

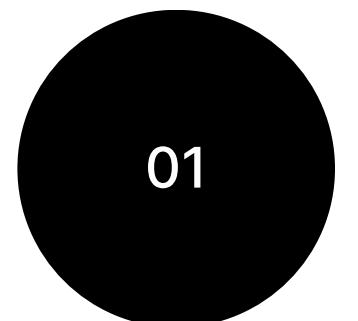
• • TRABALHO FINAL • •

GERAÇÃO DE AVATARES

COM VARIATIONAL AUTOENCODERS

Grupo: Ana Luísa Corsi, Enzo Lazzarini, Guilherme Cabral, João Lucas
Pontes e Laylla Royer

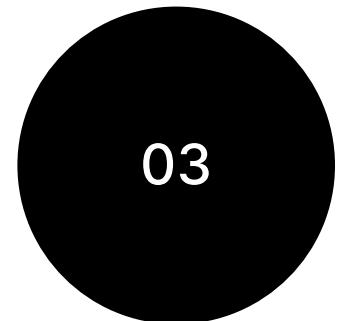
SUMÁRIO



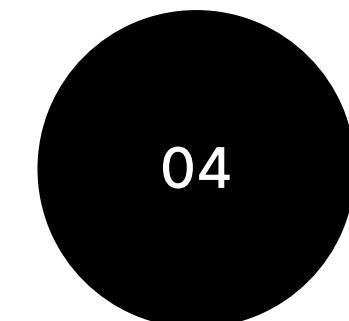
INTRODUÇÃO



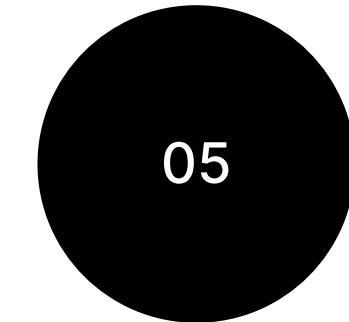
REVISÃO TEÓRICA



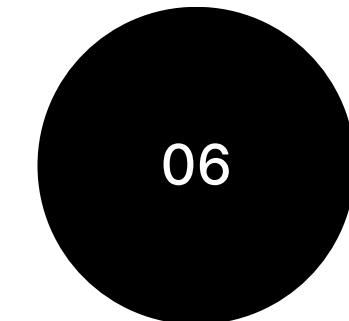
DESENVOLVIMENTO



RESULTADOS



CONCLUSÃO



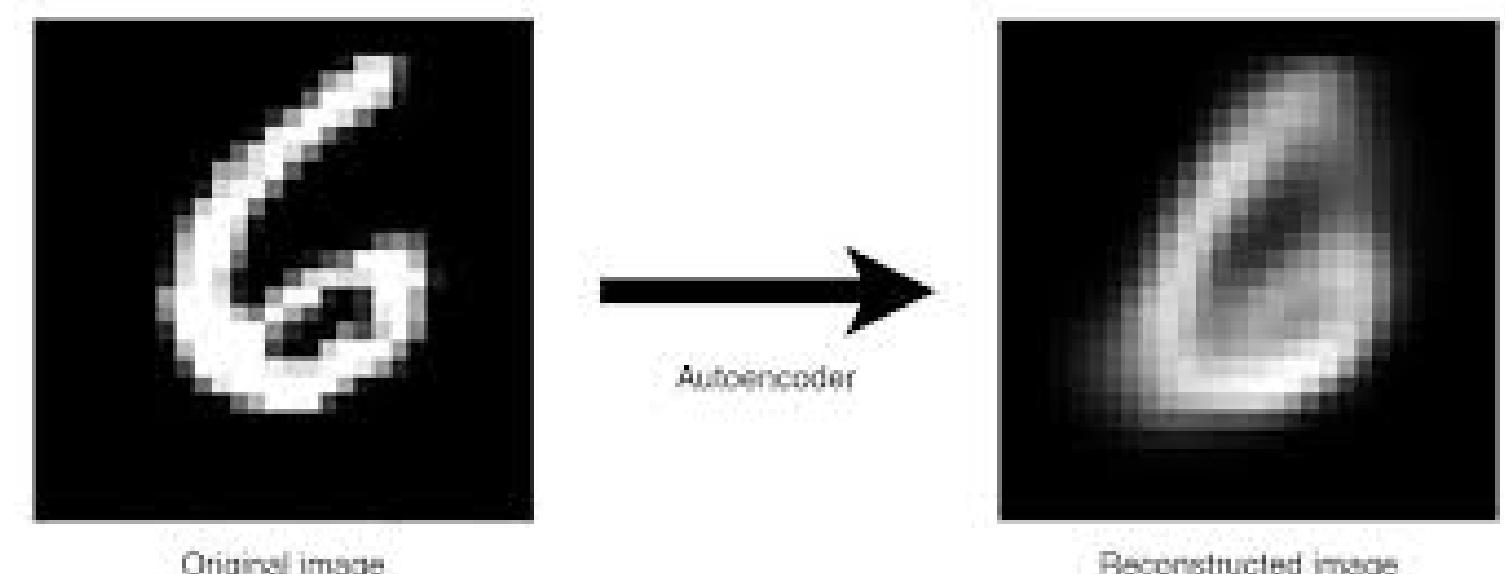
REFERÊNCIAS

INTRODUÇÃO

Autoencoders são redes neurais artificiais com a capacidade de representar dados com uma quantidade menor de dimensões, de tal forma que seja possível reconstruir os dados originais.

Porém não conseguem gerar novas imagens.

- **Objetivo principal:** não é prever uma saída y a partir de uma entrada x , mas sim aprender uma representação eficiente dos próprios dados de entrada.
- **Objetivo do treinamento:** minimizar o erro de reconstrução.



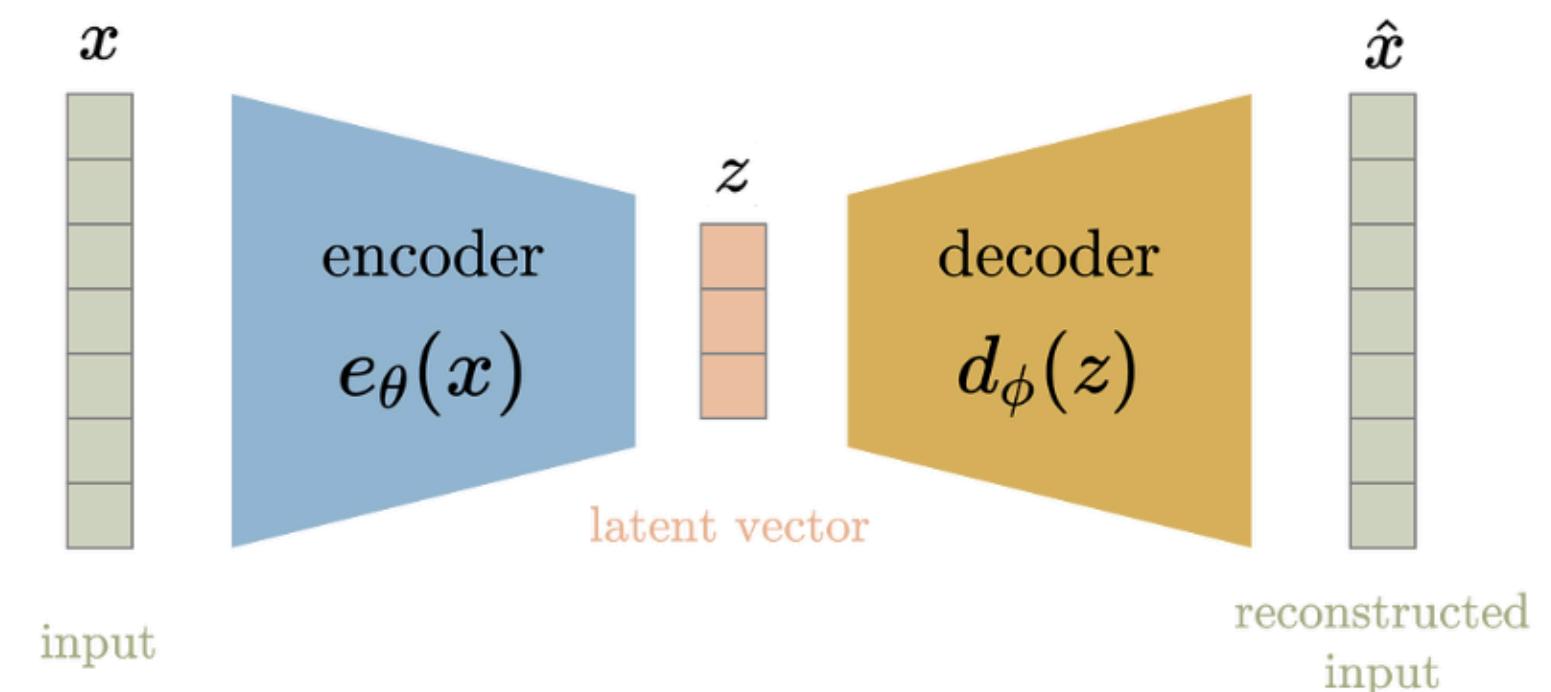
INTRODUÇÃO

ESTRUTURA

Encoder: Parte onde ocorre a redução de dimensionalidade.

Latent Space: Camada da rede onde temos menos dimensões para representar os dados sem perda ou quase sem perda de informação.

Decoder: Reconstrução do dado original.



INTRODUÇÃO

- **Limitações do AE**
 - processo de compressão-reconstrução "força" a aprender as características mais salientes dos dados.
 - entradas semanticamente similares são mapeadas para pontos próximos no espaço latente, formando clusters naturais.
- **MOTIVAÇÃO: sua falta de um viés sobre a estrutura global do espaço impede a geração de novos dados.**

DEFINIÇÃO DO PROBLEMA

Gerar instâncias artificiais a partir de um conjunto de dados reais. No caso específico, queremos gerar novos avatares por meio da técnica VAE.

OBJETIVOS DO TRABALHO

- Gerar avatares com AE e VAE;
- Comparar resultados produzidos por eles, buscando o melhor resultado.

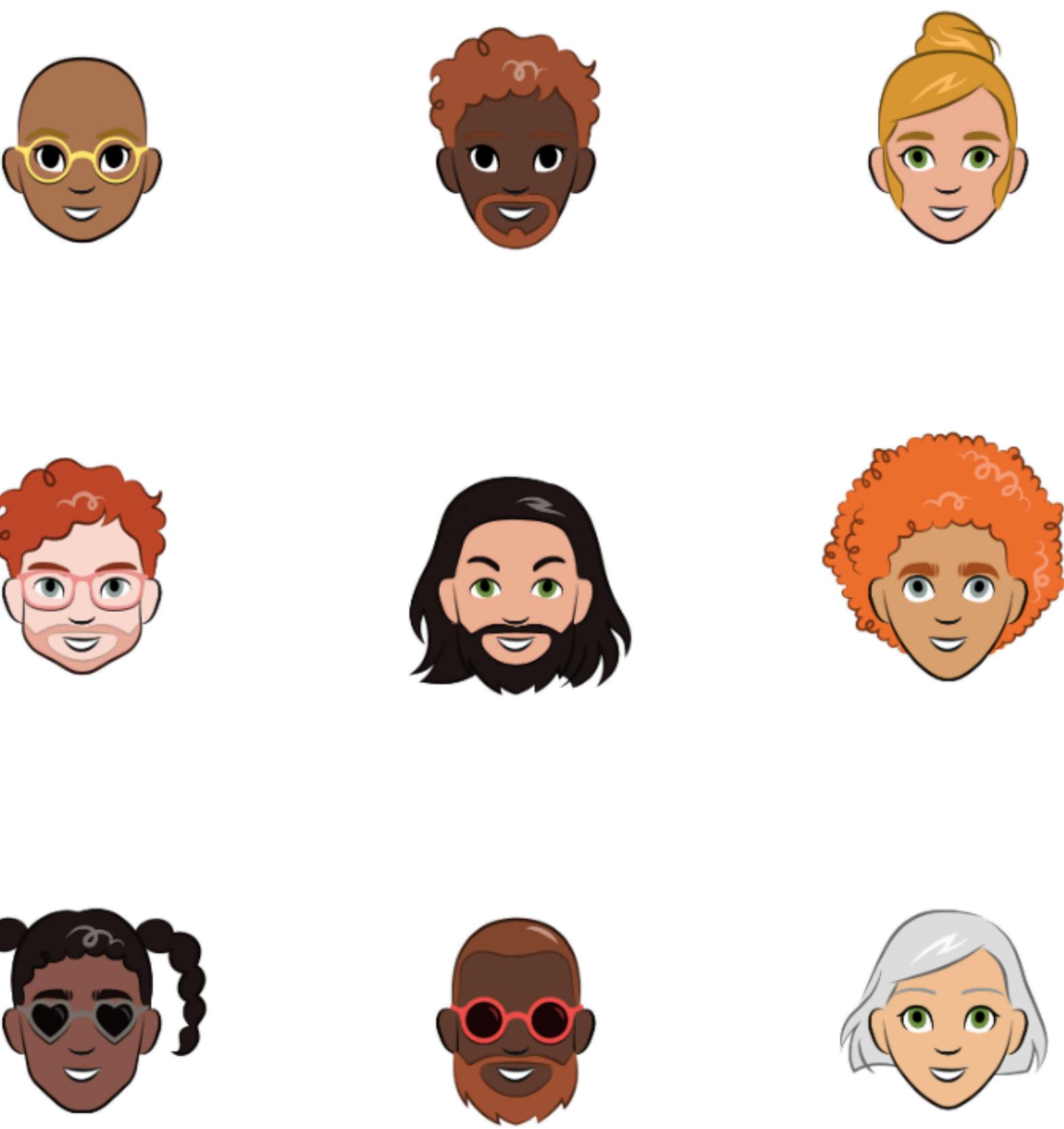
TRABALHO FINAL

INTRODUÇÃO

DATASET

Cartoon Set é uma coleção de desenhos digitais criados pelo artista Shiraz Fuman para o Google. Cada personagem é feito a partir de peças de arte criadas à mão, combinadas de forma automática para gerar muitos rostos diferentes com um estilo único e consistente.

Os avatares podem mudar no formato do rosto, olhos, boca, cabelo, acessórios e nas cores usadas. Existem duas versões, com 10k e 100k imagens.



REVISÃO TEÓRICA

VARIATIONAL AUTOENCODERS

- São uma evolução dos Autoencoders que os transforma em poderosos modelos gerativos.
- Impõe um viés indutivo muito mais forte e específico: assume que a **representação latente** dos dados pode ser descrita por uma **distribuição de probabilidade contínua e suave**.
- Força o modelo a criar um espaço latente onde não apenas os pontos são importantes, mas também as transições e interpolações entre eles.

AUTOENCODERS COMUNS

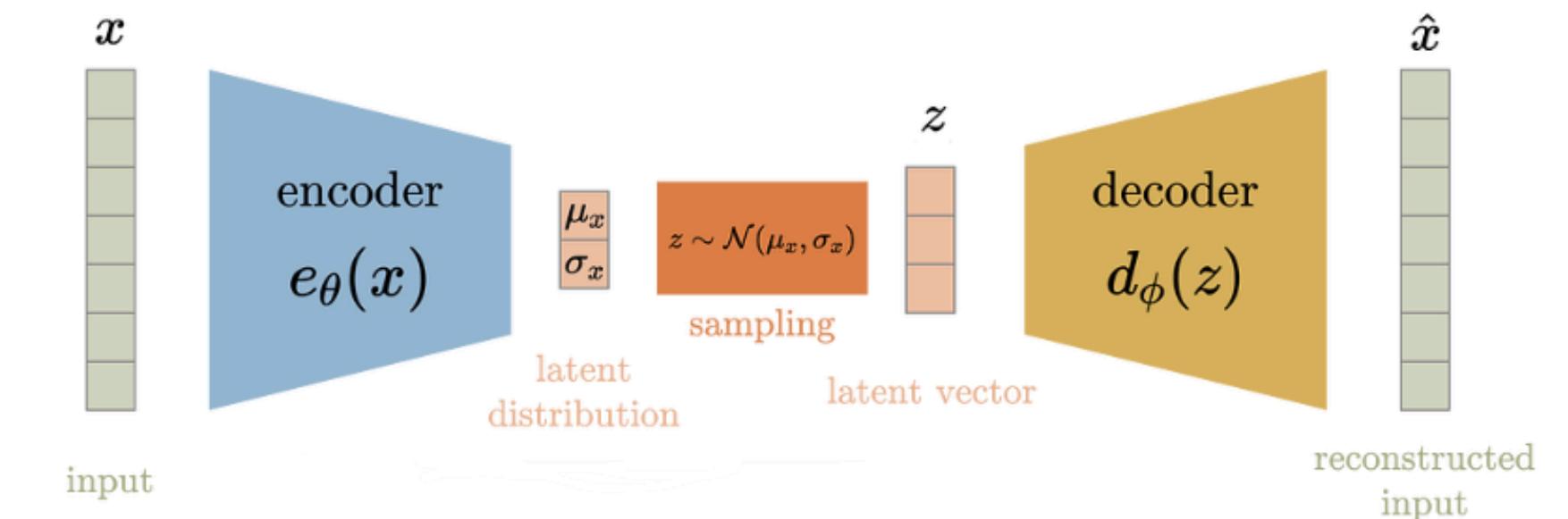
X VARIATIONAL AUTOENCODERS

- Encoder gera um vetor fixo;
- **Resultado:** Z é determinístico.
- Encoder gera uma distribuição;
- **Resultado:** Z é amostrado dessa distribuição.

REVISÃO TEÓRICA

VARIATIONAL AUTOENCODERS

- O Encoder não gera um vetor de ponto, mas sim **dois vetores** que parametrizam uma distribuição: um de **médias (μ)** e um de **desvios padrão (σ)**.



- **Ponto Chave:** O VAE não apenas aprende a representar os dados, mas aprende a representá-los de acordo com um "mapa" probabilístico pré-definido, o que o **torna generativo**.

REVISÃO TEÓRICA

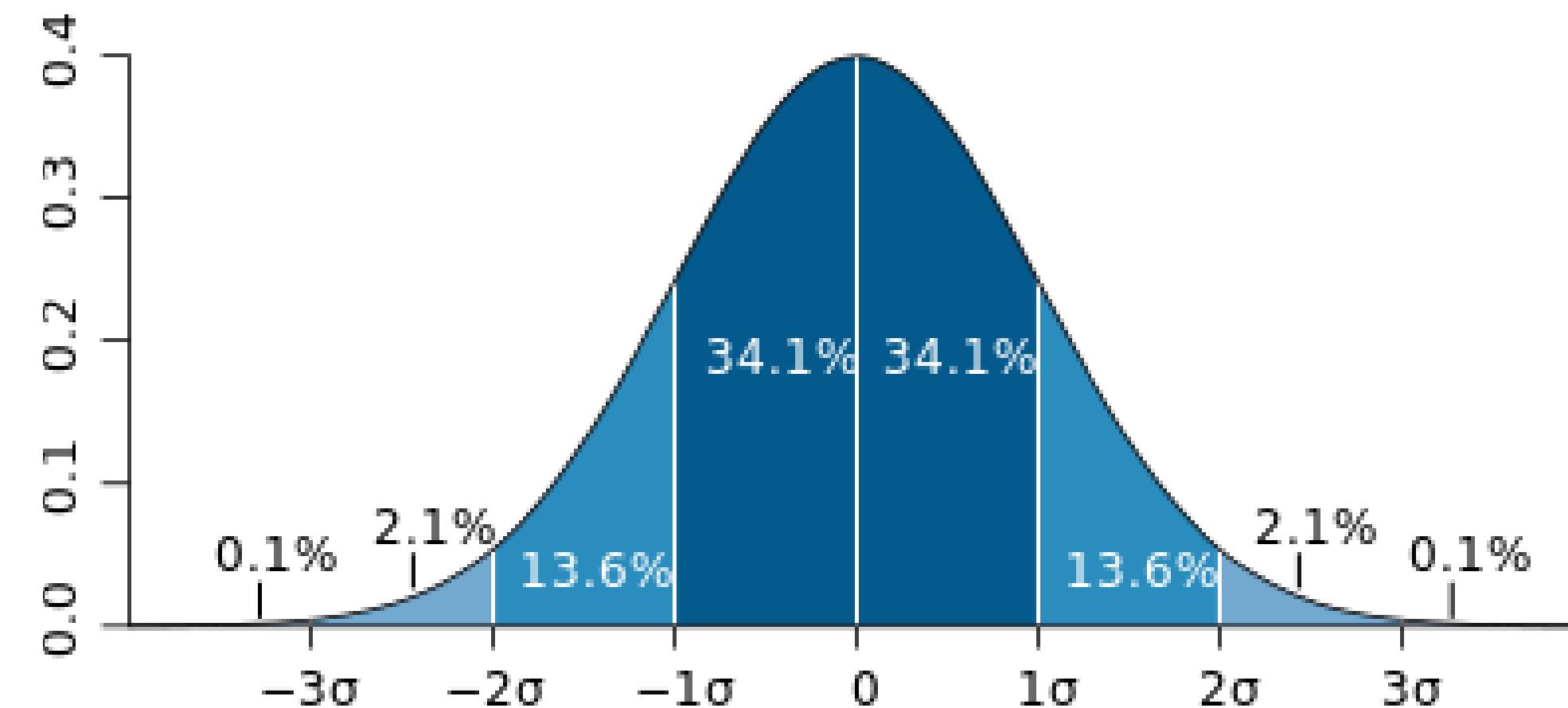
VARIATIONAL AUTOENCODERS

VAES modelam distribuições próximas as gaussianas para representar o espaço latente.

Os valores de cada dimensão são sorteados de acordo com a distribuição gerada pelo encoder.

Novos dados mais parecidos com o original tem probabilidade maior de serem gerados.

$$X \sim \mathcal{N}(\mu, \sigma^2)$$



REVISÃO TEÓRICA

VARIATIONAL AUTOENCODERS

- **Objetivo de Treinamento:**

- Erro de Reconstrução: Garante que a representação seja fiel aos dados (como no AE).
- Divergência KL (Termo de Regularização): Este é o termo que impõe o viés indutivo. Ele "pune" a rede se as distribuições aprendidas se afastarem muito de uma distribuição normal padrão, forçando a organização e a continuidade do espaço.

$$L(x) = E_{q(z|x)} [\log p(x | z)] - \text{KL}(q(z | x) \| p(z))$$

→ **Erro entre o dado original e o dado reconstruído
(Verossimilhança ou Erro Quadrático)**

→ **Diferença entre a distribuição normal e a distribuição gerada
(Regularização do Espaço)**

TRABALHO FINAL

DESENVOLVIMENTO

Autoencoder

VS

Variational Autoencoder

- Reconstrução
- Espaço Latente
- Geração de novas Imagens

CÓDIGO AUTOENCODER



ENCODER

Entrada: (None, 256, 256, 3)

- Número de amostras: indefinido
- Altura da imagem: 256px
- Largura da imagem: 256px
- Quantidade de cores: 3 (RGB)

Camadas escondidas: 16 camadas

→ 5 blocos compostos por:

- Camada de convolução
- Camada de normalização
- Camada de ativação

→ Uma camada de achatamento

Saída: Vetor de tamanho 200

→ Camada densa

```
def encoder(input_encoder):  
    inputs = keras.Input(shape=input_encoder, name='input_layer')  
    x = layers.Conv2D(32, kernel_size=3, strides=2,  
                      padding='same', name='conv_1')(inputs)  
    x = layers.BatchNormalization(name='bn_1')(x)  
    x = layers.LeakyReLU(name='lrelu_1')(x)  
  
    x = layers.Conv2D(64, kernel_size=3, strides=2,  
                      padding='same', name='conv_2')(x)  
    x = layers.BatchNormalization(name='bn_2')(x)  
    x = layers.LeakyReLU(name='lrelu_2')(x)  
  
    x = layers.Conv2D(64, 3, 2, padding='same', name='conv_3')(x)  
    x = layers.BatchNormalization(name='bn_3')(x)  
    x = layers.LeakyReLU(name='lrelu_3')(x)  
  
    x = layers.Conv2D(64, 3, 2, padding='same', name='conv_4')(x)  
    x = layers.BatchNormalization(name='bn_4')(x)  
    x = layers.LeakyReLU(name='lrelu_4')(x)  
  
    x = layers.Conv2D(64, 3, 2, padding='same', name='conv_5')(x)  
    x = layers.BatchNormalization(name='bn_5')(x)  
    x = layers.LeakyReLU(name='lrelu_5')(x)  
  
    flatten = layers.Flatten()(x)  
    bottleneck = layers.Dense(200, name='dense_1')(flatten)  
    model = tf.keras.Model(inputs, bottleneck, name="Encoder")  
    return model
```

CÓDIGO AUTOENCODER

DECODER

Entrada: Vetor de tamanho 200

→ Recebe o ponto do espaço latente gerado pelo encoder.

Camadas escondidas: 14 camadas

→ Uma camada densa para expandir o vetor latente.

→ Uma camada de *reshape*

→ 4 blocos compostos por:

- Camada de convolução transposta
- Camada de normalização
- Camada de ativação: LeakyReLU

→ Uma camada de convolução transposta final com ativação sigmoid.

Saída: (None, 256, 256, 3)

→ Imagem reconstruída

```
def decoder(input_decoder):  
    inputs = keras.Input(shape=input_decoder, name='input_layer')  
    x = layers.Dense(4096, name='dense_1')(inputs)  
    x = layers.Reshape((8,8,64), name='Reshape_Layer')(x)  
    x = layers.Conv2DTranspose(64, 3, strides=2,  
                             padding='same', name='conv_transpose_1')(x)  
    x = layers.BatchNormalization(name='bn_1')(x)  
    x = layers.LeakyReLU(name='lrelu_1')(x)  
    x = layers.Conv2DTranspose(64, 3, strides=2,  
                             padding='same', name='conv_transpose_2')(x)  
    x = layers.BatchNormalization(name='bn_2')(x)  
    x = layers.LeakyReLU(name='lrelu_2')(x)  
    x = layers.Conv2DTranspose(64, 3, 2, padding='same',  
                             name='conv_transpose_3')(x)  
    x = layers.BatchNormalization(name='bn_3')(x)  
    x = layers.LeakyReLU(name='lrelu_3')(x)  
    x = layers.Conv2DTranspose(32, 3, 2, padding='same',  
                             name='conv_transpose_4')(x)  
    x = layers.BatchNormalization(name='bn_4')(x)  
    x = layers.LeakyReLU(name='lrelu_4')(x)  
    outputs = layers.Conv2DTranspose(3, 3, 2, padding='same',  
                                 activation='sigmoid', name='conv_transpose_5')(x)  
    model = tf.keras.Model(inputs, outputs, name="Decoder")  
    return model
```

TRABALHO FINAL

CÓDIGO AUTOENCODER

TREINAMENTO



```
def train(dataset, epochs):
    for epoch in range(epochs):
        start = time.time()
        i = 0
        loss_ = []
        for image_batch in dataset:
            i += 1
            loss = train_step(image_batch)
            seed = image_batch[:25]
            display.clear_output(wait=True)
            generate_and_save_images([enc, dec],
                                     epoch + 1,
                                     enc.save_weights,
                                     dec.save_weights)

            display.clear_output(wait=True)
            generate_and_save_images([enc, dec],
                                     epochs,
                                     seed)
```



```
def ae_loss(y_true, y_pred):
    loss = K.mean(K.square(y_true - y_pred), axis = [1,2,3])
    return loss

@tf.function
def train_step(images):
    with tf.GradientTape() as encoder, tf.GradientTape() as decoder:
        latent = enc(images, training=True)
        generated_images = dec(latent, training=True)
        loss = ae_loss(images, generated_images)

    gradients_of_enc = encoder.gradient(loss, enc.trainable_variables)
    gradients_of_dec = decoder.gradient(loss, dec.trainable_variables)
    enc_optimizer.apply_gradients(zip(gradients_of_enc, enc.trainable_variables))
    dec_optimizer.apply_gradients(zip(gradients_of_dec, dec.trainable_variables))
    return loss
```

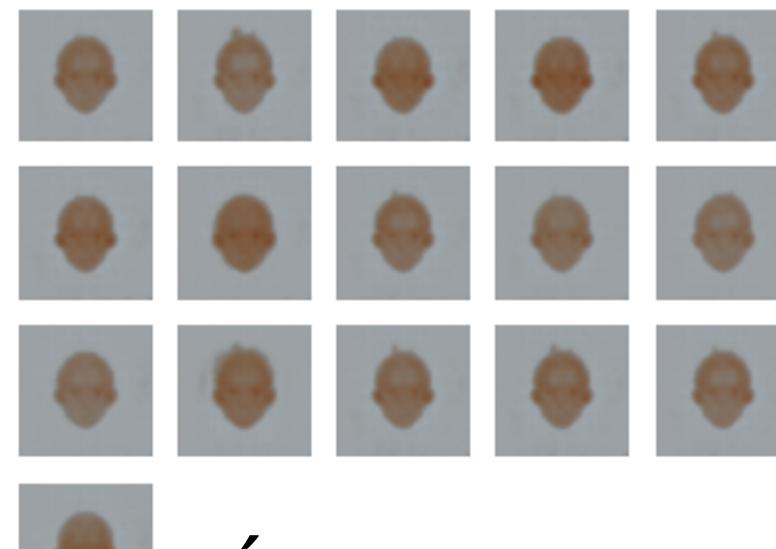
Para cada época, as imagens passam pelo **encoder**, a representação latente é construída e passada de volta para o **decoder**.

Depois, **calcula-se o erro para fazer o backpropagation**.

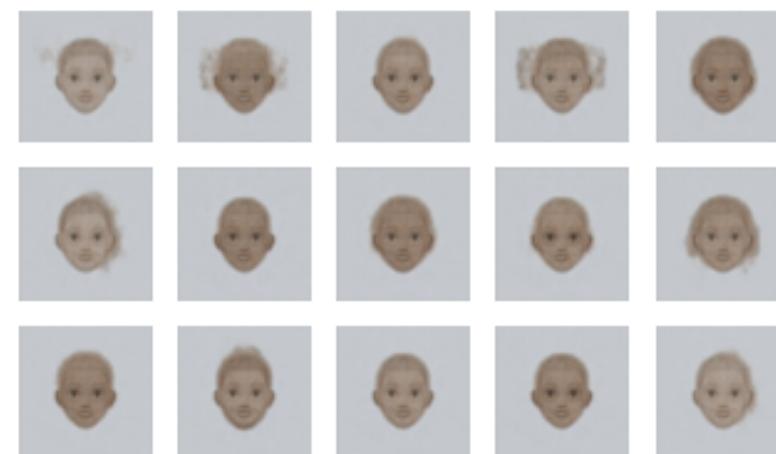
TRABALHO FINAL

CÓDIGO AUTOENCODER

TREINAMENTO



Época 1



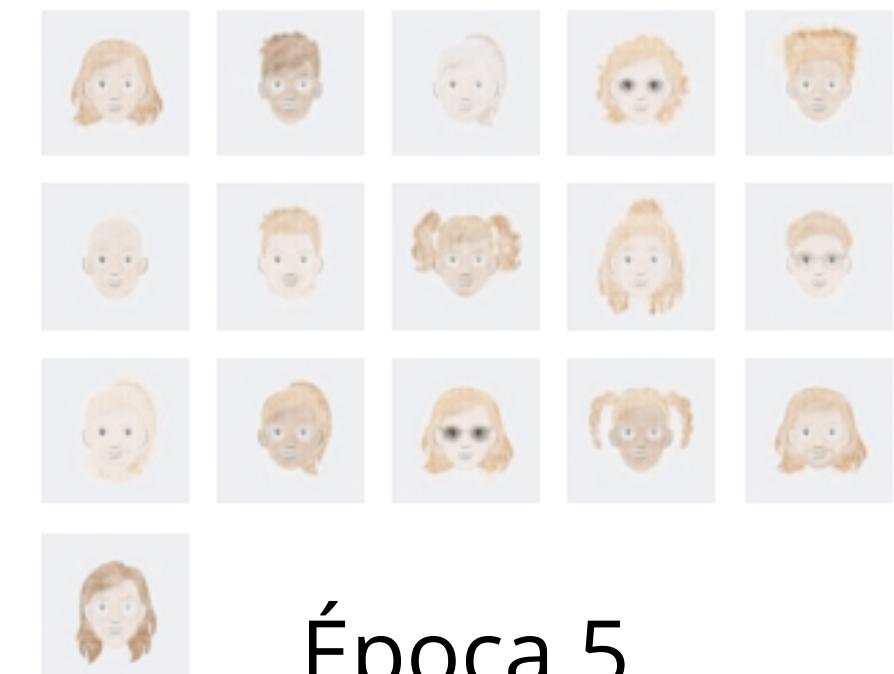
Época 2



Época 3



Época 4



Época 5

CÓDIGO VAE

ENCODER

Entrada: (None, 256, 256, 3)

- Número de amostras: indefinido
- Altura da imagem: 256px
- Largura da imagem: 256px
- Quantidade de cores: 3 (RGB)

Camadas escondidas:

- 5 blocos compostos por:
 - Camada de convolução
 - Camada de normalização
 - Camada de ativação
- Uma camada de achatamento

Totalizando 16 camadas

Saída: Dois vetores de tamanho 200
média e desvio padrão.

```

def encoder(input_encoder):
    inputs = keras.Input(shape=input_encoder, name='input_layer')
    # Block-1
    x = layers.Conv2D(32, kernel_size=3, strides=2,
                      padding='same', name='conv_1')(inputs)
    x = layers.BatchNormalization(name='bn_1')(x)
    x = layers.LeakyReLU(name='lrelu_1')(x)
    # Block-2
    x = layers.Conv2D(64, kernel_size=3, strides=2,
                      padding='same', name='conv_2')(x)
    x = layers.BatchNormalization(name='bn_2')(x)
    x = layers.LeakyReLU(name='lrelu_2')(x)
    # Block-3
    x = layers.Conv2D(64, 3, 2, padding='same',
                      name='conv_3')(x)
    x = layers.BatchNormalization(name='bn_3')(x)
    x = layers.LeakyReLU(name='lrelu_3')(x)
    # Block-4
    x = layers.Conv2D(64, 3, 2, padding='same',
                      name='conv_4')(x)
    x = layers.BatchNormalization(name='bn_4')(x)
    x = layers.LeakyReLU(name='lrelu_4')(x)
    # Block-5
    x = layers.Conv2D(64, 3, 2, padding='same',
                      name='conv_5')(x)
    x = layers.BatchNormalization(name='bn_5')(x)
    x = layers.LeakyReLU(name='lrelu_5')(x)
    # Final Block
    flatten = layers.Flatten()(x)
    mean = layers.Dense(200, name='mean')(flatten)
    log_var = layers.Dense(200, name='log_var')(flatten)
    model = tf.keras.Model(inputs, (mean, log_var), name="Encoder")
    return model

```

CÓDIGO VAE

DECODER

Entrada: Vetor de tamanho 200

→ Recebe o ponto do espaço latente gerado pela camada de sorteio.

Camadas escondidas: 14 camadas

→ Uma camada densa para expandir o vetor latente.

→ Uma camada de *reshape*

→ 4 blocos compostos por:

- Camada de convolução transposta
- Camada de normalização
- Camada de ativação: LeakyReLU

→ Uma camada de convolução transposta final com ativação sigmoid.

Saída: (None, 256, 256, 3)

→ Imagem reconstruída



```
def decoder(input_decoder):
    inputs = keras.Input(shape=input_decoder, name='input_layer')
    x = layers.Dense(4096, name='dense_1')(inputs)
    x = layers.Reshape((8,8,64), name='Reshape')(x)
    # Block-1
    x = layers.Conv2DTranspose(64, 3, strides= 2, padding='same',
                                name='conv_transpose_1')(x)
    x = layers.BatchNormalization(name='bn_1')(x)
    x = layers.LeakyReLU(name='lrelu_1')(x)
    # Block-2
    x = layers.Conv2DTranspose(64, 3, strides= 2, padding='same',
                                name='conv_transpose_2')(x)
    x = layers.BatchNormalization(name='bn_2')(x)
    x = layers.LeakyReLU(name='lrelu_2')(x)
    # Block-3
    x = layers.Conv2DTranspose(64, 3, 2, padding='same',
                                name='conv_transpose_3')(x)
    x = layers.BatchNormalization(name='bn_3')(x)
    x = layers.LeakyReLU(name='lrelu_3')(x)
    # Block-4
    x = layers.Conv2DTranspose(32, 3, 2, padding='same',
                                name='conv_transpose_4')(x)
    x = layers.BatchNormalization(name='bn_4')(x)
    x = layers.LeakyReLU(name='lrelu_4')(x)
    # Block-5
    outputs = layers.Conv2DTranspose(3, 3, 2,padding='same',
                                    activation='sigmoid',
                                    name='conv_transpose_5')(x)
model = tf.keras.Model(inputs, outputs, name="Decoder")
return model
```

CÓDIGO VAE

TREINAMENTO



```
def train_step(images):
    with tf.GradientTape() as encoder, tf.GradientTape() as decoder:
        mean, log_var = enc(images, training=True)
        latent = final([mean, log_var])
        generated_images = dec(latent, training=True)
        loss_vec = vae_loss(images, generated_images, mean, log_var)
        loss = tf.reduce_mean(loss_vec)
    gradients_of_enc = encoder.gradient(loss, enc.trainable_variables)
    gradients_of_dec = decoder.gradient(loss, dec.trainable_variables)
    optimizer.apply_gradients(zip(gradients_of_enc, enc.trainable_variables))
    optimizer.apply_gradients(zip(gradients_of_dec, dec.trainable_variables))
    return loss
```



```
def sampling(input_1, input_2):
    mean = keras.Input(shape=input_1, name='input_layer1')
    log_var = keras.Input(shape=input_2, name='input_layer2')
    out = layers.Lambda(sampling_model, name='encoder_output')([mean, log_var])
    enc_2 = tf.keras.Model([mean, log_var], out, name="Encoder_2")
    return enc_2
```

Para cada época, as imagens passam pelo **encoder**, a representação latente é construída e passada de volta para o **decoder**.

Depois, **calcula-se o erro para fazer o backpropagation**.

A representação latente é feita através de um **sorteio na distribuição normal** a partir do vetor de **média e desvio padrão**.

TRABALHO FINAL

CÓDIGO VAE

AVALIAÇÃO



```
def mse_loss(y_true, y_pred):  
    return 1000.0 * K.mean(K.square(y_true - y_pred), axis=[1,2,3])  
  
def kl_loss(mean, log_var):  
    return -0.5 * K.sum(1 + log_var - K.square(mean) - K.exp(log_var), axis=1)  
  
def vae_loss(y_true, y_pred, mean, log_var):  
    return mse_loss(y_true, y_pred) + kl_loss(mean, log_var)
```

O erro é calculado através da **soma dos dois tipos de erros**:

- **kl_loss** - Diferença entre a distribuição gerada e a normal.
- **mse_loss** - Perda de reconstrução da imagem.

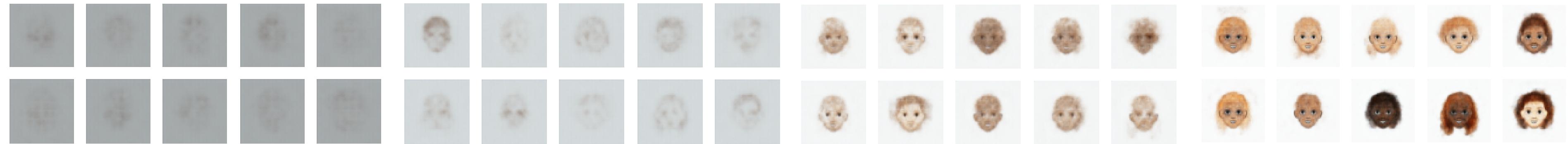
TRABALHO FINAL

CÓDIGO VAE

CONVERGÊNCIA



Época 1



Época 2



Época 5



Época 10



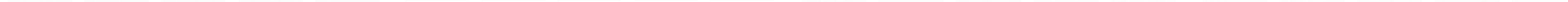
Época 15



Época 20



Época 25



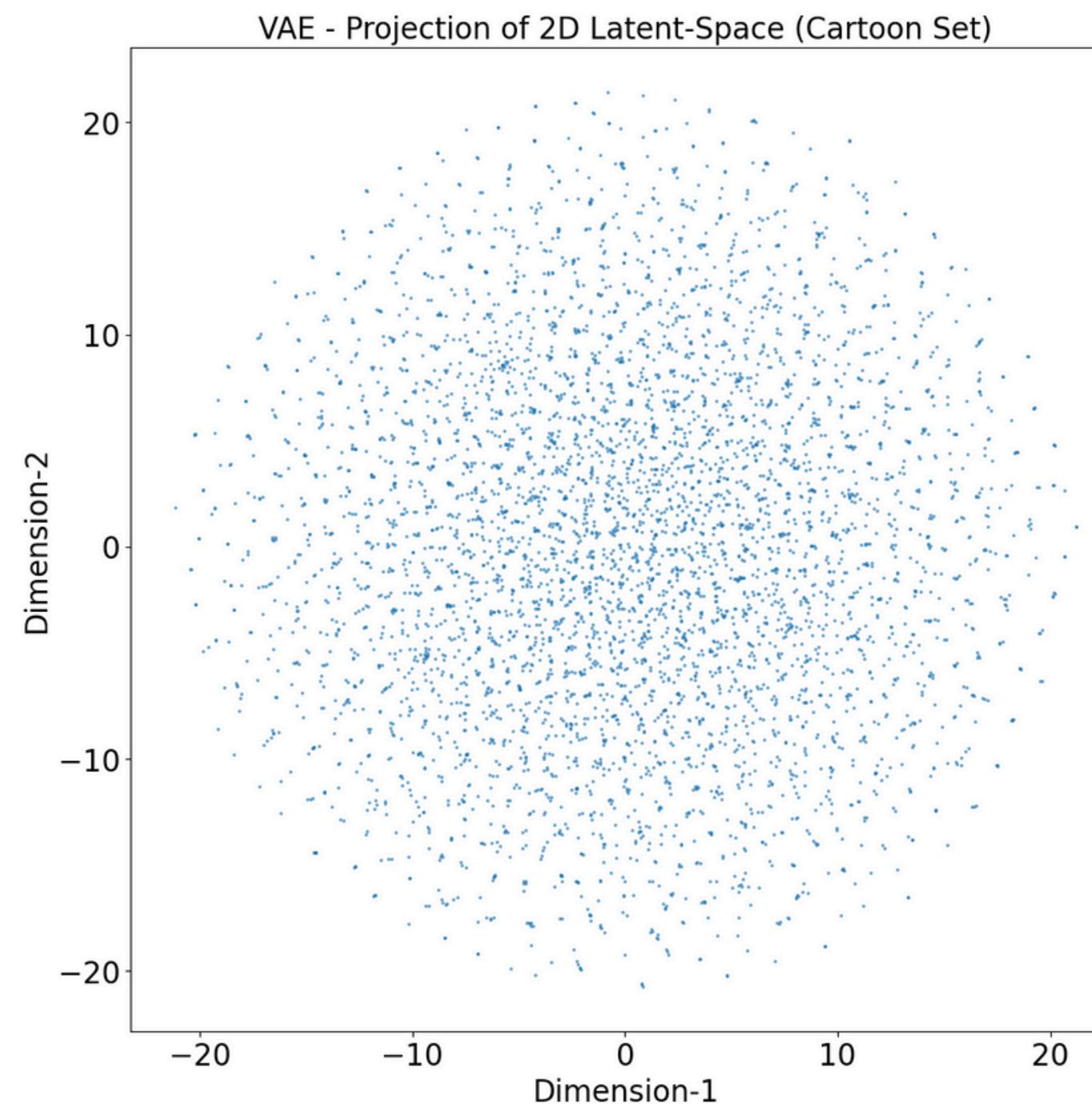
Época 30

TRABALHO FINAL

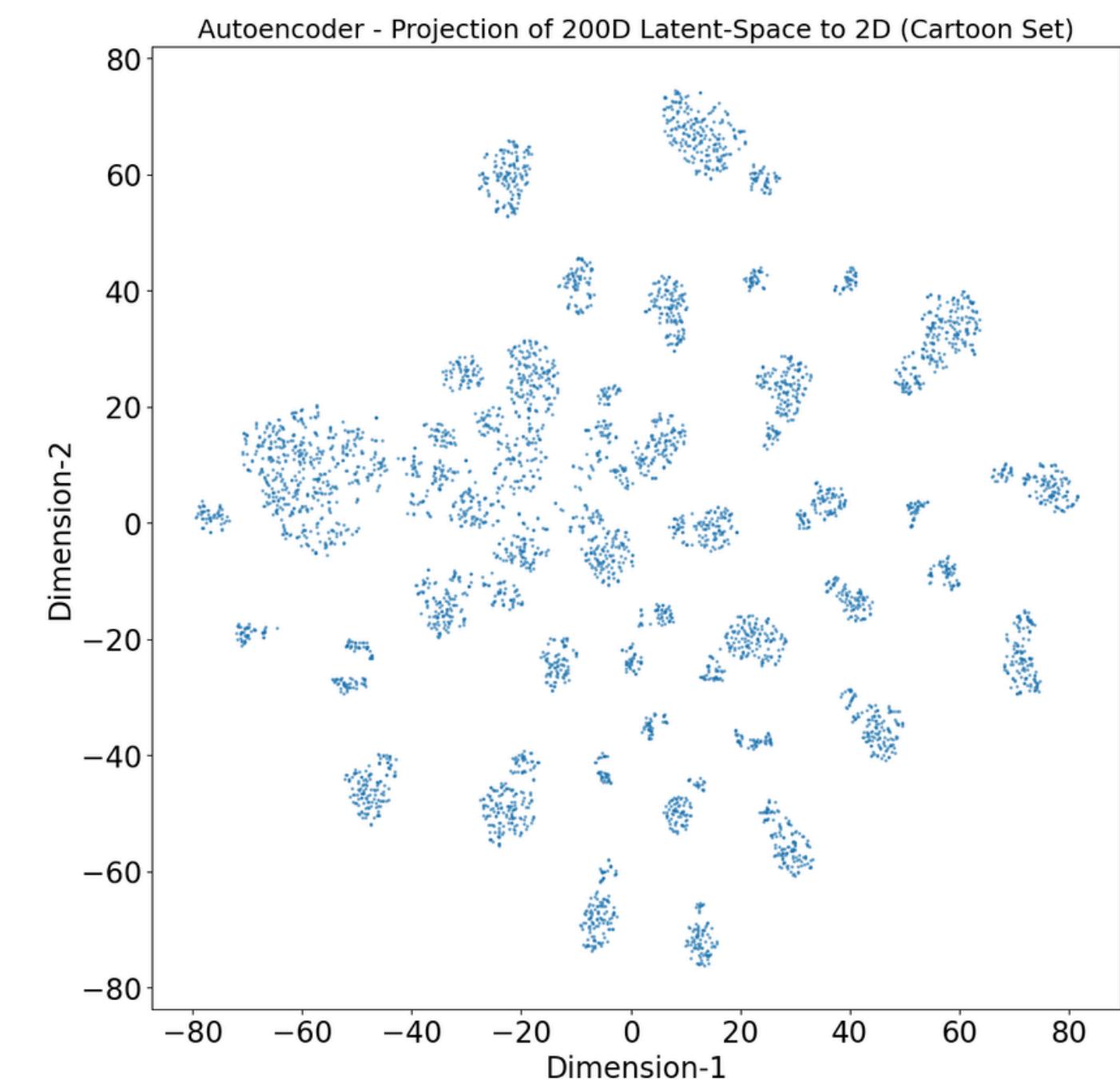
RESULTADOS

ESPAÇO LATENTE

VAE



AUTOENCODER

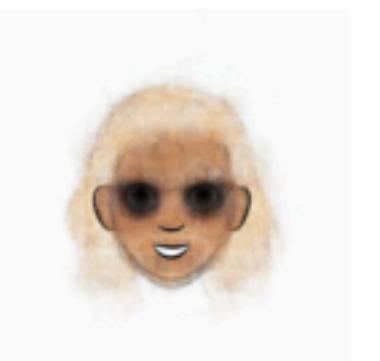


TRABALHO FINAL

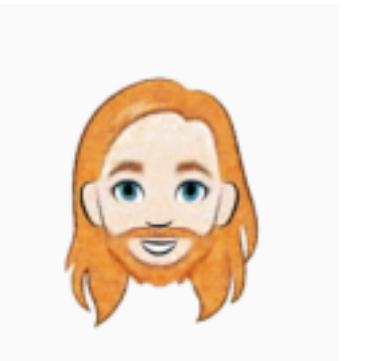
RESULTADOS

RECONSTRUÇÃO DE IMAGENS

VAE



AUTOENCODER

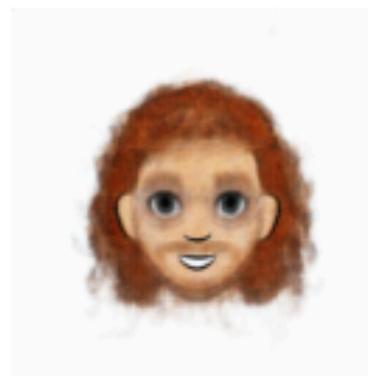


TRABALHO FINAL

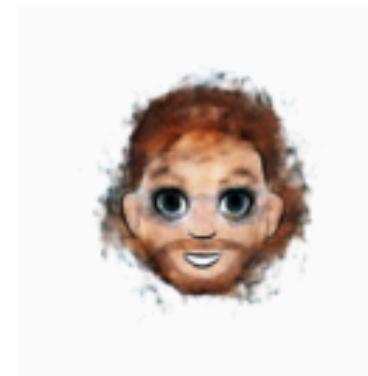
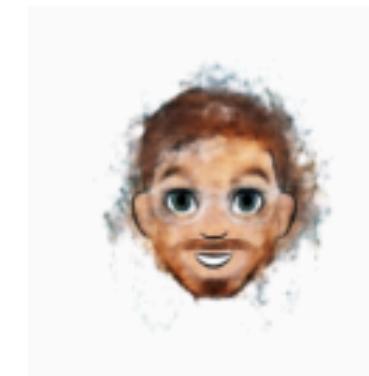
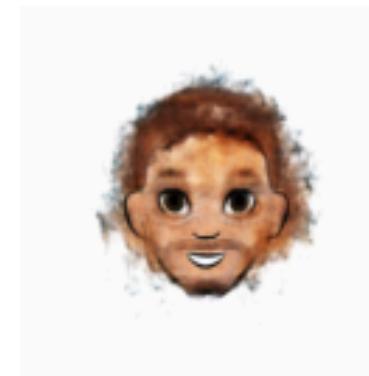
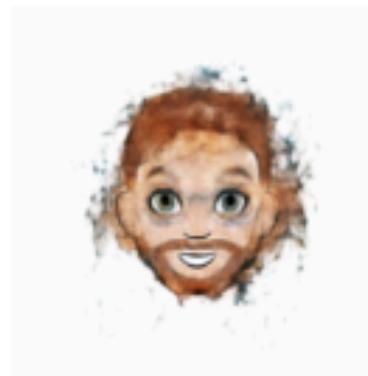
RESULTADOS

GERAÇÃO DE NOVAS IMAGENS POR SORTEIO (SAMPLE) NO ESPAÇO LATENTE

VAE



AUTOENCODER



TRABALHO FINAL

CONCLUSÃO

DIFERENÇA ENTRE AE X VAE

Autoencoder

- Reconstrói melhor (imagens mais nítidas)
- Espaço latente disperso → ruim para geração

Variational Autoencoder (VAE)

- Reconstrução menos detalhada
- Espaço latente contínuo e estruturado
- Gera imagens novas e plausíveis

CONCLUSÃO

VANTAGENS E DESVANTAGENS DO VAE

Vantagens

- Espaço latente regularizado (mais interpretável).
- Geração de amostras novas a partir de distribuição conhecida.
- Suporte a interpolação suave entre imagens.
- Melhor generalização para dados não vistos.

Desvantagens

- Reconstruções tendem a ser menos nítidas (imagens “borradas”).
- Ajuste delicado entre perda de reconstrução e divergência KL.
- Maior custo computacional em comparação ao AE simples.

CONCLUSÃO

- VAE é excelente para:
 - Aprender representações.
 - Gerar dados novos.
 - Visualizar e organizar espaços latentes.
- Limitações:
 - Imagens geradas podem ser borradadas.
 - Necessidade de ajuste fino entre regularização e reconstrução.

Código disponível em:
[GitHub](#)

TRABALHO FINAL

REFERÊNCIAS

<https://learnopencv.com/variational-autoencoder-in-tensorflow/#:~:text=In%20our%20previous%20post%2C%20we,and%20Google's%20Cartoon%20Set%20Data>

<https://learnopencv.com/autoencoder-in-tensorflow-2-beginners-guide>

dataset: <https://google.github.io/cartoonset/download.html>

Variational Autoencoders: How They Work and Why They Matter. (13 de agosto de 2024). DataCamp. Acesso em 2025, de

[Difference between Autoencoder \(AE\) and Variational Autoencoder \(VAE\). \(sem data\). Towards Data Science. Acesso em 2025, de](#)

[Variational Autoencoder Tutorial: VAEs Explained. \(sem data\). Codecademy. Acesso em 2025, de Variational Autoencoder Tutorial: VAEs Explained. \(sem data\). Codecademy.](#)

TRABALHO FINAL

REFERÊNCIAS

Doersch, C. (2016, 19 de junho). Tutorial on Variational Autoencoders (arXiv:1606.05908). arXiv.
<https://arxiv.org/abs/1606.05908>

Bergmann, D., & Stryker, C. (2024, 12 de junho). What is a variational autoencoder? IBM.
<https://www.ibm.com/think/topics/variational-autoencoder>

Variational Autoencoders: How They Work and Why They Matter. (13 de agosto de 2024). DataCamp. Acesso em 2025, de <https://www.datacamp.com/tutorial/variational-autoencoders>

Difference between Autoencoder (AE) and Variational Autoencoder (VAE). (sem data). Towards Data Science.
Acesso em 2025, de https://towardsdatascience.com/difference-between-autoencoder-ae-and-variational-autoencoder-vae-ed7be1c038f2/

Variational Autoencoder Tutorial: VAEs Explained. (sem data). Codecademy. Acesso em 2025, de
<https://www.codecademy.com/article/variational-autoencoder-tutorial-vaes-explained>

TRABALHO FINAL

REFERÊNCIAS

Diederik P Kingma, Max Welling (2013, 20 de dezembro). Auto-Encoding Variational Bayes (arXiv:1312.6114). arXiv. <https://arxiv.org/abs/1606.05908>

Bergmann, D., & Stryker, C. (2024, 12 de junho). What is a variational autoencoder? IBM. <https://www.ibm.com/think/topics/variational-autoencoder>

Variational Autoencoders | Generative AI Animated - Deepia

Variational Autoencoders - Arxiv Insights

Understanding Variational Autoencoders (VAEs) | Deep Learning - DeepBean