APLICAÇÃO DE REDES NEURAIS GENERATIVAS ADVERSARIAIS NA GERAÇÃO DE IMAGENS COM EMOÇÕES

Murilo Filheiro de Paula*
Prof. Dr. Alessandro Bof de Oliveira**

RESUMO

Este trabalho aborda o desenvolvimento de uma Rede Neural Generativa Adversarial (GAN) voltada à geração de imagens sintéticas de rostos humanos expressando emoções básicas, tais como felicidade, tristeza, raiva, surpresa, medo e estado neutro. O contexto da pesquisa está relacionado à crescente demanda por modelos virtuais mais realistas e pela necessidade de bases de dados equilibradas para treinamento de sistemas de reconhecimento facial. A motivação principal reside na dificuldade de obtenção de grandes volumes de imagens emocionais balanceadas, o que compromete a qualidade e a generalização dos modelos preditivos. Para justificar essa proposta, destaca-se que as GANs se consolidaram como técnicas eficazes na criação de imagens fotorrealistas, utilizando um gerador que aprende a produzir amostras e um discriminador responsável por distinguir entre dados reais e sintéticos. O objetivo central foi desenvolver uma arquitetura de GAN capaz de gerar rostos artificiais que expressem emoções específicas, com potencial aplicação na criação de avatares virtuais e no aprimoramento de bancos de dados. Como principais resultados, a rede implementada, após ajustes de hiperparâmetros, validação preliminar com Fashion MNIST e treinamento com a base KDEF, demonstrou capacidade progressiva de síntese de imagens realistas, embora ainda apresente limitações em termos de fidelidade e diversidade visual. O trabalho reforça a viabilidade do uso de GANs nesse contexto e indica caminhos para futuros aprimoramentos.

Palavras-chaves: Rede Neural Generativa; Emoções Faciais; Banco de Dados; Avatares Virtuais.

ABSTRACT

This work presents the development of a Generative Adversarial Network (GAN) designed to synthesize images of human faces displaying basic emotions, such as happiness, sadness, anger, surprise, fear, and neutral. The study is situated in the context of growing demand for more realistic virtual models and the challenge of assembling balanced emotional datasets for facial recognition training. The primary motivation lies in the difficulty of obtaining large volumes of evenly distributed emotional images, which undermines model quality and generalization. GANs have proven effective for photorealistic image creation by pitting a generator, which produces synthetic samples, against a discriminator, which distinguishes real from generated data. The main objective was to implement a GAN architecture capable of generating emotion-specific artificial faces,

*Aluno do Curso de Ciência da Computação da Universidade Federal do Pampa, Alegrete, Rio Grande do Sul, Brasil
E-mail: murilopaula.aluno@unipampa.edu.br
**Orientador, Professor do Curso de Ciência da Computação da Universidade Federal do Pampa, Alegrete, Rio Grande do Sul, Brasil, E-mail: alessandrooliveira@unipampa.edu.br

with applications in virtual avatar creation and dataset augmentation. Key results indicate that, following hyperparameter tuning, preliminary validation on Fashion MNIST, and training on the KDEF database, the network progressively synthesized increasingly realistic images, albeit with remaining limitations in fidelity and visual diversity. This work confirms the feasibility of GAN-based emotional face generation and outlines directions for future enhancements.

Keywords: Generative Adversarial Network; Facial Emotions; Dataset Augmentation; Virtual Avatars.

1. INTRODUÇÃO

Redes Neurais Generativas Adversariais (GANs) são modelos de redes neurais que recebem um ruído aleatório como entrada e geram saídas que se assemelham a amostras da distribuição de seu conjunto de treinamento (P et al., 2022). Elas são constituídas por dois componentes: um modelo gerador, que captura a distribuição do conjunto de treinamento e, a partir de um ruído aleatório, produz imagens realistas; e um modelo discriminador, que estima a probabilidade de uma amostra recebida ter vindo dos dados de treino (em vez de ter sido gerada pelo modelo gerador) (SHARMA, 2021).

Para treinar esses modelos, utiliza-se uma função conhecida como Perda de Entropia Cruzada Binária (perda BCE). Essa função calcula a diferença entre as previsões do modelo e os valores reais. O resultado desse cálculo corresponde à probabilidade de uma amostra de dados ser verídica (quando o valor da perda é próximo de 1) ou falsa (quando o valor é próximo de 0). O objetivo é minimizar essa função de perda durante o treinamento (GOODFELLOW et al., 2014).

No ramo das redes neurais, é comum se deparar com situações em que as amostras dos conjuntos de treinamento estão desequilibradas, ou seja, classes podem ter quantidades diferentes de amostras. Isso tende a fazer com que o classificador da rede produza resultados inferiores nas classes com menos representação. Como uma das classes tem uma base muito maior, o modelo treinado com dados desequilibrados pode apresentar alta precisão geral e, ainda assim, não prever corretamente nenhuma observação da classe minoritária, gerando uma falsa impressão de bom desempenho (MIOTO; MIRANDA; PREMEBIDA, 2023).

Além disso, o uso de GANs tem se tornado cada vez mais comum para auxiliar na criação de avatares em jogos online, simulações e ambientes de realidade virtual, como o Metaverso (PLATFORMS, 2021). No entanto, mesmo com diversas empresas treinando seus modelos para gerar imagens convincentes, muitos resultados acabam causando estranheza e fogem do padrão realista esperado (PEREZ, 2017).

1.1. PROBLEMA

Em um contexto que envolve a criação de avatares para simulações, realidade virtual ou jogos digitais, o uso de GANs pode ser crucial para gerar expressões faciais sintéticas, pois é essencial que esses avatares consigam expressar uma ampla variedade de emoções de maneira convincente e realista (WU et al., 2022).

Nesse sentido, tecnologias como *deepfakes* também são tipicamente gerados com redes neurais generativas (PASQUARELLI, 2019) e dependem de uma base de

dados diversa e de um treinamento extensivo, de modo que o conteúdo gerado seja visualmente realista e convincente.

Assim, o desenvolvimento de uma rede neural generativa adversarial capaz de gerar imagens sintéticas de rostos com expressões faciais contribuirá para que essas tecnologias produzam resultados sólidos e realistas (ROSADO; FERNáNDEZ; REVERTER, 2021).

1.2. OBJETIVOS

O objetivo geral deste trabalho é desenvolver uma Rede Neural Generativa Adversarial (GAN) capaz de gerar imagens sintéticas de rostos humanos que expressem seis emoções básicas — felicidade, tristeza, raiva, surpresa, medo e estado neutro — com foco na melhoria de realismo em avatares virtuais e no balanceamento de bases de dados emocionais.

Para alcançar esse objetivo, definem-se os seguintes objetivos específicos:

1. Realizar um levantamento bibliográfico de redes neurais generativas voltadas à síntese de expressões faciais, com foco em GANs.
2. Desenvolver uma arquitetura de GAN capaz de gerar rostos com as seis emoções básicas.
3. Implementar subclasses especializadas por emoção (uma por classe), otimizando o processo de treinamento.
4. Criar uma interface que permita a seleção da emoção desejada para geração de imagens.
5. Balancear o banco de dados de treinamento, garantindo distribuição equitativa de imagens por emoção.
6. Avaliar a qualidade das imagens geradas, discutindo seu impacto potencial na criação de avatares para jogos, modelos 3D e realidade virtual.

1.3. ORGANIZAÇÃO

Este Trabalho de Conclusão de Curso está organizado em cinco seções. A Seção 1 apresenta a introdução, contextualizando o tema, definindo o problema, justificando a proposta e estabelecendo os objetivos do estudo.

Na Seção 2, são discutidos trabalhos relacionados, com ênfase em abordagens e arquiteturas de redes neurais utilizadas para síntese de rostos com emoções. O levantamento serve como base para embasar as decisões técnicas adotadas no projeto.

A Seção 3 descreve a metodologia aplicada, incluindo as ferramentas utilizadas, os conjuntos de dados empregados, os ajustes de hiperparâmetros e a arquitetura da GAN desenvolvida.

A Seção 4 apresenta os resultados obtidos com os experimentos, tanto preliminares quanto com o conjunto de dados emocional, além de discutir os avanços obtidos, limitações e desafios identificados durante a implementação.

Por fim, a Seção 5 traz as considerações finais, destacando as contribuições do trabalho e propondo direções para a continuidade da pesquisa em trabalhos futuros.

Desse modo, o trabalho apresenta uma estrutura organizada, revelando uma análise teórica aprofundada e detalhes práticos do desenvolvimento da rede, que resultam em uma avaliação crítica e criteriosa das contribuições do projeto e das atividades futuras planejadas.

2. ARTIGOS RELACIONADOS

Com o intuito de pesquisar trabalhos relacionados que apresentem o desenvolvimento de uma rede neural generativa adversarial para gerar imagens de emoções humanas, realizou-se uma ampla pesquisa em plataformas como Google Scholar (Google Scholar, 2025) e arXiv (ARXIV,). Para esquematizar a busca foram utilizados os termos "*emotional faces*" e "*generative adversarial networks*". Dentre os trabalhos encontrados, foram selecionados aqueles que apresentam a implementação de uma rede neural, não necessariamente uma GAN, para a geração de imagens.

O trabalho de Sinha, Biswas, Yadav e Bhomwick (SINHA et al., 2022) apresenta um novo método que gera rostos falantes controláveis com emoções. É utilizado uma rede neural convolucional gráfica, que incorpora recursos de conteúdo de fala e uma entrada de emoção independente para criar emoções e movimentos induzidos pela fala. A metodologia utilizada inclui uma rede óptica de geração de textura guiada por fluxo com ramificações de movimento e de textura, que são projetadas para considerar o movimento e textura de forma independente, aprimorando o realismo das texturas geradas. Os resultados apresentavam flexibilização e generalização da abordagem, mostrando que o método era capaz de se adaptar a diversos rostos, ajustando com apenas uma imagem de uma pessoa em expressão neutra.

O trabalho de Yueqiao, Wenxia, Xi e Huan (CHEN et al., 2023) apresenta um novo modelo de conclusão de imagem baseado em GAN controlado por emoções faciais, projetado para inferir e personalizar emoções faciais. É proposto um módulo de inferência de emoções para inferir expressões faciais com base em regiões não mascaradas, treinando de forma supervisionada para identificar e isolar os elementos relevantes para reconhecer emoções em dados, mesmo quando partes do rosto estão ocultas. É proposto também um módulo de controle de emoções, que permite o ajuste das emoções iniciais às desejadas. Essa rede neural foi avaliada em dois conjuntos de dados faciais, Celeba-HQ e CFEED, mostrando a capacidade de produzir imagens com diversas expressões, até mesmo quando os rostos estão sendo ocultados por algum objeto.

Entretanto, o modelo proposto possui alguns aspectos negativos. Dentre eles, destaca-se a possibilidade de uma sobrecarga computacional adicional dependendo da complexidade dos módulos de controle e inferência de emoções que impactam na velocidade e eficiência do modelo. Ademais, em imagens nas quais as características faciais estão fortemente ocultas, a GAN possui dificuldades de inferir e controlar emoções, o que acaba por prejudicar seus resultados.

O trabalho de Domingos, Cortes e Lobato (DOMINGOS; CORTES; LOBATO, 2022) possuía o objetivo de aprimorar a rede Deep-Emotive, implementando arquitetura de redes neurais convolucionais usando algoritmos genéticos para melhorar a detecção de emoções a partir de expressões faciais. Foi utilizado para treinamento o conjunto de

dados FER-2013 para classificar sete emoções em imagens de rostos. A rede inicial alcançou uma precisão superior a 60%, que foi aprimorada neste estudo. Essa abordagem resultou em uma arquitetura CNN com uma precisão de 63,84% no conjunto de treinamento e 62,39% no conjunto de validação. Os resultados também sugerem que com recursos computacionais adicionais a rede consegue obter resultados ainda melhores. Certos pontos devem ser ressaltados, como o fato do conjunto de treinamento apresentar desafios como ruído, desequilíbrio e imagens não relacionadas a emoções, o que leva a impactos negativos no desempenho da rede.

No trabalho de Esmaeili e Kiani (ESMAEILI; KIANI, 2024) é proposto um modelo híbrido baseado em GAN que reconstrói uma expressão facial personalizada baseada em seus sinais emocionais de eletroencefalograma correspondentes, utilizando uma cGAN. É proposto um novo método chamado CSP-BiLSTM que combina as redes CSP e BiLSTM para explorar as dependências dos sinais brutos de eletroencefalograma. Uma camada "*Fully Connected*" com uma função de ativação "*Softmax*" é aplicada às características extraídas com o intuito de reconhecer o rótulo dos sinais de eletroencefalograma. Após esse processo, o rótulo emocional é inserido em uma cGAN, juntamente com a imagem de um rosto com expressão neutra, para reconstruir emocionalmente a imagem de entrada. O modelo alcançou precisão superior a 99% para classificação de duas e três emoções, tanto em experimentos trans-sujeitos quanto em experimentos sujeito-dependentes, mostrando ser um modelo interdisciplinar de alta precisão. No entanto, o modelo foi considerado fraco em alguns critérios baseados em qualidade, incluindo a geração de imagens de alta dimensão.

No projeto de (XU et al., 2023) é proposto uma estrutura que incorpora o estilo de emoção em solicitações de texto e utiliza um codificador de emoção multimodal alinhada para mesclar modalidades de texto, áudio e imagem em um espaço unificado. Também foi projetado um gerador de rosto emocional de alta fidelidade baseado em estilo para gerar imagens realistas de altas resoluções. A estrutura proposta integra com sucesso o estilo de emoção em solicitações de texto, e o gerador de faces garante resultados de qualidade. Extensos experimentos mostraram a flexibilização e a generalização do método no controle da emoção, permitindo suporte para modalidades de emoção arbitrárias durante o teste e a generalização para estilos de emoção despercebidas. A rede pode enfrentar dificuldades para lidar com estilos de emoção despercebidas, devido à semântica limitada, já que os métodos existentes geralmente sofrem com configurações únicas ou com a qualidade dos rostos gerados, impactando o desempenho geral. Além disso, mesmo com a proposta de oferecer controle flexível em aplicações práticas, ainda pode haver limitações em capturar totalmente a complexidade e as nuances das emoções em diferentes modalidades.

O artigo de Carlsson e Kollias (CARLSSON; KOLLIAS, 2019) discute extensivamente as GANs, explorando várias arquiteturas, aplicativos, métodos de treinamento e tratamento de espaço latente. A pesquisan envolveu a compilação de um conjunto de dados de imagens representando sete emoções humanas básicas. Tal conjunto de dados foi utilizado em colaboração com a base FER2013 citada anteriormente, com o intuito de treinar um modelo denominado de StarGAN. O objetivo do treinamento era gerar e reconhecer imagens relacionadas a diferentes emoções. Os modelos treinados variavam em complexidade de três a cinco camadas, com o modelo de três camadas não conseguindo produzir resultados fotorrealistas, enquanto o modelo de cinco camadas gerava imagens mais realistas mas sem capturar as emoções preten-

didas de forma convincente. A dificuldade da rede neural em produzir imagens de qualidade é atribuída à natureza diversificada dos dados e ao tamanho reduzido das imagens, o que prejudica o aprendizado de detalhes mais específicos. Outrossim, a combinação das duas bases de dados acabou introduzindo mais ruído do que contribuindo para o aprendizado, porém, quando treinados somente com a base FER2013, os resultados se mostravam de melhor qualidade.

Esses trabalhos apresentam estratégias distintas para geração de imagens com emoções utilizando redes neurais. O primeiro gera rostos falantes com diferentes expressões, utilizando uma CNN gráfica. O segundo apresenta uma GAN capaz de gerar e classificar emoções, mesmo quando partes do rosto estão borradas ou ocultas. O terceiro implementa uma CNN para classificar sete emoções humanas básicas em imagens de rostos. O quarto implementa uma cGAN, com um banco de dados conjunto, para reconstruir uma emoção personalizada baseada em sinais emocionais de um eletroencefalograma. O quinto apresenta um gerador de rosto emocional, baseado nas emoções transmitidas por áudios, textos e imagens. Por fim, o sexto treina uma StarGAN, com o intuito de gerar e reconhecer imagens relacionadas a diferentes emoções. Dessa forma, destaca-se que o presente trabalho surge com o diferencial separar cada emoção em uma classe de GAN e ser capaz de gerar uma imagem de um rosto com a emoção que o usuário desejar, dentro das limitações da base de dados.

Tabela 1 - Comparação entre trabalhos relacionados

Trabalho	**Características / Similaridades**	**Diferenças / Limitações**
Sinha *et al.* (2022) – Talking Face Generation	• Geração de rostos controláveis com emoção • Foco no realismo • Usa CNN gráfica	• Gera rostos falantes, não foca em emoções isoladas • Não é um GAN puro • Sem separação por classes de emoções
Chen *et al.* (2023) – EC-GAN	• GAN condicionada por emoção • Geração e controle de expressões faciais	• Dificuldade em rostos parcialmente ocultos • Não faz sub-GANs por emoção
Domingos *et al.* (2022) – Deep-Emotive CNN	• Classificação de emoções com CNN • Geração básica de rostos emocionais	• Não é de fato uma GAN • Foco em classificação, não geração
Esmaeili & Kiani (2024) – cGAN + EEG	• cGAN com rótulo de emoção via sinais EEG • Reconstrói rosto neutro em expressão desejada	• Baseado em EEG, menos foco no processamento visual puro • Qualidade reduzida em alta resolução
Xu *et al.* (2023) – Multimodal Emotion Fusion	• Integra texto, áudio e imagem em espaço latente • Gerador de alta-fidelidade baseado em estilo	• Complexidade computacional elevada • Limitações para emoções não previstas
Carlsson & Kollias (2019) – StarGAN Emotions	• GAN unificada para múltiplas emoções • Usou FER2013 + base própria	• Número de camadas limitado afeta realismo • Combinação de bases introduziu ruído

3. METODOLOGIA

Para o desenvolvimento da Rede Generativa Adversarial (GAN), adotou-se a linguagem de programação Python 3 (Python Software Foundation,) em conjunto com o *framework* PyTorch (PASZKE et al., 2019), selecionado por sua eficiência computacional. O ambiente integrado de desenvolvimento (IDE) utilizado foi o Visual Studio Code (Microsoft,), executado em uma estação de trabalho equipada com processador Intel Core i5-11400H (Intel Corporation, 2024) e placa de vídeo NVIDIA GeForce RTX 3050 Mobile (NVIDIA Corporation, 2024).

A validação inicial da arquitetura foi realizada com a base de dados Fashion MNIST (XIAO; RASUL; VOLLGRAF, 2017), contendo 60.000 imagens em tons de cinza (28×28 *pixels*), conforme ilustrado na Figura 1. Esta escolha fundamenta-se na simplicidade estrutural e representatividade de padrões reais, permitindo avaliação preliminar da capacidade de generalização da rede.

Para o treinamento especializado em expressões faciais, empregou-se a base KDEF (LUNDQVIST; FLYKT; ÖHMAN, 1998), totalizando 4.900 imagens de rostos humanos categorizados em seis emoções básicas: felicidade, medo, neutro, surpresa, tristeza e raiva. Cada diretório utilizado para o treinamento contém um total de 12 imagens de rostos humanos expressivos posicionados de frente, tamanho 128x128 *pixels*. Como etapa de pós-processamento, implementou-se um **filtro de mediana** (WALT et al., 2014) com kernel 3×3, técnica não linear clássica para redução de ruído impulsivo (*salt-and-pepper*) que preserva bordas ao substituir o valor central pela mediana dos vizinhos na janela espacial (WANG; BOVIK, 1981). Essa escolha se justifica porque, ao final do treinamento, as imagens geradas pela GAN frequentemente exibem artefatos de alta frequência — pequenos pontos brancos ou pretos e flutuações pontuais de pixel — que dificultam a análise qualitativa. O filtro de mediana mostrou-se eficiente para atenuar esses artefatos sem borrar contornos faciais, mantendo a nitidez das expressões e permitindo uma avaliação visual mais confiável do desempenho do modelo.

Figura 1 - Exemplo de imagem do banco de dados Fashion MNIST

4. DESENVOLVIMENTO DA REDE

O código a seguir implementa o treinamento de uma Rede Neural Generativa Adversarial (GAN) utilizando PyTorch com o banco de dados Fashion MNIST. A arquitetura consiste em um componente gerador, responsável por produzir imagens sintéticas, e um discriminador, encarregado de distinguir entre amostras reais e sintéticas. Primeiramente, importam-se os pacotes necessários, incluindo numpy, torch (CONTRIBUTORS, 2024a), torchvision (CONTRIBUTORS, 2024b) além do uso de argumentos de linha de comando para gerenciar parâmetros como número de épocas, taxa de aprendizado e tamanho do batch. Utiliza-se a biblioteca torchsummary para exibir detalhes estruturais dos modelos. O torchvision é empregado no carregamento e transformação dos dados (como redimensionamento e normalização das imagens). Durante o treinamento, calcula-se a perda de entropia cruzada binária (BCE), atualizam-se os parâmetros do gerador e discriminador por meio do otimizador Adam, e registram-se as métricas utilizando o TensorBoard para monitoramento.

Código 1 - Importações e argumentos de linha de comando

```
import torch
import numpy as np
import argparse
import os
import tqdm
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, transforms
from torch.autograd import Variable
from torchvision.utils import save_image, make_grid
from torch.utils.tensorboard import SummaryWriter
from torchsummary import summary
import datetime
from PIL import Image
from torch.utils.data import Dataset, DataLoader
import glob
import matplotlib.pyplot as plt
from skimage.filters import median
from skimage.util import img_as_ubyte
from skimage.morphology import square

parser = argparse.ArgumentParser()
parser.add_argument("--n_epochs",       type=int,   default=8000,
                    help="numero␣de␣upocas␣de␣treinamento")
parser.add_argument("--batch_size",     type=int,   default=64,
                    help="tamanho␣do␣lote␣de␣treinamento")
parser.add_argument("--lr",             type=float, default=5e-5,
                    help="Adam:␣taxa␣de␣aprendizado")
parser.add_argument("--b1",             type=float, default=0.5,
                    help="Adam:␣decaimento␣do␣primeiro␣momento")
parser.add_argument("--b2",             type=float, default=0.999,
                    help="Adam:␣decaimento␣do␣segundo␣momento")
parser.add_argument("--latent_dim",     type=int,   default=2048,
                    help="dimensao␣do␣espaco␣latente␣(entrada␣do␣
   gerador)")
parser.add_argument("--img_size",       type=int,   default=128,
                    help="tamanho␣(altura/largura)␣das␣imagens")
parser.add_argument("--channels",       type=int,   default=1,
```

```
                    help="numero␣de␣canais␣das␣imagens␣(1␣=␣cinza,␣3␣=␣
    RGB)")
parser.add_argument("--output_log_dir", type=str,
    default="diff-run/py-gan",
                    help="diretorio␣para␣logs␣do␣TensorBoard")
parser.add_argument("--output_img_dir", type=str,
    default="diff-run/images",
                    help="diretorio␣para␣salvar␣imagens␣geradas")
args = parser.parse_args()
```

Em seguida, estes parâmetros controlam:

- **n_epochs**: define quantas vezes o laço de treinamento varre todo o conjunto de dados, permitindo convergência do modelo.
- **batch_size**: número de amostras processadas em cada iteração; lotes maiores tendem a estabilizar o gradiente, mas consomem mais memória.
- **lr** (learning rate): taxa de aprendizado do otimizador Adam, controla o tamanho do passo na atualização dos pesos.
- **b1** e **b2**: coeficientes de decaimento dos momentos primeiro e segundo do Adam, usados para calcular médias móveis dos gradientes e sua variância.
- **latent_dim**: dimensão do vetor aleatório de entrada (espaço latente) que o gerador transforma em imagens; maior dimensão pode capturar mais variabilidade.
- **img_size** e **channels**: especificam o formato das imagens de entrada/saída (por exemplo, 128×128 pixels em escala de cinza).
- **output_log_dir**: pasta onde são gravados os logs do TensorBoard (perdas, métricas, imagens de amostra).
- **output_img_dir**: diretório para salvar periodicamente as imagens geradas pelo modelo durante o treino.

Dessa forma, todos os hiperparâmetros e caminhos de entrada/saída ficam facilmente configuráveis via linha de comando, facilitando experimentos e reprodutibilidade do treinamento.

Na etapa apresentada no código 2, definem-se diretórios de saída, inicializa-se o TensorBoard e seleciona-se o dispositivo de cálculo (CUDA ou CPU). Em seguida, cria-se o pipeline de transformações com ‘transforms.Compose‘, que inclui:

- Redimensionamento para ‘(args.img_size, args.img_size)‘. - Espelhamento horizontal aleatório. - Rotação aleatória de até 5°. - Conversão para escala de cinza (1 canal). - Conversão para tensor (valores em $[0, 1]$). - Normalização com média 0.0 e desvio padrão 1.0, aplicados sobre o intervalo de pixel já em $[0, 1]$ (unidade: intensidade de pixel normalizada).

Depois, define-se a classe ‘KDEFDataset‘ para carregar as imagens ‘.JPG‘ da pasta ‘args.data_path‘ usando aquelas transformações, e instancia-se um ‘DataLoader‘ com ‘batch_size=args.batch_size‘ e shuffle ativado. Por fim, calcula-se ‘image_shape‘ e ‘image_dim‘ a partir de ‘args.channels‘ e ‘args.img_size‘.

Código 2 - Configuração do DataLoader para KDEF e pipeline de transformações

```
torch.manual_seed(1)

os.makedirs(args.output_log_dir, exist_ok=True)
os.makedirs(args.output_img_dir, exist_ok=True)

writer = SummaryWriter(args.output_log_dir)
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

train_transform = transforms.Compose([
    transforms.Resize((args.img_size, args.img_size)),
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(5),
    transforms.Grayscale(num_output_channels=1),
    transforms.ToTensor(),
    transforms.Normalize([0.0], [1.0])
])

class KDEFDataset(Dataset):
    def __init__(self, root_dir, transform=None):
        self.root_dir = root_dir
        self.transform = transform
        self.image_paths = glob.glob(os.path.join(root_dir, "*.JPG"))

    def __len__(self):
        return len(self.image_paths)

    def __getitem__(self, idx):
        img_path = self.image_paths[idx]
        image = Image.open(img_path).convert("RGB")
        if self.transform:
            image = self.transform(image)
        return image, 0

train_dataset = KDEFDataset(root_dir=args.data_path,
   transform=train_transform)
print(f"Number␣of␣samples␣in␣train_dataset:␣{len(train_dataset)}")
train_loader = DataLoader(dataset=train_dataset,
   batch_size=args.batch_size, shuffle=True)

image_shape = (args.channels, args.img_size, args.img_size)
image_dim = int(np.prod(image_shape))
```

No código 3 ocorre a definição do modelo do Gerador, que recebe um vetor de ruído como entrada e libera um vetor de mais de 16000 dimensões de saída, correspondente às proporções da imagem. As camadas são organizadas sequencialmente, cada uma sendo definida com uma combinação de funções de ativação, normalização e operações lineares. O método forward propaga dados através da rede, recebendo um vetor de ruído como entrada e passando esse vetor pelo modelo sequencial que realiza as operações definidas no construtor.

Código 3 - Definição do modelo do Gerador

```
class Gerador(nn.Module):
    def __init__(self):
        super(Gerador, self).__init__()
        self.model = nn.Sequential(
            nn.Linear(args.latent_dim, 4096),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Dropout(0.2),
            nn.Linear(4096, 2048),
            nn.BatchNorm1d(2048, 0.8),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Dropout(0.2),
            nn.Linear(2048, 8192),
            nn.BatchNorm1d(8192, 0.8),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Linear(8192, image_dim),
            nn.Tanh()
        )

    def forward(self, noise_vector):
        image = self.model(noise_vector)
        image = image.view(image.size(0), *image_shape)
        return image
```

No código 4 ocorre a definição do modelo do Discriminador. Esse modelo é um classificador binário que consiste apenas de camadas totalmente conectadas. Cada uma de suas camadas é composta por uma combinação de funções de ativação e operações lineares, e a última é uma camada linear com ativação sigmoide. Dessa vez, o método forward serve para receber uma imagem como entrada, que é redimensionada para um vetor de uma dimensão. Esse vetor então é passado pelo modelo sequencial que realiza as operações definidas no construtor. A função então retorna um valor próximo de 0, caso a entrada seja uma imagem falsa, ou um valor próximo de 1, caso a entrada tenha sido interpretada como verdadeira.

Código 4 - Definição do modelo do Discriminador

```
class Discriminador(nn.Module):
    def __init__(self):
        super(Discriminador, self).__init__()
        self.model = nn.Sequential(
            nn.Linear(image_dim, 8192),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Dropout(0.2),
            nn.Linear(8192, 4096),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Dropout(0.2),
            nn.Linear(4096, 2048),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Linear(2048, 1),
            nn.Sigmoid()
        )

    def forward(self, image):
        image_flattened = image.view(image.size(0), -1)
        result = self.model(image_flattened)
        return result
```

A seguir, é realizada a instância das classes Gerador() e Discriminador(), que serão movidos para um dispositivo, podendo ser GPU, caso haja CUDA disponível, ou processador. E por fim, antes de começar o loop de treinamento, é inserida a função de perda BCE e definidos os otimizadores de cada modelo. O otimizador Adam recebe três argumentos: os parâmetros do gerador e do discriminador, a taxa de aprendizado e os coeficientes beta 'b1' e 'b2' para calcular médias de gradiente. O código 5 mostra o passo-a-passo dessa etapa.

Código 5 - Instanciação dos modelos, sumário e configuração de perdas e otimizadores

```
Gerador = Gerador().to(device)
Discriminador = Discriminador().to(device)

summary(Gerador,          (args.latent_dim,))
summary(Discriminador,    (args.channels, args.img_size, args.img_size))

adversarial_loss = nn.BCELoss()

G_optimizer = optim.Adam(
    Gerador.parameters(),
    lr=args.lr,
    betas=(args.b1, args.b2)
)
D_optimizer = optim.Adam(
    Discriminador.parameters(),
    lr=args.lr,
    betas=(args.b1, args.b2)
)
```

A última etapa do código é o loop de treinamento da rede. Definindo o número de épocas como 8000, é feito com que ocorram 8000 iterações completas sobre o banco de dados KDEF, e após isso, armazena-se os valores de perda do Discriminador e do Gerador em duas listas: *D_loss_list* e *G_loss_list*. Para iterar sobre as épocas, foi

usado um 'for' e dentro foi colocado mais um 'for', que itera sobre os dados da amostra de treino, e para cada lote, é feito o treinamento do discriminador.

Divide-se o treinamento do discriminador em duas partes: imagens reais e falsas. No começo, o otimizador é zerado para que não ocorra acúmulo de gradientes, e são criados rótulos reais e rótulos falsos, recebendo valor 1 caso a imagem lida seja interpretada como real ou 0 caso a imagem lida seja falsa. As imagens reais são alocadas para o dispositivo onde será feito o treinamento, e calcula-se a perda BCE usando a função *D_real_loss()*. Já para as imagens falsas, calcula-se a perda BCE com a função *D_fake_loss()*, e depois, o valor das duas perdas é somado com a função *D_total_loss()*. Logo após, os gradientes são calculados e os otimizadores são atualizados.

Código 6 - Loop de treinamento — passo do discriminador

```
num_epochs = args.n_epochs
D_loss_plot, G_loss_plot = [], []

for epoch in range(1, num_epochs + 1):
    D_loss_list, G_loss_list = [], []

    for index, (real_images, _) in enumerate(train_loader):

        D_optimizer.zero_grad()
        real_images = real_images.to(device)


        real_target = Variable(torch.full((real_images.size(0), 1),
   0.9, device=device))
        fake_target = Variable(torch.full((real_images.size(0), 1),
   0.1, device=device))

        # Perda com imagens reais
        D_real_loss = adversarial_loss(Discriminador(real_images),
   real_target)


        noise_vector = Variable(torch.randn(real_images.size(0),
   args.latent_dim).to(device))
        generated_image = Gerador(noise_vector)


        D_fake_loss = adversarial_loss(Discriminador(generated_image),
   fake_target)


        D_total_loss = D_real_loss + D_fake_loss
        D_loss_list.append(D_total_loss)
        D_total_loss.backward()
        D_optimizer.step()
```

Já para o gerador, é necessário os resultados do treinamento do discriminador para dar início. A variável `noise_vector`, no Código 6, mostram o vetor de ruído que é usado como entrada para o gerador, com seu tamanho sendo determinado pelo número de imagens reais no lote e pela dimensão latente. Esse vetor então é movido para o dispositivo especificado por *device*. É com essa entrada que o gerador cria a imagem que será lida pelo discriminador. Assim como foi feito com o discriminador,

o otimizador do gerador também é zerado para evitar acúmulo desnecessário de gradientes. A função *G_loss()* calcula a perda adversarial do gerador, e aqui a perda é realizada tendo em vista as interpretações do discriminador para o conteúdo produzido e os rótulos reais. Após isso, o valor da perda é armazenado em *G_loss_list*. Feito isso, ocorre o cálculo dos gradientes de perda e a atualização dos pesos do gerador com base nesse cálculo. Por fim, armazena-se os dados das imagens criadas em '*d*', além de registrar os valores de perda de ambas as partes usando o TensorBoard.

Código 7 - Loop de treinamento — passo do gerador e registro de métricas

```
        G_optimizer.zero_grad()
        generated_image = Gerador(noise_vector)
        G_loss = adversarial_loss(Discriminador(generated_image),
    real_target)
        G_loss_list.append(G_loss)
        G_loss.backward()
        G_optimizer.step()


        d = generated_image.data
        writer.add_scalar(
            'Perda BCE Discriminador',
            D_total_loss,
            epoch * len(train_loader) + index
        )
        writer.add_scalar(
            'Perda BCE Gerador',
            G_loss,
            epoch * len(train_loader) + index
        )
```

O trecho a seguir implementa a exibição dos valores de perda em cada época, o armazenamento das listas de perdas e o salvamento das imagens geradas, aplicando sobre elas um filtro de mediana conforme descrito na Metodologia (seção 3):

Código 8 - Exibição de perdas e salvamento de imagens com filtro de mediana

```
print('Epoca: [%d/%d]: Perda_D: %.3f, Perda_G: %.3f' % (
    epoch, num_epochs, avg_D_loss, avg_G_loss))

D_loss_plot.append(avg_D_loss)
G_loss_plot.append(avg_G_loss)


if epoch % 50 == 0:

    imgs = generated_image.data[:90].cpu()
    imgs_np = imgs.numpy()
    imgs_np_filtered = []
    for img in imgs_np:
        img_ = img[0]
        img_ = (img_ - img_.min()) / (img_.max() - img_.min() + 1e-8)
        img_ubyte = img_as_ubyte(img_)
        img_med = median(img_ubyte, square(3))
        img_med = img_med.astype(np.float32) / 255.0
        imgs_np_filtered.append(img_med)
    imgs_np_filtered = np.stack(imgs_np_filtered)
```

```
imgs_tensor_filtered = torch.tensor(imgs_np_filtered).unsqueeze(1)
save_image(imgs_tensor_filtered,
           os.path.join(args.output_img_dir, f'sample_{epoch}.png'),
           nrow=10, normalize=True)
```

Neste trecho:

- A função ‘print‘ exibe na tela o número da época atual, o total de épocas e as perdas médias do discriminador (‘Perda_D‘) e do gerador (‘Perda_G‘).
- As listas ‘D_loss_plot‘ e ‘G_loss_plot‘ armazenam essas médias para plotagem posterior.
- A cada 50 épocas, o bloco dentro do ‘if‘ salva uma grade de 90 imagens geradas. Primeiro, as imagens são convertidas para arrays NumPy e normalizadas em $[0, 1]$. Em seguida aplica-se um filtro de mediana 3×3 (técnica não linear para redução de ruído impulsivo salt-and-pepper que preserva contornos) usando ‘skimage.filters.median‘ e um elemento estruturante quadrado (‘square(3)‘). Após o filtro, as imagens são reconvertidas para tensor e gravadas em disco via ‘save_image‘.

Dessa forma, garante-se tanto o monitoramento das perdas durante o treino quanto o salvamento periódico de amostras visuais livres de artefatos de alta frequência.

A figura 2 mostra que ao iniciar a execução do código, antes de exibir as épocas e os valores de perdas dos modelos, é exibido diversas informações sobre a arquitetura dos modelos, como número de parâmetros treinados de cada rede, tamanho estimado em mb, camadas e formato do vetor de saída.

Conforme ilustrado na Figura 2, a execução do código inicia-se com a exibição detalhada da arquitetura dos modelos, incluindo parâmetros treináveis, consumo de memória e formato das camadas. Para otimizar o desempenho, implementaram-se significativas modificações baseadas em análise iterativa:

- **Ajustes de Hiperparâmetros**:

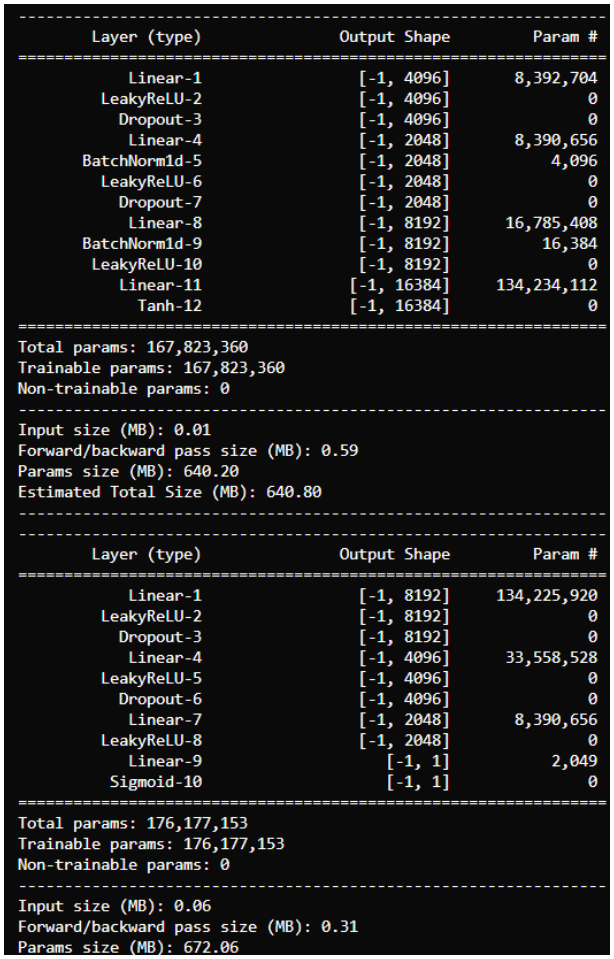
$$\begin{aligned} \text{batch size} &: 128 \rightarrow 64 \\ \text{taxa de aprendizado} &: 5 \times 10^{-5} \\ \text{dimensão latente} &: 100 \rightarrow 2048 \end{aligned}$$

  Visando estabilizar o treinamento, o *batch size* foi reduzido para mitigar *mode collapse*, enquanto a maior dimensionalidade latente ampliou a capacidade de representação de características emocionais.

- **Expansão Arquitetural**:

$$\begin{aligned} \text{Gerador} &: 2048 \rightarrow 4096 \rightarrow 8192 \\ \text{Discriminador} &: 8192 \rightarrow 4096 \rightarrow 2048 \end{aligned}$$

  Aumentou-se progressivamente os neurônios nas camadas lineares. Após testes iterativos (500 $\rightarrow$ 800 $\rightarrow$ 2000 $\rightarrow$ 3000 $\rightarrow$ 5000 épocas), estabeleceu-se 8000 épocas como ótimo empírico.



```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                [-1, 4096]       8,392,704
         LeakyReLU-2                [-1, 4096]               0
           Dropout-3                [-1, 4096]               0
            Linear-4                [-1, 2048]       8,390,656
       BatchNorm1d-5                [-1, 2048]           4,096
         LeakyReLU-6                [-1, 2048]               0
           Dropout-7                [-1, 2048]               0
            Linear-8                [-1, 8192]      16,785,408
       BatchNorm1d-9                [-1, 8192]          16,384
        LeakyReLU-10                [-1, 8192]               0
           Linear-11               [-1, 16384]     134,234,112
             Tanh-12               [-1, 16384]               0
================================================================
Total params: 167,823,360
Trainable params: 167,823,360
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 0.59
Params size (MB): 640.20
Estimated Total Size (MB): 640.80
----------------------------------------------------------------
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                [-1, 8192]     134,225,920
         LeakyReLU-2                [-1, 8192]               0
           Dropout-3                [-1, 8192]               0
            Linear-4                [-1, 4096]      33,558,528
         LeakyReLU-5                [-1, 4096]               0
           Dropout-6                [-1, 4096]               0
            Linear-7                [-1, 2048]       8,390,656
         LeakyReLU-8                [-1, 2048]               0
            Linear-9                   [-1, 1]           2,049
           Sigmoid-10                  [-1, 1]               0
================================================================
Total params: 176,177,153
Trainable params: 176,177,153
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.06
Forward/backward pass size (MB): 0.31
Params size (MB): 672.06
```

Figura 2 - Informações da arquitetura das redes

Por último, implementou-se uma interface de controle para treino focalizado por emoção. O sistema mapeia seis emoções básicas através de dicionário Python, permitindo entrada numérica ou textual, conforme mostra o código 9.

Código 9 - Interface de Seleção de Emoção

```
def selecionar_emocao():
    emocoes = {
        "1": "Felicidade",
        "2": "Medo",
        "3": "Neutro",
        "4": "Raiva",
        "5": "Surpresa",
        "6": "Tristeza"
    }

    print("\n=== Selecao de Emocao ===")
    print("Escolha a emocao que deseja treinar:")
    for key, value in emocoes.items():
        print(f"{key} - {value}")

    while True:
        escolha = input("\nDigite o numero ou nome da emocao: 
    ").strip().lower()
```

```
if escolha in emocoes:
    return emocoes[escolha]


for key, value in emocoes.items():
    if escolha == value.lower():
        return value

print("Opcao invalida! Por favor, tente novamente.")
```

4.1. TESTE COM BASE DE EXPRESSÕES FACIAIS

Para realizar o treino da GAN utilizando as bases KDEF e AKDEF, utilizou-se um diretório de 35 imagens para realizar o primeiro teste com imagens de emoções humanas. A Figura 3 contém as imagens utilizadas nesta etapa do treinamento. O número máximo de épocas aumentou de 200 para 500 devido à complexidade da nova base, sendo necessárias mais iterações para alcançar um resultado satisfatório.

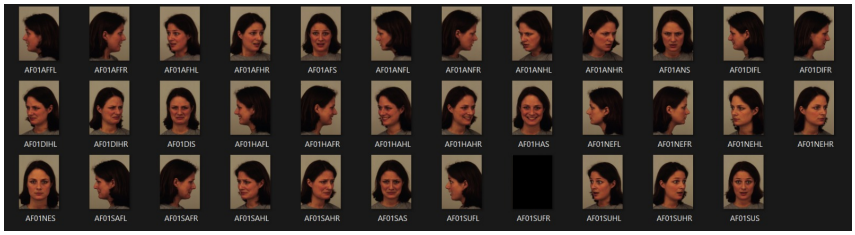


Figura 3 - Amostra de Dados de Treinamento KDEF

Após 200 épocas de treino, o resultado gerado, conforme mostra a Figura 4, ainda dista de assemelhar-se às amostras de treinamento. O rosto produzido pelo gerador apresenta-se bastante distorcido e com ruído excessivo. Já com 500 épocas de treinamento, como evidenciado na Figura 5, o resultado aproxima-se mais de um rosto humano semelhante ao das amostras de treinamento, embora ainda exibam falhas visíveis a serem corrigidas.

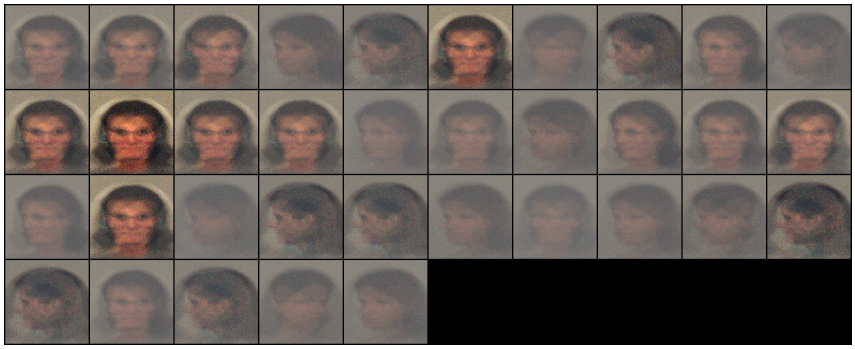


Figura 4 - Resultado Após 200 Épocas de Treinamento

Para reduzir a complexidade computacional e acelerar o treinamento, as imagens foram alteradas para serem processadas em escala de cinza (1 canal); isso diminui

Figura 5 - Resultado Após 500 Épocas de Treinamento

a dimensionalidade dos dados sem comprometer a representação de contornos faciais, fundamentais para captura de emoções. Após a definição de 5 000 épocas de treinamento, redução para um canal de imagem e refinamento dos hiperparâmetros, observou-se melhora significativa na qualidade das amostras. Contudo, os rostos sintetizados apresentavam sempre perfil lateral, conforme mostrado na figura 6. Para corrigir esse viés, filtrou-se a base KDEF/AKDEF, mantendo apenas imagens de faces frontais (12 amostras por emoção).

Com o conjunto balanceado de rostos frontais, as saídas exibiram contornos faciais mais coerentes, mas ainda persistia ruído de fundo. A aplicação do filtro de mediana (3×3) da biblioteca `scikit-image` reduziu drasticamente esses artefatos, indicando convergência estável do modelo (figura 7).

Finalmente, o método foi avaliado em cinco emoções adicionais—*medo*, *neutro*, *surpresa*, *raiva* e *tristeza*—confirmando a capacidade da GAN em reproduzir variações sutis de expressão em todas as categorias testadas.

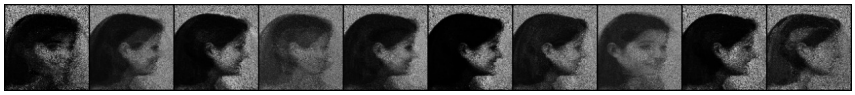


Figura 6 - Teste com 5000 épocas, parâmetros finais, rostos de frente e de lado

Figura 7 - Teste com 8000 épocas, parâmetros finais, apenas rostos de frente. Emoção Felicidade

Em paralelo à avaliação visual, a Figura 8 apresenta as curvas de perda de entropia cruzada binária (BCE) para o discriminador e o gerador ao treinar a rede com a emoção de felicidade. Observa-se que a perda do discriminador estabiliza-se em torno de 0,8–1,0, indicando que ele não fica nem totalmente convencido nem completamente enganado, o que sinaliza um equilíbrio saudável na competição adversarial. Já a perda do gerador mantém-se em valores próximos a 2,0, refletindo que as imagens, embora reconhecíveis como rostos, ainda não enganaram plenamente o discriminador. Esses valores quantitativos complementam a análise subjetiva da figura 7,

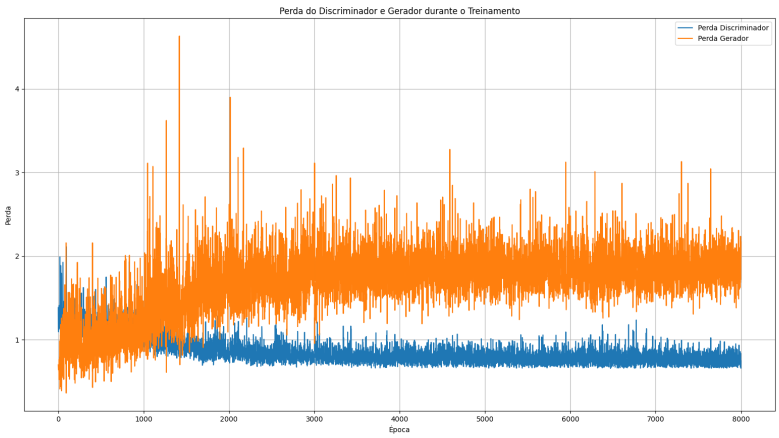


Figura 8 - Curvas de perda BCE do discriminador e do gerador ao longo de 8000 épocas

confirmando convergência do treinamento.

Figura 9 - 8000 épocas. Emoção Medo

Figura 10 - 8000 épocas. Emoção Neutro

Figura 11 - 8000 épocas. Emoção Surpresa

Figura 12 - 8000 épocas. Emoção Raiva

Figura 13 - 8000 épocas. Emopção Tristeza

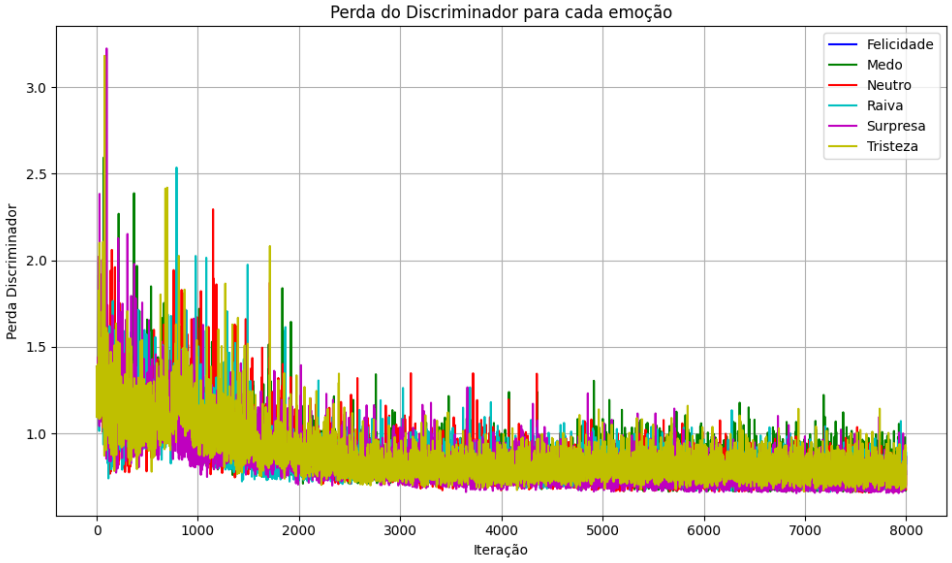


Figura 14 - Perda do Discriminador para cada emoção ao longo de 8000 épocas

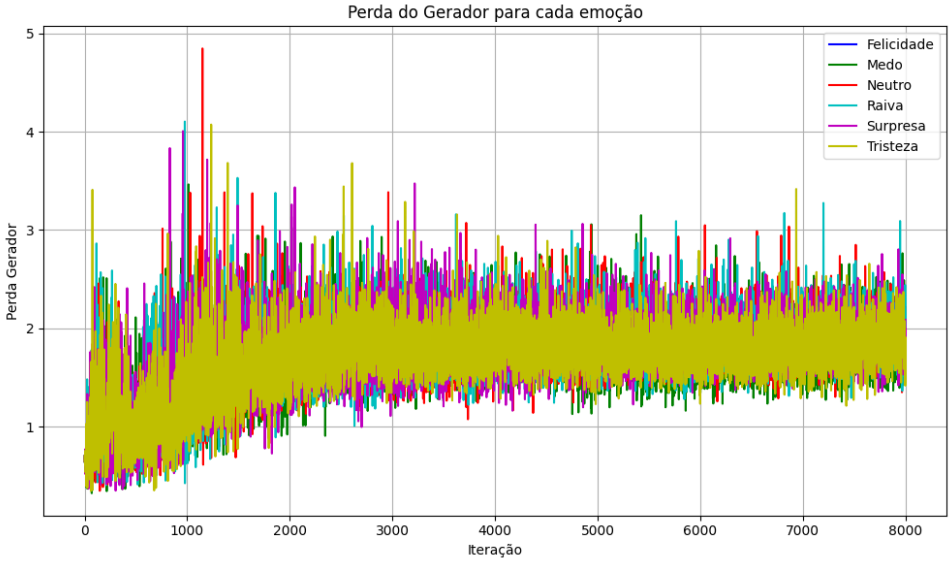


Figura 15 - Perda do Gerador para cada emoção ao longo de 8000 épocas

Em complemento às amostras visuais, as Figuras 14 e 15 comparam, em um único gráfico, as curvas de BCE para gerador e discriminador em cada emoção. Nota-se que todas as emoções apresentam comportamento muito semelhante: o discriminador estabiliza-se consistentemente entre 0,7 e 1,0, e o gerador mantém-se em torno de 1,5–2,5, indicando um ponto de equilíbrio adversarial uniforme. Pequenas variações iniciais — por exemplo, picos ligeiramente maiores em "Neutro" ou estabilização mais rápida em "Surpresa" — refletem a complexidade relativa de cada expressão, mas não comprometem o equilíbrio geral. Esses dados reforçam que a GAN aprendeu a sintetizar todas as seis emoções de forma equivalente, sem exigir ajustes específicos para cada caso.

Em síntese, os experimentos demonstram que o aumento do número de épocas contribuiu para aprimorar a coerência dos contornos faciais e reduzir parte dos artefatos de alta frequência, mas sem eliminar completamente o ruído residual nas amostras geradas. Embora a curva de perda BCE indique convergência estável, os ganhos qualitativos começam a apresentar retornos decrescentes à medida que a rede se aproxima de um ponto de equilíbrio adversarial. Além disso, o incremento de no número de épocas resultou em um aumento significativo no tempo de processamento, o que reforça a necessidade de explorar, em trabalhos futuros, alternativas de redução de custo computacional.

5. Considerações Finais

A proposta central deste projeto foi desenvolver uma ferramenta baseada em GAN para síntese de rostos humanos com emoções específicas, visando tanto o balanceamento de bases de dados de expressões faciais quanto o aprimoramento da qualidade de avatares virtuais. Tal empreendimento enfrentou desafios relacionados ao desbalanceamento das classes emocionais e à complexidade de produzir imagens fotorrealistas.

Em síntese, o trabalho desenvolveu um modelo de Rede Neural Generativa Adversarial (GAN) com uma classe especializada para cada uma das seis emoções — *felicidade*, *tristeza*, *raiva*, *medo*, *surpresa* e *neutro*. Na primeira etapa, validou-se o pipeline usando Fashion MNIST e um subconjunto frontal de KDEF; na etapa atual, aprimorou-se o ajuste de hiperparâmetros, empregou-se o filtro de mediana e implementou-se a possibilidade de selecionar a emoção, resultando em imagens significativamente mais realistas e com menor nível de ruído.

Os principais avanços desta etapa incluem:

- Estabilização do treinamento por meio do ajuste fino de batch size, learning rate e número de épocas, permitindo a convergência do modelo em 8000 iterações.
- Redução de artefatos de alta frequência através da aplicação do filtro de mediana (3×3), preservando contornos faciais essenciais.
- Implementação de função interativa que possibilita ao usuário escolher a emoção a ser gerada, garantindo treinamento balanceado por categoria.

Apesar dos resultados promissores, identificaram-se as seguintes limitações:

- Diversidade e resolução das bases de dados ainda restritas, o que pode limitar o realismo final das imagens.
- Arquitetura básica da GAN, comparada a abordagens de última geração, apresenta margens de melhoria em fidelidade.
- Ausência de métricas quantitativas robustas nesta fase de avaliação.

Recomendações e Trabalhos Futuros

Para consolidar e estender este trabalho, sugerem-se as seguintes direções:

1. Treinar a GAN em bases de dados maiores e mais diversificadas, incluindo amostras de dados sintéticas.
2. Explorar arquiteturas avançadas e métodos de regularização para reduzir artefatos e prevenir colapso de modo.
3. Desenvolver uma interface gráfica ou API web para demonstração em tempo real da geração de emoções.

Os resultados obtidos até aqui confirmam a viabilidade de GANs para síntese de expressões faciais emocionais e abrem caminho para aplicações práticas em avatares inteligentes e sistemas interativos cada vez mais realistas.

REFERÊNCIAS

ARXIV. **arXiv e-Print Archive**. Disponível em: <https://arxiv.org>.

CARLSSON, H.; KOLLIAS, D. **Image Generation and Recognition (Emotions)**. 2019.

CHEN, Y. et al. Ec-gan: Emotion-controllable gan for face image completion. **Applied Sciences**, v. 13, n. 13, 2023. ISSN 2076-3417. Disponível em: <https://www.mdpi.com/2076-3417/13/13/7638>.

CONTRIBUTORS, T. **Torch: An Open-Source Machine Learning Library**. 2024. Disponível em: <https://pytorch.org/>.

CONTRIBUTORS, T. **Torchvision: Datasets, Transforms, and Models for Computer Vision**. 2024. Disponível em: <https://pytorch.org/vision/>.

DOMINGOS, D.; CORTES, O. C.; LOBATO, F. Evoluindo redes neurais convolucionais na detecção de emoções usando micro ags. In: . [S.l.: s.n.], 2022.

ESMAEILI, M.; KIANI, K. Generating personalized facial emotions using emotional eeg signals and conditional generative adversarial networks. **Multimedia Tools and Applications**, v. 83, p. 36013–36038, 2024. Disponível em: <https://doi.org/10.1007/s11042-023-17018-w>.

GOODFELLOW, I. J. et al. **Generative Adversarial Networks**. 2014. Disponível em: <https://arxiv.org/abs/1406.2661>.

Google Scholar. **Google Scholar: uma ferramenta de busca acadêmica**. 2025. <https://scholar.google.com/>. Acessado em: 06 jun. 2024.

Intel Corporation. **Intel**. 2024. Disponível em: <https://www.intel.com>.

LUNDQVIST, D.; FLYKT, A.; ÖHMAN, A. **The Karolinska Directed Emotional Faces - KDEF**. [S.l.]: Department of Clinical Neuroscience, Psychology section, Karolinska Institutet, 1998. CD ROM.

Microsoft. **Visual Studio Code**. Disponível em: <https://code.visualstudio.com/>.

MIOTO, A. C. d. A.; MIRANDA, J. V. de; PREMEBIDA, S. M. Lidando com o desbalanceamento de dados. **Alura**, 2023. Disponível em: <https://www.alura.com.br/artigos/lidando-com-desbalanceamento-dados>.

NVIDIA Corporation. **NVIDIA**. 2024. Disponível em: <https://www.nvidia.com>.

P, P. S. et al. Generic image application using gans (generative adversarial networks): A review. **Evolving Systems**, v. 14, p. 1–15, 09 2022.

PASQUARELLI, W. **Rumo à realidade sintética: quando DeepFakes encontram AR/VR**. 2019. Disponível em: <https://oxfordinsights.com/insights/towards-synthetic-reality-when-deepfakes-meet-ar-vr/>.

PASZKE, A. et al. Pytorch: An imperative style, high-performance deep learning library. In: **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>.

PEREZ, C. E. Deep learning solves the uncanny valley problem. **Medium**, 2017. Disponível em: <https://medium.com/intuitionmachine/these-images-are-generated-by-a-deep-learning-gan-6b49062b3959>.

PLATFORMS, I. M. **The Metaverse: A Vision for the Future of Social Technology**. 2021. <https://about.fb.com/news/2021/10/founders-letter/>. Accessed: 2024-07-18. Disponível em: <https://about.fb.com/news/2021/10/founders-letter/>.

Python Software Foundation. **Python Programming Language**. Disponível em: <https://www.python.org/>.

ROSADO, P.; FERNáNDEZ, R.; REVERTER, F. Gans and artificial facial expressions in synthetic portraits. **Big Data and Cognitive Computing**, v. 5, n. 4, 2021. ISSN 2504-2289. Disponível em: <https://www.mdpi.com/2504-2289/5/4/63>.

SHARMA, A. **Generative Adversarial Networks (GANs) – An Introduction**. 2021. Disponível em: <https://learnopencv.com/introduction-to-generative-adversarial-networks/#coding>.

SINHA, S. et al. **Emotion-Controllable Generalized Talking Face Generation**. 2022. Disponível em: <https://arxiv.org/abs/2205.01155>.

WALT, S. Van der et al. scikit-image: image processing in python. **PeerJ**, v. 2, p. e453, 2014.

WANG, Z.; BOVIK, A. C. Median filters: A review. **IEEE Transactions on Circuits and Systems**, v. 35, n. 11, p. 1173–1188, 1981.

WU, Y. et al. **AniFaceGAN: Animatable 3D-Aware Face Image Generation for Video Avatars**. 2022. Disponível em: <https://arxiv.org/abs/2210.06465>.

XIAO, H.; RASUL, K.; VOLLGRAF, R. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. **arXiv preprint arXiv:1708.07747**, 2017. Disponível em: <https://arxiv.org/abs/1708.07747>.

XU, C. et al. **High-fidelity Generalized Emotional Talking Face Generation with Multi-modal Emotion Space Learning**. 2023. Disponível em: <https://arxiv.org/abs/2305.02572>.