UNIVERSIDADE ESTADUAL DE CAMPINAS

INSTITUTO DE COMPUTAÇÃO

Exame de Qualificação de Mestrado

25 de Março de 2024

Estudo Comparativo de Autoencoders

Candidato: Wilson Bagni Júnior

Orientador: Prof. Dr. Zanoni Dias

Coorientador: Prof. Dr. Hélio Pedrini

Sumário

Resumo

O uso de *Autoencoders* tornou-se crucial no aprendizado de máquina contemporâneo, desempenhando um papel essencial em tarefas como redução de dimensionalidade, geração de dados sintéticos (aumentação de dados), remoção de ruído, extração de *features*, detecção de anomalias e reconhecimento de padrões. Este projeto de mestrado visa realizar um estudo comparativo abrangente de diversos modelos de *Autoencoders*, explorando suas principais características, pontos fortes e fracos. A pesquisa abordará, por exemplo, a análise do espaço latente gerado por cada modelo, sua capacidade de adaptação a bases de dados diferentes das que foram treinados, além de investigar as propriedades generativas, examinando sua habilidade em gerar dados sintéticos. Ao final, este projeto contribuirá para o avanço do conhecimento sobre *Autoencoders*, fornecendo um material organizado e de fácil compreensão sobre esta técnica que permitirá ao leitor uma escolha efetiva do melhor modelo a ser utilizado nas diversas aplicações de aprendizado de máquina.

Capítulo 1

Introdução

Autoencoders são redes neurais que utilizam uma técnica de aprendizado de máquina não supervisionado. Nessa técnica, a rede neural tem como objetivo prever na saída o mesmo dado fornecido na entrada e a rede é projetada com um progressivo estreitamento nas camadas iniciais e um progressivo alargamento da quantidade de neurônios nas camadas finais de forma a forçar uma compactação e reconstrução dos dados ao longo do processamento da rede.

A representação básica de um *Autoencoder* pode ser vista na Figura 1.1 na qual a primeira parte da rede – responsável pela compactação dos dados – é chamada de Codificador (*Encoder*) e o trecho responsável pela descompactação é chamado de Decodificador (*Decoder*). A camada que separa esses dois conjuntos é a representação (ou espaço) latente que carrega a representação dos dados de entrada na menor dimensionalidade da rede.

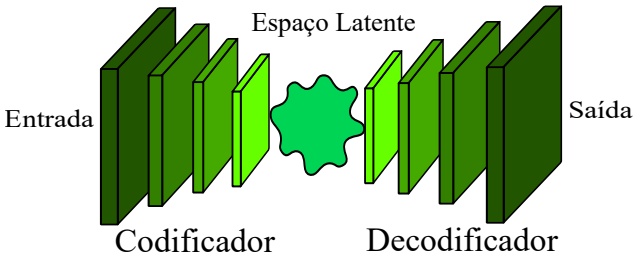

Figura 1.1: Exemplo de rede neural de um *Autoencoder*.

Durante a compressão, quando ocorre a diminuição da dimensionalidade, a rede é forçada a valorizar as informações que serão mais relevantes para reconstrução dos dados na saída da rede no momento da descompactação.

Nesse sentido, há trabalhos [3, 11] que procuraram estudar em maior profundidade o que ocorre no espaço latente da rede e como utilizá-lo de forma mais eficiente.

Apesar de parecer um processo simples, os *Autoencoders* são eficazes por seu potencial em extrair informações relevantes dos dados de uma forma não supervisionada.

1.1 *Autoencoders* e suas Aplicações

Ao revisar a literatura é comum encontrar *Autoencoders* sendo utilizados como ferramentas em diferentes contextos do aprendizado não-supervisionado [8, 18] já que são capazes de descobrir estruturas ocultas e padrões em dados não rotulados, permitindo assim a análise exploratória de conjuntos de dados, redução de dimensionalidade, detecção de anomalias e geração de dados sintéticos. Nesta seção, descrevemos uma lista, não exaustiva, de aplicações que utilizam esse tipo de rede neural.

Análise de imagens: ao comprimir imagens em dimensões menores, o codificador captura detalhes relevantes e padrões visuais das imagens. Essa abordagem permite a remoção de ruídos, a recuperação de informações importantes e a criação de representações visuais mais compactas e úteis para análise e outras tarefas de visão computacional [6, 12, 28, 34]. Dentro desse escopo, vale a pena citar o crescente uso de *Masked Autoencoders* [15] que utilizam-se de um conjunto codificador/decodificador associado a mecanismos de auto-atenção para "adivinhar" partes faltantes de uma imagem com base em apenas algumas poucas partes dela que são fornecidas como entrada.

Classificação de imagens: a tarefa de identificar e classificar imagens tem ocupado diferentes frentes de pesquisa dentro do aprendizado de máquina e, recentemente, uma arquitetura utilizando *Masked Autoencoders* em conjunto com Redes Convolucionais (ConvNeXt V2 [36]), atingiu o estado da arte na acurácia de classificação de objetos na *ImageNet* utilizando menor poder computacional que o modelo predecessor (MViTv2 [20]).

Compressão de dados: a capacidade do *Autoencoder* de comprimir os dados reside na restrição imposta pela dimensão da codificação, que força a rede a capturar apenas as características mais importantes dos dados. Isso o torna útil para a compressão de dados, reduzindo o espaço de armazenamento necessário ou acelerando a transmissão de informações, enquanto ainda permite uma recuperação razoável dos dados originais por meio do decodificador [23].

Detecção de anomalias: quando uma anomalia é apresentada ao *Autoencoder* já treinado, a decodificação tende a ser inadequada, resultando em um alto erro de reconstrução. Isso permite que a rede identifique instâncias anômalas com base em seu alto erro de reconstrução, tornando-a eficaz na detecção de padrões atípicos em várias aplicações, como segurança cibernética, manutenção preditiva e detecção de fraudes [32].

Extração de *features*: a camada latente do *Autoencoder* pode ser utilizada como um conjunto de *features* que serve como dado de entrada para modelos de aprendizado supervisionado, melhorando assim o desempenho em tarefas como classificação ou regressão, permitindo que o modelo se concentre nas informações mais essenciais dos dados de entrada [16, 21, 26].

Geração de dados sintéticos: ao alimentar o decodificador do *Autoencoder* com representações latentes modificadas por ruído, ou completamente aleatórias, é possível gerar, na

saída da rede, dados novos diferentes dos utilizados no treinamento, porém, que compartilham características semelhantes com o conjunto original treinado. Essa abordagem permite a criação de variações e ampliação do conjunto de dados, sendo útil para aumentar a diversidade nos dados de treinamento e auxiliar em tarefas de geração de conteúdo, como geração de imagens, texto e até música, com base nas informações aprendidas dos dados originais [29].

Processamento de texto: ao transformar sequências de palavras em representações densas e significativas, o *Autoencoder* permite a extração de características relevantes de dados textuais, capturando a semântica e a estrutura das sentenças. Essa codificação pode ser usada para tarefas como análise de sentimentos, sumarização de texto e tradução automática. As representações aprendidas pela rede podem melhorar a eficiência e o desempenho ao lidar com a complexidade e a variabilidade do texto [22].

Resolução de equações diferenciais: através da análise e do uso do espaço latente dos *Autoencoders*, esse tipo de rede neural tem sido utilizada para resolver Equações Diferenciais Parciais (*Partial Differential Equations* - PDE) e para a extração de informações relevantes do sistema físico descrito pela PDE analisada [1, 5, 14].

Sistemas de recomendação: o codificador converte os dados de avaliação ou preferências do usuário em uma representação compacta, capturando padrões de interesse e preferências. Essa codificação é usada para reconstruir as avaliações originais, minimizando a diferença entre as avaliações reais e as reconstruções. Essas representações latentes podem ser usadas para identificar relações entre itens e usuários, permitindo recomendações personalizadas com base em preferências semelhantes de outros usuários ou itens. Desta forma, os *Autoencoders* aprimoram a capacidade do sistema de recomendação de capturar características sutis e complexas dos dados, melhorando a precisão e a eficácia das sugestões oferecidas aos usuários [40].

Esse rol de aplicações auxilia a evidenciar o potencial de utilização dos *Autoencoders* em diversos tipos de problemas nos quais o aprendizado de máquina pode ser empregado.

1.2 Objetivos e Contribuições

O objetivo desta pesquisa é o estudo de modelos de *Autoencoders*, comparando principais características, suas aplicações, e avaliando também o espaço latente de cada modelo. Uma lista, não exaustiva de objetivos, engloba:

- Revisão da literatura sobre o tema;
- Identificação de modelos a serem estudados;
- Implementação dos modelos;

- Análise de desempenho, do espaço latente, do comportamento dos *Autoencoders* com dados distintos das bases de treinamento e de sua capacidade de gerar dados sintéticos;
- Complementação das análises aplicando *Autoencoders* em cenários específicos (por exemplo, em aumentação de dados);
- Publicação dos resultados obtidos.

Como contribuições, espera-se, além da publicação de resultados, trazer na dissertação um material consolidado sobre os *Autoencoders*, situando o leitor sobre o atual estado da arte sobre o tema e apresentar estudos comparativos entre modelos, principais conceitos-chaves e aplicações que permitam uma investigação profunda posterior.

1.3 Questões de Pesquisa

Durante o desenvolvimento da pesquisa, algumas perguntas servem como guia para as investigações realizadas:

- Quais modelos de *Autoencoders* possuem propriedades interessantes para serem estudados?
- Quais são os pontos fortes e fracos de cada modelo analisado?
- Há padrões que permitam fazer previsões ou estimativas de desempenho ou de outra propriedade de determinado modelo?
- Como é o desempenho dos modelos estudados em relação à geração de dados sintéticos?
- O quão adaptáveis são os modelos para lidar com dados diferentes daqueles utilizados no treinamento?

1.4 Organização do Texto

O Capítulo 2 traz conceitos e trabalhos relacionados ao tema de pesquisa que darão fundamentação teórica necessária para compreensão dos *Autoencoders*. No Capítulo 3, são apresentadas a metodologia, as métricas de avaliação e as bases de dados utilizadas durante o desenvolvimento da pesquisa. O Capítulo 4 apresenta o plano de trabalho e o cronograma para seu desenvolvimento. O Apêndice A apresenta alguns resultados obtidos até o momento.

Capítulo 2

Revisão Bibliográfica

Este capítulo descreve conceitos e técnicas relevantes relacionados ao tema sob investigação.

2.1 Fundamentação Teórica

Esta seção apresenta conceitos fundamentais para o entendimento do funcionamento dos diferentes tipos de *Autoencoders*.

2.1.1 Redes Neurais

Redes neurais são modelos computacionais inspirados no funcionamento do cérebro humano, projetados para realizar tarefas complexas de aprendizado e reconhecimento de padrões. Compostas por camadas de neurônios artificiais interconectados, essas redes processam informações de entrada, atribuindo pesos às conexões entre neurônios para aprender representações hierárquicas dos dados. A camada de entrada recebe os dados, que são propagados através das camadas ocultas, onde ocorre a computação ponderada. Cada neurônio nas camadas subsequentes aplica funções de ativação às suas entradas ponderadas, introduzindo não linearidades cruciais para a capacidade da rede de aprender padrões complexos.

Durante o treinamento, os pesos das conexões são ajustados iterativamente usando algoritmos de otimização, minimizando a diferença entre as saídas previstas e os rótulos reais. Isso permite que as redes neurais generalizem para novos dados, tornando-as poderosas ferramentas em tarefas como reconhecimento de imagem, processamento de linguagem natural e outras aplicações de aprendizado de máquina. As redes neurais são peças fundamentais utilizadas nas arquiteturas dos diferentes tipos de *Autoencoders* que serão descritos a seguir.

2.1.2 *Vanilla Autoencoder*

O modelo *Vanilla* representa o tipo mais básico de *Autoencoder*. Ele é composto essencialmente por duas partes: a) um Codificador, que converte os dados de entrada em uma representação latente de menor dimensão e b) um Decodificador, que reconstrói os dados originais a partir da dimensão reduzida. Estes componentes são estruturados com redes neurais simples, cujos pesos são ajustados durante o treinamento para minimizar a diferença entre

os dados de entrada e a reconstrução. Com camadas simétricas, este *Autoencoder* é amplamente utilizado em tarefas de redução de dimensionalidade e extração de características em conjuntos de dados, servindo como base para variações mais avançadas.

2.1.3 *Fully Connected Autoencoder* (FCAE)

O FCAE [31], também conhecido como *Dense Autoencoder*, é uma variação do tipo *Vanilla*, que também possui uma configuração básica, mas com a especificidade de utilizar camadas totalmente conectadas tanto no Codificador como no Decodificador. Apesar de possuir uma configuração simples, por não ser otimizado para uma tarefa específica, o FCAE é versátil e pode ser aplicado a uma variedade de tarefas de aprendizado não supervisionado, como, por exemplo, redução de dimensionalidade, remoção de ruídos e geração de dados.

2.1.4 *Denoising Autoencoder* (DAE)

O DAE [35] consiste de uma variante de *Autoencoder* que é projetada para aprender representações robustas, filtrando informações ruidosas. Como diferencial, o DAE treina a reconstrução dos dados a partir de versões originais que foram corrompidas deliberadamente com adição de ruído aos dados de entrada. Durante o treinamento, o objetivo é recuperar os dados originais a partir das versões ruidosas, incentivando a rede a aprender padrões mais significativos e ignorar o ruído. Essa abordagem ajuda a melhorar a capacidade do modelo de generalização, tornando-o mais resistente a variações e perturbações nos dados. O DAE é geralmente aplicado em tarefas de pré-processamento de dados, redução de ruído e extração de características robustas em ambientes com presença significativa de interferências ou variações nos dados.

2.1.5 *Convolutional Autoencoder* (CAE)

O CAE [38] é projetado especialmente para processar dados estruturados como os de imagens. Ele utiliza camadas convolucionais para capturar padrões locais e aprender representações espaciais dos dados. Tais camadas usam filtros que deslizam pela entrada, realizando operações de convolução para detectar características específicas. Portanto, o uso de convoluções na estrutura da rede neural deste modelo permite que o *Autoencoder* aprenda hierarquias de características complexas, tornando-o eficaz em tarefas de reconstrução e geração de imagens, bem como na extração de características robustas para diversas aplicações de visão computacional.

2.1.6 *Variational Autoencoder* (VAE)

O VAE [17] incorpora conceitos de inferência probabilística para aprender representações latentes mais robustas. Ele modela a distribuição probabilística das representações latentes, tendo como objetivo otimizar a rede para gerar representações que sigam uma distribuição específica, geralmente uma distribuição normal. Por exemplo, ao invés de gerar uma camada reduzida de forma direta a partir dos dados de entrada como num *Autoencoder* simples

(do tipo *Vanilla*), o Codificador do VAE aprende através dos treinamentos da rede a gerar um conjunto de valores de média e de logaritmo da variância (que funciona como uma aproximação do desvio-padrão) capazes de criar uma camada latente a ser processada pelo Decodificador.

Além de minimizar o erro na reconstrução dos dados na saída, a função de perda desse modelo tenta otimizar também a distribuição do espaço latente para que os valores de média e de logaritmo da variância formem uma distribuição normal. Isso permite, não apenas a geração de novas amostras na representação latente, mas também facilita a interpolação suave entre diferentes instâncias de dados. Espera-se que isso permita ao VAE criar um espaço latente melhor distribuído e com melhor capacidade generativa.

2.1.7 *Adversarial Autoencoder* (AAE)

O AAE [24] é uma variação mais avançada que incorpora elementos de redes adversariais generativas (*Generative Adversarial Networks* - GAN), para melhorar a qualidade das representações latentes geradas. Ele consiste de três partes principais: o Codificador, o Decodificador e um Discriminador. Durante o treinamento, o Discriminador é incentivado a distinguir entre representações latentes reais e falsas, forçando o modelo como um todo a criar representações latentes mais realistas. Essa abordagem adversarial aprimora a capacidade do *Autoencoder* em gerar representações mais significativas e próximas da distribuição real dos dados, resultando em uma técnica poderosa para aprendizado não supervisionado e de geração de dados.

2.1.8 *Recurrent Autoencoder* (RAE)

O RAE [7] é uma variante de *Autoencoder* que incorpora camadas recorrentes para processar sequências de dados e, portanto, é projetado para lidar com a natureza dinâmica de informações sequenciais, como séries temporais ou dados de linguagem. As camadas recorrentes permitem que a rede mantenha estados ocultos, capturando informações contextuais ao longo do tempo.

2.1.9 *Sparse Autoencoder* (SAE)

O SAE [27] incorpora mecanismos para promover a esparsidade nas camadas de aprendizado da rede neural. Ele introduz termos adicionais na função de perda durante o treinamento, incentivando a ativação seletiva de neurônios na camada oculta. O objetivo é forçar a rede a aprender representações compactas e eficientes, com muitos neurônios permanecendo inativos na maioria dos dados processados. Isso resulta em uma representação esparsa, onde apenas um subconjunto limitado de neurônios é ativado para codificar padrões específicos. O SAE costuma ser utilizado para extração de características robustas e eficientes em conjuntos de dados, contribuindo para uma representação mais interpretável e econômica de recursos computacionais.

2.1.10 *Contractive Autoencoder* (ConAE)

O ConAE [30] incorpora uma "penalização por contrato" durante o treinamento para aprender representações mais estáveis. Ele introduz um termo na função de perda do modelo que penaliza as mudanças nas ativações da camada oculta em relação às alterações nas entradas, de forma a incentivar a rede a aprender representações que sejam insensíveis a pequenas variações nos dados de entrada. Essa abordagem visa criar uma representação mais compacta e resistente a perturbações nos dados, o que é benéfico para tarefas de reconstrução e generalização. O ConAE é utilizado em situações em que é desejável que a rede ignore variações irrelevantes nos dados de entrada, como, por exemplo, extração de características (*features*), remoção de ruído e detecção de anomalias.

2.1.11 *Masked Autoencoder* (MAE)

O MAE [15] mistura técnicas de mascaramento dos dados – dividindo em recortes (*patches*) menores e omitindo parte deles no treinamento – com mecanismos de auto-atenção ao treiná-los no seu conjunto Codificador/Decodificador. Essa abordagem contribui para uma representação mais seletiva e adaptável, resultando em um desempenho aprimorado em tarefas de reconstrução e geração de dados estruturados. Ele é especialmente útil em tarefas onde a relevância de diferentes elementos dos dados pode variar ao longo do tempo, como em séries temporais, linguagem natural ou dados com padrões irregulares.

2.2 Trabalhos Relacionados

Existem trabalhos que procuraram analisar diferentes tipos e usos de *Autoencoders* e estes podem ser tomados por base para este projeto e complementados com a adição da análise de modelos mais recentes. No entanto, vale destacar que, embora seja comum encontrar na literatura diversas aplicações de *Autoencoders*, não há muito material dedicado a estudar essa técnica por si ou comparar suas variantes.

Por exemplo, Bank, Koenigstein e Giryes [2] e Zhai *et al.* [39] apresentaram trabalhos em formato de *survey* com explicações sobre o funcionamento e principais características de diferentes modelos de *Autoencoders*, porém, não desenvolveram um estudo sistemático e comparativo entre eles. Embora tenham abordado uma variedade de arquiteturas, os modelos mais recentes, como por exemplo o RAE e o MAE, não foram analisados.

Outros trabalhos encontrados na revisão bibliográfica que se propuseram a comparar modelos de *Autoencoders* estavam restritos a um contexto específico. Neste sentido, podemos destacar o trabalho de Li, Pei e Li [19] que apresenta um *survey* bem completo, porém, com enfoque para *deep learning* e, da mesma forma que os trabalhos citados anteriormente, não se aprofunda em desenvolver uma comparação sistemática entre os modelos.

Ainda numa abordagem de comparação de modelos de *Autoencoders*, encontramos trabalhos dedicados à comparação com arquiteturas muito específicas [33] nas quais não é possível generalizar as conclusões e também situações de contexto muito específico como no caso de um *survey* de modelos destinados ao uso em sistemas de recomendação [40].

Capítulo 3

Material e Métodos

Este capítulo descreve a metodologia, as bases de dados, as métricas de avaliação e os recursos computacionais que serão utilizados no desenvolvimento do projeto.

3.1 Metodologia

Esta seção apresenta os procedimentos adotados para análise e comparação dos modelos de *Autoencoders* entre si. As principais etapas descritas nesta metodologia estão ilustradas na Figura 3.1.

Implementação de modelo: após definirmos um modelo específico de *Autoencoder* a ser estudado ele será então implementado e testado para verificar se seu desempenho está de acordo com o descrito na literatura.

Definição da arquitetura da rede: uma vez em posse de um modelo funcional, realizaremos uma rodada de experimentos para determinar qual a melhor configuração de camadas a serem utilizadas com a base de dados (vide Seção 3.2) que será utilizada nos experimentos.

Variação do espaço latente: com a arquitetura do *Autoencoder* definida, ele será treinado para diferentes tamanhos de camada latente. Os pesos do modelo em cada treinamento serão armazenados para que possam ser reutilizados nos testes subsequentes. De forma geral, o treinamento consiste em otimizar os pesos da rede neural para que ela preveja na saída do Decodificador o mesmo dado fornecido na entrada do Codificador. Dependendo do modelo estudado, a função de perda do treinamento poderá otimizar também outros valores de interesse que sejam específicos do *Autoencoder*.

Performance do *Autoencoder* na reconstrução dos dados: para análise da qualidade do modelo em reconstruir na saída os dados de entrada, utilizaremos a métrica *Root Mean Squared Error* (RMSE), definida na Seção 3.3.1, em um conjunto de dados de teste que não foi visto anteriormente no treinamento do modelo. Essa medida será obtida para os diferentes tamanhos de camada latente previamente treinados e assim será possível avaliar a eficácia do

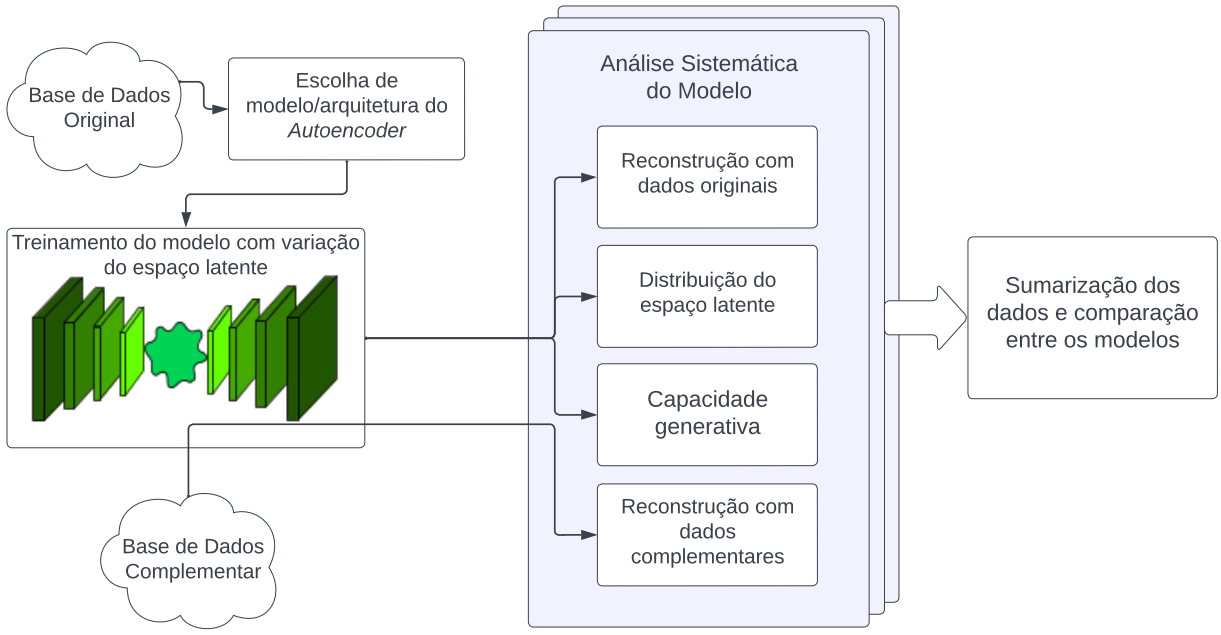

Figura 3.1: Etapas que compõem a metodologia utilizada neste projeto.

modelo com relação à redução de dimensionalidade empregada para um mesmo conjunto de dados.

Qualidade das imagens reproduzidas: caso seja utilizado um conjunto de dados de imagens para treinamento do modelo, será possível realizar uma análise visual, complementar à medição da eficácia do modelo pela métrica RMSE, descrita anteriormente. Para isso, uma imagem do conjunto de teste de cada classe da base de dados será escolhida para ser fornecida para o *Autoencoder* pré-treinado em diferentes tamanhos de camada latente. As imagens serão agrupadas lado a lado de forma que possa ser possível realizar uma comparação visual da qualidade dos dados gerados.

Distribuição dos dados no espaço latente: para avaliar as características da redução de dimensionalidade empreendida pelo Codificador, utilizaremos a técnica *Uniform Manifold Approximation and Projection* (UMAP) [25], de forma não supervisionada, em um conjunto de dados de teste. Baseada em métodos de Geometria Riemanniana e de Teoria de Grafos, o UMAP utiliza uma abordagem de otimização global para mapear dados em um espaço de menor dimensão, buscando manter a estrutura local e global dos dados de forma mais equitativa do que outras técnicas de redução de dimensionalidade. Essa técnica é especialmente eficaz na visualização de agrupamentos, preservando distâncias e revelando padrões intrínsecos em conjuntos de dados complexos, sendo amplamente utilizada em análise exploratória de dados e compreensão visual de dados multidimensionais.

Métricas de redução de dimensionalidade: de forma complementar ao método de visualização de dados utilizando o UMAP, usaremos um conjunto de 5 métricas específicas

para análise de redução de dimensionalidade entre os dados de entrada e do espaço latente: *trustworthiness, continuity, normalized stress, neighborhood hit* e *shepard goodness*. Estas métricas são descritas e explicadas em detalhes na Seção 3.3.2. Cada uma delas avalia um aspecto específico da redução de dimensionalidade e, quando analisadas em conjunto, permitem uma visão geral de suas características.

Desempenho do *Autoencoder* com dados diferentes: para essa análise, o *Autoencoder* pré-treinado em diferentes tamanhos de espaço latente processará dados de uma base diferente daquela que foi utilizada para treinamento do modelo. Serão escolhidas imagens de cada classe desta nova base de dados e as imagens resultantes serão organizadas lado a lado de forma a permitir uma análise visual da evolução das imagens processadas de acordo com a variação do tamanho do espaço latente.

Capacidade generativa: para avaliar a capacidade do *Autoencoder* de gerar novos dados (sintéticos), realizaremos a codificação de um dado de entrada do conjunto de teste e sobre o ponto gerado no espaço $d-$dimensional pela camada latente resultante dessa codificação, aplicaremos uma pequena perturbação de forma a obter um novo ponto que espera-se estar próximo o bastante do original para que seja gerado um exemplar da mesma classe ao submetê-lo ao Decodificador do *Autoencoder*, mas sem, no entanto, replicar exatamente o mesmo dado de entrada.

Considerando que o ponto no espaço latente de dimensão d pode ser representado por $P = [p_1, p_2, ...p_d]$, neste trabalho utilizaremos a seguinte equação para obter o novo ponto $P' = [p'_1, p'_2, ...p'_d]$ no espaço reduzido:

$$p'_i = p_i \pm \frac{\delta}{100} \times p_i \tag{3.1}$$

Nessa equação, δ representa um valor aleatório máximo da porcentagem de variação aceita em cada eixo. Por exemplo, para o ponto fictício $P = [0, 1, 2, 3]$ e $\delta = 15$ temos cada dimensão variando de até 15% e assim $p'_1 \in [0, 0]$, $p'_2 \in [0.85, 1.15]$, $p'_3 \in [1.70, 2.30]$ e $p'_4 \in [2.55, 3.45]$. Com esse método de perturbação do espaço latente, espera-se manter uma certa medida de aleatoriedade no processo de geração de novos dados e também priorizar as dimensões mais significativas (com valores maiores) ao se adotar como limite de variação um valor percentual.

Consolidação dos dados: após a coleta sistemática dos dados supracitados para diferentes modelos de *Autoencoders*, procederemos com a sumarização e comparação dos resultados obtidos de forma que as características avaliadas dos modelos e seus pontos fortes e fracos possam ser comparados entre si.

3.2 Bases de Dados

Esta seção descreve algumas bases de dados a serem utilizadas nos experimentos.

- *Modified National Institute of Standards and Technology* (MNIST) [10]: é uma base de dados amplamente utilizada no campo de aprendizado de máquina composta por 70.000 imagens em escala de cinza de dígitos manuscritos, cada uma com dimensões de 28 × 28 *pixels*. As imagens são divididas em conjuntos de treinamento (60.000 amostras) e teste (10.000 amostras) e abrangem dígitos de 0 a 9. Esta base é frequentemente utilizada para avaliar a eficácia de algoritmos de classificação e redes neurais em tarefas de visão computacional, oferecendo uma base sólida para desenvolvimento e comparação de modelos de aprendizado de máquina.

- *Fashion*-MNIST [37]: é uma base de dados projetada como uma alternativa ao conjunto de dados MNIST, específica para tarefas de classificação de imagens de moda. Ela consiste em 70.000 imagens em escala de cinza de artigos de vestuário, divididas em 10 classes distintas, incluindo camisetas, vestidos, tênis e outros itens relacionados à moda. Cada imagem possui dimensões de 28 × 28 *pixels*, semelhante ao MNIST. Esta base também é amplamente utilizada na avaliação e comparação de algoritmos de aprendizado de máquina, servindo como um conjunto de dados padrão para testar a capacidade de modelos em reconhecer padrões em imagens de vestuário. A diversidade das classes e a estrutura semelhante a do MNIST tornam a *Fashion*-MNIST uma escolha popular para pesquisas e desenvolvimento de modelos de visão computacional.

- *Extended* MNIST (EMNIST) [9]: é uma base de dados também amplamente utilizada no campo de aprendizado de máquina composta por mais 800.000 imagens em escala de cinza de letras manuscritas, cada uma com dimensões de 28 × 28 *pixels*. Essa base foi desenvolvida para ter um formato compatível com o conjunto de dados MNIST.

- *Painter by Numbers*: é uma base de dados apresentada na competição de mesmo nome hospedada na plataforma *Kaggle* [1] na qual são disponibilizadas cerca de 80.000 imagens de obras de arte de diferentes tamanhos e artistas. Estas imagens devem ser utilizadas para treinamento de um modelo que, ao final do processo, seja capaz de receber duas imagens de obras de arte e dizer se são ou não de um mesmo artista.

3.3 Métricas de Avaliação

As métricas utilizadas neste trabalho podem ser divididas em dois subgrupos: (a) um destinado a medir a qualidade do *Autoencoder* em replicar os dados de entrada em sua saída e (b) outro que avalia a qualidade da redução de dimensionalidade no espaço latente da rede.

3.3.1 Replicação dos Dados de Entrada

Para avaliação do desempenho do *Autoencoder* em replicar os dados de entrada, costumam ser utilizadas [4, 26, 40] as métricas de erro quadrático médio (*Mean Squared Error* - MSE) ou a raiz quadrada do erro quadrático médio (*Root Mean Squared Error* - RMSE). Neste trabalho, utilizamos o RMSE definido pela Equação 3.2 na qual x_i e $\hat{x}_i$ representam, respectivamente,

[1] https://www.kaggle.com/competitions/painter-by-numbers

os valores de entrada e saída dos dados processados pelo *Autoencoder* para todos os d valores que compõem os dados de entrada.

$$\text{RMSE} = \sqrt{\frac{1}{d}\sum_{i=1}^{d}(\hat{x}_i - x_i)^2} \tag{3.2}$$

3.3.2 Qualidade da Redução de Dimensionalidade

Antes de definir o conjunto de métricas utilizadas para avaliar o desempenho da redução de dimensionalidade, precisamos entender que cada dado a ser processado pelo *Autoencoder* é composto por d características que representam um ponto no espaço d-dimensional. Neste espaço, a partir da escolha de uma medida de distância (por exemplo, euclidiana), é possível determinar para cada ponto um conjunto de outros k pontos que estejam mais próximos dele no espaço d-dimensional, que são chamados de k-vizinhos mais próximos. Para exemplificar, considere os $n = 4$ pontos fictícios na dimensão $d = 4$: $P_1 = [0, 1, 2, 3]$, $P_2 = [5, 3, 2, 1]$, $P_3 = [3, 2, 5, 6]$ e $P_4 = [7, 3, 4, 2]$ cujas distâncias euclidianas entre si estão calculadas na Tabela 3.1a.

-	P_1	P_2	P_3	P_4
P_1	0.00	5.74	5.29	7.62
P_2	5.74	0.00	6.24	3.00
P_3	5.29	6.24	0.00	5.83
P_4	7.62	3.00	5.83	0.00

(a) Distância euclidiana entre os pontos.

-	1º	2º	3º
P_1	P_3	P_2	P_4
P_2	P_4	P_1	P_3
P_3	P_1	P_4	P_2
P_4	P_2	P_3	P_1

(b) Vizinhos mais próximos para cada ponto.

Tabela 3.1: Distâncias e vizinhos mais próximos para os pontos $P_1 = [0, 1, 2, 3]$, $P_2 = [5, 3, 2, 1]$, $P_3 = [3, 2, 5, 6]$ e $P_4 = [7, 3, 4, 2]$.

Reordenando cada linha da Tabela 3.1a considerando as distâncias, podemos construir a Tabela 3.1b ordenada pelos vizinhos mais próximos para cada ponto. Podemos observar, por exemplo, que os dois vizinhos mais próximos do ponto P_1 são os pontos P_3 e P_2.

Para avaliar o desempenho da redução de dimensionalidade, neste trabalho usamos o conjunto de métricas propostas por Espadoto *et al.* [13] que, de forma geral, baseiam-se em comparações entre os k-vizinhos mais próximos no espaço original e no espaço de dimensão reduzida, ou latente. Para o ponto P_2, por exemplo, seus dois ($k = 2$) vizinhos mais próximos são P_4 e P_1 (em ordem) conforme é possível observar na Tabela 3.1b O conjunto de métricas *trustworthiness* (confiabilidade), *continuity* (continuidade), *normalized stress* (estresse normalizado), *neighborhood hit* (vizinhança atingida) e *shepard goodness* (bondade de shepard) serão detalhadas na sequência e podem ser observadas, de forma resumida, na Tabela 3.3.

Trustworthiness ($\mathbf{M_t}$): variando entre $[0, 1]$, sendo 1 o valor da melhor eficácia, esta métrica indica a proporção de k-vizinhos mais próximos no espaço original que também estão próximos nos espaço reduzido e representa a confiança de que padrões definidos no espaço original estão mantidos no espaço reduzido. Esta métrica está definida na Tabela 3.3

-	P_1	P_2	P_3	P_4
P_1	0.00	5.72	1.86	7.17
P_2	5.72	0.00	3.86	1.45
P_3	1.86	3.86	0.00	5.31
P_4	7.17	1.45	5.31	0.00

(a) Distância euclidiana entre os pontos.

-	1^{o}	2^{o}	3^{o}
P_1	P_3	P_2	P_4
P_2	P_4	$\mathbf{P_3}$	$\mathbf{P_1}$
P_3	P_1	$\mathbf{P_2}$	$\mathbf{P_4}$
P_4	P_2	P_3	P_1

(b) Vizinhos mais próximos para cada ponto.

Tabela 3.2: Distâncias e vizinhos mais próximos para os pontos P_1, P_2, P_3 e P_4 após sofrerem redução de dimensionalidade. Em destaque estão os pontos que tiveram sua ordem alterada em relação à Tabela 3.1b.

na qual $U_i^{(k)}$ representa o conjunto de pontos entre os k-vizinhos mais próximos no espaço original para o ponto i que não estão entre os k-vizinhos mais próximos no espaço reduzido e $r(i,j)$ é a posição do ponto j (pertencente a $U_i^{(k)}$) em relação ao ponto i no espaço original. O cálculo é realizado considerando os n pontos ($N = [P_1, P_2, ..., P_n]$) que compõem o conjunto de dados. Para exemplificar, considere que os pontos descritos nas Tabelas 3.1a e 3.1b, após passarem por um processo de redução de dimensionalidade pela aplicação da técnica *Principal Component Analysis* (PCA) para $d = 1$, passam a ter valores $P_1 = 3.69$, $P_2 = -2.03$, $P_3 = 1.83$ e $P_4 = -3.48$. Sendo assim, as distâncias e a relação de vizinhança entre esses pontos no espaço 1-dimensional estão indicadas, respectivamente, na Tabela 3.2a e na Tabela 3.2b.

Para o ponto $i = P_2$, por exemplo, ao considerar $k = 2$ temos que $U_1^{(2)}$ é composto pelo ponto P_3, pois dentre seus dois vizinhos mais próximos (P_4 e P_3), apenas P_3 não aparece entre os dois vizinhos mais próximos de P_2 no espaço original. Para cada ponto j no conjunto (neste caso apenas o P_3), verifica-se a posição de vizinhança dele no espaço original que define o valor $r(P_2, P_3)$. Na Tabela 3.1b observa-se que P_3 é o terceiro vizinho de P_2 e, portanto, têm-se que $r(P_2, P_3) = 3$.

Repetindo o procedimento com todos os $n = 4$ pontos, obtém-se, para este exemplo, um valor $M_t = 1 - \frac{2}{4\times2(2\times4-3\times2-1)}(0 + 1 + 1 + 0) = 0.5$.

Continuity ($\mathbf{M_c}$): variando entre $[0, 1]$, sendo 1 o valor da melhor eficácia, esta métrica é semelhante à *trustworthiness* nos cálculos mas atua de forma invertida: ela indica a proporção de k-vizinhos mais próximos no espaço reduzido que também estão próximos no espaço original estando assim diretamente relacionada com a avaliação dos vizinhos faltantes na projeção.

Ela está definida na Tabela 3.3 na qual $V_i^{(k)}$ representa o conjunto de pontos entre os k-vizinhos mais próximos no espaço reduzido para ponto i que não estão entre os k-vizinhos mais próximos no espaço original e $\hat{r}(i,j)$ é a posição do ponto j (pertencente a $V_i^{(k)}$) em relação ao ponto i no espaço reduzido. O cálculo é realizado considerando os n pontos ($N = [P_1, P_2, ..., P_n]$) que compõem o conjunto de dados.

Para o ponto $i = P_2$, por exemplo, ao considerar $k = 2$ e observando as Tabelas 3.1b e 3.2b, temos que $V_1^{(2)}$ é composto pelo ponto P_1, pois dentre seus dois vizinhos mais próximos (P_4 e P_1), apenas P_1 não aparece entre os dois vizinhos mais próximos de P_2 no espaço

reduzido. Para cada ponto j no conjunto (neste caso apenas o P_1), verifica-se a posição de vizinhança dele no espaço reduzido que define o valor $\hat{r}(P_2, P_1)$. Na Tabela 3.2b, observa-se que P_1 é o terceiro vizinho de P_2 e, portanto, tem-se que $\hat{r}(P_2, P_1) = 3$.

Repetindo o procedimento com todos os $n = 4$ pontos, obtém-se, para este exemplo, um valor $M_c = 1 - \frac{2}{4\times2(2\times4-3\times2-1)}(0 + 1 + 1 + 0) = 0.5$.

Normalized stress ($\mathbf{M_{ns}}$): variando entre $[0, 1]$, sendo 0 o valor da melhor eficácia, esta métrica mede a preservação das distâncias "par a par" dos pontos entre os espaços original e reduzido. De forma geral, boas projeções têm baixos níveis de variação (ou estresse). A métrica está definida na Tabela 3.3, na qual $\Delta o_{i,j}$ e $\Delta q_{i,j}$ representam as distâncias entre os pontos i e j ($i, j \in N = [P_1, P_2, ..., P_n]$) no espaço original e reduzido, respectivamente. O denominador da fórmula foi modificado em relação ao apresentado no artigo original [13], trocando $(\sum_{i,j} \Delta o_{i,j}^2)$ por $\max(\sum_{i,j} \Delta o_{i,j}^2, \sum_{i,j} \Delta q_{i,j}^2)$ para garantir que os limites propostos para a métrica sejam mantidos nos casos em que as distâncias entre os pontos no espaço reduzido sejam maiores que as do espaço original. Utilizando os pontos que estamos adotando como exemplo, temos $\sum_{i,j} \Delta o_{i,j}^2 = 0.00^2 + 5.74^2 + 5.29^2 + ... + 3.00^2 + 5.83^2 + 0.00^2 = 201.00$ com base nas distâncias indicadas na Tabela 3.1a e $\sum_{i,j} \Delta q_{i,j}^2 = 0.00^2 + 5.72^2 + 1.86^2 + ... + 1.45^2 + 5.31^2 + 0.00^2 = 132.67$ com base nas distâncias indicadas na Tabela 3.2a e, seguindo o mesmo raciocínio temos $\sum_{i,j}(\Delta o_{i,j} - \Delta q_{i,j})^2 = (0.00 - 0.00)^2 + (5.74 - 5.72)^2 + (5.29 - 1.86)^2 + ... + (3.00 - 1.45)^2 + (5.83 - 5.31)^2 + (0.00 - 0.00)^2 = 20.35$. Consequentemente, temos $M_{ns} = \frac{20.35}{\max(201.00,132.67)} = 0.10$.

Neighborhood hit ($\mathbf{M_{nh}}$): variando entre $[0, 1]$, sendo 1 o valor da melhor eficácia, esta métrica quantifica a proporção de k-vizinhos mais próximos de um ponto que possuam a mesma classe (*label*) do ponto analisado. Idealmente, espera-se que pontos próximos no espaço d-dimensional de características sejam semelhantes e estejam rotulados na mesma classe. A métrica está definida na Tabela 3.3 na qual $K_i^{(k)}$ representa a quantidade de k-vizinhos mais próximos do ponto i que possuem a mesma classe dele. Esta métrica não depende de uma comparação entre espaço original e reduzido e pode ser aplicada separadamente em cada um deles. Neste trabalho, ela é empregada para análise do espaço reduzido, a menos que haja indicação expressa dizendo o contrário. Recorrendo novamente aos pontos do nosso exemplo, suponhamos que eles representem duas classes possíveis A e B, sendo que P_1 e P_2 estão associados à classe A e P_3 e P_4 à classe B. Considerando o espaço reduzido e $k = 2$, observamos a partir da Tabela 3.2b, que para P_1 (classe A), entre seus dois vizinhos mais próximos, P_3 (classe B) e P_2 (classe A), há apenas um ponto (P_2) que possui a mesma classe de P_1 e, portanto, neste caso, $K_1^{(2)} = 1$. Para P_2 (classe A), entre seus dois vizinhos mais próximos P_4 (classe B) e P_3 (classe B), não há ponto que possua a mesma classe de P_2 e portanto, neste caso, $K_2^{(2)} = 0$. Repetindo o processo com os demais pontos temos $M_{nh} = \frac{(1+0+0+1)}{4\times2} = 0.25$.

Shepard goodness ($\mathbf{M_s}$): variando entre $[0, 1]$, sendo 1 o valor da melhor eficácia, esta métrica quantifica correlação de classificação de Spearman (*Spearman rank correlation*) em um gráfico de dispersão entre todos os pontos ($\Delta o_{i,j}$, $\Delta q_{i,j}$) que representam as distâncias entre os pontos i e j ($i, j \in N = [P_1, P_2, ..., P_n]$) no espaço original e reduzido, res-

Métrica	Definição	Intervalo
Trustworthiness (M_t)	$1 - \frac{2}{nk(2n-3k-1)} \sum_{i \in N} \sum_{j \in U_i^{(k)}} (r(i,j) - k)$	$[0, \mathbf{1}]$
Continuity (M_c)	$1 - \frac{2}{nk(2n-3k-1)} \sum_{i \in N} \sum_{j \in V_i^{(k)}} (\hat{r}(i,j) - k)$	$[0, \mathbf{1}]$
Normalized stress (M_{ns})	$\frac{\sum_{i,j} (\Delta o_{i,j} - \Delta q_{i,j})^2}{\max(\sum_{i,j} \Delta o_{i,j}^2, \sum_{i,j} \Delta q_{i,j}^2)}$	$[\mathbf{0}, 1]$
Neighborhood hit (M_{nh})	$\sum_{i \in N} \frac{K_i^{(k)}}{nk}$	$[0, \mathbf{1}]$
Shepard goodness (M_s)	$\frac{\text{Spearman coef.}(\Delta o_{i,j}, \Delta q_{i,j}) + 1}{2}$	$[0, \mathbf{1}]$

Tabela 3.3: Métricas utilizadas no trabalho. Na coluna "Intervalo", está em destaque o valor de melhor eficácia.

pectivamente. Um valor de $M_s = 1$ indica uma correlação perfeita das distâncias. A fórmula foi modificada em relação à apresentada no artigo original [13], somando o valor 1 e dividindo por 2, para garantir que os limites propostos para a métrica sejam mantidos nos casos em que haja uma correlação negativa. Para os pontos do nosso exemplo, ao aplicar a correlação de classificação de Spearman nos dois conjuntos distâncias $\Delta o_{i,j} = [0.00, 5.74, 5.29, ..., 3.00, 5.83, 0.00]$ e $\Delta q_{i,j} = [0.00, 5.72, 1.86, ..., 1.45, 5.31, 0.00]$ obtemos o valor 0.77 e, portanto, $M_s = \frac{0.77+1}{2} = 0.89$.

A princípio, neste trabalho utilizaremos um valor de vizinhos mais próximos de $k = 5$ e distância euclidiana, a menos que haja indicação expressa dizendo o contrário. Estabelece-se ainda uma última métrica que considera todas as anteriores, definida na Equação 3.3 que varia entre $[0, 1]$, sendo 1 o valor da melhor eficácia:

$$M = \frac{M_t + M_c + (1 - M_{ns}) + M_{nh} + M_s}{5} \tag{3.3}$$

Para os pontos do nosso exemplo, consolidando todas as métricas apresentadas, temos $M = \frac{0.5+0.5+(1-0.10)+0.25+0.89}{5} = 0.61$.

3.4 Recursos Computacionais

Python [2] é a linguagem de programação utilizada para implementação dos modelos desenvolvidos neste projeto, uma linguagem amplamente consolidada, que possui um vasto conjunto de bibliotecas voltadas para aprendizado de máquina, execução eficiente de operações científicas e numéricas, além de geração de gráficos. Dentre as bibliotecas mais utilizadas neste projeto, podemos citar: NumPy [3], Pandas [4], scikit-learn [5], TensorFlow [6], Keras [7], PyTorch [8]

[2] https://www.python.org
[3] https://www.numpy.org
[4] https://pandas.pydata.org
[5] https://scikit-learn.org
[6] https://www.tensorflow.org
[7] https://keras.io
[8] https://pytorch.org

e Matplotlib [9].

Os experimentos serão realizados em três ambientes principais:

- Plataforma Google Colab [10], que disponibiliza recursos de computação em nuvem gratuitos como CPU, GPU e armazenamento de dados. Diante da atual parceria entre Unicamp e Google para armazenamento de dados, o volume de informação processada nos experimentos será armazenado, majoritariamente, na conta institucional da Unicamp vinculada ao Google;

- Recursos computacionais do Instituto de Computação (IC) da Unicamp, que disponibiliza um ambiente virtual hospedado em nuvem com recursos como CPU, GPU e armazenamento de dados aos alunos do Programa de Pós-Graduação.

- Supercomputador Santos Dumont [11], através do projeto "Aprendizado de Máquina Profundo para Problemas de Visão Computacional e Biologia Computacional" coordenado pelo Prof. Zanoni Dias, que oferece ambiente virtual com uso de GPUs com alto poder computacional para processamento de dados.

[9] https://matplotlib.org
[10] https://colab.research.google.com
[11] https://sdumont.lncc.br

Capítulo 4

Plano de Trabalho e Cronograma de Execução

O plano de trabalho é composto pelas seguintes atividades:

1. Pesquisa bibliográfica.
2. Aproveitamento de disciplinas do Programa de Pós-Graduação cursadas previamente na condição de estudante especial.
3. Cumprimento da proficiência de inglês exigida pelo Programa de Pós-Graduação.
4. Definição dos modelos de *Autoencoders* a serem analisados e o escopo da análise restrito a certos tipos de aplicação do modelo.
5. Realização do Exame de Qualificação de Mestrado (EQM).
6. Replicação dos trabalhos a serem a comparados.
7. Análise e comparação dos resultados.
8. Documentação e publicação dos resultados.
9. Escrita do documento da dissertação.
10. Defesa da dissertação.

O cronograma de execução das atividades propostas, em um prazo de 24 meses, é apresentado na Tabela 4.1.

Atividades	**1^o ano**						**2^o ano**					
	1	**2**	**3**	**4**	**5**	**6**	**1**	**2**	**3**	**4**	**5**	**6**
Pesquisa bibliográfica	•	•	•	•	•	•		•		•		•
Aproveitamento de disciplinas	•											
Proficiência de inglês		•										
Definição dos modelos a serem analisados			•	•								
Realização do EQM				•								
Replicação dos códigos a serem comparados				•	•	•	•					
Análise e comparação dos resultados.							•	•	•			
Publicação dos resultados										•	•	
Escrita da dissertação										•	•	•
Defesa da dissertação												•

Tabela 4.1: Cronograma de atividades dividido em bimestres.

Bibliografia

[1] J. Bakarji, K. Champion, J. N. Kutz, and S. L. Brunton. Discovering governing equations from partial measurements with deep delay autoencoders. *arXiv preprint arXiv:2201.05136*, 2022.

[2] D. Bank, N. Koenigstein, and R. Giryes. *Autoencoders*, pages 353–374. Springer International Publishing, Cham, 2023.

[3] P. Bojanowski, A. Joulin, D. Lopez-Paz, and A. Szlam. Optimizing the latent space of generative networks. *arXiv preprint arXiv:1707.05776*, 2017.

[4] N. Bonettini, C. A. Gonano, P. Bestagini, M. Marcon, B. Garavelli, and S. Tubaro. Comparing autoencoder variants for real-time denoising of hyperspectral X-ray. *IEEE Sensors Journal*, 22(18):17997–18007, 2022.

[5] K. Champion, B. Lusch, J. N. Kutz, and S. L. Brunton. Data-driven discovery of coordinates and governing equations. *National Academy of Sciences*, 116(45):22445–22451, 2019.

[6] Z. Cheng, H. Sun, M. Takeuchi, and J. Katto. Deep convolutional autoencoder-based lossy image compression. In *Picture Coding Symposium (PCS)*, pages 253–257, 2018.

[7] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[8] H. Choi, M. Kim, G. Lee, and W. Kim. Unsupervised learning approach for network intrusion detection system using autoencoders. *The Journal of Supercomputing*, 75(9):5597–5621, 2019.

[9] G. Cohen, S. Afshar, J. Tapson, and A. Van Schaik. EMNIST: Extending MNIST to handwritten letters. In *International Joint Conference on Neural Networks (IJCNN)*, pages 2921–2926. IEEE, 2017.

[10] L. Deng. The MNIST database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[11] B. M. Dillon, T. Plehn, C. Sauer, and P. Sorrenson. Better latent spaces for better autoencoders. *SciPost Physics*, 11(3):061, 2021.

[12] W. El-Shafai, S. A. El-Nabi, E.-S. M. El-Rabaie, A. M. Ali, N. F. Soliman, A. D. Algarni, A. El-Samie, and E. Fathi. Efficient deep-learning-based autoencoder denoising approach for medical image diagnosis. *Computers, Materials & Continua*, 70(3), 2022.

[13] M. Espadoto, R. M. Martins, A. Kerren, N. S. Hirata, and A. C. Telea. Toward a quantitative survey of dimension reduction techniques. *IEEE Transactions on Visualization and Computer Graphics*, 27(3):2153–2173, 2019.

[14] L. Gao and J. N. Kutz. Bayesian autoencoders for data-driven discovery of coordinates, governing equations and fundamental constants. *arXiv preprint arXiv:2211.10575*, 2022.

[15] K. He, X. Chen, S. Xie, Y. Li, P. Dollár, and R. Girshick. Masked autoencoders are scalable vision learners. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 16000–16009, 2022.

[16] K. Jun, D.-W. Lee, K. Lee, S. Lee, and M. S. Kim. Feature extraction using an RNN autoencoder for skeleton-based abnormal gait recognition. *IEEE Access*, 8:19196–19207, 2020.

[17] D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[18] D. Li, H. Guo, Z. Wang, and Z. Zheng. Unsupervised fake news detection based on autoencoder. *IEEE Access*, 9:29356–29365, 2021.

[19] P. Li, Y. Pei, and J. Li. A comprehensive survey on design and application of autoencoder in deep learning. *Applied Soft Computing*, 138:110176, 2023.

[20] Y. Li, C.-Y. Wu, H. Fan, K. Mangalam, B. Xiong, J. Malik, and C. Feichtenhofer. MViTv2: Improved multiscale vision transformers for classification and detection. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4804–4814, 2022.

[21] E. Lin, S. Mukherjee, and S. Kannan. A deep adversarial variational autoencoder model for dimensionality reduction in single-cell RNA sequencing analysis. *BMC Bioinformatics*, 21(1):64, Feb 2020.

[22] C.-Y. Liou, W.-C. Cheng, J.-W. Liou, and D.-R. Liou. Autoencoder for words. *Neurocomputing*, 139:84–96, 2014.

[23] T. Liu, J. Wang, Q. Liu, S. Alibhai, T. Lu, and X. He. High-ratio lossy compression: exploring the autoencoder to compress scientific data. *IEEE Transactions on Big Data*, 9(1):22–36, 2023.

[24] A. Makhzani, J. Shlens, N. Jaitly, I. Goodfellow, and B. Frey. Adversarial autoencoders. *arXiv preprint arXiv:1511.05644*, 2015.

[25] L. McInnes, J. Healy, and J. Melville. UMAP: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*, 2018.

[26] Q. Meng, D. Catchpoole, D. Skillicom, and P. J. Kennedy. Relational autoencoder for feature extraction. In *International Joint Conference on Neural Networks (IJCNN)*, pages 364–371. IEEE, 2017.

[27] A. Ng. Sparse autoencoder. *CS294A Lecture Notes*, 72(2011):1–19, 2011.

[28] T. Park, J.-Y. Zhu, O. Wang, J. Lu, E. Shechtman, A. Efros, and R. Zhang. Swapping autoencoder for deep image manipulation. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pages 7198–7211. Curran Associates, Inc., 2020.

[29] G. Parmar, D. Li, K. Lee, and Z. Tu. Dual contradistinctive generative autoencoder. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 823–832, June 2021.

[30] S. Rifai, P. Vincent, X. Muller, X. Glorot, and Y. Bengio. Contractive auto-encoders: explicit invariance during feature extraction. In *International Conference on Machine Learning (ICNML)*, pages 833–840, 2011.

[31] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, Oct 1986.

[32] M. Said Elsayed, N.-A. Le-Khac, S. Dev, and A. D. Jurcut. Network anomaly detection using lstm based autoencoder. In *ACM Symposium on QoS and Security for Wireless and Mobile Networks (Q2SWinet)*, pages 37–45, 2020.

[33] C. C. Tan and C. Eswaran. Performance comparison of three types of autoencoder neural networks. In *2nd Asia International Conference on Modelling & Simulation (AMS)*, pages 213–218, 2008.

[34] M. Tripathi. Facial image denoising using autoencoder and UNet. *Heritage and Sustainable Development*, 3(2):89–96, 2021.

[35] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol. Extracting and composing robust features with denoising autoencoders. In *International Conference on Machine Learning (ICML)*, page 1096–1103, New York, NY, USA, 2008. Association for Computing Machinery.

[36] S. Woo, S. Debnath, R. Hu, X. Chen, Z. Liu, I. S. Kweon, and S. Xie. ConvNeXt V2: Co-designing and scaling convnets with masked autoencoders. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 16133–16142, 2023.

[37] H. Xiao, K. Rasul, and R. Vollgraf. Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

[38] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *European Conference on Computer Vision (ECCV)*, pages 818–833, Cham, 2014. Springer International Publishing.

[39] J. Zhai, S. Zhang, J. Chen, and Q. He. Autoencoder and its various variants. In *IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 415–419. IEEE, 2018.

[40] G. Zhang, Y. Liu, and X. Jin. A survey of autoencoder-based recommender systems. *Frontiers of Computer Science*, 14:430–450, 2020.

Apêndice A

Resultados Preliminares

Neste apêndice, estão descritos os experimentos iniciais desenvolvidos até o momento e os resultados obtidos. Eles têm como finalidade demonstrar a viabilidade de implementação da metodologia descrita na Seção 3.1.

A.1 Experimentos Iniciais com *Autoencoders*

Esta seção descreve os resultados dos experimentos com dois modelos: o *Fully Connected Autoencoder* (FCAE) e o *Variational Autoencoder* (VAE).

A.1.1 *Fully Connected Autoencoder* (FCAE)

Como primeiro objeto de estudo foi escolhido um dos modelos mais simples de *Autoencoder*, o FCAE, para análise e experimentos iniciais. A base de dados utilizada para avaliação do modelo foi a MNIST e por possuir dados no tamanho de 28×28 *pixels* (784 *pixels* no total), as camadas de entrada do Codificador e de saída do Decodificador foram fixadas em 784 neurônios.

Para definir a arquitetura (quantidade de camadas) do modelo, foram realizados testes com as seguintes configurações de Codificador, sendo que o Decodificador testado em cada caso foi simétrico:

- **1 camada:** $784 \rightarrow 392 \rightarrow$ Camada Latente
- **2 camadas:** $784 \rightarrow 523 \rightarrow 260 \rightarrow$ Camada Latente
- **3 camadas:** $784 \rightarrow 588 \rightarrow 392 \rightarrow 196 \rightarrow$ Camada Latente
- **4 camadas:** $784 \rightarrow 627 \rightarrow 470 \rightarrow 313 \rightarrow 156 \rightarrow$ Camada Latente
- **5 camadas:** $784 \rightarrow 654 \rightarrow 524 \rightarrow 394 \rightarrow 264 \rightarrow 134 \rightarrow$ Camada Latente
- **6 camadas:** $784 \rightarrow 672 \rightarrow 560 \rightarrow 448 \rightarrow 336 \rightarrow 224 \rightarrow 112 \rightarrow$ Camada Latente

Para cada cenário o modelo foi treinado por 50 épocas com tamanhos da Camada Latente (z) com valores 10, 20, 40, 80, 160 e 320. Com os modelos treinados, o conjunto de teste foi

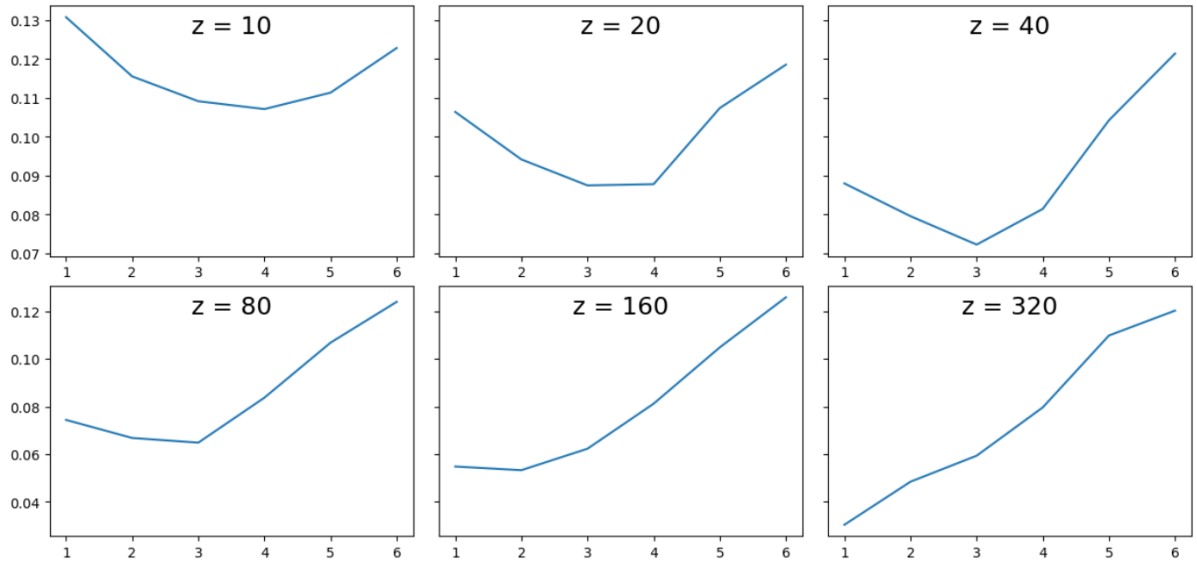

Figura A.1: Variação do RMSE (eixo y) no conjunto de testes de acordo com as camadas (eixo x) do modelo para diferentes valores z da Camada Latente.

avaliado e a raiz quadrada do erro quadrático médio (*Root Mean Squared Error* - RMSE) foi usada para medir a eficácia do *Autoencoder*. O resultado pode ser observado na Figura A.1.

Quanto maior a quantidade de neurônios na Camada Latente (z), menos camadas na rede são necessárias para obter o melhor desempenho do modelo. Considerando que os experimentos posteriores seriam realizados com diferentes valores de z, foi adotada a configuração de 3 camadas que possui boa eficácia para diferentes valores de Camada Latente.

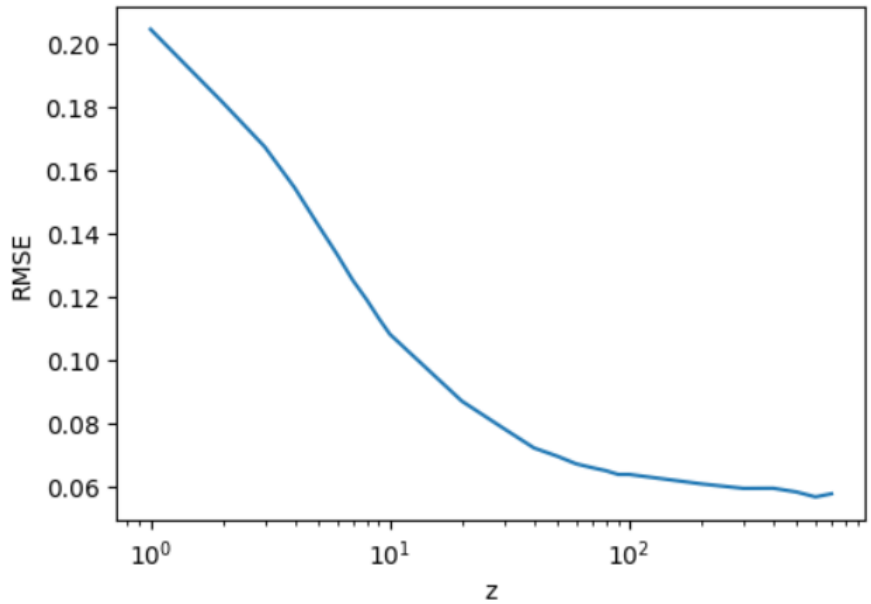

Figura A.2: Variação do RMSE (eixo y) no conjunto de testes de acordo com a variação da quantidade de neurônios z da Camada Latente (eixo x em escala logarítmica) para o modelo FCAE.

Definida a arquitetura, o modelo foi treinado por 50 épocas para z assumindo os seguintes valores de quantidade de neurônios: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 200, 300, 400, 500, 600 e 700.

A evolução do RMSE para diferentes valores de z pode ser observada na Figura A.2, na qual é possível notar que a performance do modelo em replicar na saída os dados de entrada melhora com o aumento de neurônios na Camada Latente. Tal comportamento está de acordo com o esperado uma vez que o modelo será capaz de guardar mais informações para replicar os dados a medida que houver aumento da Camada Latente – no limite superior utilizado no experimento ela chega a ficar próxima do tamanho original da imagem.

Apesar da redução do erro na reprodução das imagens com o aumento da Camada Latente, é possível observar na Figura A.3 que poucos neurônios já são suficientes para reproduzir na saída do Decodificador imagens corretas. Portanto, embora as medições tenham sido realizadas para valores de z superiores a 40, elas não serão apresentadas nesta seção.

O espaço latente foi investigado na sequência para os diferentes valores de z através do UMAP e, como já era possível prever pela boa replicação de imagens obtidas pelo modelo, os dados foram bem separados no espaço latente. Para evitar o estouro de memória, apenas uma parte dos dados de teste foi selecionada aleatoriamente, processada pelo *Autoenconder* e o conjunto de pontos da Camada Latente foi então processado pelo UMAP.

O resultado para os valores $z = 2$ e $z = 40$ podem ser vistos na Figura A.4. Apesar dos dados estarem bem separados no espaço latente desde valores baixos de z é possível notar na figura da esquerda ($z = 2$) que existem mais sobreposições de pontos de dígitos diferentes (por exemplo, entre o dígito 4 e 9) do que na figura da direita, na qual eles estão melhor separados.

As métricas definidas na Seção 3.3 também foram calculadas e podem ser observadas na

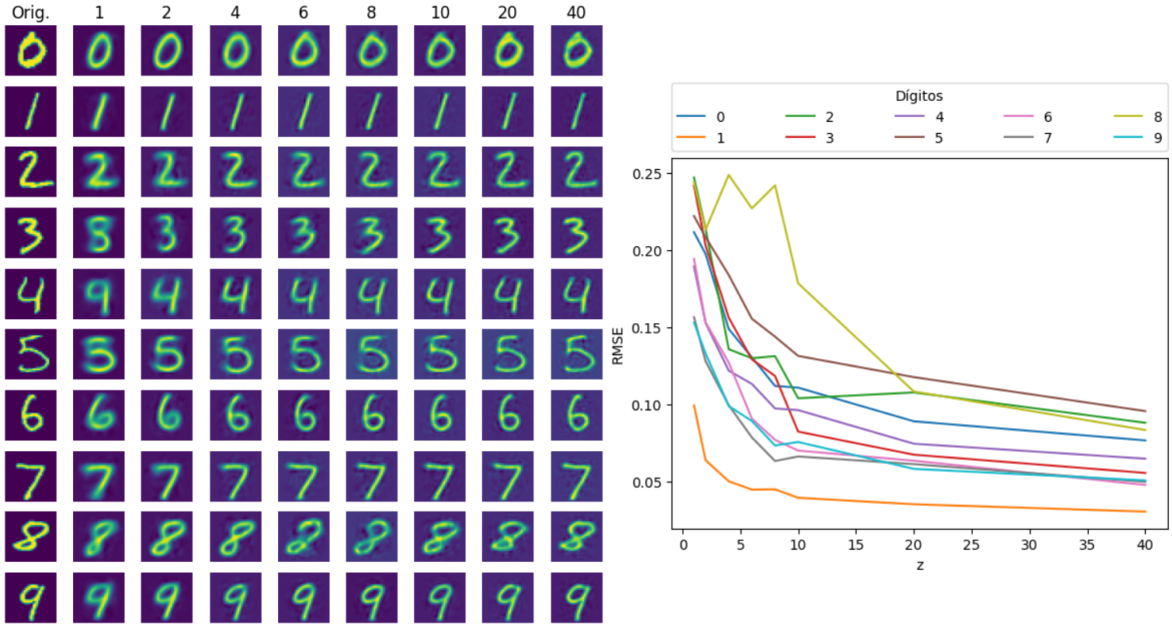

Figura A.3: FCAE: (a) à esquerda, imagens originais e processadas para diferentes valores de z; (b) à direta, evolução do RMSE em relação a z.

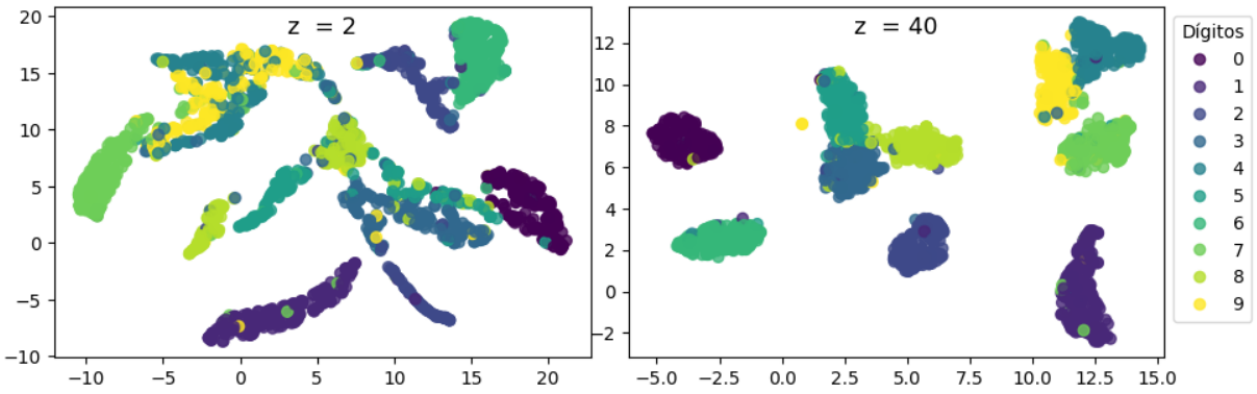

Figura A.4: FCAE: redução de dimensionalidade realizada pelo UMAP. Na figura à esquerda, é apresentado o resultado para $z = 2$ e, à direta, para $z = 40$.

Figura A.5 na qual é possível notar que, em sua maioria, todas evoluem para valores próximos de 1.0 (melhor valor) com o aumento de z. A exceção ocorre para a métrica *neighborhood hit* que aumenta para valores até $z = 10$ e passa a diminuir suavemente depois disso. Essa métrica mede a quantidade de k-vizinhos mais próximos de um ponto analisado que tenham a mesma classe e tal queda na métrica pode ser explicada pelo fato de que o aumento das dimensões que compõem o espaço latente permitirá mais possibilidades de organização dos dados neste hiperespaço, fazendo com que pontos vizinhos de classes diferentes – antes distantes – possam estar mais próximos nessa nova configuração pela distância euclidiana.

Para investigar a capacidade do FCAE em processar dados diferentes daqueles com os quais o modelo foi treinado, utilizamos as bases de dados EMNIST e *Fashion*-MNIST. Além da análise visual, o RMSE entre os dados de entrada e saída também foram calculados e os resultados podem ser observados nas figuras A.6 e A.7.

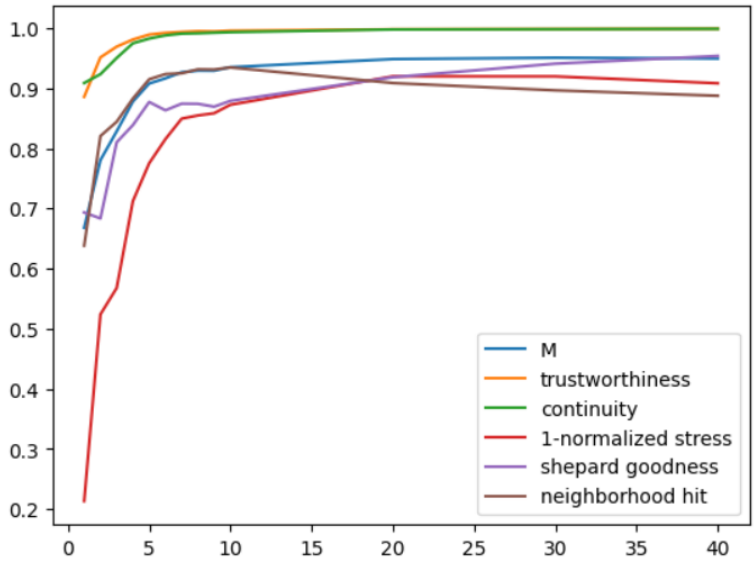

Figura A.5: FCAE: evolução das métricas definidas na Seção 3.3 de acordo com a variação de z. O valor M é definido pela Equação 3.3.

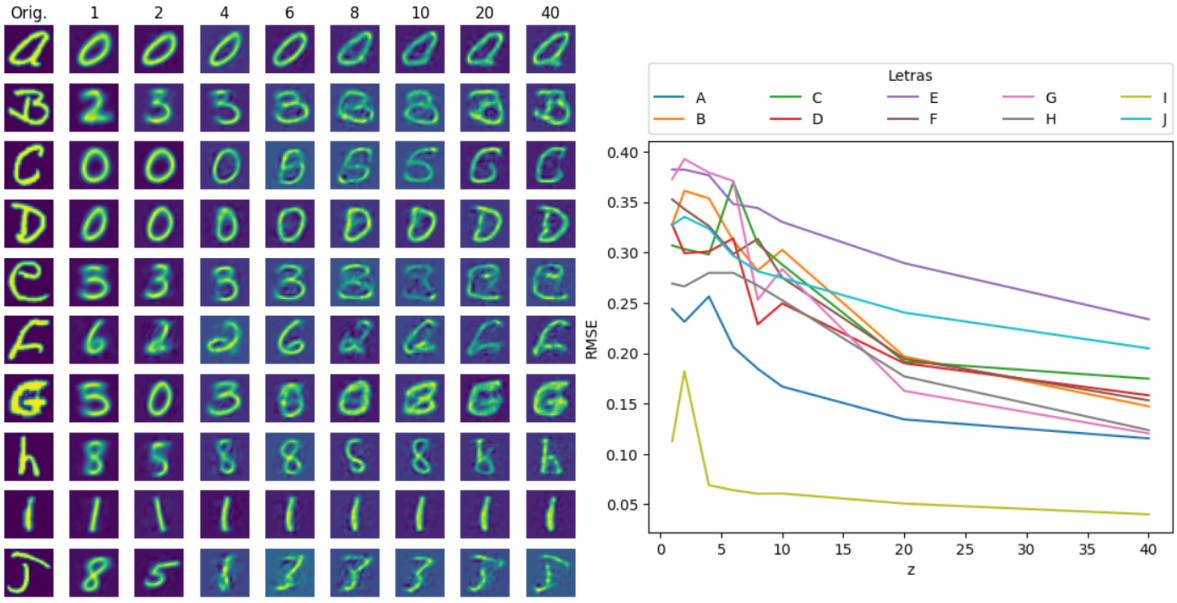

Figura A.6: FCAE: (a) à esquerda, imagens originais (EMNIST) e processadas para diferentes valores de z; (b) à direta, evolução do RMSE em relação a z.

É possível observar na Figura A.6 que os modelos treinados com valores menores de tamanho de Camada Latente tiveram dificuldade em reproduzir os dados de entrada e como resultado, na saída foram geradas imagens de dígitos. Porém, com 40 neurônios na Camada Latente o modelo já foi capaz se ajustar e reproduzir imagens de letras que se assemelhavam às da entrada. Um comportamento semelhante pode ser observado na Figura A.7. No entanto,

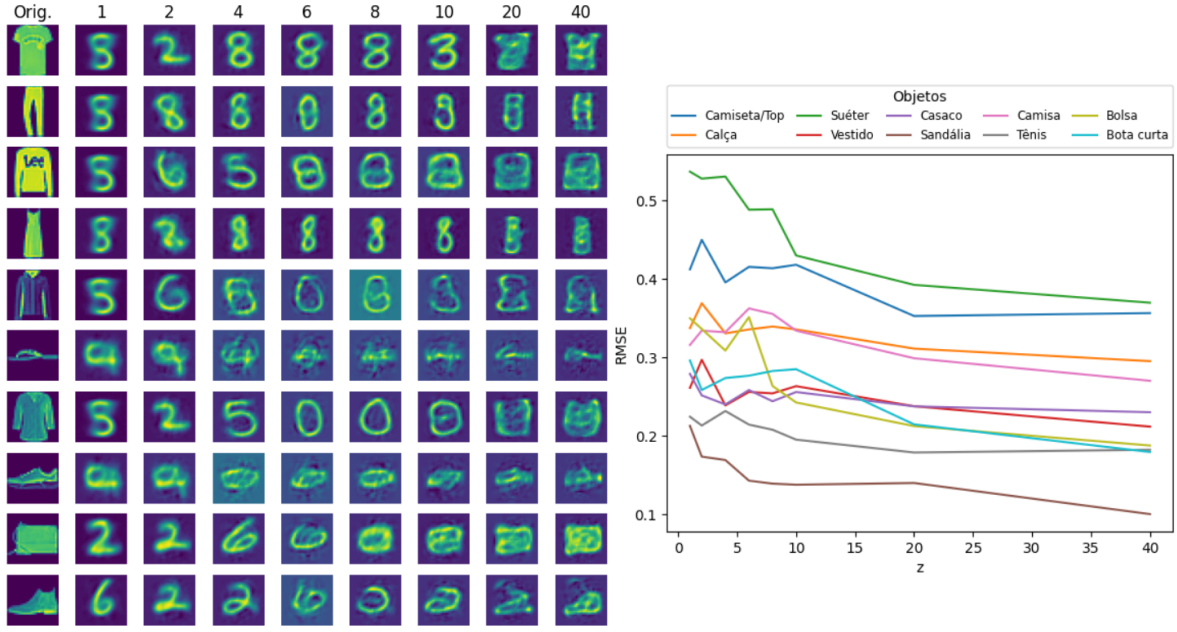

Figura A.7: FCAE: (a) à esquerda, imagens originais (*Fashion*-MNIST) e processadas para diferentes valores de z; (b) à direta, evolução do RMSE em relação a z.

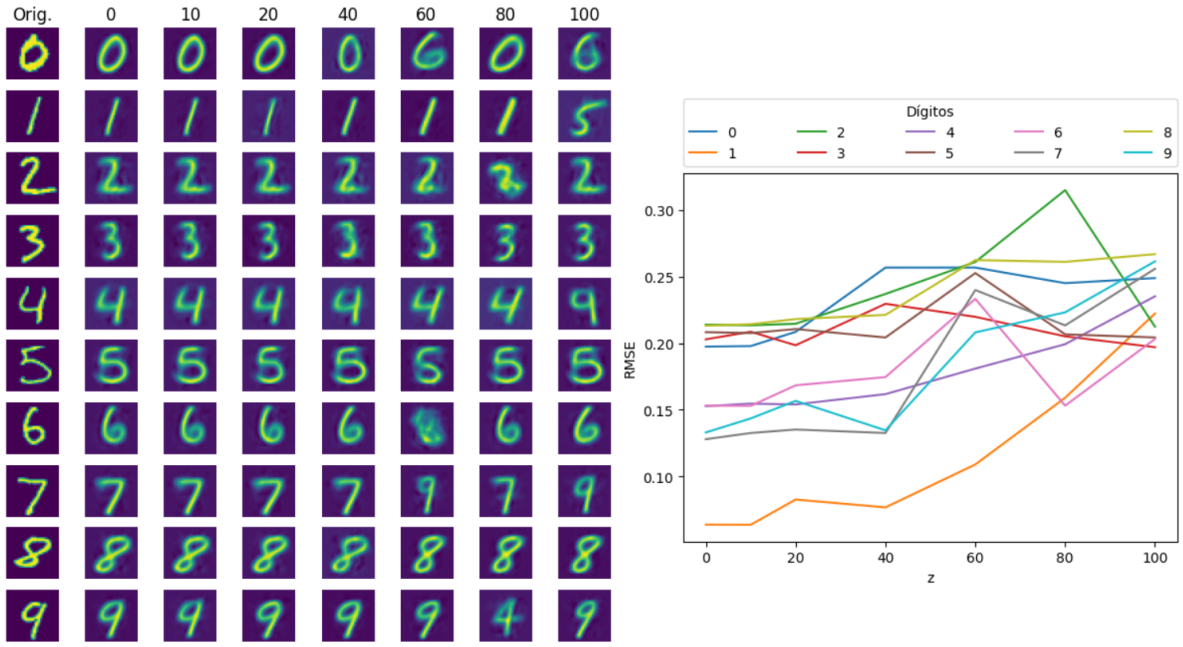

Figura A.8: FCAE: (a) à esquerda, imagens originais e processadas para $z = 2$ com perturbação máxima do espaço latente de 0 a 100%; (b) à direta, evolução do RMSE em relação à perturbação aplicada.

por ter uma base de dados mais complexa, mesmo com 40 neurônios, o modelo consegue apenas reproduzir parcialmente os contornos das imagens da entrada.

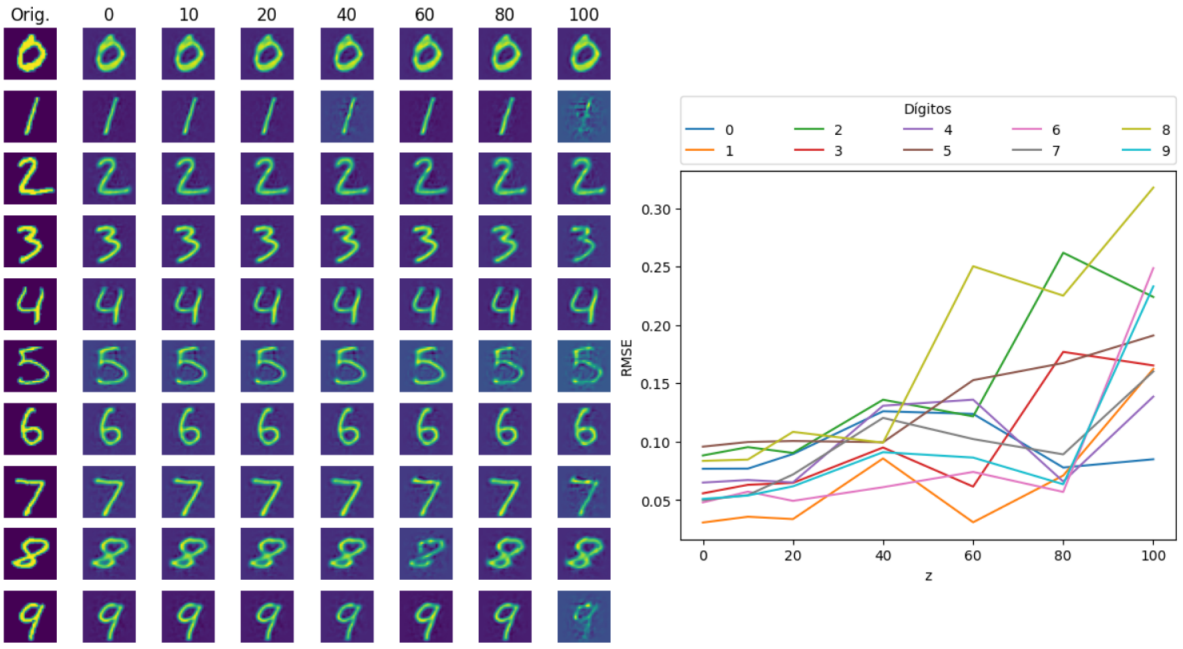

Figura A.9: FCAE: (a) à esquerda, imagens originais e processadas para $z = 40$ com perturbação máxima do espaço latente de 0 a 100%; (b) à direta, evolução do RMSE em relação à perturbação aplicada.

A seguir, estudamos as propriedades generativas do modelo FCAE passando uma imagem do conjunto de teste pelo Codificador, causando perturbação no ponto resultante na Camada Latente (conforme descrito na Seção 3.1) e obtendo um novo ponto que foi repassado pelo Decodificador. As figuras A.8 e A.9 apresentam imagens processadas pelo *Autoencoder* e a evolução do RMSE entre as imagens de entrada e saída para valores de z iguais a 2 e 40, respectivamente, compreendendo variações máximas no espaço latente de 0 a 100%.

É possível observar que, à medida que aumentamos a variação da perturbação no espaço latente, o RMSE aumenta, ou seja, a figura resultante na saída do Decodificador está mais distante da original. Para $z = 2$, o aumento da perturbação faz com que modelo passe a gerar imagens diferentes da classe original, mas para $z = 40$, mesmo os valores altos de perturbação, o modelo ainda gera imagens na mesma classe original.

A.1.2 *Variational Autoencoder* (VAE)

O VAE foi o segundo modelo estudado. Na tentativa de proporcionar uma comparação mais justa, a arquitetura do modelo seguiu a mesma configuração do FCAE estudando anteriormente (3 camadas: $784 \rightarrow 588 \rightarrow 392 \rightarrow 196 \rightarrow$ Camada Latente). Após a condução de alguns testes exploratórios, foi decidido treinar o modelo com 250 épocas e usar na função de perda do modelo a *Binary Cross Entropy*. O VAE trabalha tentando minimizar dois termos na função de perda do treinamento: um que mede a qualidade da reprodução dos dados de entrada na saída e outro que procura otimizar a distribuição do espaço latente. Apesar de não terem sido realizados experimentos com dados de classificação binária, a *Binary Cross Entropy* como recurso para medir a qualidade da reprodução dos dados funcionou melhor na função de perda juntamente com o termo que procurava otimizar o espaço latente do que

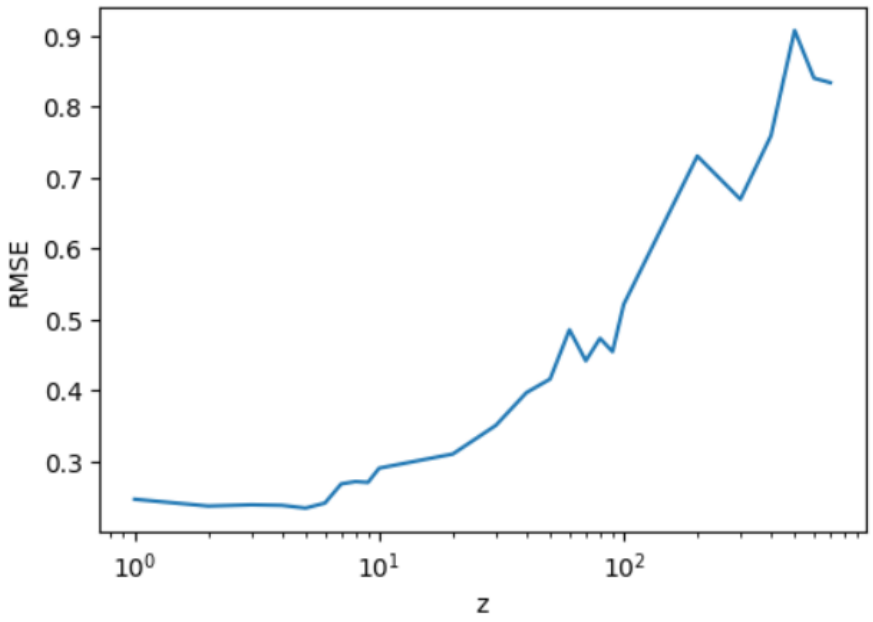

Figura A.10: Variação do RMSE (eixo y) no conjunto de testes de acordo com a variação da quantidade de neurônios z da Camada Latente (eixo x em escala logarítmica) para o modelo VAE.

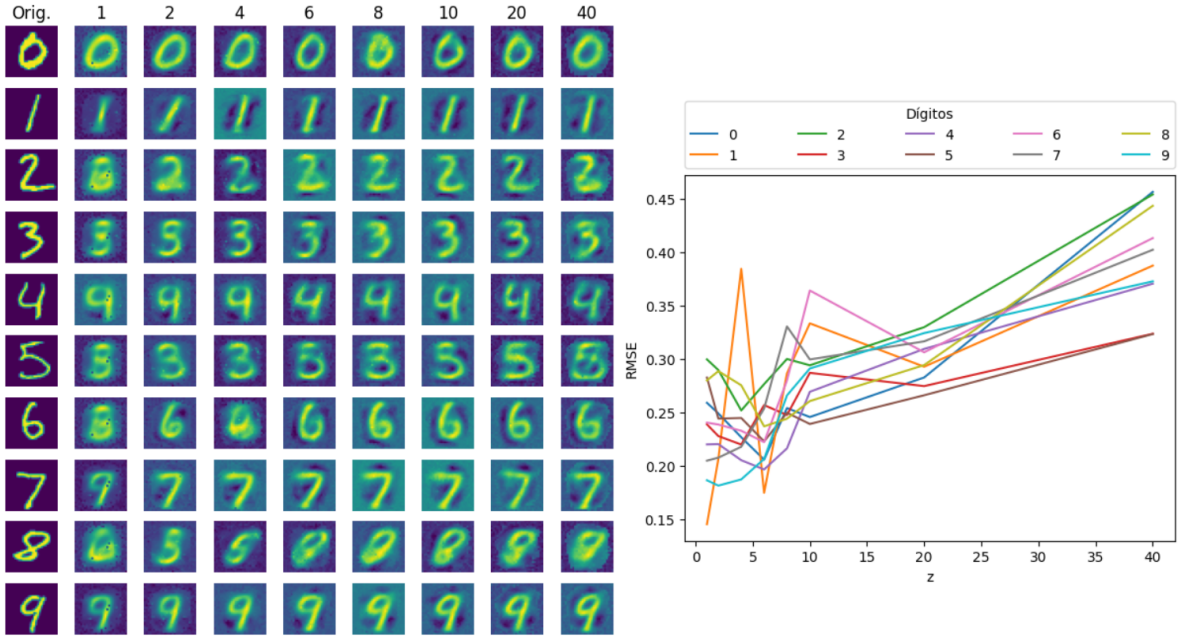

Figura A.11: VAE: (a) à esquerda, imagens originais e processadas para diferentes valores de z; (b) à direta, evolução do RMSE em relação a z.

utilizar apenas o erro quadrático médio.

A evolução do RMSE para diferentes valores de z pode ser observada na Figura A.10, na qual é possível notar que a performance do modelo em replicar na saída os dados de entrada piora com o aumento de neurônios na Camada Latente. Esse comportamento pode ser explicado ao considerarmos que, com o aumento no número de dimensões do espaço latente, o modelo gastará mais recursos otimizando a distribuição do espaço e, consequentemente, prejudicará o treinamento de sua capacidade de reprodução dos dados.

O resultado da reprodução das imagens pelo *Autoencoder* e a evolução do RMSE de acordo com a quantidade de neurônios na Camada Latente (z) pode ser visto na Figura A.11. Este modelo teve mais dificuldades em reproduzir as imagens em comparação ao FCAE para um

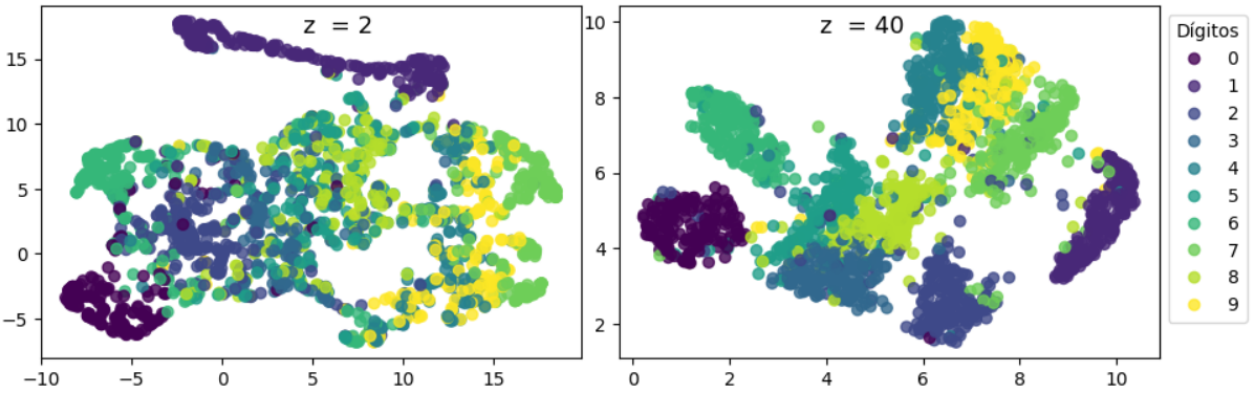

Figura A.12: VAE: redução de dimensionalidade realizada pelo UMAP. Na figura à esquerda, é apresentado o resultado para $z = 2$ e, à direta, para $z = 40$.

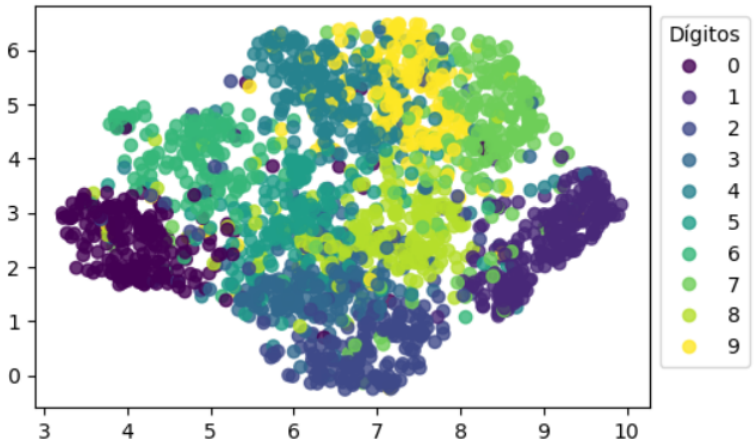

Figura A.13: VAE: redução de dimensionalidade realizada pelo UMAP para $z = 100$. Nesta imagem, a melhor organização do espaço latente está mais evidenciada.

mesmo intervalo de z. Apesar do aumento do RMSE ao longo do aumento de z, podemos notar que as imagens geradas na camada $z = 40$ estão visualmente próximas das originais. Como o modelo divide esforços entre a replicação dos dados e a organização do espaço latente, é esperado que ele tenha maior dificuldade na reprodução das imagens.

Com respeito à organização do espaço latente, ao aplicarmos o UMAP em um conjunto de pontos dos dados de teste, podemos ver na Figura A.12 que os dados estão melhores organizados para $z = 40$ (à direita) do que para $z = 2$ (à esquerda).

Especialmente na imagem da direita, é possível notar que dígitos parecidos (por exemplo, 4, 9 e 7, ou 3 e 8) ficam mais próximos entre si no espaço latente, o que deve dar melhor capacidade generativa ao modelo uma vez que itens com características semelhantes estão próximos na dimensão reduzida.

Tal comportamento fica mais evidente ao analisarmos a distribuição dos pontos para

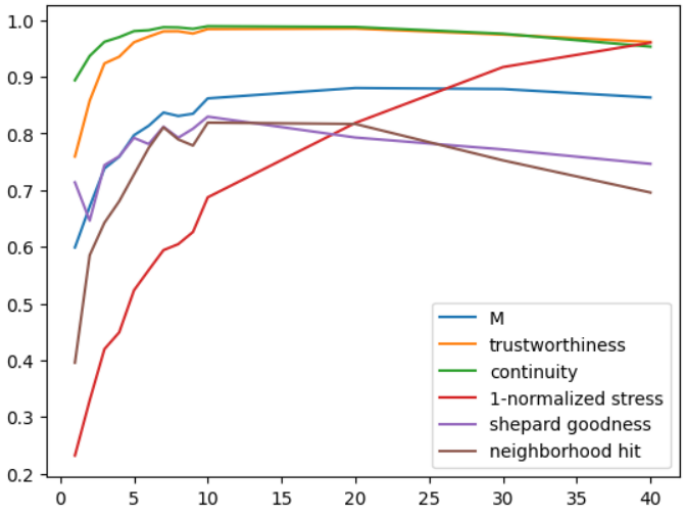

Figura A.14: VAE: evolução das métricas definidas na Seção 3.3 de acordo com a variação de z. O valor M é definido pela Equação 3.3.

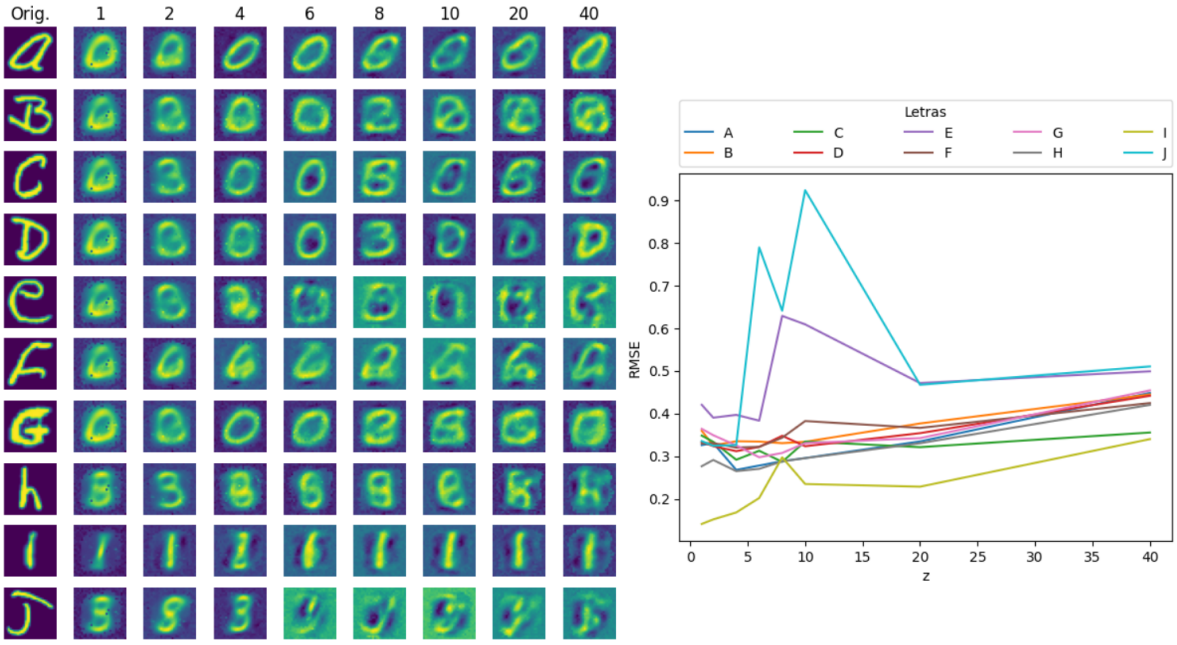

Figura A.15: VAE: (a) à esquerda, imagens originais (EMNIST) e processadas para diferentes valores de z; (b) à direta, evolução do RMSE em relação a z.

$z = 100$ conforme indicado na Figura A.13. Nela podemos observar que todas as classes vão se organizando no espaço latente de forma que aquelas com características mais parecidas ficam mais próximas entre si e portanto é esperado que regiões próximas de um determinado ponto nesse espaço de dimensão reduzida compartilhem as mesmas propriedades.

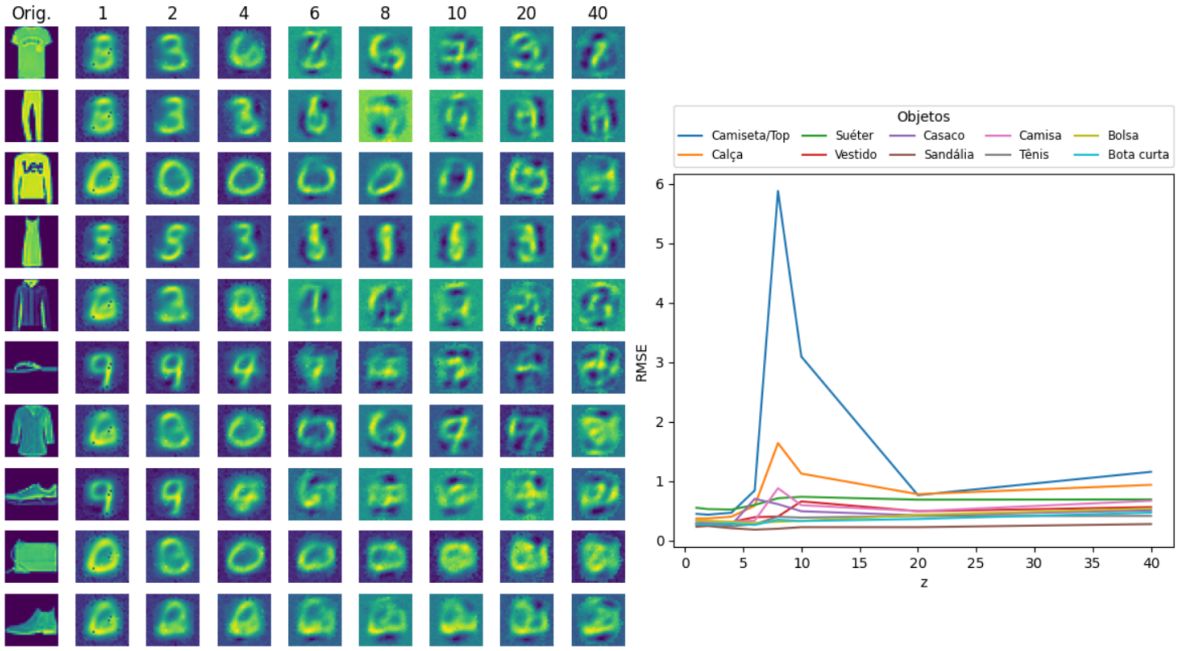

Figura A.16: VAE: (a) à esquerda, imagens originais (*Fashion*-MNIST) e processadas para diferentes valores de z; (b) à direta, evolução do RMSE em relação a z.

As métricas relacionadas à qualidade da redução de dimensionalidade foram computadas e podem ser observadas na Figura A.14. Nela podemos notar que, de forma geral, as métricas melhoram com o aumento do tamanho de Camada Latente dentro da faixa de variação observada. É possível notar que a métrica relacionada ao *normalized stress*, que mede a preservação das distâncias dos pontos entre os espaços original e reduzido, melhora significativamente com o aumento das dimensões do espaço latente.

Prosseguindo com os experimentos, testamos a capacidade do VAE em lidar com dados diferentes daqueles em que o modelo foi treinado, utilizando imagens das bases EMNIST e *Fashion*-MNIST. As imagens processadas e a evolução do RMSE podem ser observadas nas figuras A.15 e A.16. Através destas imagens, podemos observar que esse modelo de *Autoencoder* teve mais dificuldade em lidar com dados diferentes daqueles que foram usados no treinamento, em relação ao FCAE visto anteriormente.

A.2 Uso de *Autoencoder* para Aumentação de Dados

Como parte complementar dos experimentos com modelos de *Autoencoders*, avaliamos a capacidade generativa do FCAE para aumentação de dados envolvendo um problema real da competição *Painter by Numbers* promovida pela plataforma Kaggle[1]. Nessa competição, são fornecidas cerca de 80.000 imagens de obras de arte de diferentes tamanhos e artistas e o objetivo é desenvolver um modelo que, ao final do processo, seja capaz de receber duas imagens de obras de arte e dizer se elas são ou não de um mesmo artista.

Para isso, inicialmente utilizamos o modelo Inception v3[2], uma rede neural convolucional pré-treinada em imagens, como extrator de *features*. Cada imagem de obra de arte foi redimensionada para o tamanho de entrada da rede ($299 \times 299 \times 3$) e no topo da rede foi gerado um vetor de tamanho de 2048 *features* que serviu de entrada para outra rede que seria responsável pelo treino proposto na competição.

Dadas duas imagens, o modelo recupera o vetor de 2048 *features* extraído para cada uma delas pelo Inception v3, passa cada um deles por duas camadas densas de 1024 neurônios com função de ativação ReLU, agrupa os dois resultados através de uma multiplicação entre eles, elemento a elemento, resultando em um vetor de tamanho 1024 que é então conectado a uma camada de um neurônio com ativação do tipo sigmoide. O modelo é treinado de forma que o neurônio de saída tenha valor 1.0 para imagens de um mesmo artista e 0.0 para o caso contrário. O modelo treinado avalia o conjunto de teste promovido pela competição e a medida de performance é área sob a curva (AUC - *Area Under the Curve*) do gráfico ROC (*Receiver Operating Characteristic*) que traça a curva entre as taxas de verdadeiros positivos e falso positivos para diferentes limiares de classificação. Este modelo foi treinado por 70 épocas e então foi aplicado no conjunto de teste da competição (*Private Score*) obtendo um valor de AUC de 0.8397.

A aumentação de dados foi obtida através de um *Autoencoder* do tipo FCAE composto por 3 camadas ($2048 \rightarrow 1536 \rightarrow 1024 \rightarrow 512 \rightarrow$ Camada Latente) treinado por 45 épocas. O mesmo modelo foi treinado por 70 épocas com dados aumentados por duas estratégias principais: a) apenas nas classes de pintores com menos obras de arte e b) em todas as classes.

[1] https://www.kaggle.com/competitions/painter-by-numbers

[2] https://cloud.google.com/tpu/docs/inception-v3-advanced

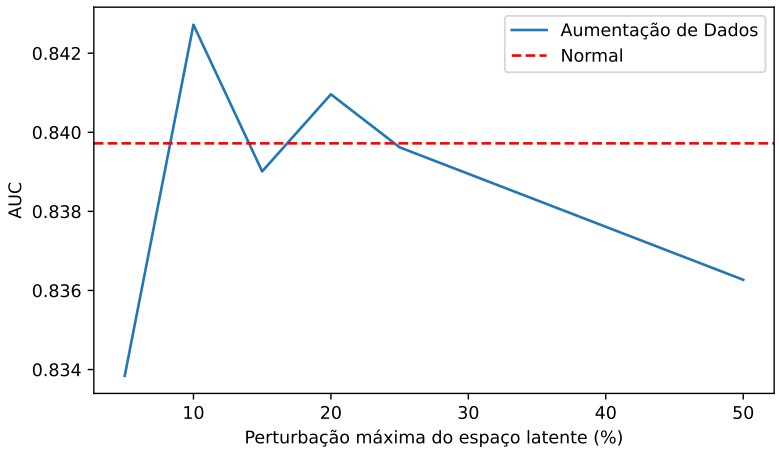

Figura A.17: Variação da AUC no conjunto de teste com uso de aumentação de dados de acordo com a porcentagem de perturbação máxima do espaço latente. Artistas com menos de 20 obras tiveram dados sintéticos criados até esse valor.

Artistas com poucas obras de arte: supondo que o modelo de classificação teria mais dificuldade em reconhecer obras de arte a que ele teria sido menos exposto na etapa de treinamento, os primeiros experimentos priorizaram a geração de dados sintéticos para pintores com poucas imagens na base de dados. Uma análise dos dados mostrou que dos cerca de 1600 diferentes artistas, o pintor com mais imagens na base de dados tinha cerca de 375 obras e aquele com menos imagens tinha 4. Metade deles tinha 20 obras ou menos e, portanto, a primeira estratégia foi usar a aumentação de dados para elevar todos os artistas a terem pelo menos 20 amostras. A perturbação no espaço latente (conforme definida na Seção 3.1) variou entre os valores 5%, 10%, 15%, 20%, 25% e 50%. O resultado da AUC no conjunto de

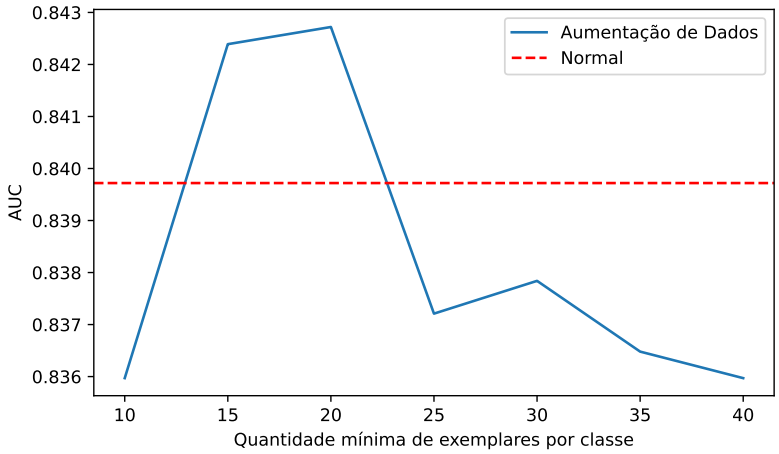

Figura A.18: Variação da AUC no conjunto de teste com uso de aumentação de dados de acordo com a quantidade mínima de exemplares por classe que foram aumentadas. A perturbação do espaço latente foi de 10% em todos os casos.

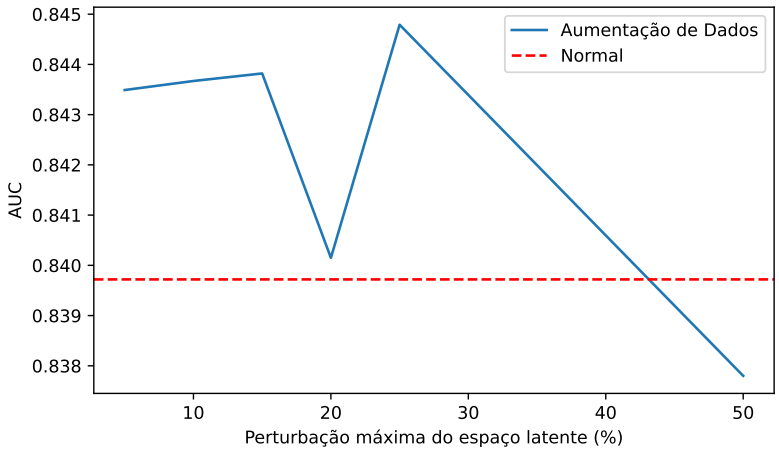

Figura A.19: Variação da AUC no conjunto de teste com uso de aumentação de dados de acordo com a porcentagem de perturbação máxima do espaço latente. Foi gerada uma imagem sintética adicional para cada amostra original.

teste da competição em comparação ao resultado obtido sem a aumentação de dados pode ser visto na Figura A.17.

Como o melhor aumento ocorreu para uma perturbação de 10%, foi realizada uma nova rodada de experimentos fixando a perturbação neste valor e variando a quantidade de classes a serem aumentadas para que cada pintor tivesse ao menos 10, 15, 20, 25, 30, 35 e 40 exemplares no conjunto de treinamento. A variação da AUC no conjunto de teste para essas variações pode ser visto na Figura A.18.

Com base nessas figuras, é possível notar que nesse problema o *Autoencoder* foi capaz de promover uma pequena melhora no resultado para perturbações da ordem de 10 a 20% em artistas com poucas amostras. Para um aumento de até 20 amostras por classe, os dados de treinamento aumentaram em cerca de 10% e, portanto, o tempo de treinamento não foi significativamente prejudicado com esta abordagem de geração de dados sintéticos.

Aumentação das amostras: uma segunda estratégia foi criar um dado sintético para cada amostra da base de dados no treinamento. Com este cenário, o tempo de treinamento do modelo foi diretamente impactado (aumentando aproximadamente o dobro), mas a eficácia foi um pouco melhor do que aquela obtida na estratégia anterior, de analisar apenas as classes com poucos exemplares. Foram testadas perturbações de 5%, 10%, 15%, 20%, 25% e 50%. Os valores de AUC obtidos podem ser observados na Figura A.19.

De forma geral, foi possível observar que um modelo de *Autoencoder* de arquitetura simples como o FCAE obteve sucesso, mesmo que pequeno, no aumento da eficácia do treinamento do modelo para determinados valores de perturbação do espaço latente. Portanto, esta ferramenta, com uma arquitetura mais complexa e associada com outras estratégias, pode ser efetiva na melhoria do treinamento de modelos de classificação em aprendizado de máquina.