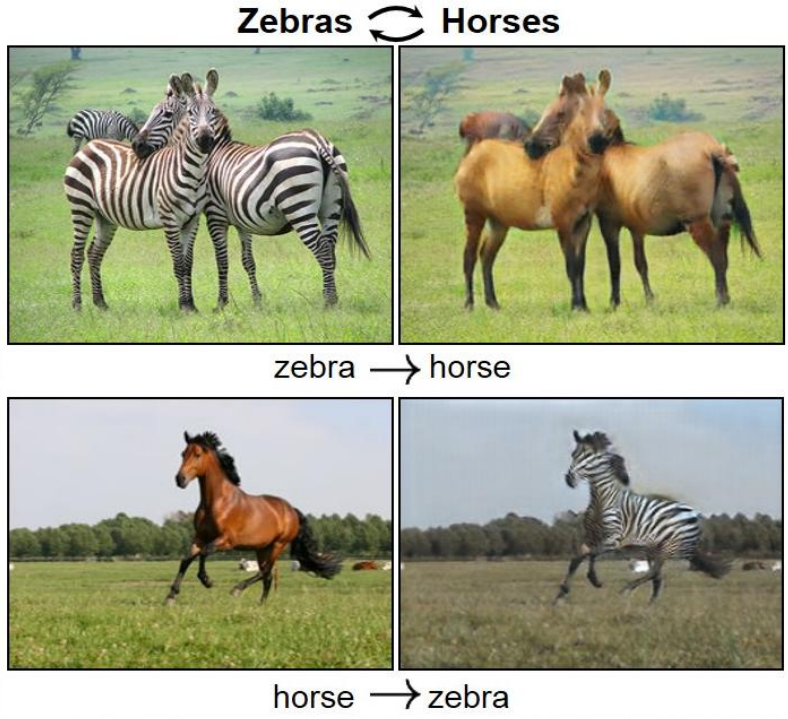
Desde a sua criação, importantes avanços e descobertas foram feitos, o que permitiu a evolução acelerada desse tipo de rede no que tange ao seu desempenho e qualidade de resultados gerados, especialmente em tarefas que envolvem imagens. A Figura 1 exemplifica os avanços recentes na tarefa de geração de imagens realistas de rostos. Por conta desse progresso, as GANs hoje podem ser utilizadas para diversas aplicações, como o aumento de dados (*data augmentation*) (TANAKA; ARANHA, 2019), (MARIANI et al., 2018); o aumento de resolução de imagens (LEDIG et al., 2016), (BIN et al., 2017); geração de imagens realistas (KARRAS et al., 2017), (BROCK; DONAHUE; SIMONYAN, 2018); tradução de textos em imagens (ZHANG et al., 2016); tradução de imagens para imagens (ISOLA et al., 2016), (ZHU et al., 2017), entre outras.

Figura 1 – Evolução das GANs ao longo dos anos



Fonte: (BRUNDAGE et al., 2018). Exemplo do progresso da capacidade das GANs nos anos recentes, na tarefa de geração de imagens realistas de rostos.

Figura 2 – Tradução imagem para imagem



Fonte: (ZHU et al., 2017). Exemplo de tarefa de tradução imagem para imagem com a rede CycleGAN.

Diante dessa quantidade extensa de aplicações e modelos desenvolvidos, decidimos por trabalhar em um simples problema de geração de imagens de dígitos escritos à mão, utilizando o conjunto de dados MNIST (DENG, 2012). Não obstante, abordamos essa questão sob o prisma de três arquiteturas distintas: GAN, CGAN (*Conditional* GAN) e DCGAN (*Deep Convolutional* GAN). Assim, além de entender o comportamento de tais redes, nosso objetivo foi comparar a performance desses modelos na tarefa escolhida, fazendo uso de métricas quantitativas e qualitativas. Tendo em vista obter ganhos de desempenho, propomos também algumas modificações nas arquiteturas e hiperparâmetros utilizados inicialmente e mensuramos o impacto dessas mudanças no resultado final produzido por cada rede estudada.

O projeto foi estruturado da seguinte forma: no capítulo 2 apresentamos todos os conceitos básicos a respeito de redes neurais, desde o *perceptron* mais simples até as arquiteturas de GAN trabalhadas. A seguir, no capítulo 3 são relatados os experimentos base, assim como as métricas e parâmetros utilizados para tais experimentos. No capítulo 4, denotamos as modificações abordadas com o intuito de melhorar o desempenho das redes. Finalmente, no capítulo 5, explicamos nossas conclusões do projeto, assim como algumas dificuldades encontradas e citamos os possíveis trabalhos futuros.

O código produzido para esse trabalho está disponível para visualização e download em https://github.com/rodcf/PyTorch-GAN.

2 REDES NEURAIS

O objetivo deste capítulo é prover o conhecimento prévio de redes neurais que o leitor precisa para ter compreensão plena do escopo desse projeto. Para tal, iniciaremos trazendo os conceitos chave de redes neurais. Logo após, abordaremos as redes neurais convolucionais, que são muito utilizadas para aprendizado de máquina relacionado a imagens. Por fim, seguiremos com as redes específicas do projeto em questão: as redes generativas adversárias e suas extensões.

2.1 A REDE NEURAL

Atualmente, redes neurais (GURNEY, 1997) são amplamente utilizadas para tarefas de classificação de informação (DREISEITL; OHNO-MACHADO, 2002), agrupamento de dados (DU, 2010), previsão (EL_JERJAWI; ABU-NASER, 2018) e todo um conjunto extenso de problemas (ABIODUN et al., 2018). Elas são sistemas de computação inspirados em redes neurais biológicas que constituem o cérebro. Para entender melhor o que isto significa, vamos olhar primeiro para a neurobiologia básica. O cérebro consiste de células nervosas ou neurônios. Cada neurônio se comunica por meio de sinais elétricos, no qual cada conexão é mediada por junções chamadas de sinapses. Cada neurônio recebe, tipicamente, milhares de conexões de outros neurônios e, portanto, recebe uma multitude de sinais elétricos. Aqui, eles são integrados ou somados de tal forma que se o sinal resultante exceder algum limite (*threshold*), então o neurônio irá gerar um impulso de resposta que é transmitido aos outros neurônios.

Não obstante, um forte poder de uma rede neural, seja biológica ou artificial, está em sua capacidade de aprendizado. Tal aprendizado é alcançado por meio de experiências que, no caso biológico, geram novas conexões entre neurônios, ou seja, novas sinapses. Já para as redes neurais artificiais, a estrutura e conexões entre nós (neurônios) é previamente definida, assim como uma função objetivo, ou de perda, a ser minimizada, e o processo de aprendizagem comum é o de utilizar um conjunto de treinamento (experiências) como entrada para a rede de modo que os pesos (sinapses) são alterados constantemente até convergirem para um ponto ótimo. As definições de peso, função objetivo e conjunto de treinamento são apresentadas em conjunto com a rede neural mais básica — o Perceptron.

2.1.1 Perceptron

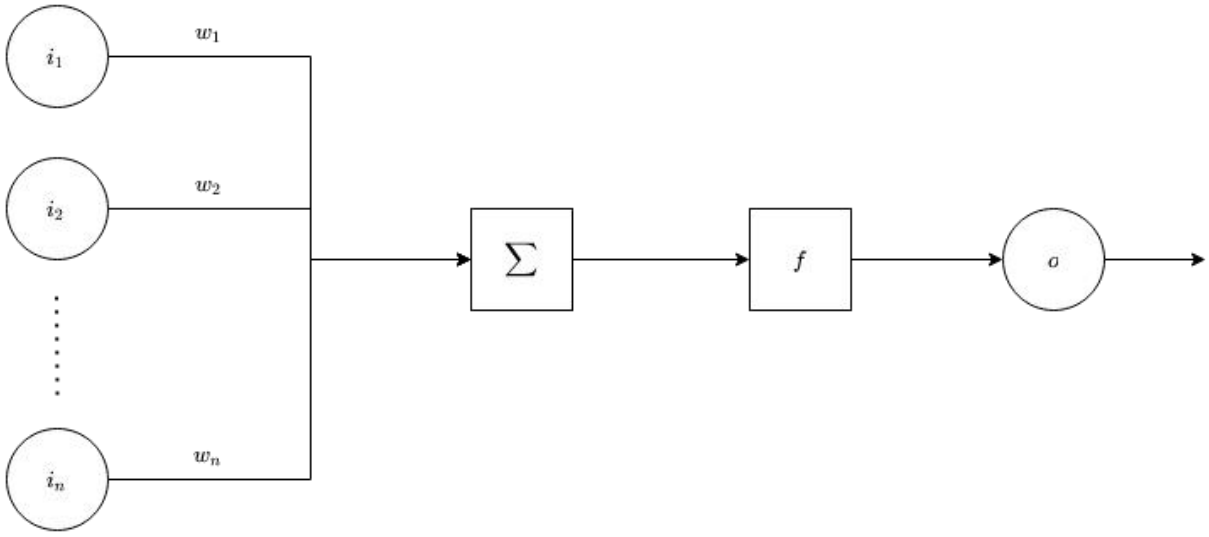
O formato base e mais simples para redes neurais é o Perceptron (SHALEV-SHWARTZ, 2008). Ele funciona da seguinte forma: um sinal, como um vetor, é recebido pelos nós de entrada de tal maneira que cada elemento do vetor é associado a um único e distinto nó de entrada. Então, o nó de entrada dispara para o nó de saída o valor recebido multiplicado

pelo seu peso associado, no qual o peso é apenas um número. Após tal propagação, é feito um somatório entre os valores dos nós de entrada multiplicados pelos pesos. Por fim, o resultado passa pela função de ativação e é devolvido no nó de saída.

Formalmente, o perceptron (Figura 3) é definido assim: sejam os nós de entrada i_k e os pesos w_k para $k \in \{1, 2, 3, ..., n\}$, a função de ativação f e o nó de saída o. O valor a ser retornado em o é dado por:

$$o = f(\sum_{k=1}^{n} i_k w_k) \tag{2.1}$$

Figura 3 – Perceptron



Em uma tarefa de classificação, por exemplo, o perceptron é utilizado para separar linearmente o espaço entre as classes. Isso é feito por meio de uma atualização iterativa em w até que a rede produza as saídas esperadas. O processo de atualização do vetor de pesos w é chamado de aprendizado ou treinamento. Para tal, precisamos das seguintes definições:

- Conjunto de treinamento $D = \{(x_1, d_1), ..., (x_s, d_s)\}$, onde:
  - s é a quantidade de dados ou amostras.
  - x_j é o vetor de entrada n-dimensional, para $j \in \{1, ..., s\}$.
  - d_j é a saída esperada para a entrada x_j, para $j \in \{1, ..., s\}$.
- Taxa de aprendizagem α no intervalo $[0, 1]$ que configura a volatilidade das mudanças nos pesos. Valores mais altos configuram mudanças mais abruptas.

- Função perda ou objetivo l que configura o valor a ser minimizado. Para o caso mais simples, pode ser definida como

$$l = |d_j - o(x_j)| \tag{2.2}$$

  onde o é dado pela equação 2.1.

- $w(t)$ é o vetor de pesos no tempo t.

A partir dessas definições podemos apresentar o seguinte algoritmo de aprendizado:

1. Inicializar os pesos (w) com valores aleatórios ou nulos.
2. Para cada exemplo j em D, faça:
   - Para cada nó de entrada $0 \leq i \leq n$:
     - Atualize os pesos com:

$$w_i(t+1) = w_i(t) + \alpha * l * x_{j,i} \tag{2.3}$$

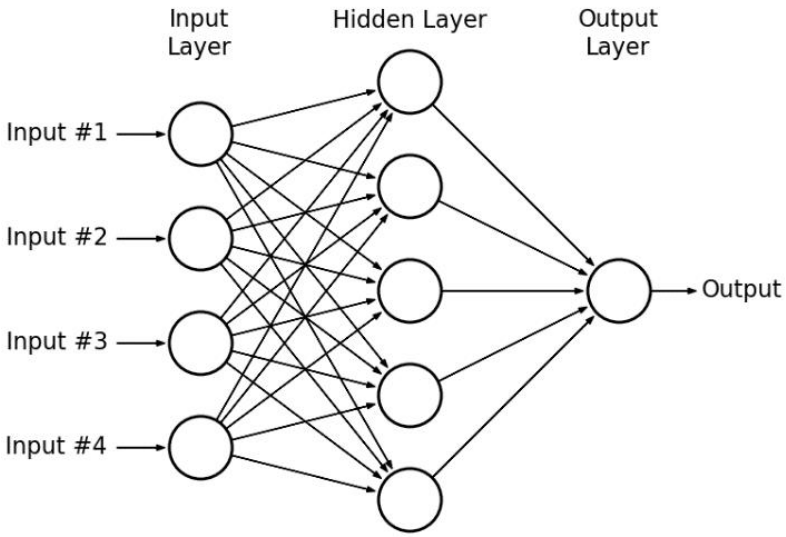
       onde $x_{j,i}$ é o elemento i do vetor de entrada x_j.

2.1.2 Perceptron multi-camadas (*Multilayer Perceptron* - MLP)

Uma extensão natural do perceptron é o MLP (perceptron multi-camadas) (ABIRAMI; CHITRA, 2020) que adiciona novas camadas de nós à rede tornando-a capaz de realizar operações mais complexas, já que ela pode distinguir dados que não são linearmente separáveis. Essas novas camadas são chamadas de intermediárias ou escondidas e são a parte em que reside o verdadeiro poder computacional do MLP.

Assim como no perceptron, a camada de entrada recebe os dados a serem processados e a camada de saída realiza a tarefa requerida como predição ou classificação. A Figura 4 ilustra a arquitetura da rede.

Figura 4 – MLP



Fonte: (HASSAN et al., 2015). *Input* #1 #2, #3 e #4 são os sinais de entrada, *Input Layer* é a camada de entrada, *Hidden Layer* é a camada intermediária, *Output Layer* é a camada de saída e *Output* é a saída gerada pela rede.

A forma mais comum de aprendizado para o MLP é o algoritmo chamado de *backpropagation*. A intuição é semelhante à do perceptron simples: compara-se a saída da rede com a saída esperada em uma função de perda ou erro e a atualização dos pesos é derivada da minimização desse erro. Assim, o erro é propagado da camada de saída até a camada de entrada possibilitando a atualização dos pesos de cada camada. Após a atualização dos pesos, o erro no nó de saída é calculado mais uma vez e o ciclo se repete até que a rede alcance um erro aceitável, isto é, pequeno.

Para mais informações a respeito de perceptron, MLP, *backpropagation* e o processo de aprendizagem, ver (POPESCU et al., 2009).

2.2 REDE NEURAL CONVOLUCIONAL

Dentro da área de aprendizado de máquina há um campo chamado *Deep Learning* ou aprendizado profundo, que consiste em arquiteturas de redes neurais mais extensas, nas quais há uma grande quantidade de camadas intermediárias. Uma das técnicas de *deep learning* mais utilizadas consiste nas redes neurais convolucionais (*convolutional neural network* - CNN), que são extensamente utilizadas em tarefas que envolvem imagens ou vídeos (SULTANA; SUFIAN; DUTTA, 2018), (FRIZZI et al., 2016), (SHI et al., 2016).

2.2.1 Arquitetura

A arquitetura de redes convolucionais pode ser observada na Figura 5 e tem como padrão as seguintes camadas:

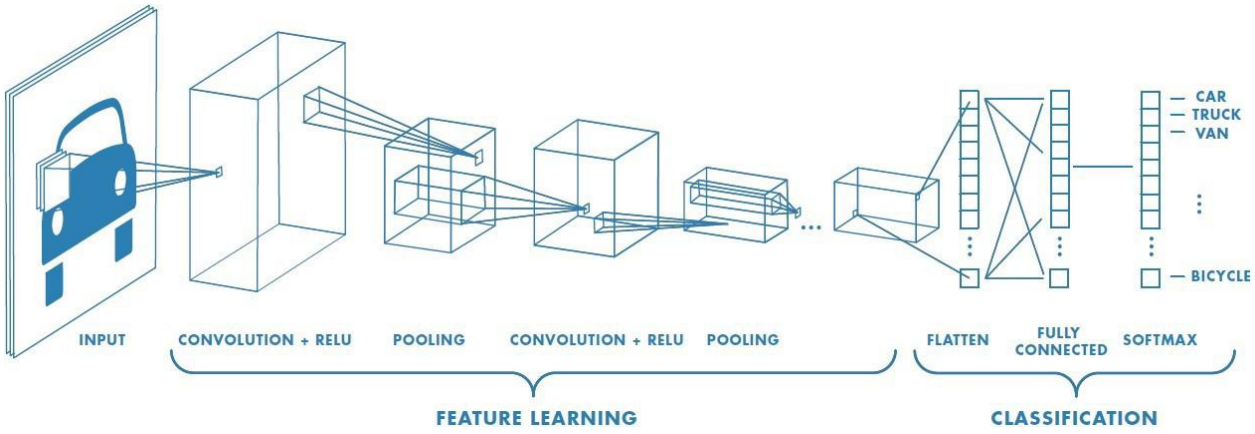
Camada convolucional Busca encontrar padrões (*features*) em sub-regiões da imagem de entrada através de filtros (*kernels*). Por exemplo, podemos ter um filtro que realiza a detecção de arestas da imagem na vertical, outro que detecta arestas na horizontal e outro que mapeia as áreas com maior variação de frequência. Na prática, porém, os filtros são determinados pelo próprio processo de aprendizagem, o que torna difícil compreender qual *feature* está sendo extraída com aquele filtro.

Camada de *pooling* Camada focada na minimização da quantidade de parâmetros de entrada sem perda de informações importantes. Útil para diminuir o uso de recursos computacionais.

Camada totalmente conectada (*fully connected layer* - FC) Rede neural padrão que recebe como entrada a saída das camadas anteriores em um vetor.

As camadas de convolução e *pooling* podem se repetir antes do resultado ser passado para a camada totalmente conectada. Intuitivamente, isso significa que a cada camada de convolução que acrescentamos, extraímos *features* ou padrões de mais alto nível na imagem.

Figura 5 – Arquitetura de uma CNN



Fonte: https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939, acessado em 08/03/22.

Uma imagem pode ser definida de diversas formas. Ao tratarmos de representações *raster*, formato mais comum, temos que a imagem é constituída por uma matriz em que cada elemento é chamado de pixel e pode receber um valor de 0 a 255 que, por sua vez, irá representar a cor daquele pixel. No caso de imagens em preto e branco, o 0 representa o preto e o 255 o branco. Qualquer valor intermediário assumirá um tom de cinza mais escuro, caso esteja mais próximo do 0, ou mais claro, caso esteja mais próximo do 255.

Para a representação de cores, o formato amplamente utilizado por computadores é o RGB (*Red*, *Green* e *Blue*) que nada mais é do que três matrizes R, G e B de mesmo

tamanho em que cada elemento varia de 0 a 255, nas quais 0 é equivalente a preto e os outros valores assumem tons de vermelho, verde e azul, respectivamente. Dessa forma, as matrizes são sobrepostas e o pixel final é representado pela composição dessas três cores.

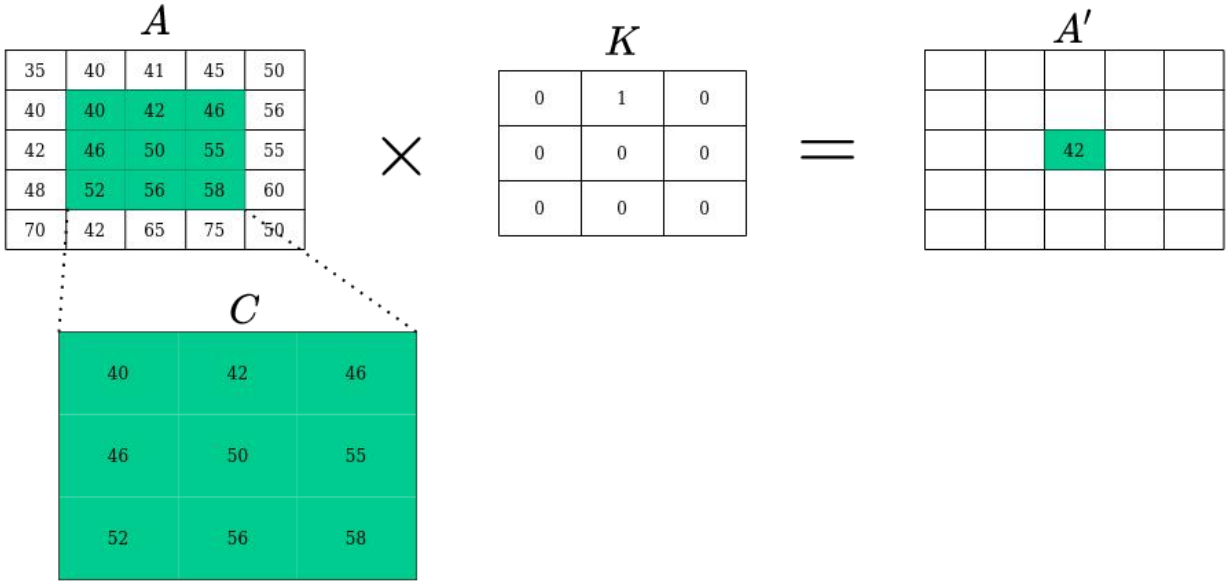
Convolução, na computação gráfica, consiste em uma matriz sendo tratada por outra chamada de filtro ou *kernel*. Seja a imagem A de tamanho $r \times s$, o pixel analisado A_{ij}, C a matriz quadrada $m \times m$, tal que $m \leq r, s$, centrada em A_{ij} e o *kernel* K, também $m \times m$. A fórmula para gerar o pixel correspondente A'_{ij} é dada por:

$$A'_{ij} = \sum_{1 \leq a,b \leq m} C_{ab} K_{ab} \tag{2.4}$$

A Figura 6 ilustra uma etapa de um processo convolucional referente ao elemento central da matriz. Neste caso, como apenas $K_{12} \neq 0$, temos que:

$$A'_{33} = C_{12} * K_{12} = 42 * 1 = 42 \tag{2.5}$$

Figura 6 – Convolução para elemento central



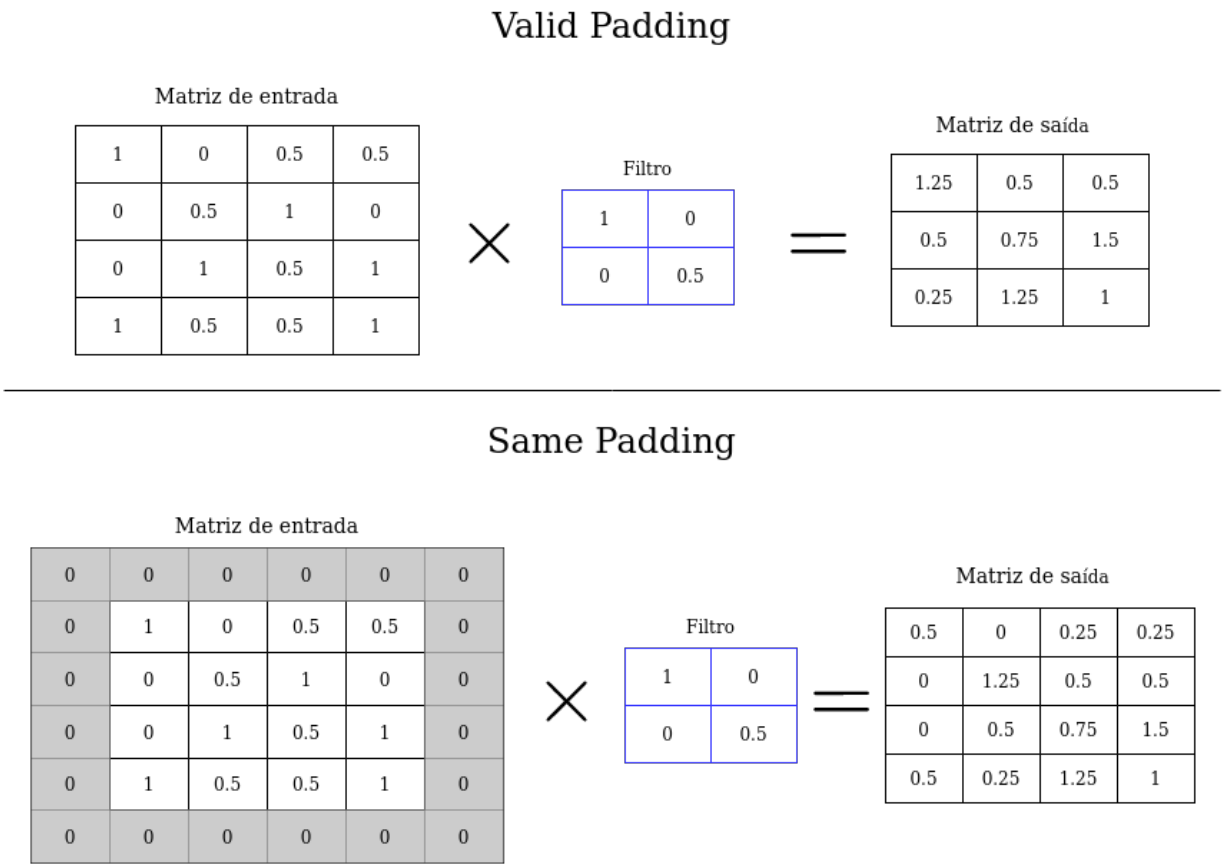
Esse processo convolucional, na verdade, é feito em cada um dos elementos das matrizes, i.e., os pixels da imagem, para gerar uma nova matriz. Após esse processo em cada um dos pixels da imagem, é incrementada uma camada chamada de ReLU (*rectified linear unit*) que aplica a função $f(x) = max(0, x)$ e remove os valores negativos para introduzir não linearidades à rede. Para entender mais sobre a ReLU, ver (AGARAP, 2018).

Para o caso dos pixels na borda da imagem original, possíveis soluções podem ser aplicadas (ver Figura 7):

Valid Padding Os pixels de borda são ignorados, o que ocasiona uma redução de dimensionalidade.

Same Padding A imagem $N \times N$ é aumentada em uma imagem $(N + c) \times (N + c)$ em que c é uma constante arbitrária que determinará se a nova matriz aumentará a dimensionalidade ou manterá a estrutura original da imagem. Os elementos das novas dimensões são preenchidos com o valor "0".

Figura 7 – *Valid padding* x *same padding*



Vimos que nas redes tradicionais temos um sinal de entrada sendo multiplicado por pesos e passado para uma função de ativação para a próxima camada e o objetivo da rede é encontrar pesos que minimizem uma função de perda. Já nas redes convolucionais temos um sinal de entrada sendo multiplicado por filtros que passam por uma função de ativação (ReLU) para a próxima camada e os filtros serão escolhidos de forma a minimizar a função de perda.

2.2.2 Camada de *Pooling*

A camada de *pooling* é responsável por reduzir a quantidade de parâmetros de entrada sem perda de informação relevante. Semelhante à aplicação de um filtro, o *pooling* irá

percorrer a matriz em blocos. Contudo, não há uma filtragem via *kernel*, mas uma função $f(C)$ que determina o valor dos novos pixels. A função mais utilizada é a de extrair o valor máximo entre os elementos de C. Por isso, muitos chamam essa etapa de *max-pooling*, considerando que $f(C) = max(C)$.

Não obstante, nessa etapa, há dois parâmetros fundamentalmente importantes que irão definir a dimensão da matriz de saída:

Tamanho do *kernel* Dimensões da submatriz C sobre a qual será aplicada a função f.

Tamanho do passo ou *stride* Tupla contendo de quantos em quantos elementos deve-se percorrer a matrix no eixo horizontal e no eixo vertical. Para entender melhor esse parâmetro, compare as iterações nas Figuras 8 e 9.

Como exemplo, temos a Figura 8. Os parâmetros dados são um tamanho de *kernel* (2,2) e um tamanho de passo (2,2). Como pode-se observar, C corresponde às submatrizes coloridas da matriz de entrada a cada iteração. Da mesma forma, $f(C)$ é representado pelo elemento colorido na matriz de saída. Já na Figura 9, é feito o mesmo procedimento com a variação de que o tamanho do passo no eixo vertical é de uma unidade, fazendo com que a matriz de saída tenha dimensão 3×2 em vez de 2×2.

Figura 8 – Pooling

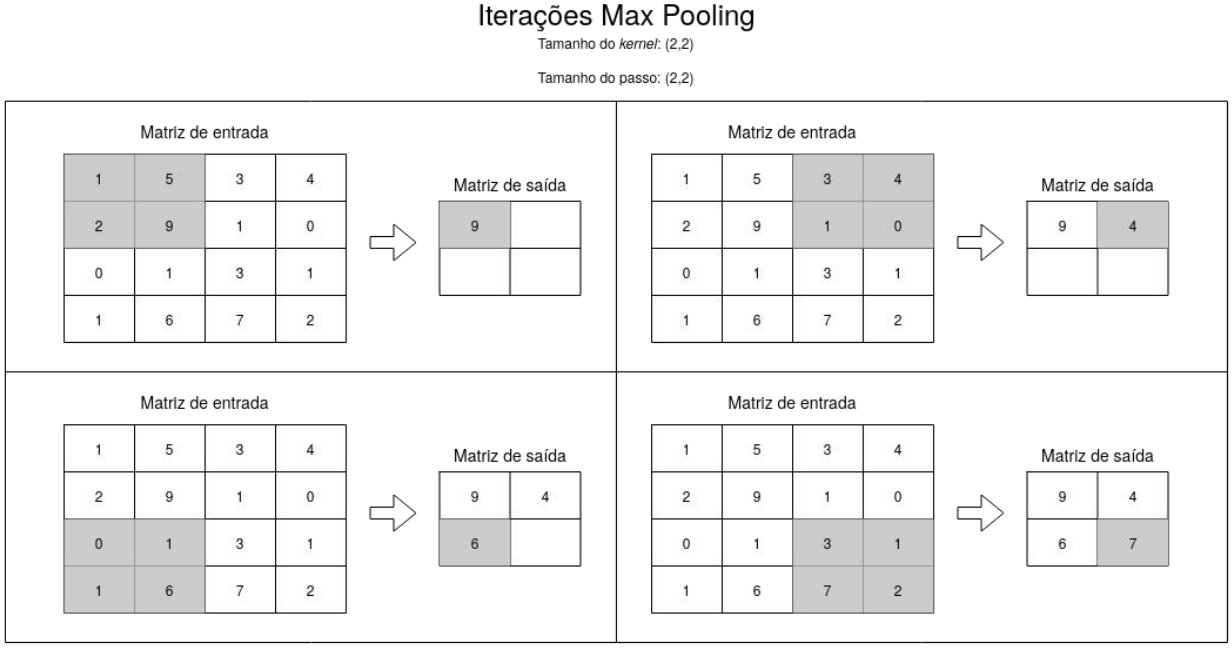
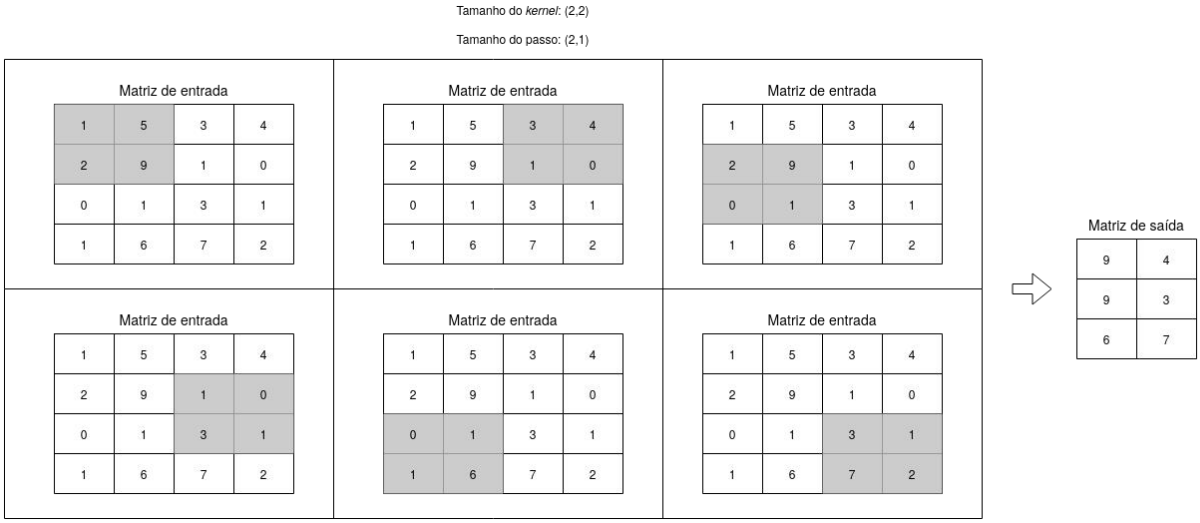


Figura 9 – Pooling com *stride* diferente



A combinação da convolução com o *pooling* se torna especialmente útil para o caso de tratamento de imagens em redes neurais, posto que, para utilizarmos um MLP necessitaríamos transformar a imagem em um vetor (processo de *flattening*). Tal processo, porém, em muitos casos, pode gerar vetores enormes, o que, por sua vez, requeriria da rede um poder computacional absurdamente vasto.

2.2.3 Camada totalmente conectada (*Fully connected layer* - FC)

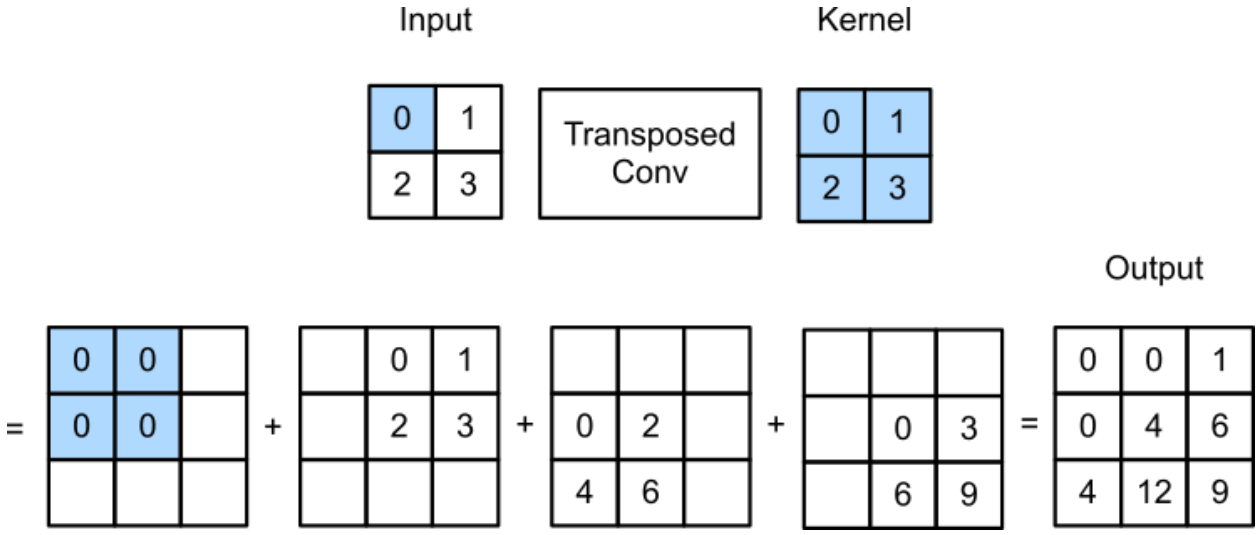
Correspondendo a uma das etapas finais da rede covolucionada, temos uma camada totalmente conectada. Nela, a fórmula do *perceptron* é aplicada em cada um dos neurônios (elementos do vetor) e gera a quantidade necessária de nós de saída. Para essa etapa, também pode-se usar o MLP para maior complexidade.

2.2.4 Convolução transposta

Outro campo dentro de visão computacional estuda o uso de *upsampling*, que significa aumentar a resolução de uma imagem ou matriz. Há diversas abordagens como interpolação bilinear e *nearest neighbors* (MAZZINI, 2018), além da convolução transposta. Aplicando os conceitos previamente mencionados sobre convolução, é possível compreender a convolução transposta a partir do exemplo a seguir.

Na Figura 10, considere a matriz M (*input*) de dimensão 2×2 com 4 elementos que precisa ser ampliada (*upsampled*) para uma matriz U (*output*) de dimensão 3×3 e o *kernel* 2×2 definido por K. Em seguida, para cada elemento $m_{i,j}$ de M, tal que $1 \leq i, j \leq 2$, é feita a multiplicação escalar $m_{i,j} * K$ e posicionado em uma matriz temporária 3×3 como mostra a figura. Por fim, todas as matrizes temporárias são somadas para gerar o resultado final da convolução transposta.

Figura 10 – Operação básica de convolução transposta

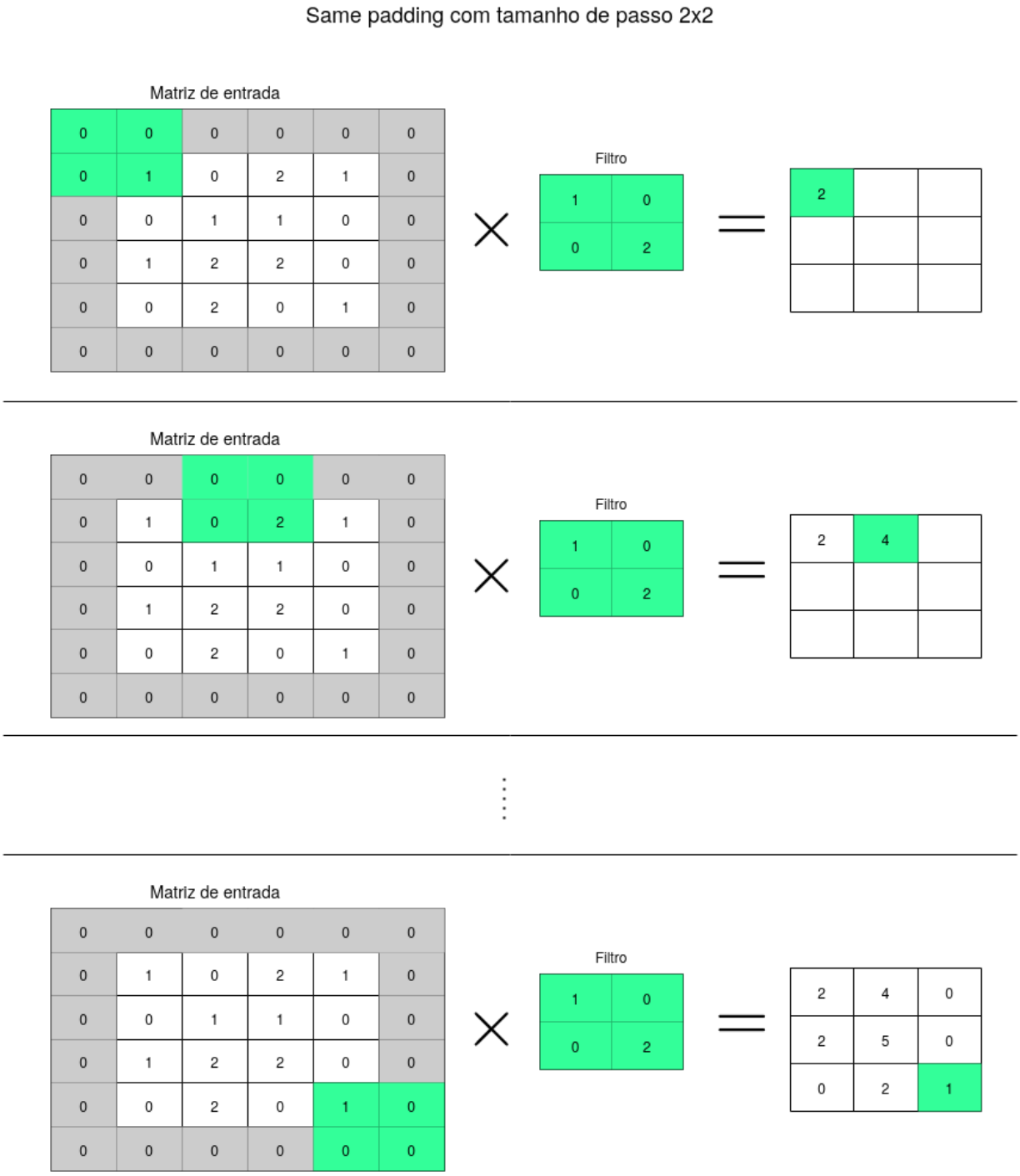


Fonte: (ZHANG et al., 2021)

2.2.5 *Strided Convolution*

Um conceito que pode ser aplicado para suprimir a etapa de *pooling* em redes convolucionais é o conceito de *strided convolution*. Ele consiste simplesmente em transportar o conceito de passo ou *stride*, presente na camada de *pooling*, para a camada de convolução. Assim, para valores de passo maiores que 1, a convolução não será aplicada em todos os pixels da imagem original. Pode-se ver na Figura 11 um tamanho de *stride* de (2,2) aplicado junto ao contexto de *same padding*.

Figura 11 – Operação de *strided convolution*



2.3 REDE ADVERSÁRIA GENERATIVA (*GENERATIVE ADVERSARIAL NETWORK* - GAN)

Além dos modelos discrimitativos, aqueles focados em tarefas de regressão e classificação, há uma área de estudo em aprendizado de máquina a respeito de modelagem generativa (FOSTER, 2019). Modelagem generativa envolve o aprendizado automático

de regularidades ou padrões no dado de entrada de forma que o modelo seja capaz de gerar novos exemplos que poderiam, de maneira plausível, fazer parte do conjunto inicial de dados.

Podemos ver a utilidade de modelos generativos para casos em que é necessário ter novos dados de acordo com um padrão específico. Por exemplo, quando há poucas imagens em um conjunto de treinamento de um modelo, o que pode trazer resultados ruins em um processo de aprendizado. Assim, em 2014, surge um novo tipo de arquitetura de redes neurais: as GANs ou redes adversárias generativas (GOODFELLOW et al., 2014).

As redes adversárias generativas funcionam por meio de um treinamento simultâneo de dois modelos de redes neural (MLPs). Há o modelo generativo (gerador) G que captura a distribuição dos dados e o modelo discriminativo (discriminador) D que diz se uma amostra veio do conjunto de treinamento em vez da saída de G.

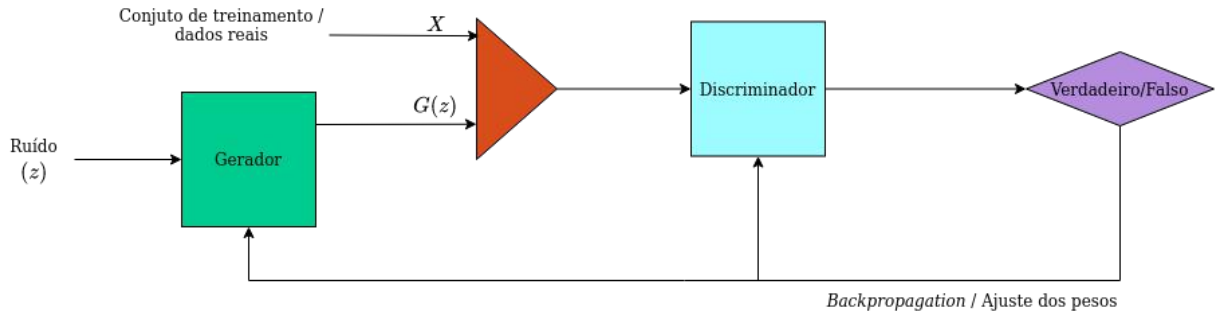
O processo de treino para G é o de maximizar a probabilidade de D cometer um erro. Isso corresponde a um *minimax* de dois jogadores em que em que há uma solução única na qual G recupera a distribuição de dados de treino enquanto D não sabe mais diferenciar entre uma amostra proveniente do conjunto de treinamento e de uma proveniente de G.

Assim sendo, para tarefas de visão computacional, podemos considerar que o objetivo final de uma GAN é obter um modelo generativo que gera imagens realistas o suficiente de modo a enganar com sucesso o melhor modelo discriminativo obtido durante as etapas de treinamento.

2.3.1 Arquitetura

Como ilustrado na Figura 12, a rede geradora G recebe um valor de ruído, que é um vetor aleatório z, e gera a saída $G(z)$, a imagem gerada. Seja X o conjunto de treinamento contendo imagens reais. A rede discriminativa D pode receber tanto um dado pertencente a X quanto a $G(z)$ e a saída será a classificação do dado em verdadeiro ou falso.

Figura 12 – Arquitetura da GAN



Assim, enquanto D quer maximizar sua probabilidade de acerto, G quer minimizar a probabilidade de acerto de D quando a imagem for falsa. Isto é representado por:

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}[logD(x)] + \mathbb{E}_{z\sim p_z(z)}[log(1-D(G(z)))] \quad (2.6)$$

onde x é o dado de entrada, z é o ruído, $\mathbb{E}$ é o valor esperado, $x \sim p_{data}(x)$ significa que x assume a distribuição de probabilidade do conjunto de treinamento e $z \sim p_z(z)$ que z assume a distribuição dos ruídos.

2.3.2 Extensões

Desde o surgimento das GANs houve diversas extensões e arquiteturas novas que fazem uso desse conceito. Como arquiteturas a serem trabalhadas nesse projeto, ainda temos a CGAN (GAN condicional) (MIRZA; OSINDERO, 2014), proposta como parte de trabalhos futuros na tese inicial da GAN, e a DCGAN (GAN convolucional profunda) (RADFORD; METZ; CHINTALA, 2016), que se utiliza de conceitos de redes convolucionais citados na seção 2.2.

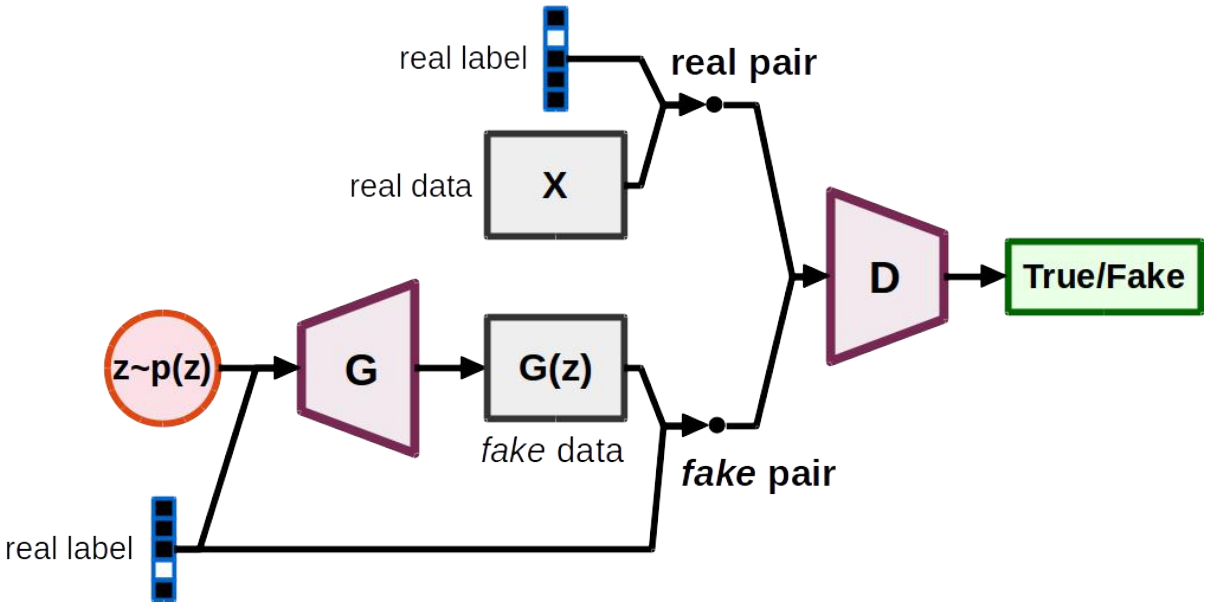
2.4 GAN CONDICIONAL (*CONDITIONAL GAN* - CGAN)

As GANs condicionais são extensões das GANs tradicionais que condicionam ambos os modelos discriminativo D e generativo G em alguma informação extra y. Essa nova variável y pode ser qualquer tipo de informação auxiliar, como rótulos (labels) de classe de imagens, tags ou anotações a cerca de imagens, outras imagens (como usado em tarefas de *image-to-image translation*) ou dados de outras modalidades. Ambos os modelos passam a receber essa nova informação y em conjunto com suas entradas usuais durante o processo de treinamento e são condicionados a ela, de forma que suas saídas passam a ser, respectivamente, $D(x|y)$ e $G(z|y)$, onde $|y$ significa que a variável precedente é condicionada em y.

Esse condicionamento permite um maior controle sobre a saída do modelo generativo, tornando possível limitar os resultados gerados por ele a um domínio mais específico. Isso faz com que seja viável, por exemplo, a geração de imagens de um tipo ou classe desejados, o que não é possível com uma GAN tradicional. Essa limitação, por sua vez, tende a aumentar a qualidade e fidelidade das imagens geradas. Além disso, modelos de GANs condicionais tendem a apresentar convergência mais rápida do que os não condicionais.

A Figura 13 ilustra a arquitetura de uma CGAN, onde tanto D como G recebem uma mesma informação adicional y simbolizada pelo vetor denotado de *real label*, que representa um rótulo de classe para a imagem.

Figura 13 – Arquitetura da CGAN



Fonte:(DONG; YANG, 2019). Apesar de estar representado duas vezes na figura, *real label* representa o mesmo elemento, que integra ambos os pares *real pair* e *fake pair*.

2.5 GAN CONVOLUCIONAL PROFUNDA (*DEEP CONVOLUTIONAL GAN* - DCGAN)

As GANs convolucionais profundas, ou DCGANs, são também extensões diretas das GANs tradicionais. Consistem na adoção de redes neurais convolucionais (CNNs) nos modelos generativo e discriminativo, substituindo os tradicionais perceptrons multi-camadas (MLPs).

Nesse tipo de GAN, o modelo generativo consiste principalmente de camadas de convolução transposta, que basicamente fazem o inverso de uma operação de convolução. Já o modelo discriminativo é um classificador binário que consiste de várias camadas de convolução e que apresenta como saída uma probabilidade de a imagem ser verdadeira ou falsa.

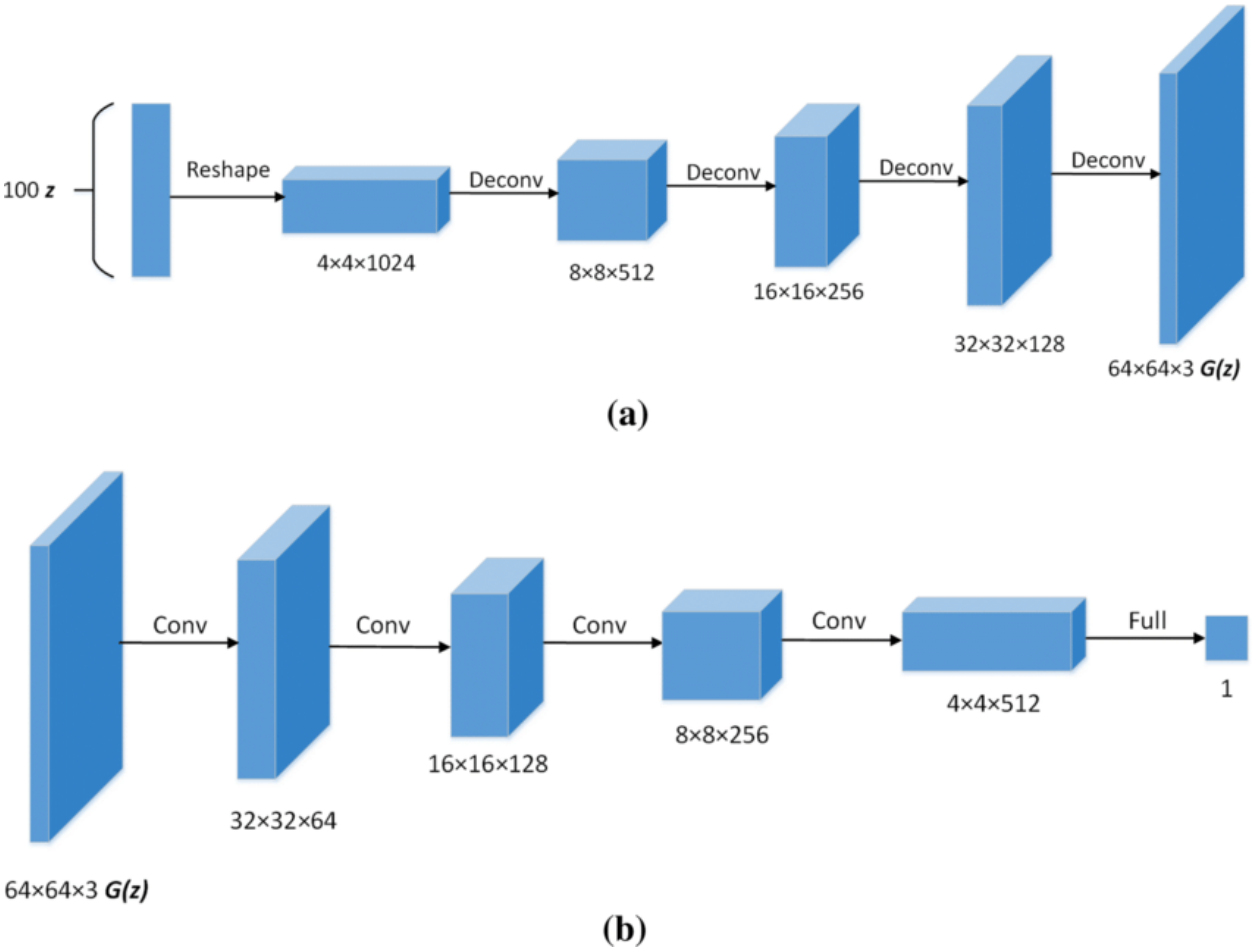
As DCGANs apresentam melhor performance e uma maior estabilidade durante o treinamento dos modelos, quando comparadas às GANs tradicionais. A Figura 14 ilustra a arquitetura de uma DCGAN, onde (a) representa o modelo generativo G e (b) o modelo discriminativo D.

Utilizando os valores de cada dimensão citada no artigo, temos que, assim como em uma GAN tradicional, G recebe o vetor de ruído z de dimensão 100, também chamado de vetor latente. Esse vetor é redimensionado para $4 \times 4 \times 1024$, onde esses valores representam largura, altura e quantidade de canais, respectivamente. As próximas etapas constituem a convolução transposta até que tenhamos uma imagem RGB (3 canais de

cor) de resolução 64 de largura por 64 de altura.

Já a arquitetura do modelo discriminativo é uma rede convolucional que recebe a imagem RGB de dimensão 64×64. As diferenças para uma CNN tradicional residem no fato dela usar *strided convolution* em vez de *pooling*, o que permite à rede a definição da própria função de redução espacial; as camadas intermediárias de um MLP na camada FC são removidas; e todas as camadas utilizam outra função de ativação chamada LeakyReLU (XU et al., 2020), pois foi verificado através de testes que o modelo aprende mais rápido com essas mudanças.

Figura 14 – Arquitetura da DCGAN



Fonte:(LI; SONG; ZHANG, 2020). (a) representa o modelo generativo e (b) representa o modelo discriminativo

A diferença da LeakyReLU para a ReLU é que ela aceita valores negativos multiplicados por uma inclinação de reta α, como ilustrado nas Figuras 15 e 16 . Assim, ela é definida como:

$$LeakyReLU(x, \alpha) = \begin{cases} x & : \ x \geq 0 \\ \alpha * x & : \ x < 0 \end{cases} \tag{2.7}$$

Figura 15 – ReLU

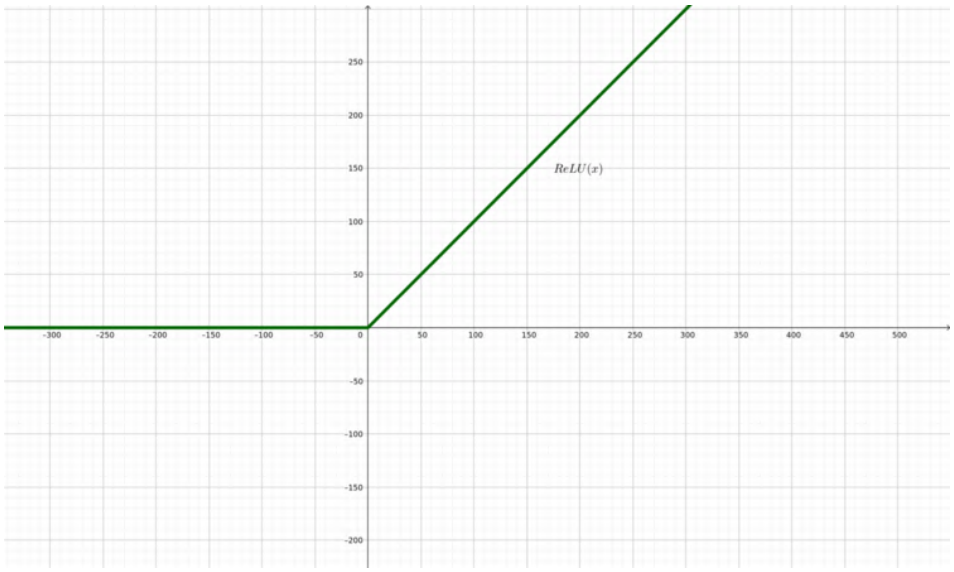


Gráfico para função de ativação ReLU

Figura 16 – LeakyReLU

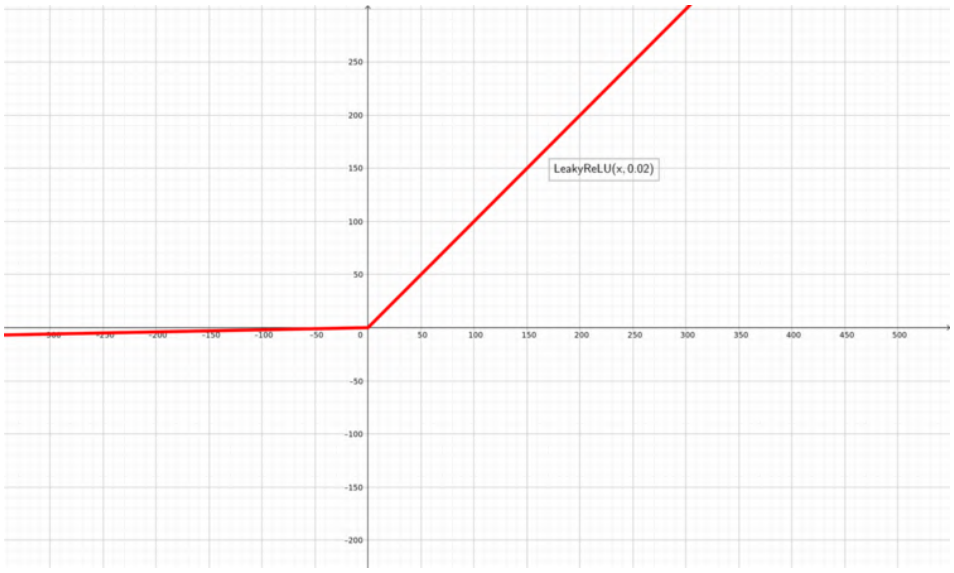


Gráfico para função de ativação LeakyReLU com inclinação de reta igual a 0.02

2.6 CONCLUSÃO

Neste capítulo, discorremos sobre o que é uma rede neural e suas representações mais básicas: o *perceptron* e o MLP. Em seguida, apresentamos a arquitetura de redes convolucionais e sua importância para aplicações com imagens para, enfim, apresentarmos as GANs e duas de suas variações — DCGAN e CGAN. No capítulo seguinte seguiremos para fazer experimentos comparativos entre as três redes adversariais retratadas.

3 EXPERIMENTOS

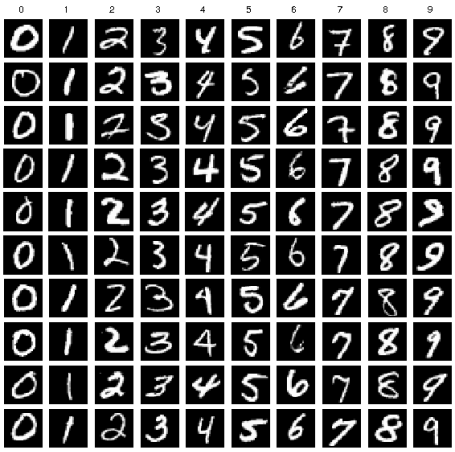
Neste capítulo iremos relatar os resultados obtidos nos experimentos realizados como base para nosso trabalho. Esses experimentos tiveram como finalidade avaliar e comparar o desempenho dos modelos de GAN, GAN condicional (CGAN) e GAN convolucional profunda (DCGAN) na tarefa de gerar imagens realistas, a partir de um mesmo conjunto de dados. Para cada um dos modelos, detalharemos os parâmetros utilizados e os resultados obtidos de acordo com as métricas que também serão estabelecidas neste capítulo.

3.1 BASE DE DADOS

A base de dados adotada nos experimentos foi a MNIST (Modified National Institute of Standards and Technology) (DENG, 2012), comumente utilizada para o treinamento de sistemas de processamento de imagens e visão computacional. A base consiste de cerca de 60000 imagens dos dígitos 0 a 9 – aproximadamente 6000 imagens por dígito – escritos à mão por 500 pessoas diferentes, onde cada imagem tem resolução de 28x28 pixels e é exibida em tons de cinza. Cada imagem é representada como um vetor de tamanho 784, onde cada elemento corresponde a um pixel e possui valor entre 0 (preto) e 255 (branco), que representa o seu tom de cinza. A base inclui também, para cada imagem, um rótulo que indica qual é o dígito nela retratado.

Apesar de reconhecer que as GANs possuem capacidade suficiente para trabalhar com conjuntos de imagens muito mais complexos, resolvemos utilizar uma base de dados mais simples por se tratar de um estudo introdutório e por conta do menor tempo de treinamento necessário para realização dos experimentos. A Figura 17 apresenta alguns exemplos de imagens retiradas dessa base de dados.

Figura 17 – Exemplos do dataset MNIST



Fonte: (DENG, 2012).

3.2 TAREFA

O experimento envolve o treinamento dos modelos generativos e discriminativos dos diferentes tipos de GAN escolhidos, utilizando como conjunto de treinamento a base MNIST. Mais especificamente, a tarefa dos geradores é aprender a gerar imagens realistas de dígitos entre 0 e 9 escritos à mão, enquanto que os discriminadores terão como objetivo aprender a distinguir entre imagens reais da base MNIST e as novas imagens produzidas pelo modelo generativo. Dessa forma, a saída produzida pelos geradores deverá ser uma imagem 28x28, em tons de cinza, e a saída dos discriminadores deverá ser um número entre 0 e 1 que representa a probabilidade de a imagem ser verdadeira e, para uma probabilidade maior ou igual à 0.5, a imagem é considerada verdadeira.

Como foco de nosso trabalho, estaremos mais interessados em avaliar e comparar o desempenho dos diferentes modelos generativos na tarefa de gerar imagens realistas de digítos escritos à mão. Para isso, ao fim do processo de treinamento, iremos separar algumas amostras aleatórias de imagens geradas por esses modelos e então avaliá-las tanto de maneira qualitativa (visualmente), quanto de maneira quantitativa (usando algumas métricas a serem apresentadas).

3.3 MÉTRICAS

Uma questão muito relevante para o estudo é como avaliar o desempenho dos modelos de GAN utilizados. Diferentemente de outros modelos de rede neural em que métricas como precisão, *recall*, *f1-score* e acurácia (VAKILI; GHAMSARI; REZAEI, 2020) já dão uma ideia suficientemente boa da eficácia da rede, as GANs não tem um padrão claro de avaliação.

(GOODFELLOW et al., 2014) propõe o uso de um estimador de densidade das amostras chamado de *Kernel Density Estimation* (KDE) ou *average log-likelihood*. Contudo, como (THEIS; OORD; BETHGE, 2016) demonstram, tal estimação apresenta uma série de problemas como:

- *Log-likelihood* e qualidade de amostras geradas não são estritamente relacionados. Assim, um modelo pode apresentar alto KDE mas não gerar imagens boas e vice-versa.

- Essa métrica produziu resultados melhores para modelos geradores do que para a própria distribuição real dos dados, a qual deveria ser o limite superior.

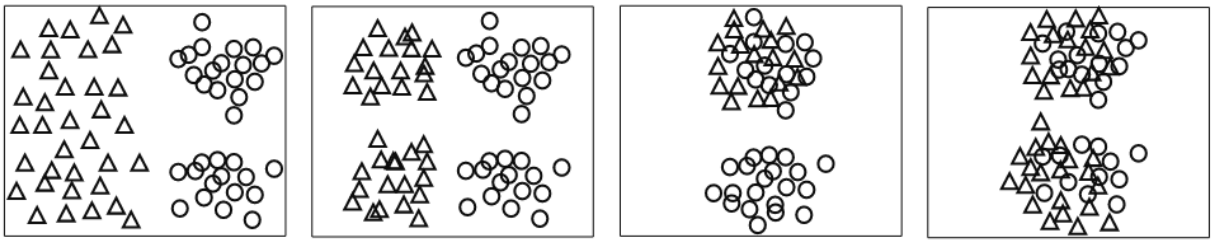
Dessa forma, utilizar *average log-likelihood* não parece ser a melhor forma de avaliação dos modelos. Mais recentemente, (BORJI, 2019) fez um comparativo de 24 métricas de avaliação de GANs incluindo formas quantitativas e qualitativas para definir as mais eficazes. Alguns dos critérios para definir uma métrica de GAN eficiente foram os seguintes:

- Deve favorecer modelos que produzem imagens que sejam dificilmente distinguíveis das imagens reais.
- Deve favorecer modelos que gerem imagens diversas. Isto se deve ao fato de que há um problema em algumas arquiteturas de GANs chamado de *mode collapse* em que a amostra gerada é sempre igual ou de classes específicas, colapsando o gerador em apenas um ou mais modos. Para o caso do MNIST, é o equivalente a gerar apenas imagens dos dígitos "1"e "9", por exemplo.
- Deve ter limites superiores e inferiores bem definidos. Como mencionado, foi verificado que o KDE falha para esse princípio, pois não possui limite superior.
- Deve ser invariante a pequenas transformações e distorções. Exemplificando, uma baixa rotação não deveria alterar muito os valores entre uma imagem e outra.
- Deve concordar com julgamentos e percepções humanas dos modelos.
- Deve ter baixo custo e complexidade computacional.

Ademais, tanto (THEIS; OORD; BETHGE, 2016) quanto (BORJI, 2019) concordam que não há consenso sobre a métrica definitiva para avaliação de GANs. O que pode-se fazer é levar em conta a aplicação específica sob a qual deve-se medir o desempenho e escolher a(s) métrica(s) a partir disso. Duas métricas amplamente utilizadas, *Inception Score* (IS) (SALIMANS et al., 2016) e *Fréchet Inception Distance* (FID) (HEUSEL et al., 2017) apresentam a desvantagem de usarem modelos profundos pré-treinados em conjuntos de dados de cenas naturais, como o *ImageNet* (DENG et al., 2009). Logo, a aplicação deles em um contexto de dígitos é questionável.

Recentemente, (BARUA et al., 2019) propuseram uma métrica chamada de *Cross Local Intrinsic Dimensionality* (CrossLID). Nela, o conceito de dimensionalidade intrínseca local (LID) (HOULE, 2013) é utilizado para caracterizar a distribuição de dados de dois conjuntos G e X em que X é o conjunto de dados reais e G é o conjunto de dados gerados pelo modelo. Como mostra a Figura 18, se as duas distribuições estiverem em alinhamento, o valor do CrossLID será o mais próximo de 0 e, conforme pior for o alinhamento, mais alto será o CrossLID.

Figura 18 – Avaliação de GANs com CrossLID



(a) CrossLID=15.16 (b) CrossLID=7.33 (c) CrossLID=4.78 (d) CrossLID=2.10

Fonte: (BARUA et al., 2019). Os triângulos referem-se a imagens do conjunto de imagens geradas G e os círculos a imagens do conjunto real X, o qual é dividido em duas classes. (a) as distribuições são distantes espacialmente uma da outra; (b) a distribuição de G conseguiu assumir as duas classes, mas eles continuam espacialmente distantes das classes reais; (c) a distribuição de G assume apenas uma classe, mas ela é alinhada com uma classe de X (*mode collapse*); e (d) G alinha-se com X em suas duas classes.

Para o cálculo da métrica, os autores utilizaram o *Maximum Likelihood Estimator* (MLE) de LID que é definido por:

$$LID(x;X) = -(\frac{1}{k}\sum_{i=1}^{k}\log\frac{r_i(x;X)}{r_{max}(x;X)})^{-1} \tag{3.1}$$

onde X é um conjunto de dados, x uma amostra pertencente a X, k é a quantidade de vizinhos observada, $r_i(x;X)$ é a distância euclidiana de x até o vizinho mais próximo em X e $r_{max}(x;X)$ denota a distância máxima na vizinhança. De tal estimador, obtém-se o CrossLID:

$$CrossLID(A;B) = \mathbb{E}_{x\in A}LID(x;B). \tag{3.2}$$

onde A e B são conjuntos de dados e $\mathbb{E}_{x\in A}$ é o valor esperado considerando a amostra x em A. Ademais, posto que avaliamos a qualidade das imagens geradas com relação à distribuição original e que $CrossLID(A;B) \neq CrossLID(B;A)$, a métrica é calculada a partir de $CrossLID(G;X)$, em que G é o conjunto de imagens geradas e X é o conjunto de imagens reais. Além disso, como mostra a Tabela 1, a eficiência do CrossLID é equivalente ou superior às duas métricas quantitativas mais amplamente utilizadas em critérios de avaliação de GANs. Outras duas vantagens de tal métrica foram o fato dela ter sido aplicada para testes usando o MNIST e a disponibilização do código oficial em (BARUA et al., 2019), que foi feito em Python e Keras (CHOLLET et al., 2015). Dessa maneira, escolhemos o CrossLID como métrica do projeto para avaliação e comparação entre modelos.

Tabela 1 – Comparativo de avaliação entre CrossLID, IS e FID

Critério de avaliação	CrossLID	IS	FID
Sensibilidade a *mode collapse*	Alta	Baixa	Alta
Robustez a ruído de entrada pequeno	Alta	Baixa	Baixa
Robustez a transformações na entrada	Moderada	Moderada	Baixa
Robustez à variação no tamanho da amostra	Alta	Moderada	Baixa

Fonte: (BARUA et al., 2019). Valores e critérios traduzidos para o português.

3.4 IMPLEMENTAÇÕES

Para a implementação dos diferentes tipos de GAN, fizemos uso da linguagem Python, em conjunto com o framework de aprendizado de máquina de código aberto PyTorch (PASZKE et al., 2019), que inclui diversas ferramentas que facilitam o desenvolvimento, treinamento e avaliação de modelos de redes neurais. Também usamos como base para o desenvolvimento dos modelos o repositório entitulado de *PyTorch-GAN* [1], hospedado no GitHub, que disponibiliza implementações em Python de diversos modelos de GAN. Além disso, para a realização dos experimentos, usamos uma máquina com processador (CPU) AMD Ryzen 7 3700X – de 8 núcleos físicos e 16 núcleos lógicos (*threads*) – placa de vídeo (GPU) NVIDIA GeForce GTX 970 e 16 GB de memória DDR4.

Nas subseções seguintes detalharemos a arquitetura das redes neurais dos modelos generativos e discriminativos que compõem cada tipo de GAN estudado, assim como os hiperparâmetros utilizados durante o processo de treinamento.

3.4.1 Hiperparâmetros gerais

Alguns hiperparâmetros foram mantidos constantes durante todo o processo de treinamento, assumindo assim os mesmos valores para os três diferentes modelos. São eles:

- Número de épocas: 200
- Tamanho de mini-lote de amostras (mini-batch size): 64
- Taxa de aprendizagem (learning rate): 0.0002
- Tamanho do vetor de ruído: 100
- Tamanho de cada dimensão das imagens: 28 (devido à base de dados utilizada)
- Quantidade de canais de cor para as imagens: 1 (tons de cinza)
- Número de threads da CPU durante a geração dos lotes: 8

[1] https://github.com/eriklindernoren/PyTorch-GAN, acessado em 16/02/2022

- Inclinação de reta α para a função LeakyReLU: 0.2
- Fator ϵ para a normalização em lotes: 0.8

Essas escolhas foram motivadas pelos artigos base dos modelos estudados, que sugerem esses valores como bons pontos de partida. Em especial, (MIRZA; OSINDERO, 2014) e (RADFORD; METZ; CHINTALA, 2016) fazem uso de valores similares para o treinamento dos modelos CGAN e DCGAN, respectivamente, com a mesma base de dados utilizada em nossos experimentos.

3.4.2 Modelo 1 - GAN

Em termos simples, o objetivo do modelo generativo G do primeiro tipo de GAN criado é, partindo de um valor de ruído, produzir uma imagem que aparente pertencer à base de dados utilizada no treinamento. Sendo assim, como entrada para G, utilizamos um vetor de tamanho 100, cujos elementos foram aleatoriamente amostrados de uma distribuição normal padrão (média 0 e desvio padrão 1), representando o valor de ruído. Já para a saída, queremos que o resultado produzido esteja no mesmo formato das amostras de nossa base de dados: um vetor de tamanho 784, onde cada elemento representa um pixel e possui valor entre 0 e 255.

Para isso, utilizamos no modelo generativo uma arquitetura de 5 camadas totalmente conectadas, com 128, 256, 512, 1024 e 784 neurônios, respectivamente, onde a primeira camada sucede a entrada e a última camada gera a saída do modelo. As quatro primeiras camadas usam como função de ativação a LeakyReLU, enquanto que a última faz uso da função tangente hiperbólica (*tanh*) para ativação. Escolhemos a função *tanh* pelo fato de ela gerar resultados no intervalo [-1, 1], que podem ser facilmente normalizados para o intervalo [0, 255]. Além disso, todas as camadas, com exceção da primeira, fazem uso de normalização em lotes (IOFFE; SZEGEDY, 2015) para tornar o treinamento mais estável.

Já o modelo discriminativo D é composto por 3 camadas totalmente conectadas de 512, 256 e 1 neurônios, respectivamente. Dessas, as duas primeiras usam a LeakyReLU e a última usa a função sigmoide para ativação, que produz como saída um valor decimal entre 0 e 1. Esse valor, por sua vez, é o que representa a probabilidade de a imagem pertencer à base de dados ou ter sido gerado pelo modelo generativo, de acordo D.

3.4.3 Modelo 2 - CGAN

A GAN condicional, diferentemente das GANs e DCGANs, recebe como entrada de seu modelo generativo G não apenas um valor de ruído, mas também um rótulo que indica a qual classe deve pertencer a imagem gerada por G. No caso da MNIST, as classes representam os dez dígitos presentes na base, totalizando dez classes possíveis. Em nossa implementação, os rótulos são representados por vetores de tamanho 10, preenchidos por

zeros em todas as posições, exceto naquela que corresponde ao dígito a ser representado, que assume o valor 1 (*one-hot encoding*).

A entrada do modelo generativo é formada então pelo vetor de tamanho 110 resultante da concatenação do vetor de tamanho 100, que representa o ruído, com o vetor de tamanho 10, que representa o rótulo da classe. A saída, por sua vez, segue o mesmo formato dos outros tipos de GAN: um vetor de tamanho 784, com cada elemento representando um pixel da imagem gerada. Esse modelo é composto de 5 camadas totalmente conectadas com 128, 256, 512, 1024 e 784 neurônios, respectivamente. As funções de ativação para cada camada e a normalização em lotes utilizadas são as mesmas do modelo generativo da GAN, apresentadas na subseção anterior.

Da mesma forma que em G, o modelo discriminativo D recebe um rótulo de classe como entrada, juntamente com a imagem a ser classificada como pertencente à base de dados ou gerada por G. A arquitetura de D é formada por 4 camadas totalmente conectadas, sendo as três primeiras compostas por 512 neurônios e a última por um único neurônio. As três camadas iniciais fazem uso da LeakyReLU como função de ativação e as duas camadas intermediárias (camadas 2 e 3) fazem uso de *dropout* com probabilidade 0.4. *Dropout* (SRIVASTAVA et al., 2014) é um método de regularização – que pode ser aplicado à camadas de uma rede – que consiste em ignorar durante o treinamento, de maneira aleatória e temporária, alguns nós da rede, juntamente com todas as suas conexões. Isso reduz o *overfitting* da rede em relação aos conjunto de treino, tornando-a mais genérica e, por consequência, melhorando seu desempenho quando usada com dados não vistos durante o processo de treino.

3.4.4 Modelo 3 - DCGAN

As DCGANs recebem a mesma entrada nos modelos generativo e discriminativo que as GANs, e também produzem saídas nos mesmos formatos. O modelo generativo G é composto de 4 camadas, onde a primeira, totalmente conectada, é responsável por redimensionar a entrada (*upsampling*), dada por um vetor de tamanho 100, para o formato $7 \times 7 \times 112$ (largura $\times$ altura $\times$ quantidade de canais). As 3 camadas seguintes são de convolução transposta, que redimensionam a saída da camada anterior por meio de operações matriciais, de maneira sucessiva, até que se obtenha, na camada final, uma nova saída no formato desejado de $28 \times 28 \times 1$ (vetor de tamanho 784). Cada camada de convolução tem tamanho do *kernel* igual a (3,3), tamanho do passo (*stride*) igual a (1,1) e *same padding* com aumento de 1 dimensão. As duas camadas intermediárias fazem uso da LeakyReLU como função de ativação, enquanto que a última utiliza *tanh*. Além disso, todas as camadas, com exceção da última, usam normalização em lotes.

O modelo discriminativo D é composto por 5 camadas, sendo as 4 primeiras de convolução (*strided convolution*) e a última totalmente conectada. Por meio de uma série de redimensionamentos (*downsampling*) nas camadas de convolução, D reduz a sua entrada,

dada por um vetor de tamanho 784 ($28 \times 28 \times 1$), para o formato $2 \times 2 \times 112$ que, por sua vez, é transformado pela última camada em um valor decimal entre 0 e 1 por meio da função de ativação sigmoide. Todas as camadas de convolução fazem uso de LeakyReLU para ativação e *dropout* com probabilidade 0.25, tamanho de *kernel* igual a (3,3), tamanho do passo igual a (1,1) e *same padding* com aumento de 1 dimensão. Além disso, todas as camadas, com exceção da primeira, utilizam normalização em lotes.

3.4.5 Resumo

A Tabela 2 traz um resumo das arquiteturas e hiperparâmetros de cada modelo utilizado nos experimentos descritos nessa seção.

Modelo / Hiperparâmetro	GAN		CGAN		DCGAN	
	G	D	G	D	G	D
Número de camadas	5	3	5	4	4	5
Normalização em lotes	2-5	-	2-5	-	1-3	2-5
LeakyReLU	1-4	1-2	1-4	1-3	2-3	1-4
tanh	5	-	5	-	4	-
sigmoide	-	3	-	-	-	5
Dropout	-	-	-	2-3	-	1-4

Tabela 2 – Resumo dos hiperparâmetros utilizados em cada modelo estudado. Os números representam as camadas em que foram utilizados cada hiperparâmetro, G representa o modelo generativo e D o modelo discriminativo. A nomenclatura *i-j* indica que da camada i até a camada j, inclusive, é feito uso, no modelo correspondente, do hiperparâmetro indicado.

3.5 RESULTADOS

Após 5 rodadas de experimento para cada um dos tipos de GAN, considerando os parâmetros já mencionados, obtivemos os resultados apresentados na Tabela 3.

Modelo	CrossLID (μ)	CrossLID (σ)	Tempo (μ)	Tempo (σ)
GAN	30.53	15.05	3408.6	317.6
DCGAN	4.52	0.04	7487.4	2328.21
CGAN	3.91	0.01	3940.8	472.85

Tabela 3 – Avaliação de cada modelo considerando o CrossLID e o tempo de execução total das etapas de treinamento e avaliação. Os valores referentes ao tempo de execução estão em segundos, μ representa o valor médio e σ o desvio padrão.

Como mostra essa tabela, podemos ver que os modelos de CGAN e DCGAN tiveram melhor desempenho se comparados ao modelo de uma GAN, o que era esperado. Paralelamente, ao analisarmos as imagens geradas pelos modelos na Figura 19 na coluna de 200 épocas (término do treinamento), percebemos que os resultados visuais condizem com a

métrica gerada. A GAN entra em colapso, produzindo apenas os dígitos "1", "3", "7", "8"e "9"com predominância do dígito "1", além de apresentar imagens de qualidade baixa visualmente. Já as imagens produzidas pela CGAN e DCGAN são próximas em termos de qualidade, mas o fato da CGAN apresentar a mesma quantidade de imagens para cada dígito, tendo em vista que a saída é condicionada, dá uma ligeira vantagem para ela.

Figura 19 – Imagens geradas ao longo do treinamento

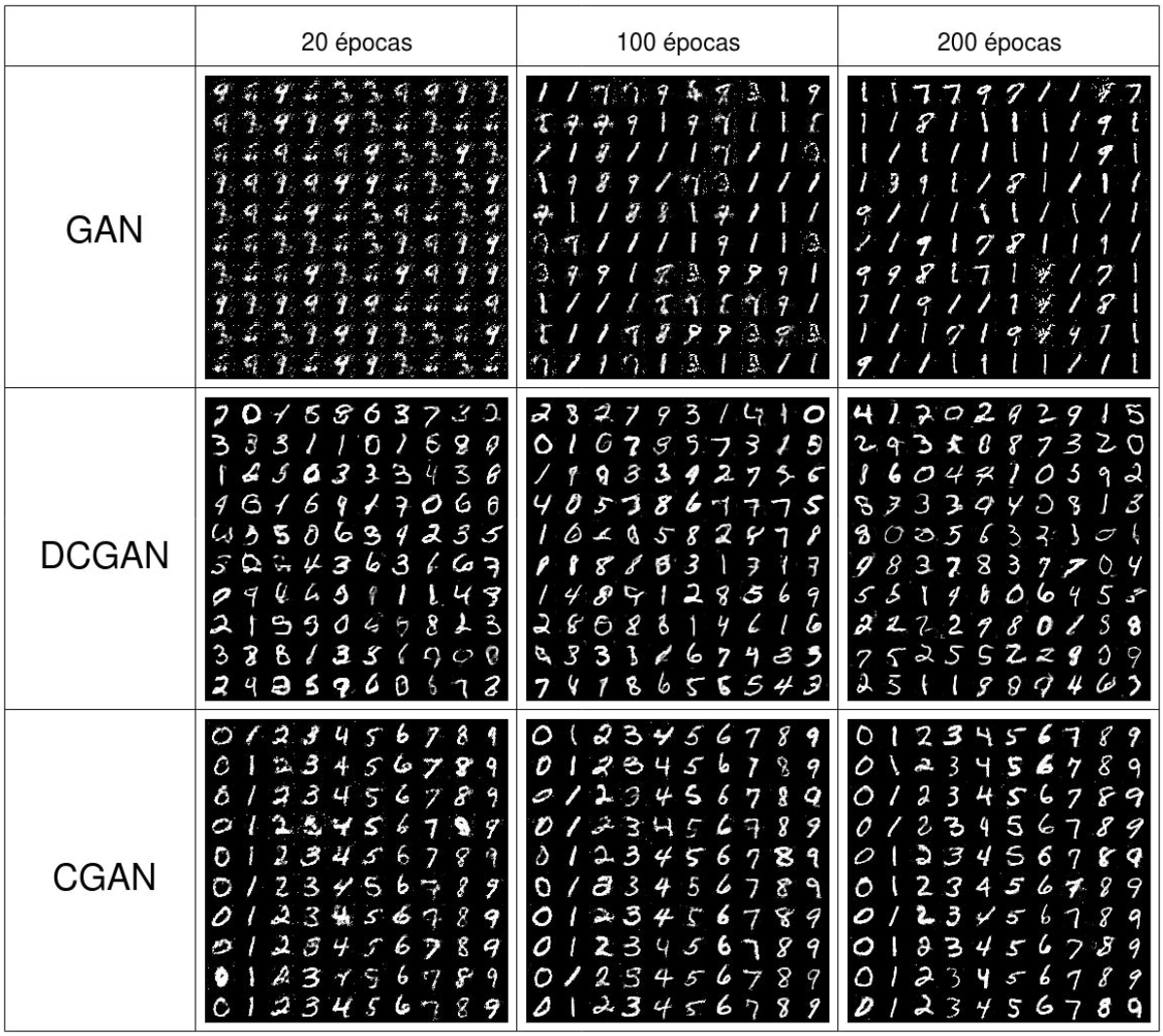


Tabela com imagens produzidas ao longo do processo de treinamento. As linhas representam os modelos e as colunas a quantidade de épocas utilizadas para treinar o modelo. 100 imagens geradas em cada passo foram concatenadas em uma matriz 10×10 para avaliar os modelos visualmente.

Ademais, com o propósito de entender melhor o funcionamento do discriminador de cada rede, medimos as suas acurácias, precisões, sensibilidades e *F1-scores* (média harmônica entre precisão e sensibilidade) ao fim de cada época de treinamento. Paralelamente, armazenamos os valores de perda (*loss*) em cada treinamento a fim de tentarmos extrair

informações adicionais ou relevantes para o projeto. Para construção de gráficos foi utilizada a ferramenta de visualização Tensorboard, parte da biblioteca Tensorflow (ABADI et al., 2016). [2]

Para facilitar a visualização dos gráficos, utilizamos o fator de suavização (*smoothing*) do Tensorboard com valor 0.95. Esse parâmetro controla o grau de "suavização" das curvas, fazendo a média móvel exponencial (*exponential moving average* - EMA), calculada recursivamente de acordo com a fórmula 3.3, onde α representa o fator de suavização, Y_t é o valor real no período de tempo t e S_t é o valor do EMA em qualquer período de tempo t.

$$S_t = \begin{cases} Y_0 & , \ t = 0 \\ \alpha Y_t + (1 - \alpha) \cdot S_{t-1} & , \ t > 0 \end{cases} \tag{3.3}$$

O fator de suavização α pode assumir valores entre 0 e 1 e, quanto maior ele for, mais rápido são desconsideradas as observações mais antigas. Assim, o valor usado de 0.95 nos permite perceber mais facilmente as tendências das curvas de cada gráfico, tornando sua variação mais suave. Nas figuras a seguir, as linhas mais claras representam os valores reais, enquanto que as linhas mais escuras representam a média móvel exponencial.

3.5.1 Acurácia do discriminador

Como apresentado na seção 2.3, o objetivo do treinamento de uma GAN é ser capaz de treinar um gerador G de tal forma que a acurácia da rede discriminativa D seja 0.5, representando a incapacidade do discriminador de distinguir entre imagens reais e imagens falsas. Por isso, o fator interessante em visualizar as acurácias dos modelos ao longo do treinamento está em compreender se as redes convergiram para esse objetivo.

Como podemos observar na Figura 21, a rede que chegou mais perto desse alvo foi a CGAN ao manter-se em torno de 0.6 a partir da época 50. Em contrapartida, a GAN (Figura 20) permaneceu com altíssima acurácia em todo o treinamento. Nossa compreensão para esse motivo é a de que a rede discriminativa foi otimizada mais eficientemente que a rede generativa, o que resulta em um gerador com maiores dificuldades de aprendizado. Ao observarmos a Figura 19 e a Tabela 3 podemos ver que a alta acurácia do discriminador é compatível com um pior desempenho das imagens geradas. Já a acurácia do discriminador da DCGAN (Figura 22) apresentou variância bem alta mas se manteve em torno de 0.65, na média.

[2] As visualizações produzidas podem ser encontradas em detalhes pelo site: https://linktr.ee/tcc_eduardo_rodrigo.

Figura 20 – Acurácia do discriminador da GAN

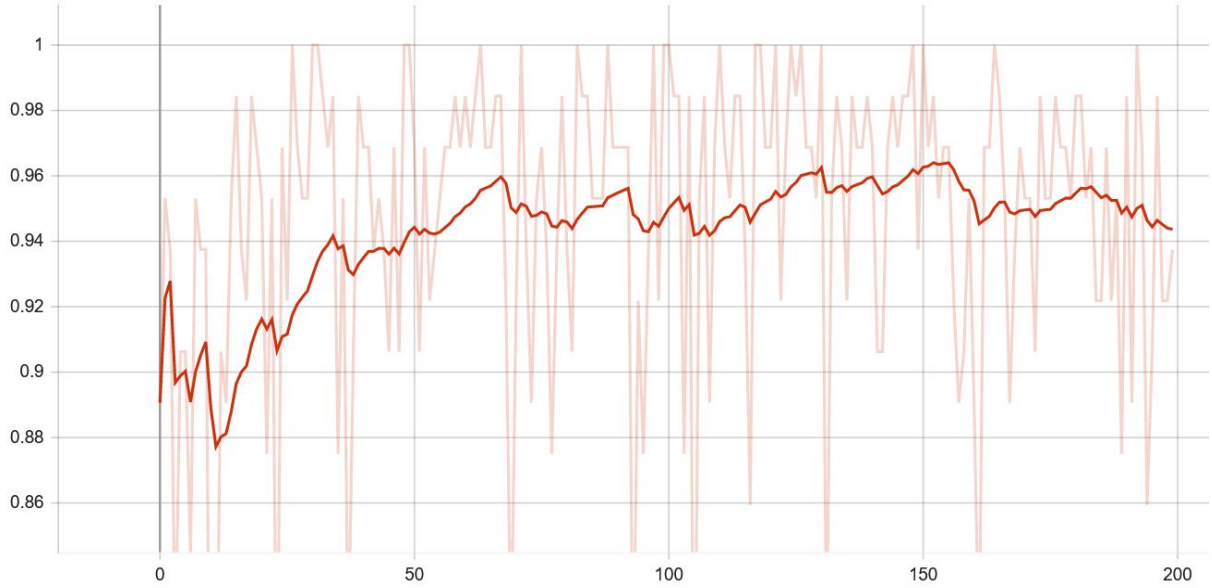


Gráfico de um experimento contendo a acurácia do discriminador do modelo GAN. As linhas claras representam os valores reais e as linhas escuras os valores após aplicação do fator de suavização. O eixo X representa o número da época e o eixo Y, a acurácia.

Figura 21 – Acurácia do discriminador da CGAN

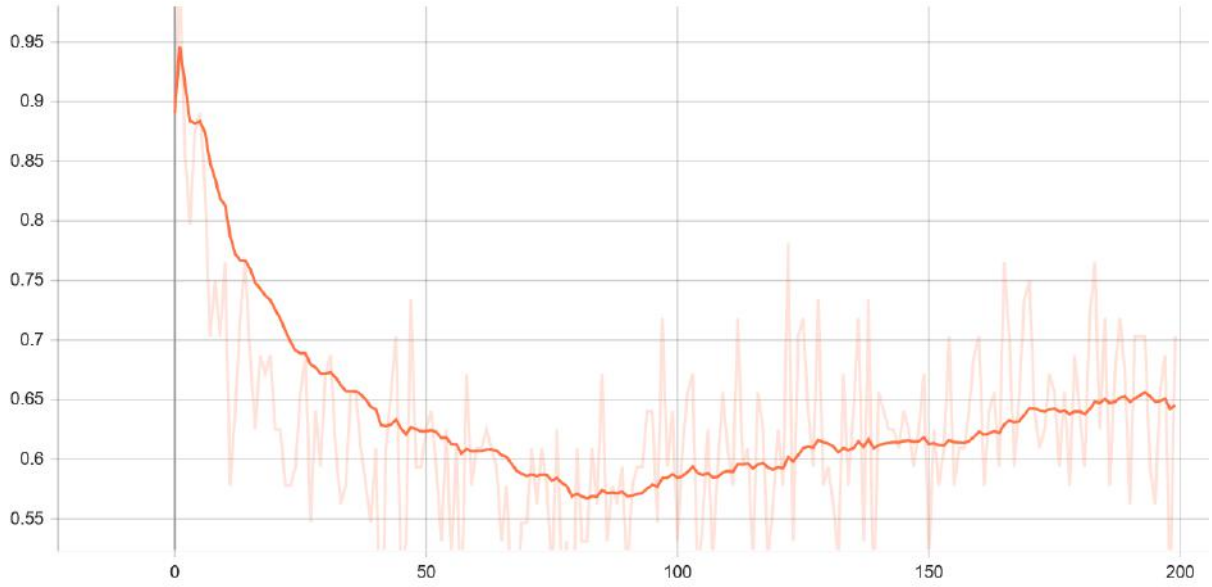


Gráfico de um experimento contendo a acurácia do discriminador do modelo CGAN. As linhas claras representam os valores reais e as linhas escuras os valores após aplicação do fator de suavização. O eixo X representa o número da época e o eixo Y, a acurácia.

Figura 22 – Acurácia do discriminador da DCGAN

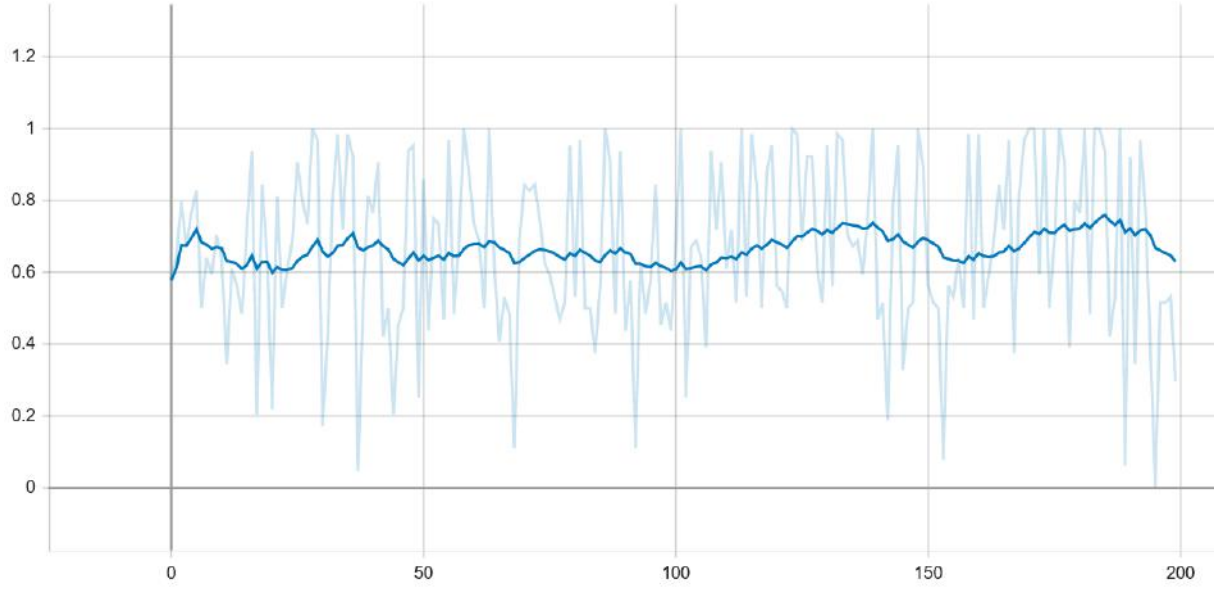
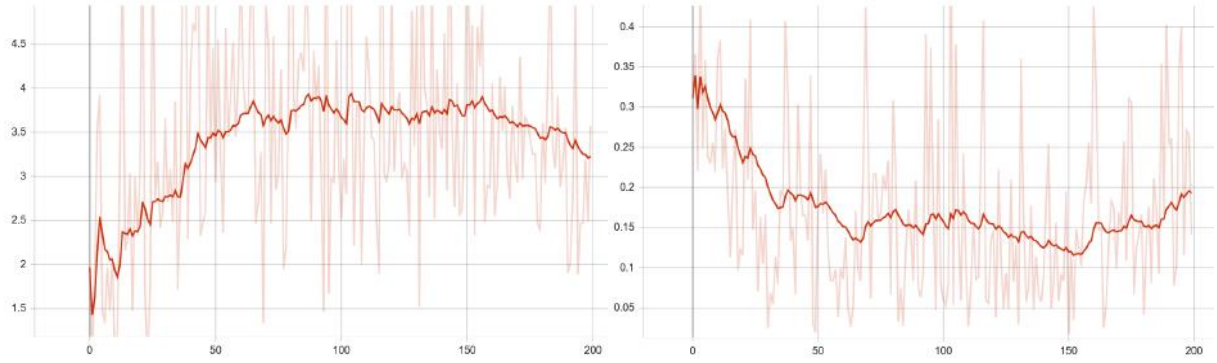


Gráfico de um experimento contendo a acurácia do discriminador do modelo DCGAN. As linhas claras representam os valores reais e as linhas escuras os valores após aplicação do fator de suavização. O eixo X representa o número da época e o eixo Y, a acurácia.

3.5.2 Perdas dos modelos

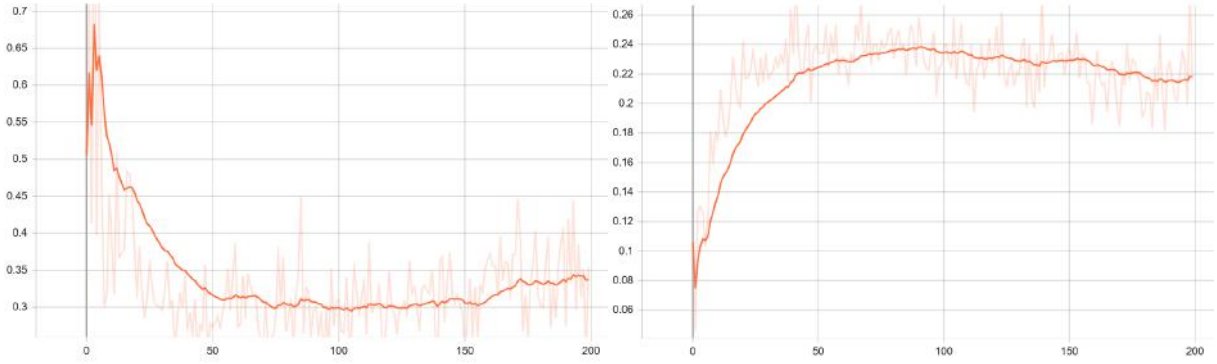
Figura 23 – Perdas da GAN



(a) (b)

Gráficos de experimentos onde (a) representa o valor da perda do gerador por cada época enquanto que (b) representa o valor da perda do discriminador para as mesmas épocas.

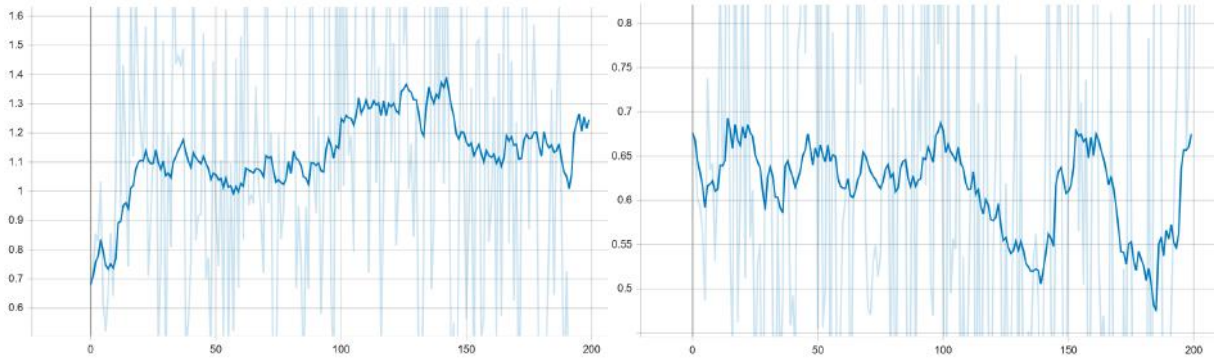
Figura 24 – Perdas da CGAN



(a) (b)

Gráficos de experimentos onde (a) representa o valor da perda do gerador por cada época enquanto que (b) representa o valor da perda do discriminador para as mesmas épocas.

Figura 25 – Perdas da DCGAN



(a) (b)

Gráficos de experimentos onde (a) representa o valor da perda do gerador por cada época enquanto que (b) representa o valor da perda do discriminador para as mesmas épocas.

De modo geral, em treinamentos de redes neurais, o decréscimo da função de perda ao longo das épocas é um resultado desejado e esperado de um aprendizado contínuo. Em contrapartida, ao colocarmos uma rede contra a outra, percebemos que esse comportamento de queda não acontece em ambos os lados. Não obstante, o comportamento observado foi mais semelhante ao de gráficos inversamente proporcionais, nos quais o aumento do valor de perda de um lado é condizente com a diminuição da perda na rede adversária. Assim, pode-se compreender que, de fato, as redes generativas estão em disputa com as redes discriminativas.

Outro fator fundamental para compreensão dos gráficos de perdas está no fato de que a CGAN utiliza uma função de perda diferente das demais. Isso se dá pelo simples fato de que a CGAN carrega uma variável condicional que diz qual dígito a figura deve gerar.

Assim, trata-se de um problema multiclasse e não apenas binário. Ademais, esse é um fator essencial para prevenir o colapso da rede, uma vez que se a rede generativa gerar uma imagem de alta qualidade, mas com o dígito diferente do valor apontado pela variável condicional, isso será contabilizado como erro pela função de perda. Por essa razão, a Figura 24 apresenta curvas com aspecto diferente das Figuras 23 e 25.

3.5.3 Matrizes de confusão

Além das medições apresentadas nas subseções anteriores, também geramos as matrizes de confusão de cada modelo discriminativo ao longo do seu treinamento. Essa visualização nos permite verificar a distribuição de classificações, feitas pelo modelo discriminativo, das 64 imagens do mini-lote em verdadeiras ou falsas (geradas pelo modelo generativo). Com isso, é possível aferir a capacidade do modelo generativo de confundir o modelo discriminativo, fazendo com que este último classifique de maneira incorreta as imagens produzidas por ele. As matrizes das Figuras 26, 27 e 28 foram geradas com o uso da biblioteca *scikit-learn* (PEDREGOSA et al., 2011).

Figura 26 – Matriz de confusão da GAN

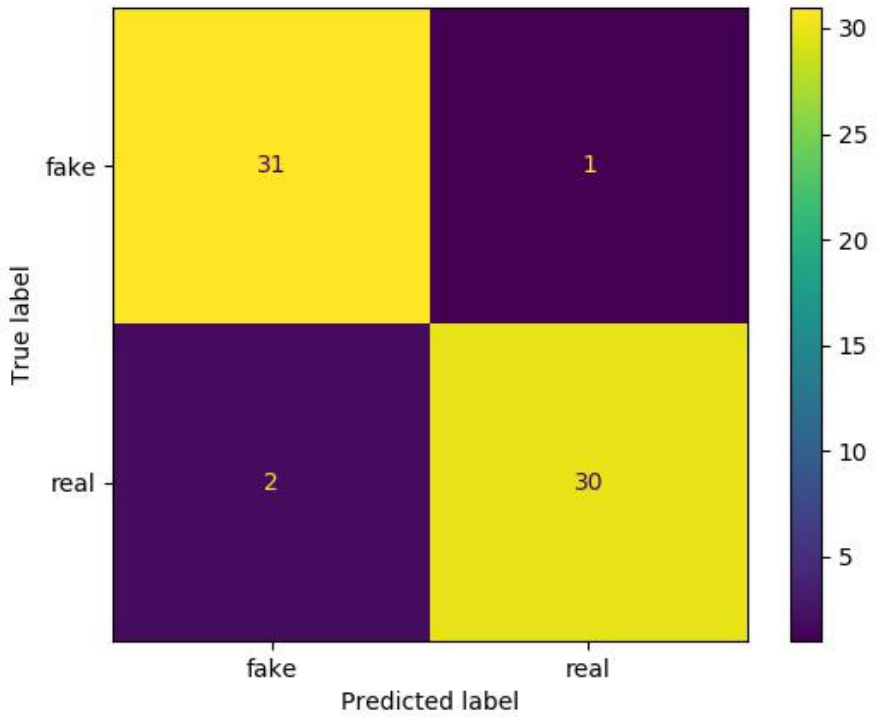


Figura 27 – Matriz de confusão da CGAN

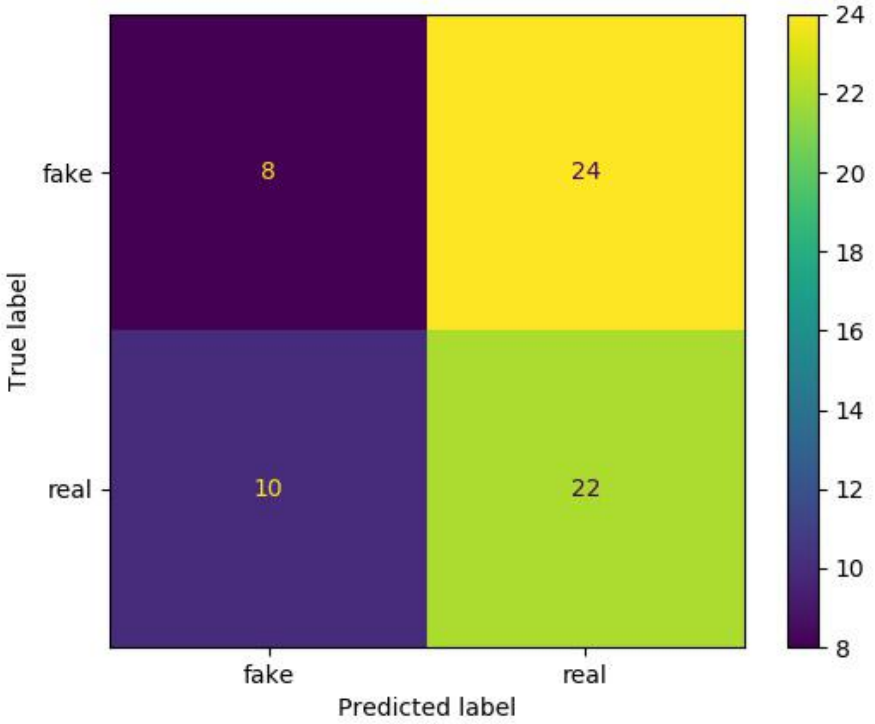
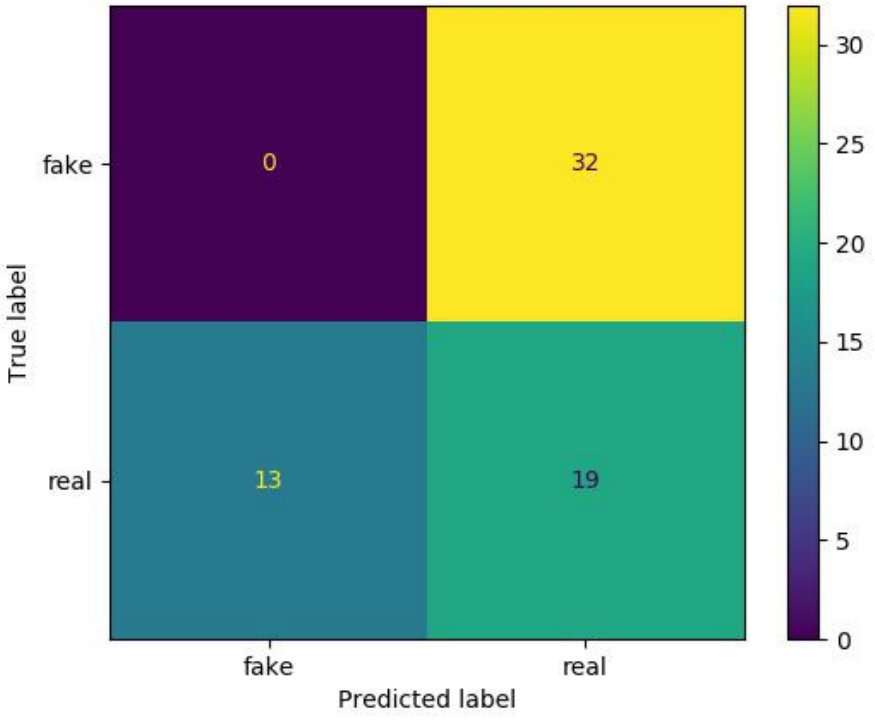


Figura 28 – Matriz de confusão da DCGAN



Escolhemos as figuras que representam as matrizes de confusão das iterações de maior desempenho do modelo generativo, para cada um dos tipos de GAN analisados. A Figura 26 mostra que o modelo generativo da GAN não foi eficaz em produzir imagens de qualidade alta o suficiente para confundir o modelo discriminativo. Isso é evidenciado pelo fato de que, das 32 imagens falsas apresentadas a ele, apenas 1 (3.1%) foi classificada incorretamente como verdadeira. Em contrapartida, os modelos generativos do CGAN e DCGAN conseguiram resultados mais satisfatórios, com o primeiro causando classificações incorretas por parte do modelo discriminativo em 24 de 32 oportunidades (75%), e o último, em todas as 32 oportunidades (100%), na sua iteração de maior desempenho.

3.6 CONLUSÃO

Neste capítulo, discorremos a respeito dos experimentos de base que fizemos para uma avaliação inicial das GANs com base na base de dados de dígitos MNIST, nos quais o objetivo foi a geração automática de imagens de dígitos escritos à mão. A principal métrica de avaliação escolhida para tal tarefa foi o CrossLID, que compara a distribuição dos dados gerados com a distribuição dos dados reais e retorna um índice que é inversamente proporcional à qualidade das imagens geradas.

Não obstante, descrevemos as implementações de cada uma das arquiteturas usando Python e Pytorch a partir do repositório base *Pytorch-GAN*. Por fim, apresentamos os resultados para o CrossLID, indicando que a CGAN é ligeiramente melhor que a DCGAN e GAN fica bem atrás de ambas, o que condiz com os resultados visuais. Ademais, a fim de entendermos a ação do discriminador, apresentamos métricas sobre a evolução do seu aprendizado em formato de gráficos e matrizes de confusão. No próximo capítulo, traremos outros experimentos específicos trazendo modificações na arquitetura da rede.

4 MODIFICAÇÕES

Neste capítulo, traremos algumas propostas de mudanças na arquitetura e parâmetros das redes com o intuito de melhorar o desempenho dos modelos e entender a influência desses parâmetros e de cada aspecto da arquitetura no resultado final. Da mesma forma, sempre trataremos das modificações e suas consequências categorizadas rede a rede.

4.1 VETOR LATENTE OU DE RUÍDO

Como mencionado nos capítulos 2 e 3, todas as GANs recebem como entrada, em seus geradores, um vetor de ruído aleatório, também chamado de vetor latente, que será transformado na imagem de um dígito. Como apresentado em (MANISHA; DAS; GUJAR, 2020), para o caso da DCGAN com o conjunto de dados de dígitos MNIST a dimensionalidade do vetor latente sugerida para ótimo desempenho foi de 10 dimensões, enquanto que para pior desempenho foi de 2 dimensões. Esses valores foram obtidos utilizando as medidas IS e FID, discutidas na seção 3.3, como avaliação de desempenho. Contudo, é apresentado que de 10 dimensões até 1000 a diferença é quase nula, tornando o de 10 dimensões ideal por necessitar de menos poder computacional. Dessa maneira, além do experimento já feito com 100 dimensões, escolhemos testar com 2 dimensões e com 10 dimensões para comparar as medidas do artigo com o resultado em CrossLID e avaliar as outras redes.

Modelo	2 Dimensões		10 Dimensões		100 Dimensões	
	CrossLID	Tempo	CrossLID	Tempo	CrossLID	Tempo
GAN	11.1 ± 5	3454 ± 208	6 ± 0.7	3105 ± 164	30.5 ± 15	3409 ± 318
DCGAN	6.4 ± 0.2	8026 ± 2154	4.6 ± 0.04	4888 ± 1053	4.5 ± 0.04	7487 ± 2328
CGAN	4.7 ± 0.6	4003 ± 353	3.9 ± 0.01	3704 ± 244	3.9 ± 0.01	3941 ± 473

Tabela 4 – Avaliação de cada modelo com 2, 10 e 100 dimensões latentes, considerando o CrossLID e o tempo de execução total das etapas de treinamento e avaliação. Os valores referentes ao tempo de execução estão em segundos, e todos são apresentados como a média ± o desvio padrão.

Conforme ilustra a Tabela 4, a mudança de dimensionalidade do vetor de ruído demonstrou ser impactante no resultado final. Primeiramente, pôde-se comprovar a expectativa de que a DCGAN manteve a qualidade em dimensão 10, com 4.58 de média comparado a 4.52 e piorou para o vetor de ruído com dimensão 2, marcando 6.38 de média no CrossLID. O mesmo comportamento foi observado para a CGAN, a qual se aproximou de 3.91 com 3.92 de média e deu resultado de 4.69 de média em dimensão 2.

Em contrapartida, o resultado de tal mudança na GAN foi um tanto surpreendente. O valor com ruído de 2 dimensões foi pior do que com 10 dimensões, como esperado,

porém, ambos os resultados foram muito melhores que o com 100 dimensões. Assim, a dimensionalidade do vetor de ruído é imensamente relevante no treinamento de GANs.

Ademais, com relação ao tempo de execução, não houve mudanças significativas.

4.2 *DROPOUT* NA GAN

De acordo com o apresentado na seção 3.5.1, a acurácia do discriminador da GAN permaneceu com valor alto durante todo o treinamento, o que gerou a hipótese de que o discriminador estava aprendendo mais rápido que o gerador. Não obstante, observamos que a GAN era a única rede que não possuía camadas de *dropout* e, portanto, uma direção natural foi adicionar camadas desse tipo no discriminador da GAN. Da mesma forma, a fim de analisar também o impacto do uso de camadas de *dropout* nos outros modelos estudados, removemos das arquiteturas do CGAN e DCGAN as camadas desse tipo.

Modelo	Com Dropout		Sem Dropout	
	CrossLID	Tempo	CrossLID	Tempo
GAN	4.59 ± 0.04	3386 ± 248	30.53 ± 15.05	3409 ± 318
DCGAN	4.52 ± 0.04	7487 ± 2328	4.42 ± 0.01	5714 ± 230
CGAN	3.91 ± 0.01	3941 ± 473	3.85 ± 0.01	3536 ± 161

Tabela 5 – Avaliação de cada modelo no cenário da GAN, CGAN e DCGAN com *dropout* e sem *dropout*, considerando o CrossLID e o tempo de execução total das etapas de treinamento e avaliação. Os valores referentes ao tempo de execução estão em segundos, e todos são apresentados como a média ± o desvio padrão.

Figura 29 – Acurácia do discriminador da GAN

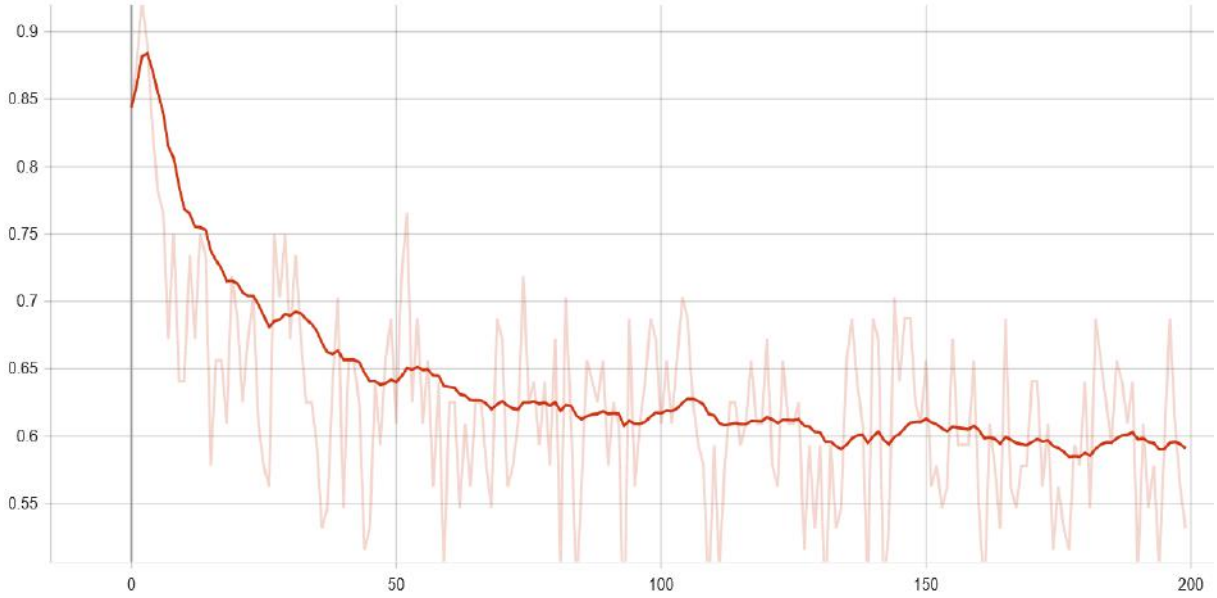


Gráfico de um experimento contendo a acurácia do discriminador do modelo GAN, após adição de camadas de *dropout* na arquitetura inicial. As linhas claras representam os valores reais e as linhas escuras os valores após aplicação do fator de suavização.

Figura 30 – Acurácia do discriminador da CGAN

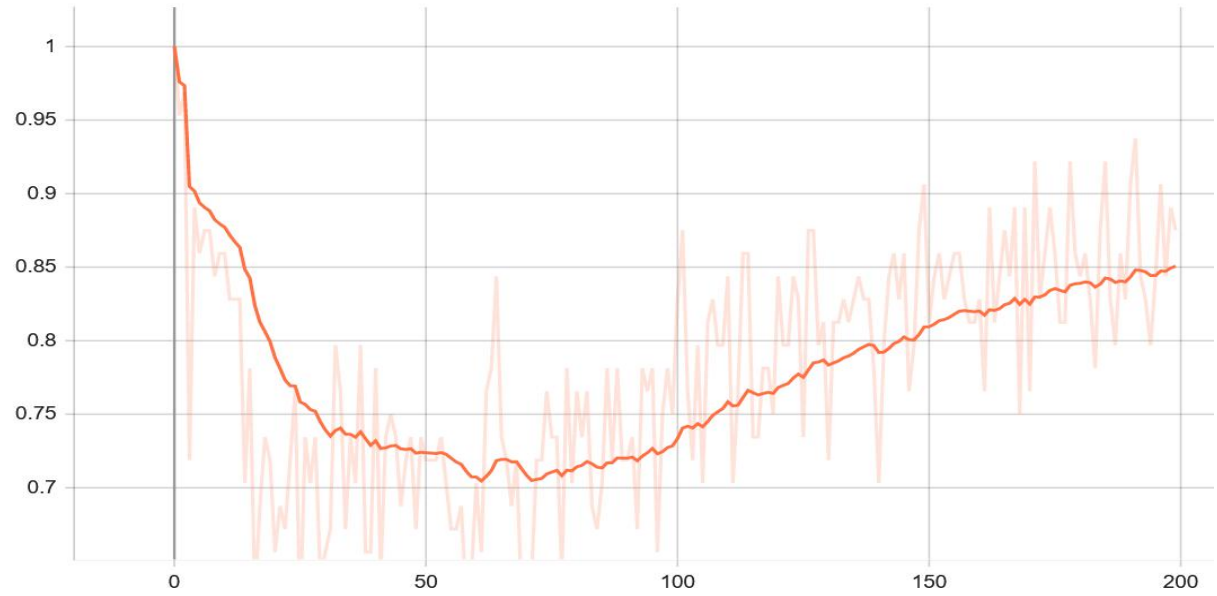


Gráfico de um experimento contendo a acurácia do discriminador do modelo CGAN, após remoção das camadas de *dropout* da arquitetura inicial. As linhas claras representam os valores reais e as linhas escuras os valores após aplicação do fator de suavização.

Figura 31 – Acurácia do discriminador da DCGAN

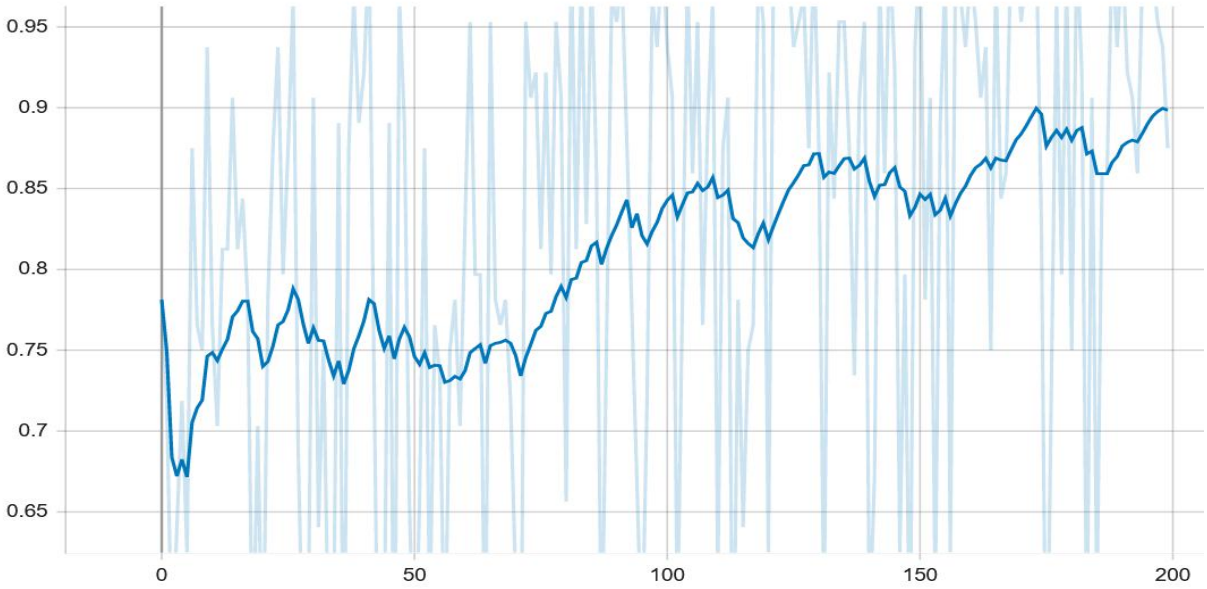


Gráfico de um experimento contendo a acurácia do discriminador do modelo DCGAN, após remoção das camadas de *dropout* da arquitetura inicial. As linhas claras representam os valores reais e as linhas escuras os valores após aplicação do fator de suavização.

O resultado obtido para o treinamento dos modelos, após essas modificações, é explicitado na Tabela 5. Podemos observar que o impacto da adição de camadas de *dropout* na GAN foi muito expressivo, fazendo com que seu valor médio de CrossLID fosse reduzido para 4.59, bem abaixo do valor de 30.53 obtido nos experimentos base (vide Tabela 19). Similarmente, a acurácia do discriminador, como apresentado na Figura 29, apresentou uma queda considerável se comparado ao resultado inicial exibido na Figura 20, passando de 0.94 para 0.6 na iteração final do treinamento. O tempo de execução total não apresentou mudanças relevantes.

No caso dos modelos de CGAN e DCGAN, a ausência das camadas de *dropout* não afetou de maneira significativa os valores de CrossLID se comparado aos resultados iniciais. A acurácia do discriminador, no entanto, passou a apresentar valores muito maiores do que havíamos registrado inicialmente nas Figuras 21 e 22. Como é possível observar a partir da comparação com os novos resultados expressos nas Figuras 30 e 31, os valores de acurácia saltaram de valores iniciais de 0.65 na CGAN e 0.6 na DCGAN – na iteração 200 do treinamento – para 0.85 e 0.9, respectivamente.

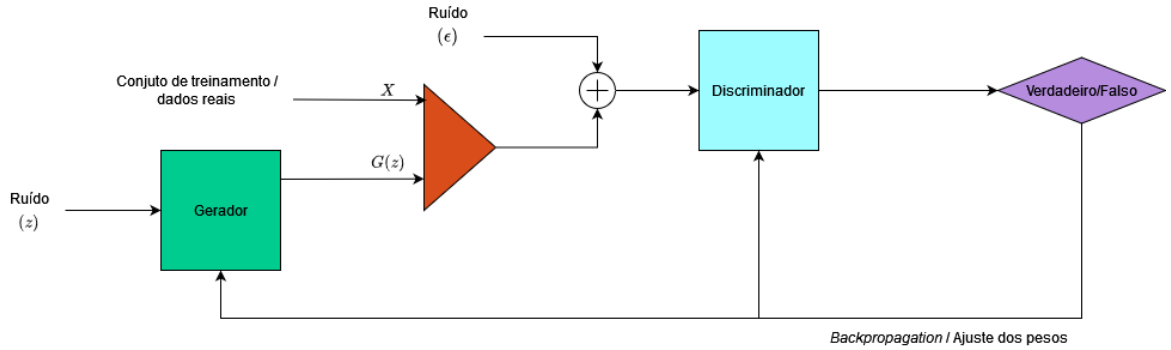
Isso nos sugere que nem sempre é possível correlacionar a acurácia do discriminador ao resultado da métrica de CrossLID visto que, mesmo com uma acurácia do discriminador altíssima – se comparado ao novo resultado do modelo GAN – ainda foi possível que os modelos generativos da CGAN e DCGAN produzissem resultados finais melhores quantitativa e qualitativamente.

4.3 ADIÇÃO DE RUÍDO NO MODELO DISCRIMINATIVO

O artigo (ARJOVSKY; BOTTOU, 2017) sugere que a adição de ruído às entradas do modelo discriminativo D de uma GAN resulta em um treinamento mais estável, na medida em que torna a tarefa de D mais díficil e, assim, permite um desempenho mais regular do modelo generativo. Para verificar se essa estabilidade adicional no treinamento seria benéfica para o resultado final produzido pelos modelos, aplicamos essa modificação em cada tipo de GAN estudado, adicionando ruído tanto nas imagens geradas pelo modelo generativo G quanto nas imagens reais, antes que elas fossem utilizadas como entrada para o modelo discriminativo D. Dessa forma, D passou a receber como entrada versões levemente modificadas das imagens selecionadas originalmente da base de dados ou geradas por G.

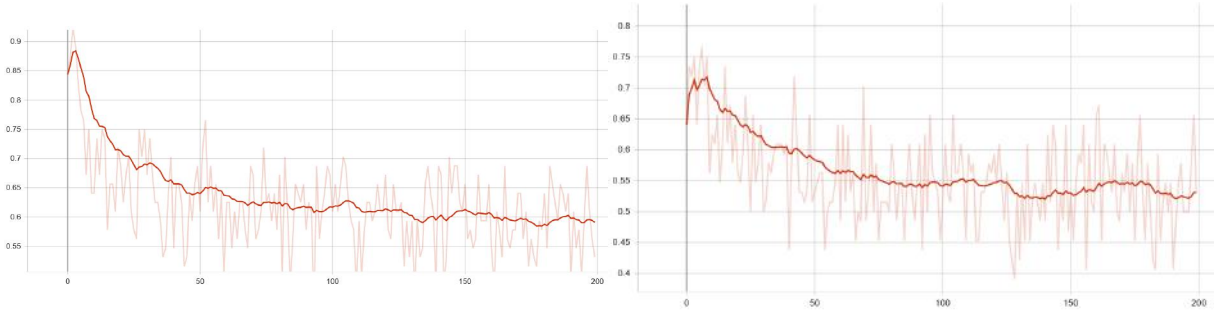
Para gerar esse valor de ruído que seria somado aos vetores de tamanho 784 que, por sua vez, representam as imagens, utilizamos vetores de mesmo tamanho cujos elementos foram amostrados aleatoriamente a partir de uma distribuição normal padrão. A Figura 32 ilustra essa modificação, para o caso do primeiro tipo de GAN apresentado, onde um novo valor de ruído ϵ é adicionado à entrada do discriminador.

Figura 32 – Arquitetura da GAN com adição de ruído nas entradas do discriminador



Como já era esperado, essa modificação afetou negativamente a performance do modelo discriminativo, tornando sua acurácia mais baixa ao longo de todo o treinamento em comparação aos experimentos iniciais. Além disso, as curvas de perdas dos três tipos de GAN estudados foram mais suaves – principalmente na GAN e CGAN – e alcançaram, ao fim do treinamento, valores menores para o modelo generativo e maiores para o modelo discriminativo, o que também é um resultado desejável. Apesar disso, em termos quantitativos, não houve alterações significativas nos valores da principal métrica que utilizamos, o CrossLID (vide Tabela 6). Similarmente, em termos qualitativos, não foi possível observar mudanças expressivas nas imagens produzidas pelo modelo generativo ao fim do treinamento. Os experimentos foram feitos com uso de *dropout* nos três modelos.

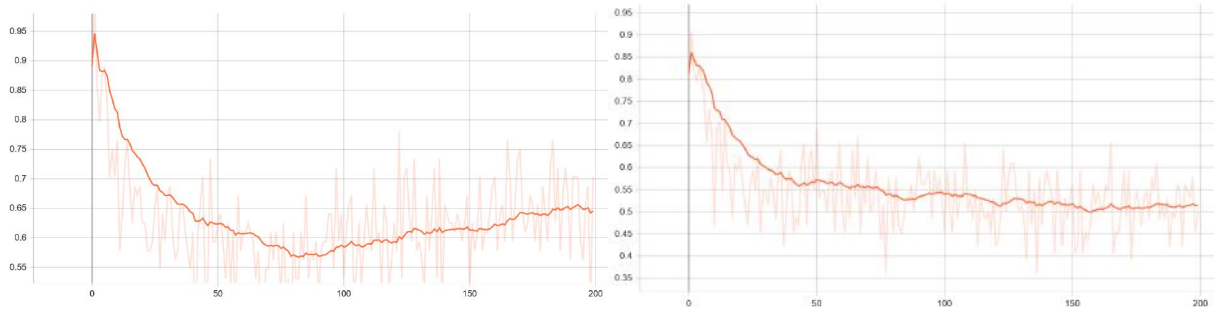
Figura 33 – Acurácia do discriminador da GAN



(a) (b)

Gráficos de experimentos onde (a) representa o valor da acurácia do discriminador da GAN por época sem a adição de ruído nas entradas, enquanto que (b) representa o mesmo valor após a adição de ruído.

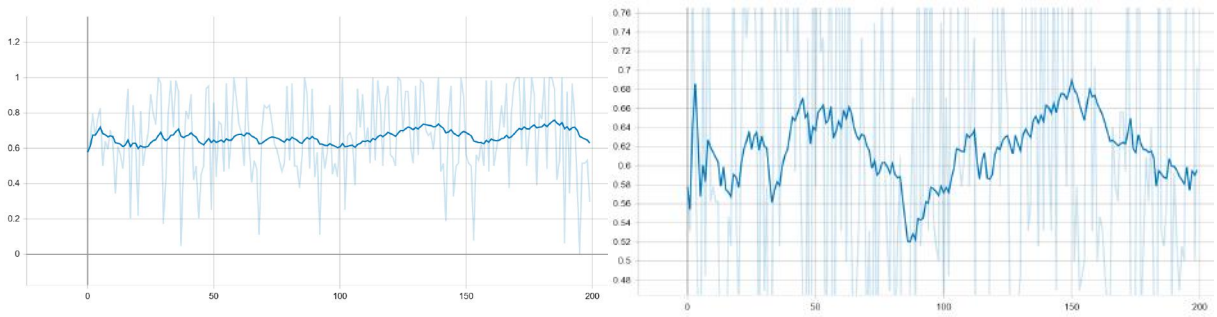
Figura 34 – Acurácia do discriminador da CGAN



(a) (b)

Gráficos de experimentos onde (a) representa o valor da acurácia do discriminador da CGAN por época sem a adição de ruído nas entradas, enquanto que (b) representa o mesmo valor após a adição de ruído.

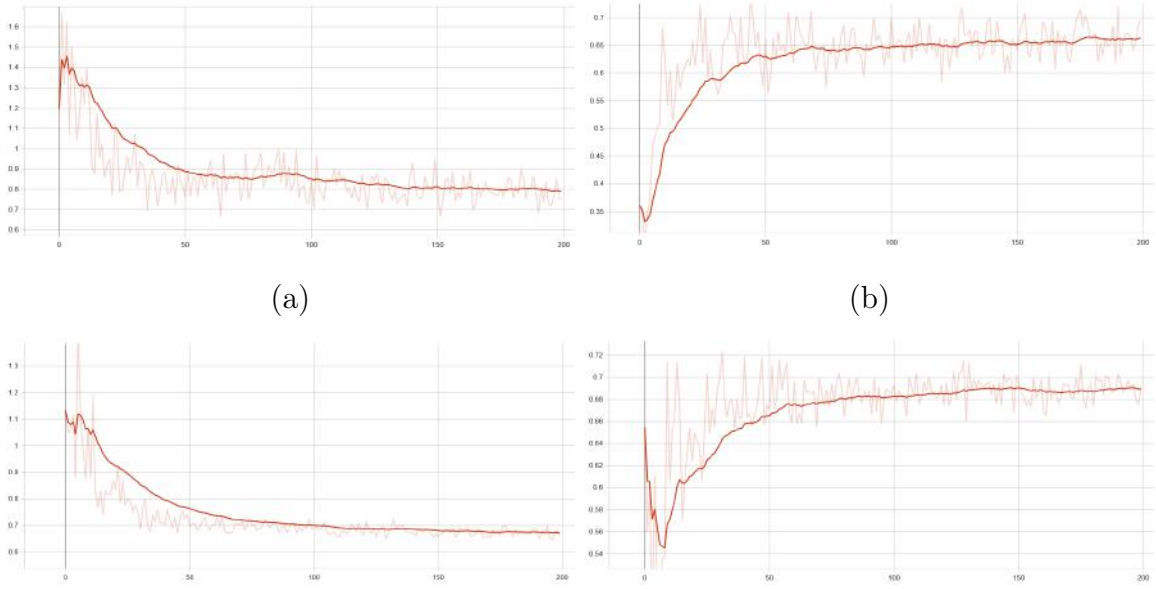
Figura 35 – Acurácia do discriminador da DCGAN



(a) (b)

Gráficos de experimentos onde (a) representa o valor da acurácia do discriminador da DCGAN por época sem a adição de ruído nas entradas, enquanto que (b) representa o mesmo valor após a adição de ruído.

Figura 36 – Perdas da GAN

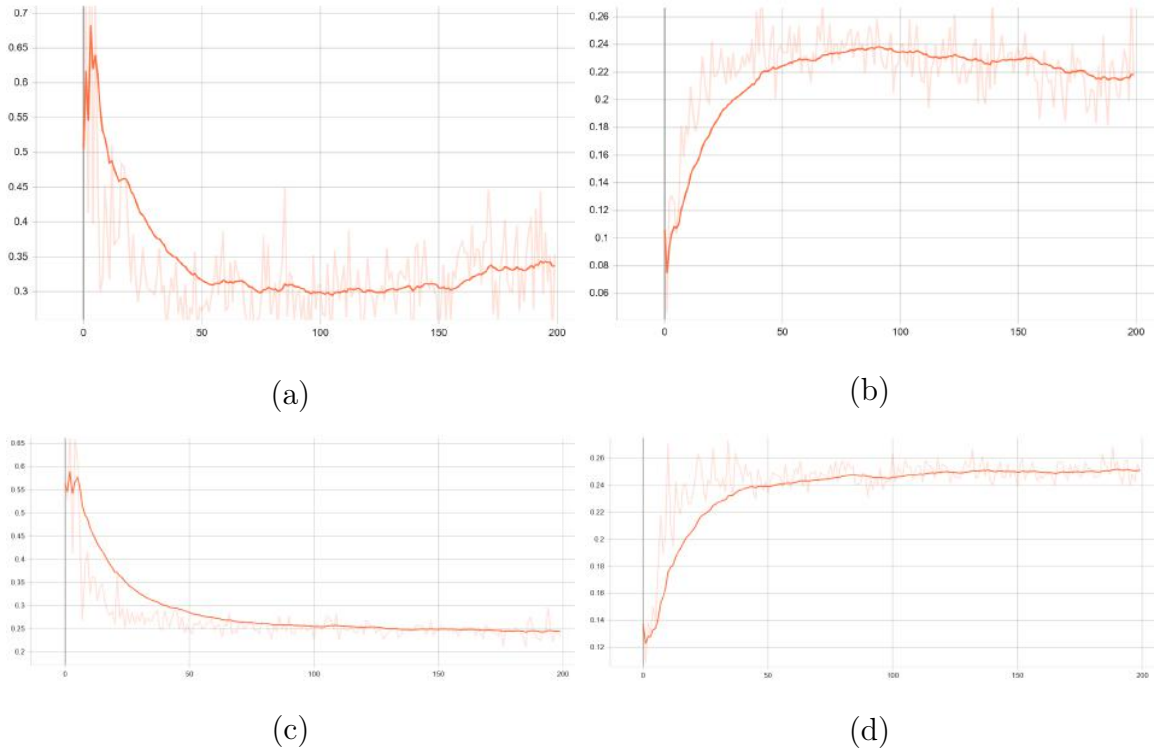


(a) (b)

(c) (d)

Gráficos de experimentos onde (a) e (b) representam os valores da perda do gerador e discriminador da CGAN por época, respectivamente, sem a adição de ruído nas entradas, enquanto que (c) e (d) representam os mesmos valores após a adição de ruído.

Figura 37 – Perdas da CGAN

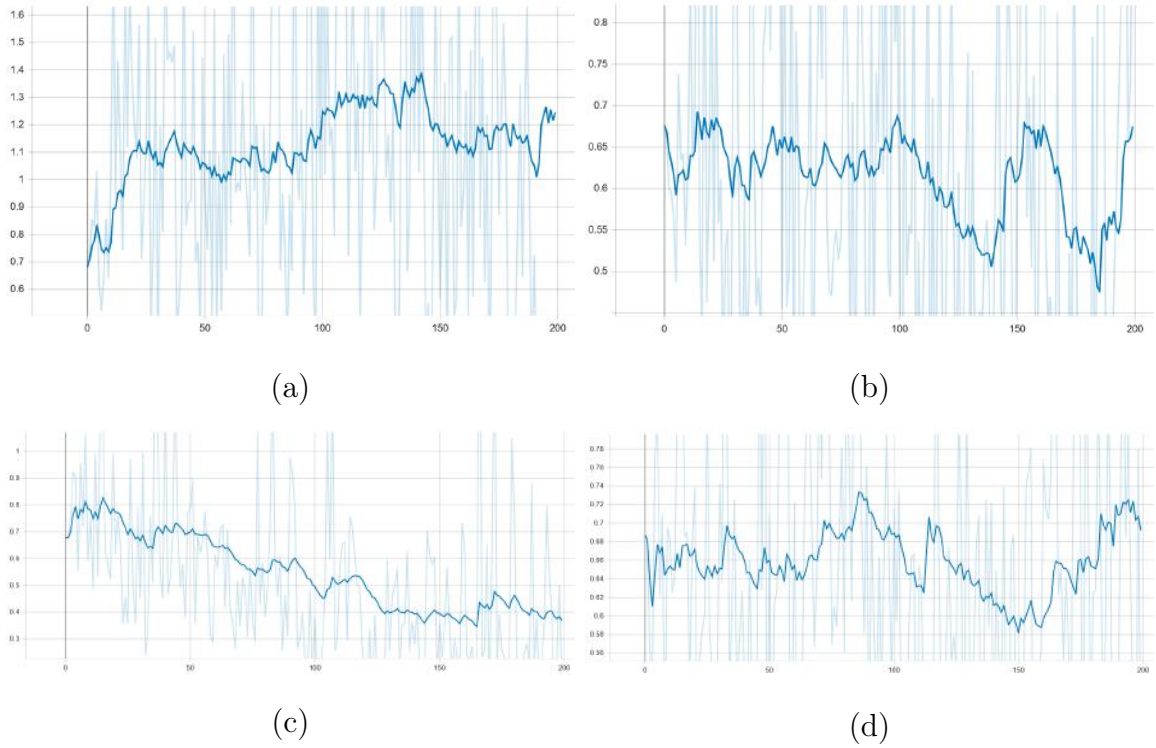


(a) (b)

(c) (d)

Gráficos de experimentos onde (a) e (b) representam os valores da perda do gerador e discriminador da CGAN por época, respectivamente, sem a adição de ruído nas entradas, enquanto que (c) e (d) representam os mesmos valores após a adição de ruído.

Figura 38 – Perdas da DCGAN



(a) (b)

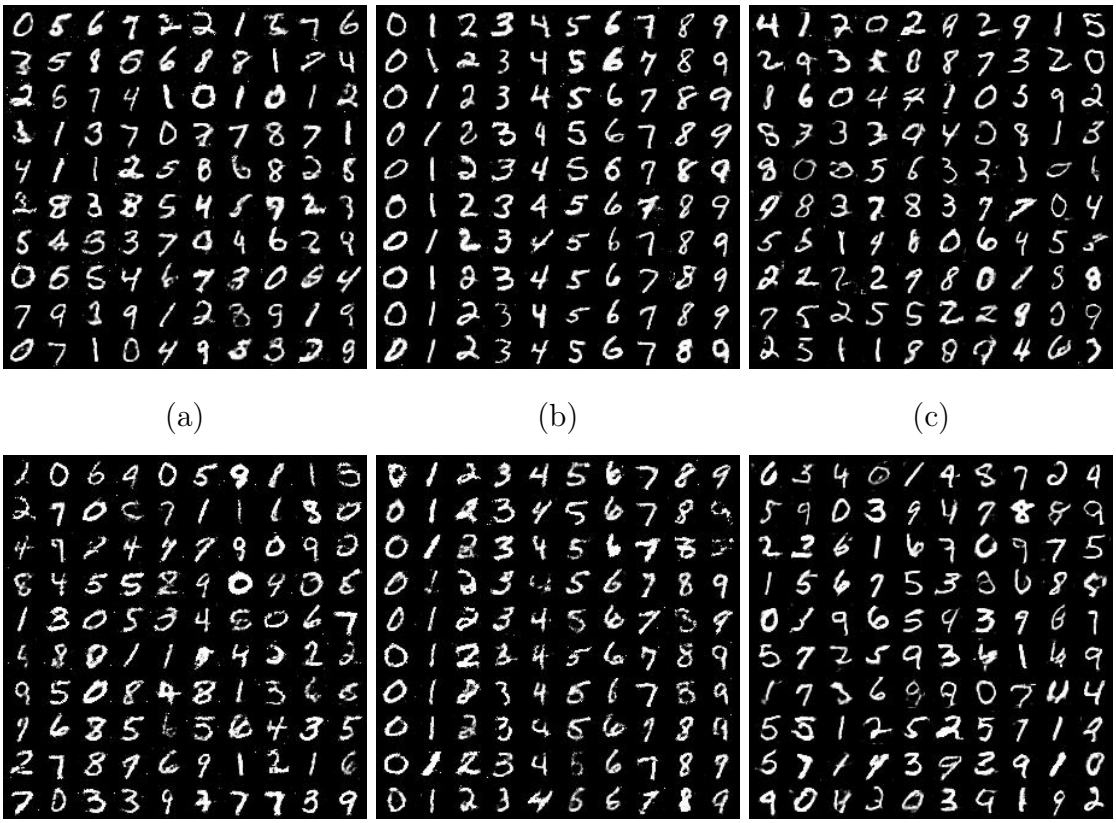
(c) (d)

Gráficos de experimentos onde (a) e (b) representam os valores da perda do gerador e discriminador da DCGAN por época, respectivamente, sem a adição de ruído nas entradas, enquanto que (c) e (d) representam os mesmos valores após a adição de ruído.

Modelo	Sem ruído no discriminador		Com ruído no discriminador	
	CrossLID	Tempo	CrossLID	Tempo
GAN	4.59 ± 0.04	3386 ± 248	4.68 ± 0.27	3505 ± 206
DCGAN	4.52 ± 0.04	7487 ± 2328	4.51 ± 0.03	5528 ± 91.85
CGAN	3.91 ± 0.01	3941 ± 473	4.2 ± 0.25	3709 ± 184

Tabela 6 – Avaliação de cada modelo no cenário da GAN, CGAN e DCGAN com e sem adição de ruído nas entradas do discriminador, considerando o CrossLID e o tempo de execução total das etapas de treinamento e avaliação. Os valores referentes ao tempo de execução estão em segundos, e todos são apresentados como a média ± o desvio padrão. Os experimentos foram feitos com uso de *dropout* nos três modelos.

Figura 39 – Imagens geradas



(a) (b) (c)

(d) (e) (f)

Imagens produzidas pelos geradores da GAN, CGAN e DCGAN, respectivamente, ao fim do processo de treinamento. (a), (b) e (c) não tiveram ruído adicionado nas entradas dos discriminadores, enquanto que (d), (e) e (f) tiveram.

4.4 CONCLUSÃO

Neste capítulo, apresentamos as modificações que fizemos nas arquiteturas e nos parâmetros dos tipos de GAN estudados, assim como os resultados obtidos após nova realização dos experimentos expostos no capítulo 3, já com as mudanças propostas. Basicamente, as alterações foram focadas em três aspectos: o número de dimensões do vetor latente ou de ruído utilizado como entrada, a utilização ou não de camadas de *dropout* e, por fim, a adição de ruído nas entradas do modelo discriminativo. Dessas três modificações, as duas primeiras apresentaram ganho significativo para o primeiro tipo de GAN estudado, assim como um ganho menos expressivo para CGAN e DCGAN, de acordo com os valores escolhidos para os paramêtros analisados. Já a última, apesar de ter o efeito que esperávamos de melhora na estabilidade do treinamento, não trouxe ganhos significativos para nenhum dos tipos de GAN estudados.

5 CONCLUSÃO

Nesse trabalho, foi apresentado um estudo comparativo entre três tipos de redes adversárias generativas: GAN, CGAN e DCGAN. Abordamos os conhecimentos básicos e trouxemos um panorama geral a respeito do funcionamento de redes neurais, incluindo as redes adversárias generativas e suas arquiteturas. Em seguida, denotamos os experimentos comparativos que serviram de base para o projeto. Nessa parte, pudemos perceber um melhor desempenho da CGAN e DCGAN, se comparadas à GAN tradicional, tendo como base a métrica quantitativa CrossLID, além de analisar o grau de aprendizado do discriminador por meio da acurácia, do valor da função de perda e de matrizes de confusão ao longo do treinamento, assim como pela análise visual das imagens geradas.

Por fim, foram feitas mudanças às redes com base em artigos e intuições provenientes dos experimentos anteriores. Algumas dessas mudanças, como o uso de camadas de *dropout* na GAN e o redimensionamento do vetor latente, melhoraram o desempenho das redes enquanto que a adição de ruído nos discriminadores não teve efeito relevante.

A respeito do desenvolvimento do projeto, além das dificuldades habituais a projetos de aprendizado de máquina, como o tempo de treinamento elevado e a necessidade de utilização de uma GPU, tivemos que usar tecnologias que nos eram desconhecidas previamente. O maior ponto de conflito, nesse sentido, foi o de integrar o cálculo do CrossLID com o restante do código, uma vez que o código está em Pytorch enquanto que o cálculo usa Tensorflow (ABADI et al., 2016). Assim, foi necessário converter os pesos para formato ONNX para fazer a interface entre os diferentes *frameworks*.

Para trabalhos e experimentos futuros o primeiro ponto é alterar a quantidade de camadas nas redes. Como abordado, a quantidade de camadas em uma rede condiz com a complexidade e capacidade de abstração de *features* daquela rede. Dessa forma, um estudo lógico seria abordar de que forma a variação no número de camadas pode ajudar ou piorar cada rede.

Outra alternativa possível é avaliar o desempenho das redes quando o discriminador é treinado em uma frequência menor do que o gerador. Assim como quando adicionamos ruído nos modelos discriminativos, espera-se que o gerador consiga alcançar um melhor desempenho, dado que uma nova dificuldade será introduzida para o discriminador.

Além disso, propomos um estudo que avalie o número de épocas de treinamento a ser utilizado, a fim de melhor verificar o impacto que o aumento ou a diminuição desse hiperparâmetro pode ter no desempenho final das redes.

Finalmente, outra possibilidade é gerar métricas utilizando IS e FID para compará-las ao CrossLID e aos resultados visuais para verificar que o CrossLID foi de fato a melhor métrica para o domínio do nosso trabalho.

REFERÊNCIAS

ABADI, M. et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. **CoRR**, abs/1603.04467, 2016. Disponível em: http://arxiv.org/abs/1603.04467.

ABIODUN, O. I. et al. State-of-the-art in artificial neural network applications: A survey. **Heliyon**, v. 4, n. 11, p. e00938, 2018. ISSN 2405-8440. Disponível em: https://www.sciencedirect.com/science/article/pii/S2405844018332067.

ABIRAMI, S.; CHITRA, P. Chapter fourteen - energy-efficient edge based real-time healthcare support system. In: RAJ, P.; EVANGELINE, P. (Ed.). **The Digital Twin Paradigm for Smarter Systems and Environments: The Industry Use Cases**. Elsevier, 2020, (Advances in Computers, 1). p. 339–368. Disponível em: https://www.sciencedirect.com/science/article/pii/S0065245819300506.

AGARAP, A. F. Deep learning using rectified linear units (relu). **CoRR**, abs/1803.08375, 2018. Disponível em: http://arxiv.org/abs/1803.08375.

ARJOVSKY, M.; BOTTOU, L. Towards principled methods for training generative adversarial networks. arXiv, 2017. Disponível em: https://arxiv.org/abs/1701.04862.

BARUA, S. et al. Quality evaluation of gans using cross local intrinsic dimensionality. arXiv, 2019. Disponível em: https://arxiv.org/abs/1905.00643.

BIN, H. et al. High-quality face image SR using conditional generative adversarial networks. **CoRR**, abs/1707.00737, 2017. Disponível em: http://arxiv.org/abs/1707.00737.

BORJI, A. Pros and cons of gan evaluation measures. **Computer Vision and Image Understanding**, v. 179, p. 41–65, 2019. ISSN 1077-3142. Disponível em: https://www.sciencedirect.com/science/article/pii/S1077314218304272.

BROCK, A.; DONAHUE, J.; SIMONYAN, K. Large scale gan training for high fidelity natural image synthesis. arXiv, 2018. Disponível em: https://arxiv.org/abs/1809.11096.

BRUNDAGE, M. et al. The malicious use of artificial intelligence: Forecasting, prevention, and mitigation. arXiv, 2018. Disponível em: https://arxiv.org/abs/1802.07228.

CHOLLET, F. et al. **Keras**. GitHub, 2015. Disponível em: https://github.com/fchollet/keras.

DENG, J. et al. Imagenet: A large-scale hierarchical image database. In: IEEE. **2009 IEEE conference on computer vision and pattern recognition**. [S.l.], 2009. p. 248–255.

DENG, L. The mnist database of handwritten digit images for machine learning research. **IEEE Signal Processing Magazine**, IEEE, v. 29, n. 6, p. 141–142, 2012.

DONG, H. W.; YANG, Y. H. Towards a deeper understanding of adversarial losses. **CoRR**, abs/1901.08753, 2019. Disponível em: http://arxiv.org/abs/1901.08753.

DREISEITL, S.; OHNO-MACHADO, L. Logistic regression and artificial neural network classification models: a methodology review. **Journal of biomedical informatics**, Elsevier, v. 35, n. 5-6, p. 352–359, 2002.

DU, K.-L. Clustering: A neural network approach. **Neural networks**, Elsevier, v. 23, n. 1, p. 89–107, 2010.

EL_JERJAWI, N. S.; ABU-NASER, S. S. Diabetes prediction using artificial neural network. **International Journal of Advanced Science and Technology**, v. 121, 2018.

FOSTER, D. **Generative Deep Learning**. [S.l.]: O'Reilly Media, Inc., 2019. ISBN 9781492041948.

FRIZZI, S. et al. Convolutional neural network for video fire and smoke detection. In: IEEE. **IECON 2016-42nd Annual Conference of the IEEE Industrial Electronics Society**. [S.l.], 2016. p. 877–882.

GOODFELLOW, I. J. et al. Generative adversarial networks. arXiv, 2014. Disponível em: https://arxiv.org/abs/1406.2661.

GURNEY, K. **An introduction to neural networks**. Londres e Nova Iorque: UCL Press, 1997.

HASSAN, H. et al. Assessment of artificial neural network for bathymetry estimation using high resolution satellite imagery in shallow lakes: Case study el burullus lake. **International Water Technology Journal**, v. 5, 12 2015.

HEUSEL, M. et al. Gans trained by a two time-scale update rule converge to a nash equilibrium. **CoRR**, abs/1706.08500, 2017. Disponível em: http://arxiv.org/abs/1706.08500.

HOULE, M. E. Dimensionality, discriminability, density and distance distributions. In: **2013 IEEE 13th International Conference on Data Mining Workshops**. [S.l.: s.n.], 2013. p. 468–473.

IOFFE, S.; SZEGEDY, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv, 2015. Disponível em: https://arxiv.org/abs/1502.03167.

ISOLA, P. et al. Image-to-image translation with conditional adversarial networks. **CoRR**, abs/1611.07004, 2016. Disponível em: http://arxiv.org/abs/1611.07004.

KARRAS, T. et al. Progressive growing of gans for improved quality, stability, and variation. **CoRR**, abs/1710.10196, 2017. Disponível em: http://arxiv.org/abs/1710.10196.

LEDIG, C. et al. Photo-realistic single image super-resolution using a generative adversarial network. **CoRR**, abs/1609.04802, 2016. Disponível em: http://arxiv.org/abs/1609.04802.

LI, J.; SONG, G.; ZHANG, M. Occluded offline handwritten chinese character recognition using deep convolutional generative adversarial network and improved googlenet. **Neural Computing and Applications**, v. 32, 05 2020.

MANISHA, P.; DAS, D.; GUJAR, S. Effect of input noise dimension in gans. **CoRR**, abs/2004.06882, 2020. Disponível em: https://arxiv.org/abs/2004.06882.

MARIANI, G. et al. BAGAN: data augmentation with balancing GAN. **CoRR**, abs/1803.09655, 2018. Disponível em: http://arxiv.org/abs/1803.09655.

MAZZINI, D. Guided upsampling network for real-time semantic segmentation. **CoRR**, abs/1807.07466, 2018. Disponível em: http://arxiv.org/abs/1807.07466.

MIRZA, M.; OSINDERO, S. Conditional generative adversarial nets. **CoRR**, abs/1411.1784, 2014. Disponível em: http://arxiv.org/abs/1411.1784.

PASZKE, A. et al. Pytorch: An imperative style, high-performance deep learning library. In: WALLACH, H. et al. (Ed.). **Advances in Neural Information Processing Systems 32**. Curran Associates, Inc., 2019. p. 8024–8035. Disponível em: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

PEDREGOSA, F. et al. Scikit-learn: Machine learning in Python. **Journal of Machine Learning Research**, v. 12, p. 2825–2830, 2011.

POPESCU, M.-C. et al. Multilayer perceptron and neural networks. **WSEAS Transactions on Circuits and Systems**, v. 8, 07 2009.

RADFORD, A.; METZ, L.; CHINTALA, S. Unsupervised representation learning with deep convolutional generative adversarial networks. **CoRR**, abs/1511.06434, 2016.

SALIMANS, T. et al. Improved techniques for training gans. In: LEE, D. et al. (Ed.). **Advances in Neural Information Processing Systems**. Curran Associates, Inc., 2016. v. 29. Disponível em: https://proceedings.neurips.cc/paper/2016/file/8a3363abe792db2d8761d6403605aeb7-Paper.pdf.

SHALEV-SHWARTZ, S. **Perceptron Algorithm**. Boston, MA: Springer US, 2008. 642–644 p. ISBN 978-0-387-30162-4. Disponível em: https://doi.org/10.1007/978-0-387-30162-4_287.

SHI, W. et al. Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network. In: **Proceedings of the IEEE conference on computer vision and pattern recognition**. [S.l.: s.n.], 2016. p. 1874–1883.

SRIVASTAVA, N. et al. Dropout: A simple way to prevent neural networks from overfitting. **J. Mach. Learn. Res.**, JMLR.org, v. 15, n. 1, p. 1929–1958, jan 2014. ISSN 1532-4435.

SULTANA, F.; SUFIAN, A.; DUTTA, P. Advancements in image classification using convolutional neural network. In: IEEE. **2018 Fourth International Conference on Research in Computational Intelligence and Communication Networks (ICRCICN)**. [S.l.], 2018. p. 122–129.

TANAKA, F. H. K. dos S.; ARANHA, C. Data augmentation using gans. **CoRR**, abs/1904.09135, 2019. Disponível em: http://arxiv.org/abs/1904.09135.

THEIS, L.; OORD, A. van den; BETHGE, M. A note on the evaluation of generative models. **CoRR**, abs/1511.01844, 2016.

VAKILI, M.; GHAMSARI, M.; REZAEI, M. Performance analysis and comparison of machine and deep learning algorithms for iot data classification. **CoRR**, abs/2001.09636, 2020. Disponível em: https://arxiv.org/abs/2001.09636.

XU, J. et al. Reluplex made more practical: Leaky relu. In: IEEE. **2020 IEEE Symposium on Computers and Communications (ISCC)**. [S.l.], 2020. p. 1–7.

ZHANG, A. et al. Dive into deep learning. **arXiv preprint arXiv:2106.11342**, 2021.

ZHANG, H. et al. Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks. **CoRR**, abs/1612.03242, 2016. Disponível em: http://arxiv.org/abs/1612.03242.

ZHU, J. et al. Unpaired image-to-image translation using cycle-consistent adversarial networks. **CoRR**, abs/1703.10593, 2017. Disponível em: http://arxiv.org/abs/1703.10593.