USP - Universidade de São Paulo
Instituto de Ciências Matemáticas e de Computação

Projeto Final

Análise de modelos geradores de faces - Generative Adversarial Networks e Modelo de Difusão

Alunos:

Bernardo Marques Costa	| 11785551
Rodrigo Lopes Assaf	| 11795530
Victor Henrique de Sa Silva	| 11795759

Disciplina:

SCC0910 - Tópicos Avançados em Ciências de Computação I (2023)

Docente:

Fernando Pereira dos Santos

São Carlos, Outubro de 2023

Descrição do projeto

Nesse projeto de Visão Computacional, optamos por apresentar a seguinte tarefa: geração de imagens de faces. A geração de imagens é uma das aplicações mais interessantes da visão computacional, pois permite criar novas imagens que podem ser usadas em diversas aplicações, como por exemplo arte, design, publicidade e até no treinamento de outros modelos.

Teremos duas etapas de análises: na primeira, atacaremos o problema utilizando redes neurais generativas adversariais (mais especificamente, Deep Convolutional GANs - DCGANs) e na segunda, tentaremos aplicar modelos de difusão para melhorar a geração dessas faces.

Com isso, nesse nosso trabalho, estudaremos sobre a criação de imagens de faces falsas, de pessoas que não existem, tendo como base um dataset composto por imagens reais de pessoas famosas.

Para este trabalho, utilizamos a base de dados CelebA (Liu et al., 2015)[1], que contém mais de 200.000 imagens de celebridades.

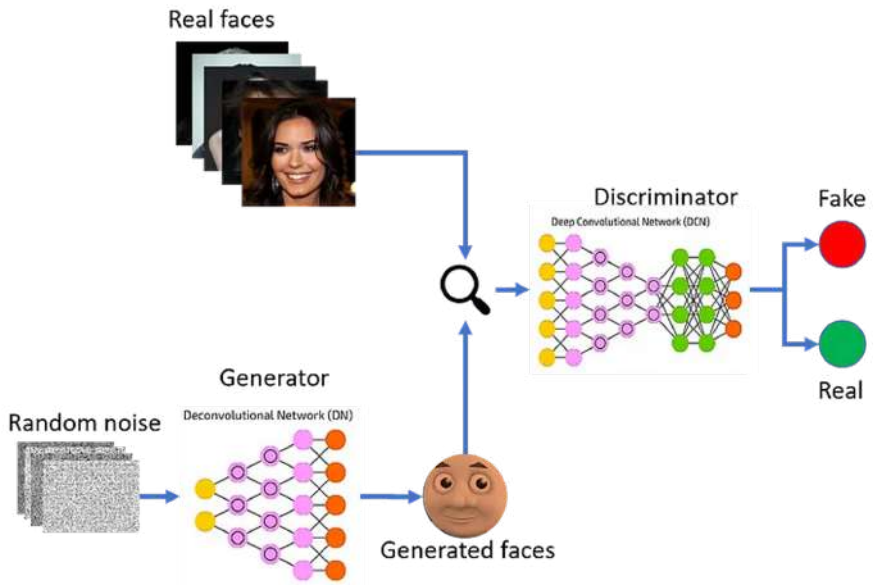
Dataset CelebA (Liu et al., 2015)

Para este trabalho, utilizamos a base de dados CelebA (Liu et al., 2015), que contém mais de 200.000 imagens de celebridades e disponibilizado na plataforma Kaggle. Exemplos de imagens utilizadas:

Introdução ao modelo GAN e DCGAN

GANs (Goodfellow et al., 2014)

As GANs foram introduzidas por Goodfellow et al. (2014)[2] e consistem em duas redes neurais, uma geradora e outra discriminadora, que são treinadas simultaneamente. A rede geradora gera imagens falsas, enquanto a rede discriminadora é treinada para distinguir imagens reais das imagens geradas. O treinamento é realizado em um processo iterativo em que a rede geradora busca gerar imagens cada vez mais realistas, enquanto a rede discriminadora busca distinguir entre imagens reais e falsas com maior precisão.



O paper apresenta as Generative Adversarial Networks como modelos de Multi-Layers Perceptrons, em que o modelo gerador aprende a distribuição pg dos dados x, definindo-se uma variável de ruído $pz(z)$, representando um mapeamento para o espaço de dados pela função $G(z; \theta g)$, em que G é uma função diferenciável representada por uma multi-layer percpetron com parâmetros θg. Assim, o modelo gerador irá receber como input imagem de ruído e fornecer como output uma imagem de mesma dimensionalidade que as imagens reais.

O modelo discriminador, é também uma multi-layer perceptron representada pela função $D(x; \theta d)$, que tem como output um único escalar. $D(x)$ representa a probabilidade que x venha dos dados reais, ao invés dos dados gerados a partir do modelo gerador, com input pg.

Treinamos D para maximizar a probabilidade de atribuir rótulos corretor para os exemplos de treinamento (amostras reais) e amostras geradas pelo modelo gerador (amostras falsas).

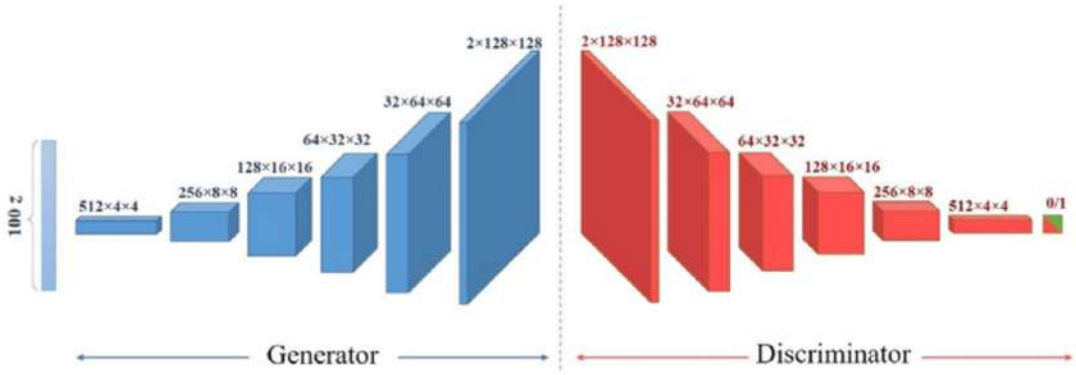
Simultaneamente, é treinado o modelo gerador G para minimizar $log(1 - D(G(z))$. Assim, temos um “two-player minmax game” (como dito no artigo original), com valores $V(G, D)$:

$$min_G max_D V(D,G) = E_{x \sim pdata(x)}[log\ D(x)] + E_{z \sim px(z)}[log(1 - D(G(z))]$$

Existem várias arquiteturas de GANs que foram propostas na literatura, incluindo a DCGAN (Radford et al., 2016), a WGAN (Arjovsky et al., 2017) e a StyleGAN (Karras et al., 2019).

DCGANs (Radford et al., 2016)

Neste trabalho, optamos por utilizar a arquitetura DCGAN, apresentadar em Radford et al (2016)[3], sendo uma extensão das GANs clássicas, especificamente projetada para a geração de imagens. As DCGANs incorporam camadas convolucionais profundas tanto no gerador quanto no discriminador, permitindo a aprendizagem hierárquica de características visuais. Essa arquitetura é fundamental para a geração de imagens de alta resolução e qualidade, pois captura informações espaciais e de contexto presentes nas imagens.



A utilização das GANs e DCGANs tem proporcionado avanços significativos em diversas aplicações. Na área de reconhecimento de padrões, essas técnicas podem gerar dados sintéticos para enriquecer conjuntos de treinamento escassos, permitindo o aprimoramento de modelos de aprendizado de máquina em tarefas como classificação de objetos, detecção de anomalias e reconhecimento facial. Além disso, as GANs e DCGANs têm sido empregadas na síntese de dados em áreas como medicina, economia e segurança, permitindo a geração de dados sintéticos realistas para análise e estudos exploratórios.

Metodologia da Primeira Iteração

Implementamos uma GAN baseada na arquitetura DCGAN, que consiste em uma rede geradora com cinco camadas convolucionais e uma rede discriminadora também com cinco camadas convolucionais. As imagens de entrada foram redimensionadas para o tamanho de 64x64 pixels e normalizadas entre -1 e 1. Usamos a função de ativação ReLU nas camadas convolucionais da rede geradora e a função de ativação LeakyReLU nas camadas convolucionais da rede discriminadora.

Treinamos a rede por 50 épocas com um *batch size* de 128 imagens e uma taxa de aprendizado de 0,0001. Para maior estabilidade, utilizamos o otimizador Adam com $\beta_1 = 0.5$ e $\beta_2 = 0.9999$. Devido a característica de classificação, utilizamos como função de custo uma entropia cruzada.

Rede geradora

A rede geradora recebe de *input* um vetor de ruído de dimensionalidade 100, realizando convoluções até transformá-lo em uma imagem normalizada entre -1 e 1, de dimensionalidade 64x64 pixels, com 3 canais (RGB).

Essa imagem será jogada para o discriminador, que será encarregado de identificar se é falsa ou não.

```
Generator(
  (main): Sequential(
    (0): ConvTranspose2d(100, 512, kernel_size=(4, 4), stride=(1, 1), bias=False)
    (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): ConvTranspose2d(512, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (4): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (5): ReLU(inplace=True)
    (6): ConvTranspose2d(256, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (7): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (8): ReLU(inplace=True)
    (9): ConvTranspose2d(128, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (10): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (11): ReLU(inplace=True)
    (12): ConvTranspose2d(64, 3, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (13): Tanh()
  )
)
```

Rede discriminadora

O modelo discriminador recebe como input uma imagem de dimensionalidade 64x64x3, realizando diversas operações convolucionais. É aplicada um operador de sigmoide para gerar um valor que é interpretado como probabilidade, que identifica a probabilidade da imagem de input pertencer ao dados de treinamento (amostras reais).

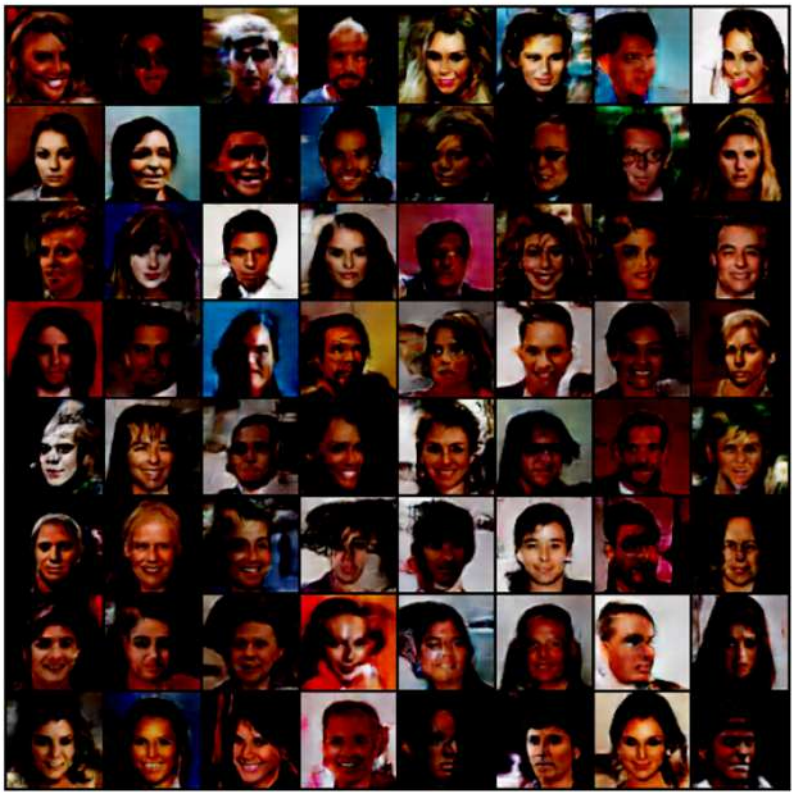
```
Discriminator(
  (main): Sequential(
    (0): Conv2d(3, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (1): LeakyReLU(negative_slope=0.2, inplace=True)
    (2): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (3): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (4): LeakyReLU(negative_slope=0.2, inplace=True)
    (5): Conv2d(128, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (6): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (7): LeakyReLU(negative_slope=0.2, inplace=True)
    (8): Conv2d(256, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (9): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (10): LeakyReLU(negative_slope=0.2, inplace=True)
    (11): Conv2d(512, 1, kernel_size=(4, 4), stride=(1, 1), bias=False)
    (12): Sigmoid()
  )
)
```

Resultados

Os resultados obtidos com a DCGAN proposta foram promissoras:

Final da primeira época

Final da décima época



Final da vigésima época

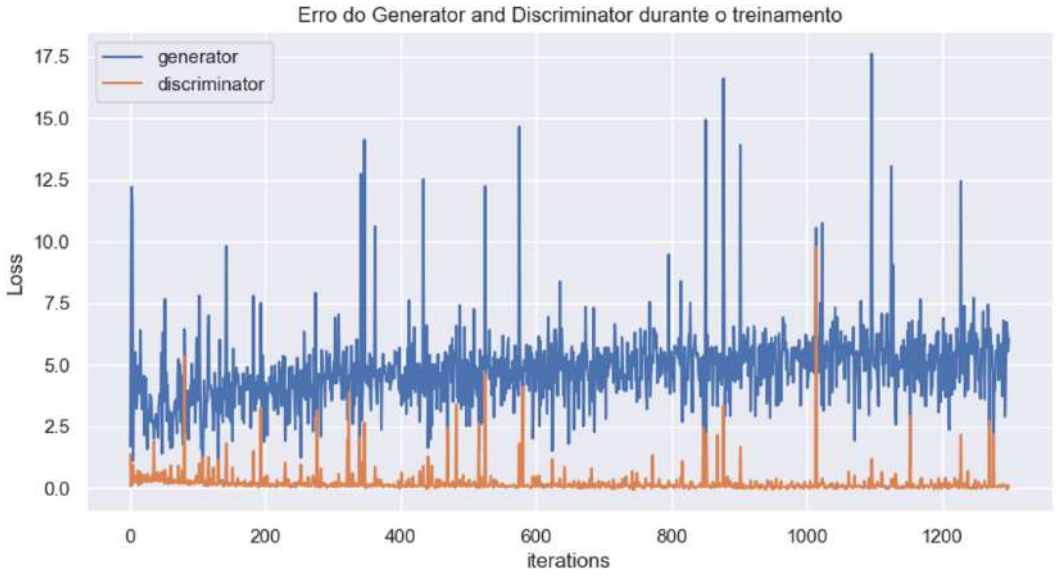
Final da trigésima época

Final da quadragésima época

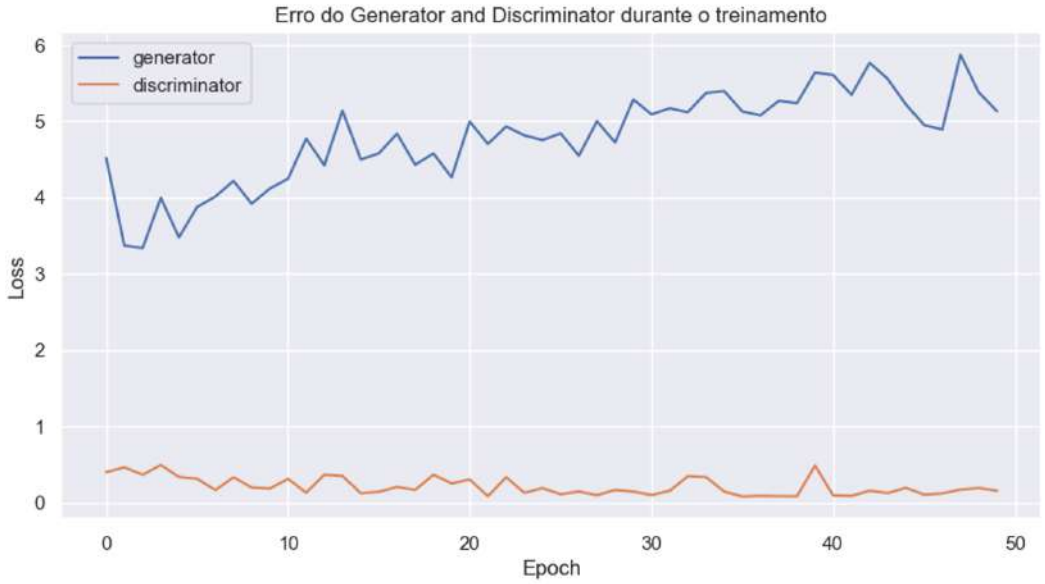
Final da quinquagésima época

As figuras acima mostram alguns exemplos de imagens geradas pela rede. Podemos ver que as imagens geradas são relativamente realistas com o passar das épocas e dos treinamentos e apresentam características de pessoas reais. Além disso, observamos que

após um certo período, temos uma estagnação, e rede não evolui tanto referente à qualidade das imagens. Abaixo temos a avaliação dos erros ao passar das iterações:



E o erro médio dos modelos por época:



Metodologia da Segunda Iteração

Introdução à modelos de Difusão

Os modelos de difusão tem como intuito espalhar informações de pixel em pixel, considerando os valores de pixels vizinhos, a fim de suavizar ou remover ruídos em imagens e melhorar a qualidade delas.

Como mencionado anteriormente, as informações dos pixels vizinhos são usadas para redistribuir os valores dos pixels da imagem original. Para isso, faz-se o uso de uma janela

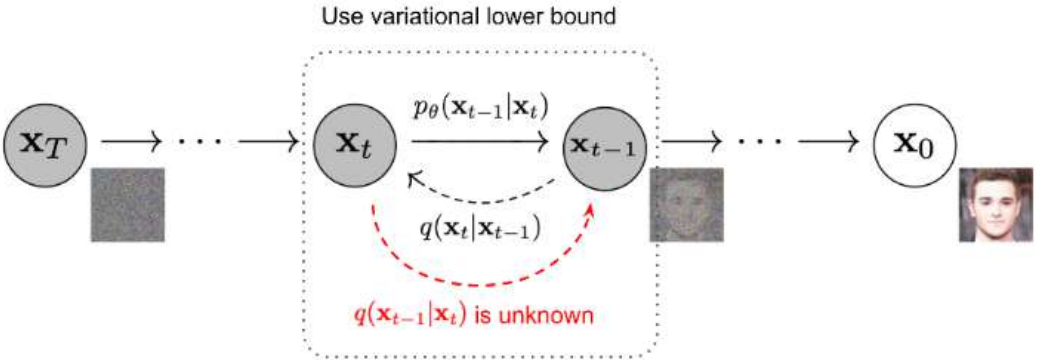
deslizante ao redor de cada pixel, que determina a região que será considerada pela função de difusão para calcular o novo valor do pixel central.

Um exemplo de função de difusão que é frequentemente utilizada é a média, que subsititui o valor do pixel central pela média dos vizinhos envolvidos pela janela, promovendo uma suavização global e removendo ruídos de alta frequência, como o *salt-and-pepper*. Podemos citar, também, a difusão anisotrópica, que considera os gradientes na imagem, o que permite que áreas mais uniformes sejam suavizadas enquanto as bordas sejam melhor preservadas.

Inspirados na área de *non-equilibrium thermodynamic*[4], modelos de difusão definem uma cadeia de markov de passos de difusão, que gradativamente incorporam ruído na imagem e aprendem a reverter esse processo, recriando imagens a partir de imagens com ruídos.

Dado uma amostra da distribuição real dos dados, definimos o processo de *forward* do modelo de difusão em que ruídos gaussianos são aplicados a amostra em T passos, produzindo uma sequência de amostras com ruído $x1,\ x2,\ \ldots,\ xT$.

O dado de amostra $x0$ gradualmente perde suas características, conforme t incrementa, uma vez que mais ruído é adicionado na imagem.



Processo de Markov que adiciona ruído na imagem original

Uma vez com os dados adicionados sistematicamente com ruído gaussiano, utilizaremos o modelo que será responsável por aprender a reverter o processo anterior, recuperando informações da imagem a partir do ruído. Assim, temos um processo de reconstrução de imagem a partir de um espaço latente.

Essa reconstrução requer a estimação da função densidade de probabilidade do passo anterior, dado contexto do estado atual do sistema. Isso é, estimar $q(x_{T-1}|\ x_T)$ quando $t = T$ e portanto, gerar os dados a partir do ruído gaussiano.Uma rede neural é utilizada para estimar $\rho\theta(x_{T-1}|\ x_T)$ baseado nos pesos do modelo θ e no estado atual t. Isso pode ser estimado da seguinte maneira:

$$p_\theta(x_{t-1}|x_t) = \mathcal{N}\left(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t)\right) \qquad p_\theta(x_{0:T}) = p(x_T)\prod_{t=1}^{T} p_\theta(x_{t-1}|x_t)$$

A parametrização para a função média foi proposta por Ho. et al. e pode ser computada por

$$\mu_\theta(x_t, t) = \frac{1}{\sqrt{a_t}}\left(x_t - \frac{\beta_t}{\sqrt{1-\overline{\alpha_t}}}\epsilon_\theta(x_t, t)\right)$$

Os autores em Ho. et. al., sugerem utilizar uma função de variância fixada como $\Sigma\theta = \beta T$. A amostra no tempo $t - 1$ pode ser computada como:

$$x_{t-1} = \frac{1}{\sqrt{a_t}}\left(x_t - \frac{1-a_t}{\sqrt{1-\overline{\alpha_t}}}\epsilon_\theta(x_t, t)\right) + \sigma_t z$$

A função de custo da rede neural será responsável por otimizar a seguinte expressão

$$\mathcal{L} = E_q\left(-logp(x_T) - \sum_{t\geq 1}\log\frac{p_\theta(x_{t-1}|\,x_t)}{q(x_t|x_{t-1})}\right)$$

Uma proposta de Sohl-Dickstein et al. resulta na seguinte simplificação, que formula a *loss* em termos da combinação linear da KL-divergence entre duas distribuições gaussianas em um conjunto de entropias:

$$\mathcal{L} = -E_q\left(D_{KL}(q(x_{t-1}|x_t, x_0)\;||\;p_\theta(x_{t-1}|x_t)) + H_q(x_T|x_0) - H_q(x_1|x_0) - H_p(x_T)\right)$$

Uma segunda simplificação, proposta por Ho et al., em que é utilizada a parametrização da média, que foi descrita acima:

$$\mathcal{L}_{simple} = -E_{t,x_0,\epsilon}\left(\left\|\epsilon - \epsilon_\theta\left(\sqrt{\overline{\alpha_t}}x_0 + \sqrt{1-\overline{\alpha_t}}\epsilon, t\right)\right\|^2\right)$$

Existem várias propostas de modelos generativos baseados em difusão, como *diffusion probabilistic models* (Sohl-Dickstein et al., 2015), *noise-conditioned score network* (NCSN; Yang & Ermon, 2019), e *denoising diffusion probabilistic models* (DDPM; Ho et al. 2020).

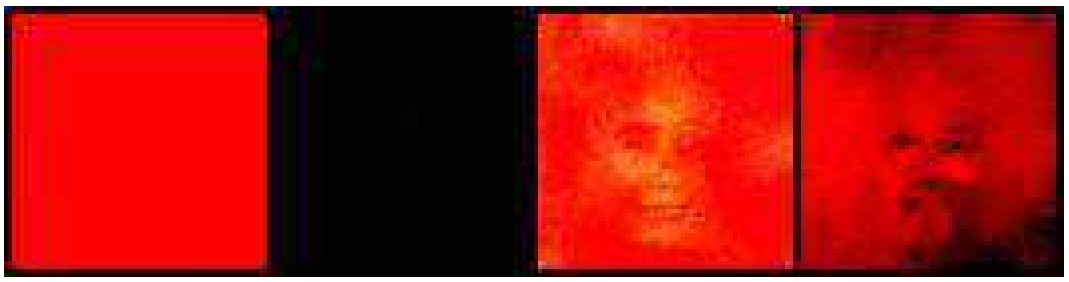
Nossa implementação

Nesse trabalho, foi utilizado um modelo mais simples, chamado *Unconditional Latent Diffusion,* com base no modelo UNet (Ronneberger et al., 2015). Esse modelo será capaz de gerar imagens a partir de um *latent space*, que é um espaço abstrato de vetores multidimensionais no qual são mapeadas informações latentes de uma determinada imagem real.

Além disso, treinamos o modelo por um total de 40 épocas, com um batch size limitado à 5 amostras (baixo devido ao limite de suporte de memória da GPU utilizada) e learning rate 3e-4. Esse treinamento decorreu durante aproximadamente 58 horas em uma placa de vídeo RTX3070.

Resultados

Para apresentar os resultados, a cada época utilizamos o modelo para gerar um conjunto de 4 imagens, e apresentamos agora os resultados para a primeira, segunda, terceira e última época de treinamento do modelo.

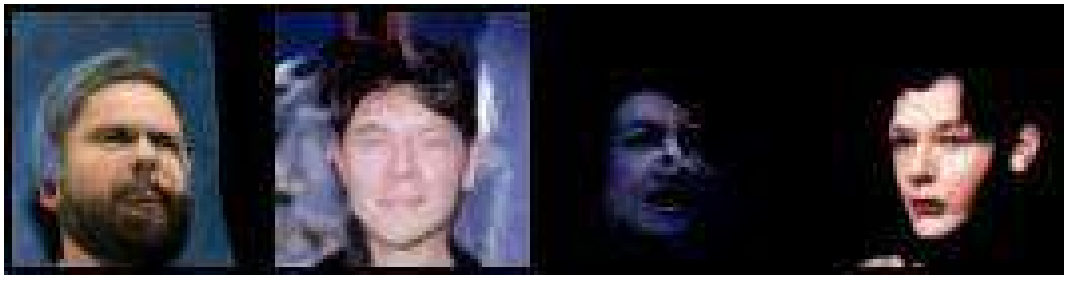


Primeira época

Décima época

Vigésima época

Trigésima época



Quadragésima época

As figuras acima mostram alguns exemplos das imagens geradas pela rede de difusão. É possível ver que as imagens geradas ficam mais realistas com o passar das épocas e dos treinamentos e apresentam características de pessoas reais.

Conclusões

Como podemos observar, comparando os resultados das imagens, observamos que ambos métodos resultaram em imagens relativamente parecidas com a realidade, considerando as capacidades de processamento utilizado, que impossibilitou um treinamento em mais épocas, com mais dados e maiores tamanhos de *batch*.

Analisando os resultados do modelo de difusão com o modelo anterior de redes adversariais, observamos os melhores resultados (as imagens falsas mais parecidas com a realidade) gerados pelos diferentes modelos mostram um melhor desempenho do método mais recente. Entretanto, não é algo tão constante, já que muitas imagens possuem ou uma iluminação saturada ou falta de luz na imagem, impossibilitando a visualização da face.

Uma questão a ser considerada, é a dificuldade para o treinamento do modelo de difusão, que levou cerca de 3 dias para terminar seu treinamento. Acreditamos que aumentando o poder computacional, os resultados poderiam ser muito melhores, uma vez que o modelo ainda não atingiu convergência e, ao contrário do modelo adversarial, que poderia apenas gerar *overfitting.* Assim, temos um trade-off entreresultado e custo de treinamento.

Repositório do projeto

O código do projeto pode ser encontrado no repositório do projeto no github

Referência bibliográfica

[1] CelebA. Dataset utilizado
Disponível em: https://arxiv.org/pdf/1406.2661.pdf
[2] Ian J. Goodfellow: Generative Adversarial Nets
Disponível em: https://arxiv.org/pdf/1406.2661.pdf
[3] Alec Radford: Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks
Disponível em: https://arxiv.org/pdf/1511.06434.pdf
[4] Jascha S.: Deep Unsupervised Learning using Nonequilibrium Thermodynamics
Disponível em: https://arxiv.org/pdf/1503.03585.pdf
[5] Face Generation with GANs. Blog post by Bharath K.
Disponível em: https://blog.paperspace.com/face-generation-with-dcgans/
[6] Olaf Ronneberger: U-Net - Convolutional Networks for Biomedical Image Segmentaion
Disponível em: https://arxiv.org/pdf/1505.04597.pdf
[7] Jonathan Ho: Denoising Diffusion Probabilistic Models
Disponível em: https://arxiv.org/pdf/2006.11239.pdf
[8] Alex Nichol: Improved Denoising Diffusion Probabilistic Models
Disponível em: https://arxiv.org/pdf/2102.09672.pdf
[9] Prafulla Dhariwal: Diffusion Models Beat GANs on Image Synthesis
Disponível em: https://arxiv.org/pdf/2105.05233.pdf