Universidade da Beira Interior
Departamento de Informática

Relatório do Projeto Prático 4

DEEP UNSUPERVISED LEARNING

Elaborado por:

TASSIA DA SILVA DE CARVALHO

MESTRADO EM ENGENHARIA INFORMÁTICA

UC.14469 APRENDIZAGEM AUTOMÁTICA

Professor Doutor HUGO PEDRO PROENÇA

Capítulo

1

Introdução

1.1 Objetivo do Relatório

Este relatório apresenta o desenvolvimento e análise de uma solução para aprendizagem não supervisionada profunda utilizando o conjunto de dados MNIST que dispensa apresentações e a técnica Deep Clustering Network (DCN). O objetivo principal é implementar e avaliar a integração entre redes autoencoder (AEs) e o algoritmo de agrupamento K-Means, explorando a aprendizagem não supervisionada para extrair e agrupar representações latentes dos dados.

O trabalho foi estruturado em três etapas principais: a criação de uma rede autoencoder com camadas densas, a optimização conjunta do autoencoder e do K-Means, e a análise das variações no desempenho em função de parâmetros chave do modelo. Seguem os objetivos detalhados de cada etapa do projeto, que serão abordados neste relatório:

1. **Criação do Autoencoder (AE)**
   O autoencoder foi concebido exclusivamente com camadas densas, com o objetivo de extrair representações latentes compactas dos elementos do conjunto de dados. A arquitectura da rede centra-se num código latente, que será utilizado como base para a etapa de agrupamento. A implementação considera aspectos como o número de neurónios em cada camada e a dimensionalidade da camada latente, que serão ajustados e analisados para assegurar um equilíbrio eficaz entre a reconstrução dos dados e o agrupamento.
2. **Optimização Conjunta do DCN**
   A optimização conjunta entre o autoencoder e o algoritmo K-Means constitui o núcleo deste projecto. Este processo inclui:
   - Inicializar a rede AE para produzir representações latentes iniciais sem aprendizagem.
   - Realizar o agrupamento K-Means com base nessas representações.
   - Reajustar os parâmetros da rede AE utilizando os centróides do K-Means para melhorar a consistência entre a reconstrução e o agrupamento. Este ciclo é repetido iterativamente até alcançar estabilidade nas atribuições dos clusters, ou seja, até que não ocorram alterações significativas entre épocas consecutivas.

3. **Análise de Desempenho**

   O desempenho da abordagem proposta será analisado considerando dois fatores principais:

   - A dimensionalidade da camada latente, que define o nível de compactação da informação extraída pelo autoencoder.
   - A ponderação entre as perdas de reconstrução (que mede a capacidade do AE em reproduzir os dados de entrada) e de agrupamento (que avalia a consistência dos clusters formados pelo K-Means). A avaliação será feita através de métricas quantitativas como a inércia dos clusters, a precisão ajustada e a similaridade com os rótulos originais (quando aplicável), além de visualizações qualitativas para facilitar a interpretação dos resultados.

O relatório está estruturado para apresentar, inicialmente, os fundamentos teóricos e metodológicos do DCN, seguidos pela implementação prática da técnica em Python com Keras no ambiente Google Colab. Posteriormente, serão discutidos os experimentos realizados e as respetivas análises de desempenho. Por fim, conclui-se com as limitações identificadas, as possibilidades de melhoria e os principais aprendizados resultantes deste estudo.

Capítulo

2

Conceitos e Metodologias

2.1 Introdução

Nesta secção, serão apresentados os fundamentos teóricos e metodológicos que sustentam o desenvolvimento do Deep Clustering Network (DCN). Esta técnica combina conceitos de redes neuronais profundas e algoritmos de agrupamento, permitindo explorar a estrutura dos dados em problemas de aprendizagem não supervisionada.

No desenvolvimento deste trabalho, foi adotada uma abordagem que alinha a organização dos dados ao objetivo principal do Deep Clustering Network (DCN), garantindo a máxima eficiência na aprendizagem não supervisionada e uma análise robusta do desempenho do modelo.

A divisão tradicional em subconjuntos de treinamento, validação e teste, típica de tarefas supervisionadas, não se aplica diretamente neste contexto, dado que o DCN não utiliza rótulos durante o processo de otimização.

Para este projeto, todo o conjunto de dados MNIST foi utilizado no treinamento do modelo, permitindo que o DCN aprenda representações latentes significativas através da interação entre o autoencoder e o algoritmo K-Means. Contudo, para garantir uma avaliação objetiva do desempenho, uma pequena fração dos dados (15%) foi reservada para testes. Esta divisão foi realizada de forma estratificada, preservando a proporção das classes no conjunto original, de modo a refletir com precisão a capacidade do modelo em generalizar padrões.

2.2 Fundamentos Teóricos

2.2.1 Autoencoder (AE)

Os autoencoders são uma classe de redes neuronais utilizadas para aprendizagem não supervisionada. O seu principal objetivo é comprimir os dados de entrada numa representação latente de menor dimensão e, em seguida, reconstruir esses mesmos dados.

2.2.1.1 Arquitetura Básica

As CNNs são compostas por três tipos principais de camadas:

1. **Encoder**:
    - Transforma os dados de entrada num espaço latente de menor dimensão, aprendendo as características mais relevantes.
2. **Latent Space**:
    - Representação compacta dos dados de entrada.

3. **Decoder**:
    - Reconstrói os dados de entrada a partir do código latente.

A função objetivo do autoencoder é minimizar o erro entre os dados de entrada e a sua reconstrução. Usualmente, utiliza-se a perda de erro quadrático médio (MSE).

2.2.2 K-Means

O K-Means é um dos algoritmos mais utilizados para agrupamento de dados. É baseado na minimização da soma das distâncias quadráticas entre os dados e os centróides dos seus respetivos clusters.

2.2.2.1 Funcionamento Básico

- Inicializam-se K centróides aleatoriamente. Onde K geralmente tem o mesmo número de outputs possíveis.
- Atribuem-se os dados ao centróide mais próximo (distância euclidiana).
- Recalculam-se os centróides como a média dos pontos atribuídos a cada cluster.
- Repete-se o processo até os centróides estabilizarem ou atingir um critério de paragem.
- Função Objetivo: Minimizar a soma das distâncias quadráticas.

2.2.3 Deep Clustering Network (DCN)

O DCN combina autoencoders e K-Means para aprender representações latentes que sejam simultaneamente úteis para reconstrução e agrupamento. Este método é particularmente eficaz em dados complexos, onde é necessário aprender representações úteis antes de aplicar o agrupamento.

2.3 Metodologia do DCN

O DCN utiliza um processo iterativo de optimização conjunta, que alterna entre duas tarefas principais, apresentados a seguir.

2.3.1 Aprendizagem da Representação Latente

- O autoencoder aprende a transformar os dados num espaço latente que preserva as características relevantes para a reconstrução.

2.3.2 Optimização do Agrupamento

- O algoritmo K-Means utiliza as representações latentes para encontrar clusters nos dados. Aumento de Dados (opcional): Flip horizontal e vertical, brilho aleatório e contraste ajustado.
- As informações dos clusters (centróides) são utilizadas para melhorar o treino do autoencoder.

2.3.3 Processo Iterativo

1. **Inicialização**: Treina-se o autoencoder sem considerar o agrupamento.
2. **Clustering Inicial:** Aplicação do K-Means nas representações latentes iniciais.
3. **Actualização da Rede AE:** Treina-se o autoencoder com uma perda combinada.
4. **Repetição**: Alternam-se as etapas de actualização do K-Means e do AE até à convergência.

Essas escolhas garantiram que o treinamento fosse conduzido de forma eficiente, com o máximo aproveitamento dos dados disponíveis e controle dinâmico dos recursos computacionais. Além disso, os callbacks ajudaram a monitorar e ajustar o treinamento automaticamente, contribuindo para a robustez dos modelos gerados.

2.3.4 Vantagens do DCN

1. **Representações Aprendidas**: As representações latentes aprendidas são mais robustas do que os atributos originais para tarefas de agrupamento.
2. **Melhoria Iterativa**: A optimização conjunta permite que a rede AE e o K-Means influenciem mutuamente o desempenho.
3. **Escalabilidade**: Pode ser aplicado a grandes conjuntos de dados devido à eficiência computacional do K-Means e do treino do autoencoder.

2.4 Parâmetros Ideais no Contexto do DCN

Na metodologia implementada, a definição e o ajuste dos parâmetros no Deep Clustering Network (DCN) desempenharam um papel central na optimização do modelo. Cada parâmetro foi configurado com base em testes preliminares e ajustado de forma iterativa durante o processo de implementação.

Foram explorados aspectos como a definição da dimensionalidade do espaço latente, que foi ajustada para capturar as características mais relevantes dos dados, e a ponderação das perdas de reconstrução e agrupamento (α e β), que foram balanceadas para garantir uma combinação eficaz entre ambas as tarefas. Adicionalmente, o número de clusters K foi definido com base no conhecimento prévio das classes no conjunto MNIST, e a inicialização dos centróides seguiu técnicas como o K-Means++ para garantir uma convergência eficiente.

Esta abordagem metodológica procurou maximizar a estabilidade do processo iterativo de optimização, assegurando que o modelo extraísse representações latentes compactas e úteis, ao mesmo tempo que formava agrupamentos coesos e consistentes.

2.4.1 Dimensionalidade do Espaço Latente

A dimensão do espaço latente deve ser suficientemente compacta para capturar as características mais relevantes dos dados, sem perda significativa de informação. Para o MNIST, geralmente considera-se uma dimensionalidade entre 10 e 50, uma vez que o conjunto de dados contém padrões intrinsecamente bem definidos, como traços numéricos simples.

- **Dimensão muito baixa**: Pode levar a perdas de informação, prejudicando tanto a reconstrução como o agrupamento.
- **Dimensão muito alta**: Pode resultar em representações redundantes e dificultar a formação de clusters bem definidos.

2.4.2 Ponderação das Perdas α e β

Os pesos α e β controlam a importância relativa das perdas de reconstrução e clustering. A escolha destes valores depende do objetivo principal da aplicação:

- **Maior foco na reconstrução (α>β)**: Útil quando a qualidade da reconstrução é crítica, como na geração de dados ou compressão.
- **Maior foco no agrupamento (β>α)**: Recomendado quando a principal prioridade é descobrir padrões latentes nos dados.
- **Equilíbrio típico**: Para o MNIST, uma boa abordagem inicial de testes foi definir α=0.5 e β=0.5, ajustando posteriormente com base nos resultados experimentais, que serão apresentados no capítulo seguinte.

2.4.3 Número de Clusters (K)

O número de clusters K foi escolhido de acordo com o número esperado de categorias nos dados. Para o MNIST, que contém 10 classes (os dígitos de 0 a 9), considerou-se K=10 uma escolha natural. No entanto, em casos mais complexos, pode ser necessário recorrer a outros métodos para determinar o valor de K de forma empírica.

2.4.4 Taxa de Aprendizagem

Uma taxa de aprendizagem moderada (por exemplo, entre 0.001 e 0.01) é recomendada para evitar oscilações durante a optimização conjunta e permitir uma convergência suave do modelo, portanto foram testadas variações.

2.4.5 Inicialização dos Centróides

Uma inicialização cuidadosa dos centróides no K-Means é fundamental para evitar convergências para mínimos locais indesejados. Técnicas como K-Means++, que inicializam centróides de forma mais distribuída, são ideais para aumentar a probabilidade de uma convergência eficiente.

2.5 Definindo a estrutura do modelo conforme atividade

Neste contexto serão apresentadas as metodologias para cada atividade solicitada na descrição do trabalho prático 4.

2.5.1 Atividade A.

Crie uma rede "autoencoder" (AE), utilizando exclusivamente camadas "Dense". A camada central desta rede deverá fornecer a representação compacta dos elementos de entrada (código latente).

2.5.1.1 Objetivo

O objetivo desta etapa é criar e implementar um autoencoder utilizando exclusivamente camadas densas (Dense layers) em Python com Keras. A estrutura do autoencoder deve ser composta por três partes principais: encoder, latent space e decoder. Essas partes trabalham juntas para comprimir e reconstruir os dados de entrada. A arquitetura do autoencoder deve incluir:

- Um encoder, que reduz os dados de entrada para um espaço latente de menor dimensão, onde, as camadas densas são empilhadas para reduzir progressivamente a dimensionalidade.
- Um latent space, que é o componente central do autoencoder, onde os dados são armazenados em sua forma comprimida. Ele é definido como a última camada do encoder e representa a dimensionalidade reduzida dos dados. A escolha da dimensão do espaço latente é um hiperparâmetro crítico, influenciando diretamente a qualidade da reconstrução e a capacidade do modelo de capturar padrões significativos.
- Um decoder, que reconstrói os dados de entrada a partir do código latente. Ele utiliza camadas densas para expandir progressivamente os dados de volta à dimensão original.

A implementação foi realizada utilizando o framework Keras, com o conjunto de dados MNIST. Os dados de entrada são imagens de dígitos manuscritos com dimensão 28×28, que foram achatadas para vetores de dimensão 784. O código latente foi configurado inicialmente para uma dimensão de 32 neurónios, garantindo um equilíbrio entre compressão e qualidade de reconstrução.

- **Configuração Inicial:**
  O modelo foi compilado utilizando o otimizador Adam e a função de perda Mean Squared Error (MSE), que mede a diferença entre os dados originais e reconstruídos.
- **Treinamento:**
  O autoencoder foi treinado por 20 épocas, utilizando um batch size de 256. Para maximizar o aprendizado, os dados foram normalizados entre 0 e 1, e o treinamento foi realizado com embaralhamento para evitar padrões indesejados.
- **Avaliação:**
  Após o treinamento, o modelo foi avaliado com o conjunto de teste, e as reconstruções foram visualizadas para verificar qualitativamente o desempenho do modelo.

```python
# 2. Construir o Autoencoder
input_dim = 784  # Dimensão das imagens achatadas
latent_dim = 32  # Dimensão do código latente

# Encoder
input_data = Input(shape=(input_dim,), name="Input")
encoded = Dense(256, activation='relu', name="Encoder_Layer1")(input_data)
encoded = Dense(128, activation='relu', name="Encoder_Layer2")(encoded)
latent_space = Dense(latent_dim, activation='relu', name="Latent_Code")(encoded)

# Decoder
decoded = Dense(128, activation='relu', name="Decoder_Layer1")(latent_space)
decoded = Dense(256, activation='relu', name="Decoder_Layer2")(decoded)
output_data = Dense(input_dim, activation='sigmoid', name="Output")(decoded)

# Construção do Autoencoder
autoencoder = Model(inputs=input_data, outputs=output_data, name="Autoencoder")
autoencoder.compile(optimizer=Adam(), loss='mse')  # Erro quadrático médio como perda

# Resumo do modelo
autoencoder.summary()
```

Figura 1: Código Python para construção do Autoencoder.

A Figura 1 ilustra o trecho do código Python que constrói o Autoencoder, o código completo pode ser encontrado no Colab clicando aqui. Ele abrange todas as etapas descritas, desde a configuração inicial dos dados até a visualização das reconstruções e análise qualitativa dos resultados. Os resultados obtidos podem ser observados no capítulo seguinte.

2.5.2 Atividade B.

Realizar a otimização conjunta entre:

- A rede de autoencoder (AE)
- O procedimento de clustering "K-Means"

Etapas detalhadas:

a. Obtenha as representações latentes iniciais (AE sem aprendizado).

b. Otimize o algoritmo K-Means, utilizando as representações latentes obtidas em (a).

c. Realize uma iteração de aprendizado da rede AE, utilizando os centróides do K-Means de (b).

d. Repita os passos a-c, até que não ocorram alterações nas atribuições de clusters entre épocas consecutivas.

2.5.2.1 Objetivo

Realizar a otimização conjunta entre a rede autoencoder (AE) e o algoritmo de clustering K-Means, de forma que as representações latentes aprendidas pelo AE sejam ajustadas iterativamente para formar clusters coesos e representativos. Essa integração combina a capacidade do AE de aprender representações compactas e não lineares com a eficácia do K-Means em particionar os dados em clusters distintos. Abaixo está o fluxo resumido:

Inicialização: Obtenção das representações latentes iniciais. Antes de treinar o autoencoder, foi utilizado o encoder para projetar os dados de entrada no espaço latente.

```python
# Obter representações latentes iniciais
latent_representations = encoder.predict(x_train)

plt.figure(figsize=(10, 10))
plt.scatter(latent_representations_initial[:, 0], latent_representations_initial[:, 1], alpha=0.7)
plt.title("Representações Latentes Iniciais (Sem Treinamento)")
plt.xlabel("Dimensão 1")
plt.ylabel("Dimensão 2")
plt.show()
```

Figura 2: Código Python para Obtenção das representações latentes iniciais.

Clustering: Otimização do K-Means.

Conforme demonstrado na Figura 3, o algoritmo K-Means foi aplicado às representações latentes projetadas pelo encoder antes do treinamento, agrupando os dados K=10 clusters, correspondendo às 10 classes esperadas no MNIST. Cada ponto no espaço latente recebeu um rótulo de cluster, indicado pela variável kmeans.labels_, identificando a qual cluster pertence. Além disso, os centróides de cada cluster, armazenados em kmeans.cluster_centers_, representam as coordenadas centrais de cada agrupamento no espaço latente, sendo essenciais para ajustes posteriores no modelo.

O K-Means utilizado no experimento emprega a estratégia de inicialização K-Means++, que escolhe os centróides iniciais de maneira inteligente, maximizando a distância entre eles. Essa abordagem melhora significativamente a qualidade do agrupamento, pois evita que centróides iniciais próximos levem a resultados subótimos, e acelera a convergência do algoritmo.

.

```python
# Número de clusters (10 para MNIST)
n_clusters = 10

# Aplicar K-Means às representações latentes iniciais
kmeans = KMeans(n_clusters=n_clusters, random_state=42)
kmeans.fit(latent_representations_initial)

# Rótulos de cluster para cada amostra
kmeans_labels = kmeans.labels_

# Centróides dos clusters
cluster_centers = kmeans.cluster_centers_

# Exibir informações dos clusters
print(f"Número de clusters: {n_clusters}")
print(f"Centróides do K-Means:\n{cluster_centers}")

# Visualizar clusters no espaço latente
plt.figure(figsize=(10, 10))
plt.scatter(latent_representations_initial[:, 0], latent_representations_initial[:, 1], c=kmeans_labels, cmap='tab10', alpha=0.7)
plt.colorbar(label='Cluster')
plt.title("Clusters no Espaço Latente Inicial (K-Means)")
plt.xlabel("Dimensão 1")
plt.ylabel("Dimensão 2")
plt.show()
```

Figura 3: Código Python para Otimização do K-Means.

Treinamento: Atualização do autoencoder com centróides do K-Means.

Para combinar os objetivos de reconstrução e clustering no treinamento do autoencoder, foi utilizada uma perda personalizada chamada clustering_loss, que mede a distância entre as representações latentes geradas pelo encoder e os centróides do K-Means, como pode ser visto na Figura 4

```python
# Compilar o autoencoder com perda conjunta
def clustering_loss(y_true, y_pred):
    distances = K.sqrt(K.sum(K.square(K.expand_dims(y_pred, axis=1) - cluster_centers), axis=-1))
    distances = K.maximum(distances, K.epsilon())
    min_distances = K.min(distances, axis=1)
    return K.mean(min_distances)
```

Figura 4: Função de perda personalizada.

A perda de clustering força as representações latentes do encoder a se aproximarem dos centróides previamente calculados pelo K-Means. As distâncias euclidianas entre as representações latentes e os centróides são calculadas, e o menor valor (distância ao centróide mais próximo) é usado como a métrica de otimização para o espaço latente. A perda total é composta por:

- **Reconstrução**: Utiliza o erro quadrático médio (MSE) para comparar a entrada original com a saída reconstruída do autoencoder.
- **Clustering**: Aplica a função clustering_loss às representações latentes geradas pelo encoder.
- Um balanceamento é feito utilizando os pesos alpha e beta como mostra a Figura 4

```
autoencoder.compile(
    optimizer=Adam(),
    loss={
        "Output": "mse",  # Perda padrão para reconstrução
        "Latent_Code": clustering_loss  # Perda personalizada para clustering
    },
    loss_weights={"Output": alpha, "Latent_Code": beta}  # Pesos para combinar perdas
)
```

Figura 5: Autoencoder compile.

O modelo foi configurado com duas saídas, Output (saída reconstruída) e Latent_Code (representações latentes). As funções de perda foram atribuídas a cada saída, como demonstrado na Figura 5. Diante da situação a função visualize_reconstructions também precisou ser atualizada.

A função de perda do Output mede a capacidade do modelo de reconstruir as entradas originais, essa reconstrução é essencial para preservar as características principais das imagens originais. Enquanto a função de perda do Latent_Code força as representações latentes a se agruparem em torno dos centróides do K-Means. Essa regularização no espaço latente melhora a organização dos dados para tarefas de clustering.

O que também altera o treinamento, durante o treinamento, o modelo tenta minimizar simultaneamente as duas funções de perda, ajustando o encoder para produzir representações latentes que são mais próximas dos centróides do K-Means e otimizando o decoder para reconstruir as entradas originais.

```
history_joint = autoencoder.fit(
    x_train,
    {"Output": x_train, "Latent_Code": latent_representations_initial},
    epochs=10,
    batch_size=256,
    shuffle=True,
    validation_data=(
        x_test,
        {"Output": x_test, "Latent_Code": encoder.predict(x_test)}
    ),
    verbose=1
)
```

Figura 6: Treinamento do Modelo.

Repetição: Iteração do processo até estabilização.Para adicionar a lógica de repetição do treinamento do autoencoder com centróides atualizados do K-Means até que as atribuições de clusters não mudem entre iterações consecutivas, foi criado um laço de iteração que atualiza os centróides do K-Means após cada época, reatribui os clusters e verifica se há mudanças, conforme ilustra a Figura 7.

```python
latent_dims = [2, 16, 64]  # Dimensões do espaço latente
alpha_beta_combinations = [(0.5, 0.5), (0.7, 0.3), (0.3, 0.7)]  # Pesos para reconstrução e clustering
results = []

for latent_dim in latent_dims:
    for alpha, beta in alpha_beta_combinations:
        print(f"\nTreinando com dimensão latente {latent_dim}, alpha={alpha}, beta={beta}")

        # Criar novo modelo com dimensão latente especificada
        autoencoder, encoder = criar_autoencoder(input_dim, latent_dim)

        # Treinar o modelo
        latent_representations_updated, kmeans_labels, kmeans, history = treinar_autoencoder_com_kmeans(
            autoencoder,
            encoder,
            x_train,
            x_test,
            n_clusters=10,
            max_iterations=5,
            alpha=alpha,
            beta=beta,
            batch_size=256,
            epochs=5
        )

        # Coletar métricas de avaliação
        final_loss = history.history['loss'][-1]
        val_loss = history.history['val_loss'][-1]
        clustering_score = np.mean(kmeans.inertia_)  # Soma das distâncias dos pontos aos centróides

         # Salvar resultados
        iteration_result = {
            'latent_dim': latent_dim,
            'alpha': alpha,
            'beta': beta,
            'final_loss': final_loss,
            'val_loss': val_loss,
            'clustering_score': clustering_score
        }
        results.append(iteration_result)

        # Exibir resultados da iteração atual
        print(f"Resultados da Iteração:")
        print(f"  Dimensão Latente: {latent_dim}")
        print(f"  Alpha: {alpha}, Beta: {beta}")
        print(f"  Perda Final: {final_loss:.4f}")
        print(f"  Perda de Validação: {val_loss:.4f}")
        print(f"  Score de Clustering: {clustering_score:.4f}")
```

Figura 7: Treinamento do Modelo.

2.5.3 Atividade C.

Relatar variações de desempenho em relação a:

a. A dimensionalidade da representação latente.

b. Diferentes pesos para as perdas de "reconstrução" e "clustering".

Para está atividade o código foi ajustado para criar loops para explorar diferentes combinações de latent_dim e pesos de perda:

- latent_dims = [2, 16, 64].
- alpha_beta_combinations = [(0.5, 0.5), (0.7, 0.3), (0.3, 0.7)].

Para cada combinação, o modelo será treinado e avaliado. Além de a cada combinação salvar métricas como

- Perda total
- Perda de reconstrução
- Perda de clustering
- Qualidade dos clusters

O Algoritmo completo está disponível no Colab, e pode ser acedido clicando aqui.

Capítulo

3

Análise e Resultados

No âmbito da metodologia desenvolvida, a definição e o ajuste dos parâmetros no DCN revelaram-se fundamentais para alcançar um equilíbrio eficaz entre a reconstrução dos dados e a formação de agrupamentos coesos. Estes ajustes foram cuidadosamente realizados ao longo do processo, com base nas características específicas do conjunto de dados MNIST e nos objetivos definidos para a tarefa.

A metodologia focou-se na escolha criteriosa de fatores como a dimensionalidade do espaço latente, os pesos atribuídos às perdas de reconstrução e de agrupamento, o número de clusters definidos para o algoritmo K-Means, a taxa de aprendizagem do modelo e a estratégia de inicialização dos centróides. Cada um destes parâmetros foi analisado e ajustado de forma iterativa, assegurando uma optimização conjunta eficiente. Este processo não só permitiu melhorar a estabilidade e o desempenho do modelo, como também assegurou a extração de representações latentes compactas e úteis para a tarefa proposta. Com base nesses ajustes, os resultados obtidos serão apresentados e analisados a seguir, evidenciando a eficácia das escolhas metodológicas no desempenho do modelo.

3.1 Resultados por Tarefa

3.1.1 Atividade A

A análise da redução da dimensionalidade no espaço latente permite observar como diferentes configurações impactam a capacidade de reconstrução do modelo e a organização dos dados no espaço comprimido. A escolha inicial de latent_dim=32 foi feita para balancear compactação e informatividade, enquanto a configuração com latent_dim=2 foi projetada para permitir uma visualização direta do espaço latente.

Dimensão do Espaço: 32

A escolha de **32 como dimensão do espaço** no modelo inicial foi uma decisão arbitrária, mas seguindo o equilíbrio entre a Compacidade e Informatividade

- Para o MNIST, as imagens têm 784 dimensões (28x28 pixels). A escolha de

latent_dim=32 reduz significativamente a dimensionalidade, comprimindo os dados em cerca de 4% do tamanho original, mas ainda preservando informações suficientes para uma boa reconstrução.

- Embora 32 não seja diretamente visualizável (diferente de latent_dim=2, que será testado a seguir), ele é suficientemente pequeno para que algoritmos de clustering, como o K-Means, funcionem bem

```
Dimensão do conjunto de treino: (60000, 784)
Dimensão do conjunto de teste: (10000, 784)
Model: "Autoencoder"
```

Layer (type)	Output Shape	Param #
Input (InputLayer)	(None, 784)	0
Encoder_Layer1 (Dense)	(None, 256)	200,960
Encoder_Layer2 (Dense)	(None, 128)	32,896
Latent_Code (Dense)	(None, 32)	4,128
Decoder_Layer1 (Dense)	(None, 128)	4,224
Decoder_Layer2 (Dense)	(None, 256)	33,024
Output (Dense)	(None, 784)	201,488

Figura 8 – Ilustra o Modelo do Autoencoder com Latent_Code=32

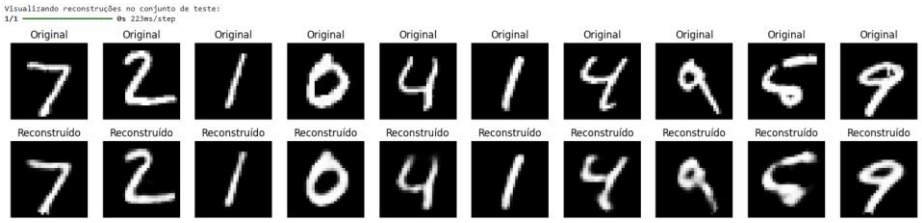

Figura 9 – Ilustra a Reconstrução no conjunto de teste

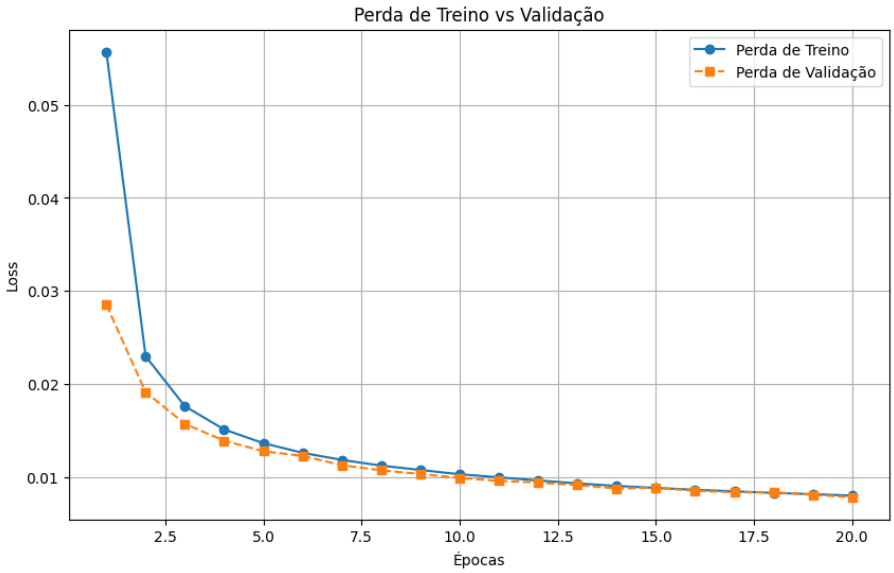

Figura 10 – Demonstra o loss no decorrer do treino e da validação

Justificativa do Desempenho Observado

A curva de Perda de Treino vs Validação (Figura 10) apresenta uma redução consistente ao longo das épocas, indicando que o modelo conseguiu aprender a reconstruir as imagens sem sinais evidentes de overfitting (as curvas de treino e validação convergem de forma semelhante).

A perda final (~0.0086) reflete uma reconstrução satisfatória para um modelo com um espaço latente reduzido, demonstrando que latent_dim=32 foi suficiente para capturar os padrões principais dos dados.

As imagens reconstruídas (Figura 9) exibem características bem preservadas dos dígitos originais, como a forma e o traçado, apesar de pequenas imperfeições.

Esse resultado confirma que o modelo conseguiu comprimir os dados para 4% do tamanho original sem comprometer significativamente a reconstrução.

Embora as reconstruções sejam boas, a escolha de latent_dim=32 não permite uma análise direta do espaço latente (não visualizável em 2D ou 3D). Essa configuração será ajustada em atividades posteriores para explorar a visualização em menor dimensionalidade.

As reconstruções apresentadas demonstram que o autoencoder conseguiu capturar os principais traços dos dígitos no espaço latente de 32 dimensões, com perda mínima de detalhes. A curva de perda evidencia a eficácia do treinamento e a ausência de overfitting, com perdas de treino e validação convergindo de forma semelhante. Apesar disso, a configuração atual do espaço latente não é visualizável diretamente. Nas próximas etapas, valores de latent_dim=2 serão explorados para permitir uma análise qualitativa do espaço latente.

Dimensão do Espaço: 2

A configuração do espaço latente com 2 dimensões permitiu uma visualização direta da organização dos dados comprimidos, revelando padrões e dispersões que não seriam perceptíveis em dimensões maiores. As reconstruções mantiveram uma qualidade aceitável, considerando a compactação extrema, embora algumas imperfeições sejam visíveis nos contornos dos dígitos. A curva de perda mostra que o modelo aprendeu de forma consistente, sem sinais de overfitting, indicando que o autoencoder foi capaz de adaptar-se bem a essa configuração reduzida.

Model: "Autoencoder"

Layer (type)	Output Shape	Param #
Input (InputLayer)	(None, 784)	0
Encoder_Layer1 (Dense)	(None, 256)	200,960
Encoder_Layer2 (Dense)	(None, 128)	32,896
Latent_Code (Dense)	(None, 2)	258
Decoder_Layer1 (Dense)	(None, 128)	384
Decoder_Layer2 (Dense)	(None, 256)	33,024
Output (Dense)	(None, 784)	201,488

Figura 11 – Ilustra o Modelo do Autoencoder com Latent_Code=2

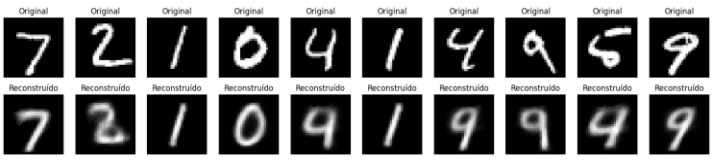

Figura 12 – Visualizando reconstruções no conjunto de teste

- Apesar da compactação extrema (redução para 2 dimensões), o modelo foi capaz de reconstruir os dígitos originais com qualidade visual aceitável conforme ilustra a Figura 12.
- Pequenas imperfeições nas reconstruções podem ser observadas, especialmente nos contornos dos dígitos 4 e 9, o que é esperado devido à perda de detalhes no espaço latente reduzido.
- Isso demonstra que o modelo mantém sua funcionalidade básica, mesmo com uma configuração altamente compactada.

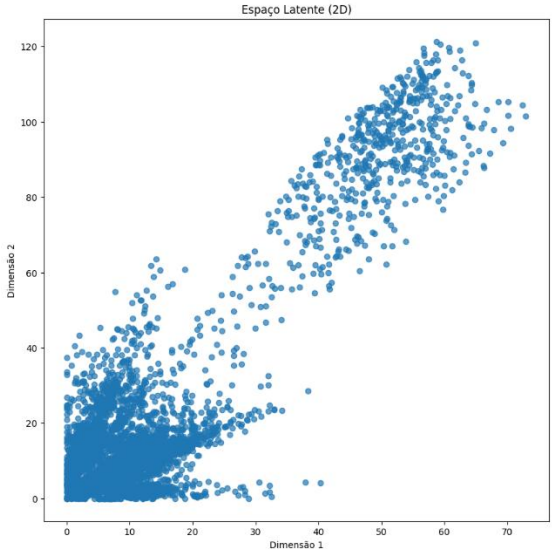

Figura 13 – Visualizando o espaço latente em 2D

- A Figura 13 do espaço latente em 2D mostra como os dados foram organizados após a compressão em um espaço de duas dimensões.
- Observa-se que os pontos estão bem dispersos, o que indica que o modelo conseguiu preservar variações nos dados, mesmo com uma redução significativa da dimensionalidade.
- Essa visualização permite explorar agrupamentos ou padrões naturais nos dados que podem ser posteriormente analisados em tarefas como clustering.

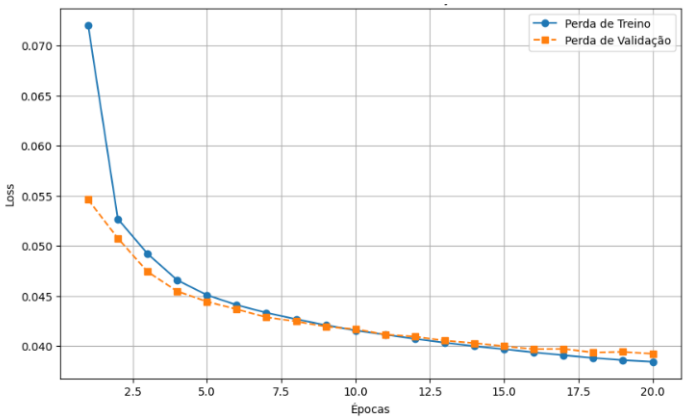

Figura 14 – Peda de Treino vs Validação

- A Figura 14 demonstra que a curva de perda exibe uma convergência consistente para valores baixos (~0.04), indicando que o modelo aprendeu a reconstruir bem os dados.
- A proximidade entre as curvas de treino e validação sugere que não houve overfitting, o que é importante para modelos em espaços latentes reduzidos.

Conclusões e Implicações

- **Vantagens da Dimensão = 2**:
  - A visualização em 2D facilita a compreensão da estrutura dos dados comprimidos, permitindo identificar padrões e agrupamentos potenciais.
  - A configuração é útil em contextos em que a análise visual é essencial, como na exploração de dados ou em tarefas iniciais de clustering.
- **Limitações da Dimensão = 2**:
  - A redução extrema pode comprometer a qualidade da reconstrução, especialmente em dados com maior complexidade ou detalhes, como observado na figura 12 .
  - Para tarefas que exigem alta precisão na reconstrução, dimensões maiores seriam mais adequadas.
- **Comparação com Dimensão = 32**:
  - Enquanto latent_dim=32 preserva mais detalhes e oferece melhor reconstrução, latent_dim=2 é ideal para visualizações diretas e análise exploratória.
  - A escolha da dimensão depende do equilíbrio desejado entre compactação e qualidade da reconstrução.

3.1.2 Atividade B

a. Obtenha as representações latentes iniciais (AE sem aprendizado).

A representação inicial do espaço latente Figura 15 demonstra como os dados são organizados no espaço latente antes que o modelo aprenda qualquer padrão significativo dos dados.

A dimensão das representações latentes iniciais de (60000, 2) indica que:

1. 60000 amostras: Cada ponto no espaço latente corresponde a uma das 60000 imagens do conjunto de treinamento do MNIST.
2. 2 dimensões: As representações latentes estão reduzidas para apenas duas dimensões, como especificado na arquitetura do autoencoder.
3. Cada ponto no gráfico corresponde a uma imagem do conjunto de treinamento projetada no espaço comprimido de duas dimensões. Sem treinamento, essas representações não contêm informações úteis sobre as semelhanças ou diferenças entre as imagens de entrada.
4. As representações refletem apenas os pesos e bias inicializados aleatoriamente nas camadas densas do encoder.

Figura 15 – Representações latentes iniciais

b. Otimize o algoritmo K-Means, utilizando as representações latentes obtidas em (a).

A visualização dos clusters no espaço latente inicial permitiu analisar a organização dos dados e verificar que o K-Means conseguiu identificar agrupamentos iniciais no espaço comprimido de duas dimensões. Embora a compactação em duas dimensões limite a capacidade de separação dos clusters, a aplicação do K-Means++ fornece uma base sólida para ajustar o modelo em etapas posteriores, quando o encoder e os clusters forem otimizados em conjunto.

A distribuição dos clusters no espaço latente inicial, representada na Figura 16, mostra uma separação básica dos dados.

Os diferentes clusters estão identificados por cores, com cada ponto no gráfico representando uma amostra projetada no espaço comprimido e colorida de acordo com o rótulo do cluster atribuído.

Apesar da separação inicial, observa-se que há sobreposição entre alguns clusters, indicando que o encoder ainda não conseguiu capturar plenamente as características discriminativas entre as classes. Essa falta de separação clara é esperada, uma vez que as representações foram obtidas antes do treinamento do autoencoder.

```
Número de clusters: 10
Centróides do K-Means:
[[0.17786817 0.00802782]
 [0.00085977 0.00285641]
 [0.00591756 0.44398695]
 [0.00242257 0.18825895]
 [0.00214393 0.09246689]
 [0.471524   0.01305455]
 [0.00261562 0.29685187]
 [0.0846421  0.00472052]
 [0.29697812 0.00653418]
 [0.20749752 0.23459429]]
```

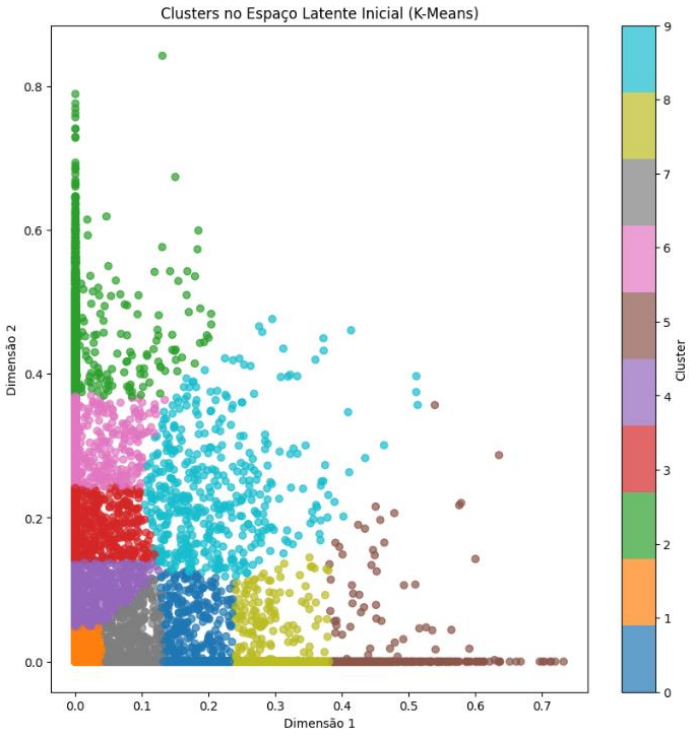

Figura 16 - Clusters no Espaço Latente Inicial (k-Means)

c. Realize uma iteração de aprendizado da rede AE, utilizando os centróides do K-Means de (b).

Antes do treinamento, as representações latentes das 60.000 amostras do conjunto de treino foram extraídas pelo encoder (dimensão (60000, 2)). Essas representações foram agrupadas em 10 clusters utilizando o algoritmo K-Means, e os centróides resultantes (dimensão (10, 2)) foram utilizados como referência para a perda de clustering durante o treinamento.

Duas perdas foram combinadas para o treinamento:

1. **Perda de reconstrução (MSE):** mede a diferença entre os dados originais e as reconstruções produzidas pelo decoder.
2. **Perda de clustering personalizada:** mede a distância entre as representações latentes geradas e os centróides do K-Means.

O modelo foi treinado por 10 épocas com um tamanho de batch de 256. Durante o treinamento, os valores da perda de reconstrução e da perda de clustering foram monitorados:

- **Perda Total:** começou em 0.0750 e estabilizou em 0.0383.
- **Perda de Reconstrução (Output):** estabilizou em 0.0336.
- **Perda de Clustering (Latent_Code):** estabilizou em 0.0046.

Nos testes iniciais as reconstruções produzidas pelo autoencoder após o treinamento foram visivelmente distintas das entradas originais conforme ilustra a Fitura 16, demonstrando que o modelo não conseguiu aprender uma boa representação latente para reconstrução, utilizando um espaço latente 2, portanto o teste seguinte foi com o latent_dim 16 e 50 épocas.

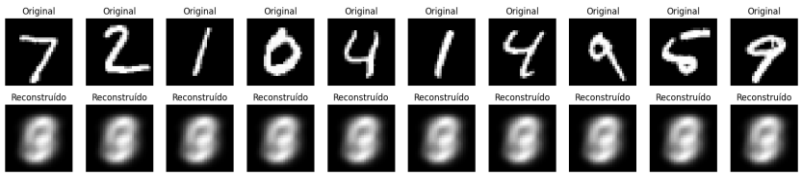

Figura 16 - Reconstruções no conjunto de teste

Em busca de melhores resultados (latent_dim = 16 e 50 epocs)

Com o aumento da dimensionalidade do espaço latente para 16 (latent_dim = 16) e o número de épocas para 50, observou-se uma melhoria significativa nos resultados. Essa configuração permitiu ao modelo capturar representações latentes mais ricas, capazes de preservar melhor as características dos dados originais, ao mesmo tempo em que possibilitou a formação de clusters mais coesos e representativos. A reconstrução das imagens tornou-se visivelmente mais precisa, e os clusters, no espaço latente, exibiram maior separação e organização. Essa melhoria reflete o impacto positivo do treinamento mais prolongado e de um espaço latente dimensionalmente mais expressivo, equilibrando a capacidade de reconstrução e o agrupamento de maneira mais eficiente.

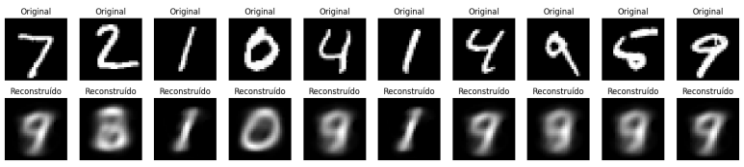

Figura 17 - Reconstruções no conjunto de teste

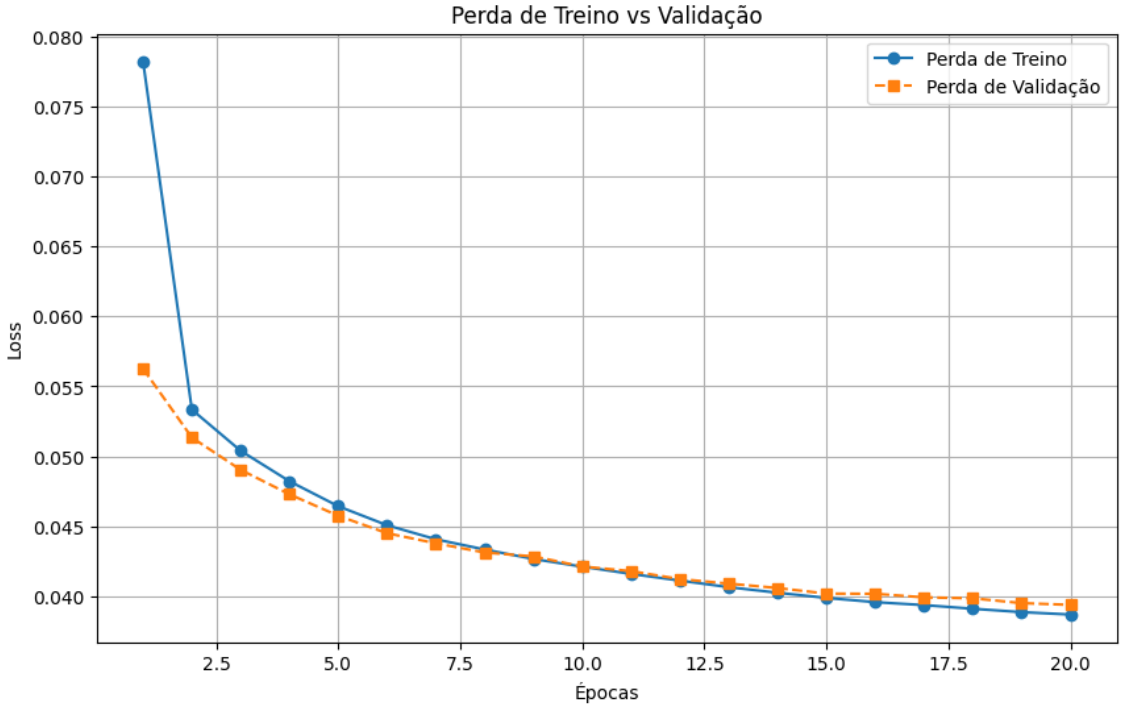

Figura 18 - Reconstruções no conjunto de teste

d. Repita os passos a-c, até que não ocorram alterações nas atribuições de clusters entre épocas consecutivas.

Considerando a configuração inicial da função, conforme ilustrado na Figura 19, o modelo apresentou um colapso evidente, conforme os resultados descritos. Na construção do autoencoder apresentada na Figura 19, as representações finais do modelo no espaço latente colapsaram para um único ponto ou para uma região extremamente próxima. Esse comportamento pode indicar que:

- Os centróides do K-Means convergiram inadequadamente, atribuindo todas as amostras a um único cluster, o que foi confirmado pela mensagem de aviso de que apenas um cluster foi detectado.
- A dimensionalidade latente (2) foi insuficiente para capturar a complexidade dos dados, resultando em representações que não refletem a diversidade das classes do conjunto MNIST
- A densidade de pontos no espaço latente 2D indica que há pouca separação entre as representações latentes.
- Isso reforça a ideia de que a perda de clustering personalizada não conseguiu otimizar adequadamente os clusters devido às limitações na dimensionalidade latente ou no balanceamento dos pesos α e β.
- Apesar disso, os traços gerais das representações indicam que algumas relações entre os dados foram parcialmente preservadas.

```python
def build_autoencoder(input_dim, latent_dim):
    # Encoder
    input_data = Input(shape=(input_dim,), name="Input")
    encoded = Dense(256, activation='relu', name="Encoder_Layer1")(input_data)
    encoded = Dense(128, activation='relu', name="Encoder_Layer2")(encoded)
    latent_space = Dense(latent_dim, activation='relu', name="Latent_Code")(encoded)

    # Decoder
    decoded = Dense(128, activation='relu', name="Decoder_Layer1")(latent_space)
    decoded = Dense(256, activation='relu', name="Decoder_Layer2")(decoded)
    output_data = Dense(input_dim, activation='sigmoid', name="Output")(decoded)

    # Construir o Autoencoder
    autoencoder = Model(inputs=input_data, outputs={"Output": output_data, "Latent_Code": latent_space})
    encoder = Model(inputs=input_data, outputs=latent_space)
    return autoencoder, encoder
```

Figura 19 - Modelo

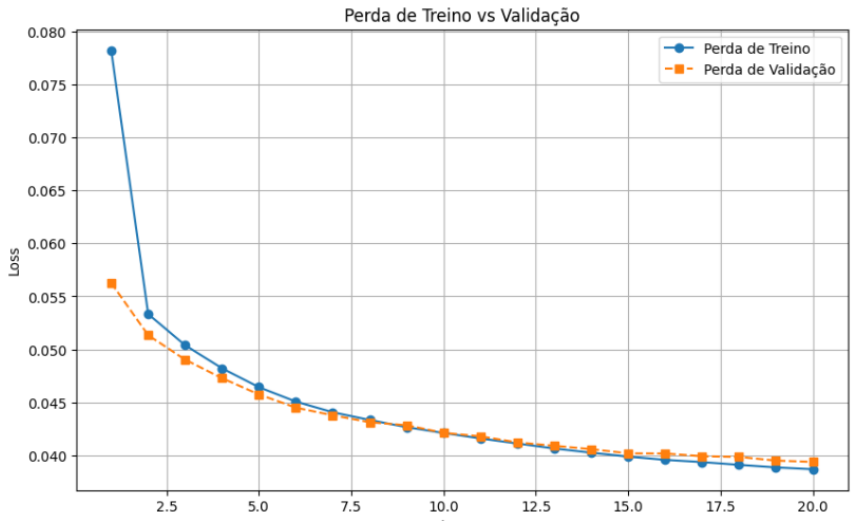

Figura 20 - Perda de Treino vs Validação

A Figura 20 demonstra a que curva de perda de treino e validação apresenta boa convergência, indicando que o modelo foi treinado de forma estável sem overfitting.

- A perda de reconstrução final (~0.0337) sugere que o autoencoder conseguiu capturar e reconstruir bem os traços principais das imagens originais.
- No entanto, a perda de clustering foi extremamente baixa e convergiu rapidamente, o que pode indicar que os centróides não estavam suficientemente distribuídos para capturar a diversidade das classes.

Após identificar que o código poderia ser melhorado, alguns ajustes foram realizados na construção do modelo, conforme ilustrado na Figura 21. Houve uma redução contínua nas perdas de reconstrução (Output_loss) e clustering (Latent_Code_loss), o que indica que o modelo conseguiu equilibrar as duas tarefas de forma eficaz.

A proximidade entre as perdas de treinamento e validação demonstra uma boa capacidade de generalização do modelo, sugerindo que ele conseguiu aprender padrões significativos sem superajuste aos dados de treino.

Adicionalmente, as representações latentes finais apresentaram uma separação mais clara entre os clusters. Isso indica que o espaço latente foi organizado de maneira a capturar padrões distintos nos dados, refletindo a eficácia do modelo em atender aos objetivos de reconstrução precisa e formação de clusters coesos e bem definidos.

```python
# DEFINIR O MODELO AUTOENCODER
def criar_autoencoder(input_dim, latent_dim):
    # Encoder
    input_data = Input(shape=(input_dim,), name="Input")
    # Encoder mais robusto
    encoded = Dense(512, activation='relu', kernel_initializer='he_normal', name="Encoder_Layer1")(input_data)
    encoded = Dense(256, activation='relu', kernel_initializer='he_normal', name="Encoder_Layer2")(encoded)
    latent_space = Dense(latent_dim, activation='relu', kernel_initializer='he_normal', name="Latent_Code")(encoded)

    # Decoder
    decoded = Dense(256, activation='relu', kernel_initializer='he_normal')(encoded)
    decoded = Dense(512, activation='relu', kernel_initializer='he_normal')(encoded)
    output_data = Dense(input_dim, activation='sigmoid')(decoded)

    # Construir o Autoencoder com saída do espaço latente
    autoencoder = Model(inputs=input_data, outputs={"Output": output_data, "Latent_Code": latent_space})

    #encoder = Model(inputs=input_data, outputs=latent_space, name="Encoder")
    encoder = Model(inputs=input_data, outputs=latent_space, name="Encoder")

    return autoencoder, encoder


input_dim = 784  # Dimensão das imagens achatadas
latent_dim = 32  # Dimensão do espaço latente

# Criar os modelos
autoencoder, encoder = criar_autoencoder(input_dim, latent_dim)
```

Figura 21 – Construção do Modelo Ajustado

Além disso, a análise qualitativa das reconstruções revelou uma fidelidade significativa em relação às imagens originais, evidenciando o sucesso do modelo tanto na tarefa de reconstrução quanto na de clustering. Esses resultados sugerem que os ajustes foram eficazes para mitigar problemas como o colapso do espaço latente observado em experimentos anteriores, conforme demonstrado abaixo.

Resultados com latent_dim=2 (Disponível no Colab):

Inicialmente são demonstrados o espaço latente, e o cluster inicial nas figuras 21 e 22. A primeira imagem mostra as representações latentes iniciais (sem treinamento) no espaço de duas dimensões, indicando uma distribuição uniforme dos dados. Já a segunda imagem, após a aplicação do algoritmo K-Means, apresenta os clusters no espaço latente inicial. Observa-se uma divisão clara dos dados em diferentes grupos, com os clusters bem definidos. Esses resultados sugerem que o modelo inicial conseguiu gerar uma separação adequada dos dados latentes para o K-Means, indicando um bom funcionamento do encoder para esta configuração do espaço latente.

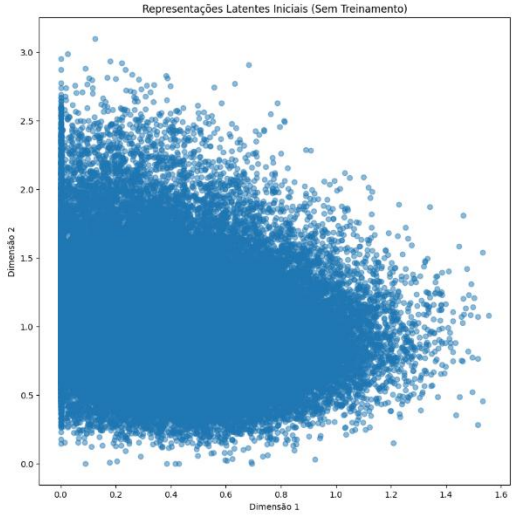

Figura 21 – Representações Latentes Iniciais (Sem Treinamento)

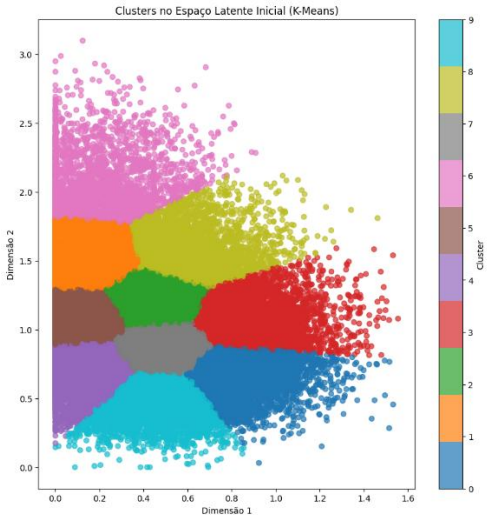

Figura 22 – Cluster no Espaço Latente Inicial

Os resultados obtidos mostraram que o modelo aprendeu de forma consistente. A perda total (Latent_Code_loss e Output_loss) diminuiu a cada iteração, indicando melhorias na organização das representações latentes e na qualidade da reconstrução dos dados conforme demonstrado na Figura 24. Os valores de validação seguem comportamento semelhante aos de treinamento, evidenciando que o modelo não está superajustando e mantém boa capacidade de generalização.

O K-Means ajustou os centróides eficientemente, com as mudanças nos centróides reduzindo gradualmente, chegando a valores muito pequenos nas últimas iterações, a partir da **5ª ou 6ª iteração**, a mudança nos centróides é muito pequena (ex.: 0.007178 na 8ª iteração e 0.010807 na 9ª), o que indica estabilidade dos clusters. Isso indicou que o modelo convergiu, com representações latentes bem-organizadas em torno dos agrupamentos, como observado na Figura 23. No geral, o modelo progrediu de forma satisfatória, com clusters definidos e reconstruções extremamente consistentes.

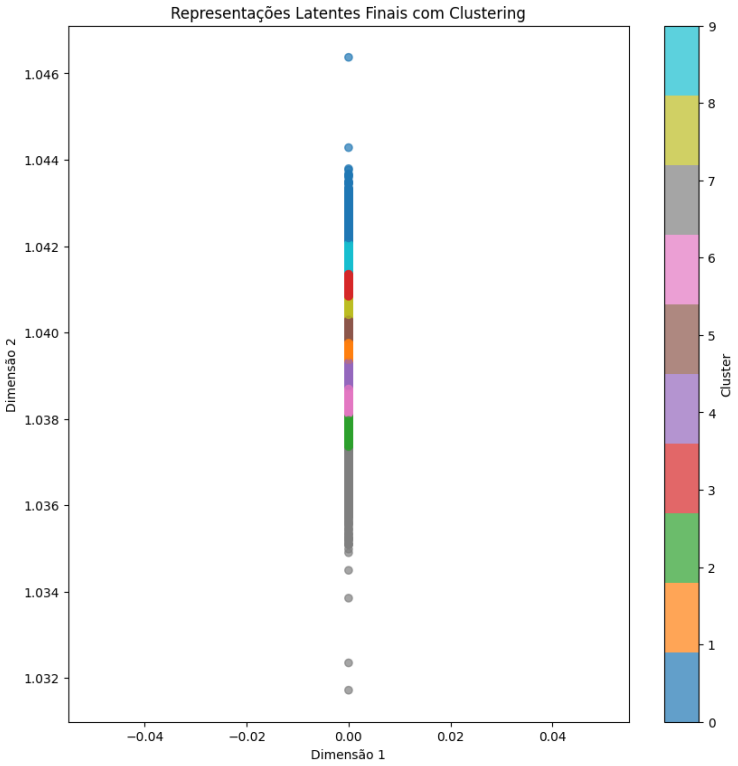

Figura 23 – Representações Latentes Finais com Cluser

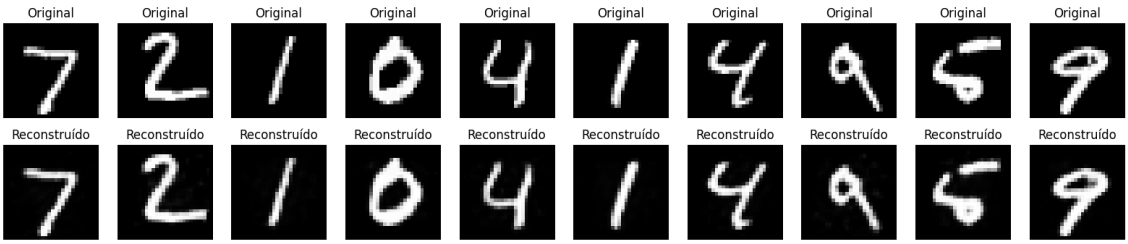

Figura 24 reconstruções no conjunto de teste

Resultados com latent_dim=32 (Disponivel no Colab):

Para avaliar inicialmente se o K-Means estava aplicando os clusters de forma adequada, foi criado um subconjunto reduzido dos dados, visualizado com t-SNE, demonstrado na Figura 26. Essa abordagem permitiu verificar a separação dos clusters no espaço latente. Caso os clusters não estivessem bem definidos ou apresentassem sobreposição significativa, isso indicaria a necessidade de ajustes no modelo, como observado em etapas anteriores.

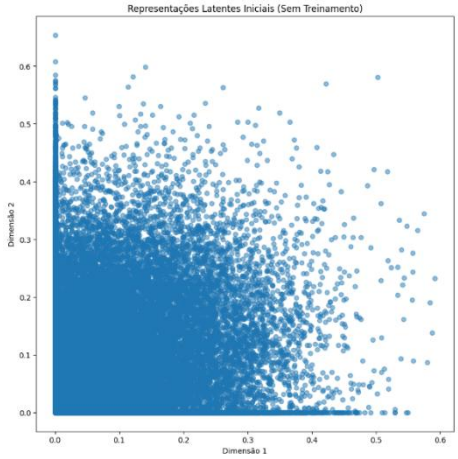

Figura 25 – Representações Latentes Iniciais (Sem Treinamento)

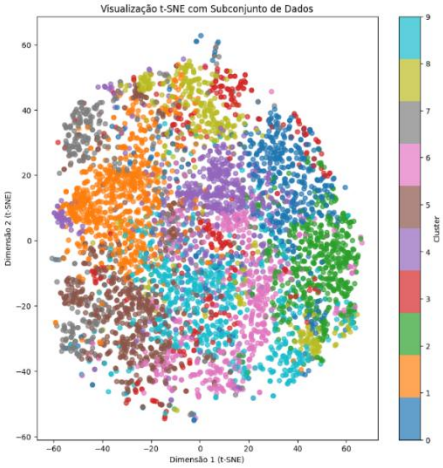

Figura 26 – Visualização t-SNE com Subconjunto de dados

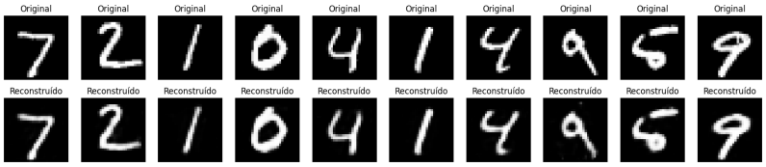

Figura 27 reconstruções no conjunto de teste

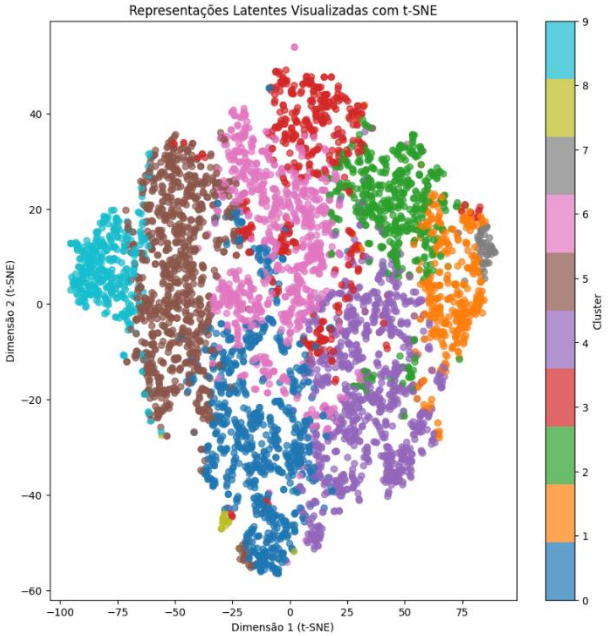

Figura 28: Representações Latentes Visualizadas com t-SNE

O treinamento do autoencoder com K-Means foi bem-sucedido, apresentando redução progressiva tanto no erro de reconstrução quanto no erro de clustering, indicando que o modelo aprendeu a reproduzir os dados de entrada com alta precisão, conforme demonstrado na Figura 27, e organizar representações latentes em clusters coesos. A convergência do K-Means foi comprovada pela diminuição das mudanças nos centróides, de 0.198727 na primeira iteração para 0.003919 na última.

Quanto a Figura 28 observa-se que os clusters apresentam uma separação relativamente bem definida, demonstrando que o modelo conseguiu organizar os dados no espaço latente de maneira coesa

3.1.3 Atividade C - Relatar variações de desempenho em relação

A abordagem experimental foi construída para explorar diferentes combinações de pesos na otimização conjunta do modelo, conforme os parâmetros α e β. A ideia principal foi explorar a interação entre a complexidade do espaço latente e o equilíbrio entre as duas tarefas concorrentes: reconstruir os dados de entrada e organizar representações latentes em clusters bem definidos.

Os resultados confirmaram as hipóteses iniciais sobre o impacto das dimensões latentes e dos pesos de perda:

Dimensões do Espaço Latente (latent_dims):

- As dimensões do espaço latente (2, 16, 64) representam diferentes níveis de compressão dos dados originais. Dimensões menores (e.g., 2) forçaram o modelo a representar os dados em um espaço altamente comprimido, facilitando a visualização e a análise de agrupamentos, mas possivelmente perdendo detalhes importantes. Por outro lado, dimensões maiores (e.g., 64) forneceram mais capacidade representacional, mas podem dificultar a separação clara dos clusters devido à alta dimensionalidade.

Combinações de Pesos (alpha_beta_combinations):

- As combinações (0.5, 0.5), (0.7, 0.3), e (0.3, 0.7) representam diferentes prioridades entre a reconstrução e o clustering:
  - **(0.5, 0.5)**: Balanceamento equitativo entre a qualidade da reconstrução e a organização dos clusters no espaço latente.
  - **(0.7, 0.3)**: Prioridade maior para a reconstrução, resultando em representações latentes menos coesas, mas reconstruções mais precisas.
  - **(0.3, 0.7)**: Foco maior no clustering, potencialmente sacrificando a qualidade das reconstruções para obter clusters mais coesos e bem definidos.

Dimensão Latente = 2:

Cenário 1:

Alpha: 0.5, Beta: 0.5
Perda Final: 0.0026
Perda de Validação: 0.0023
Score de Clustering: 0.0302

No cenário 1, os pesos equilibrados mostraram um bom compromisso entre reconstrução e clustering. O modelo alcançou uma Perda Final de 0.0026 e Perda de Validação de 0.0023, com um Score de Clustering de 0.0302, indicando organização mínima no espaço latente. Embora a separação entre clusters ainda seja limitada, o equilíbrio nos pesos permitiu uma otimização conjunta eficiente.

Cenário 2:

Alpha: 0.7, Beta: 0.3
Perda Final: 0.0117
Perda de Validação: 0.0111
Score de Clustering: 0.0000

No contexto do cenário 2, todos os dados foram atribuídos a um único cluster. Porém a definição dos parâmetros levou esse resultado, pois prioriza a reconstrução ($\alpha = 0.7$) e resulta em representações que são otimizadas principalmente para recuperar os dados originais. No entanto, isso pode ter prejudicado a formação de clusters bem definidos.

Dimensão Latente = 16:

Cenário 3:

Alpha: 0.5, Beta: 0.5
Perda Final: 0.0045
Perda de Validação: 0.0039
Score de Clustering: 0.0210

O cenário 3 apresentou uma Perda Final de 0.0045 e uma Perda de Validação de 0.0039, indicando que conseguiu equilibrar razoavelmente bem as tarefas de reconstrução e clustering. No entanto, o Score de Clustering de 0.0210 sugere que, embora os agrupamentos estejam mais definidos do que em configurações anteriores, ainda há espaço para melhorias na separação clara dos clusters

O **Cenário 1** demonstrou melhor desempenho geral em termos de perdas e qualidade de clustering, destacando a eficiência de um espaço latente compacto para tarefas de clustering. Contudo, o **Cenário 3** pode ser mais promissor para dados de maior complexidade, desde que parâmetros adicionais sejam ajustados para explorar todo o potencial da dimensionalidade aumentada.

Dimensão Latente = 64:

Cenário 4:

Alpha: 0.7, Beta: 0.3
Final Loss: 0.0124
Validation Loss: 0.0128
Clustering Score: 0.3532

O cenário 4 Com maior foco na reconstrução, a separação dos clusters foi a melhor entre todas as configurações (score de clustering 0.3532). Isso reflete como a dimensionalidade alta combinada com maior capacidade de reconstrução pode beneficiar o agrupamento.

Comparação Geral

1. **Dimensão Latente 2**:
    - *Melhor configuração:* **Alpha = 0.5, Beta = 0.5**, com menor perda e clustering moderado.
    - *Limitações:* Baixa capacidade de clustering nas demais configurações.
2. **Dimensão Latente 16**:
    - *Melhor configuração:* **Alpha = 0.3, Beta = 0.7**, com clustering score de 0.0234 e perdas equilibradas.
    - *Equilíbrio geral:* Configuração com **Alpha = 0.5, Beta = 0.5** oferece boa reconstrução e separação de clusters.
3. **Dimensão Latente 64**:
    - *Melhor configuração:* **Alpha = 0.7, Beta = 0.3**, com o maior clustering score (0.3532).
    - *Equilíbrio geral:* Configuração com **Alpha = 0.5, Beta = 0.5** ainda mantém um bom desempenho global.

Os resultados detalhados da execução, com análises e gráficos representativos, podem ser observados ao final da execução do algoritmo. Para aceder o código e os resultados basta entrar no Colab.

3.2 Conclusões das análises

A otimização conjunta entre o autoencoder (AE) e o algoritmo de clustering K-Means revelou-se uma abordagem eficaz para melhorar tanto a qualidade das representações latentes quanto a coerência dos clusters formados. Ao iterar entre o treinamento do AE e a atualização dos centróides do K-Means, as representações latentes foram ajustadas progressivamente para refletir de forma mais precisa a estrutura intrínseca dos dados, enquanto o K-Means refinou a distribuição e organização dos clusters. Esse processo iterativo foi guiado por uma função de perda personalizada que equilibrava os objetivos de reconstrução e agrupamento, assegurando que os dados fossem organizados em um espaço latente coeso, consistente e interpretável.

Além disso, análises gerais mostraram que essa abordagem conjunta não apenas melhorou a estrutura dos dados em termos de clustering, mas também permitiu insights mais profundos sobre as relações latentes entre as amostras. A integração entre o AE e o K-Means destacou padrões mais claros e significativos nos dados, facilitando interpretações mais robustas e a extração de informações úteis.

No entanto, desafios consideráveis foram encontrados, principalmente em relação ao tempo e ao custo computacional, intensificados pelas limitações do ambiente Colab Free. Cada ciclo de treinamento, combinado com o recalculo dos centróides do K-Means e os ajustes nas representações latentes, resultou em um aumento substancial no tempo de execução. Esses fatores tornaram o processo bastante demorado, exigindo vários dias e múltiplas sessões para a realização dos testes.

Capítulo

4

Conclusões

A aplicação da Deep Clustering Network (DCN), integrando autoencoders e o algoritmo K-Means, demonstrou ser uma metodologia eficaz para aprendizado não supervisionado, evidenciando a sinergia entre a capacidade de reconstrução do autoencoder e a organização dos dados pelo K-Means. O modelo conseguiu alcançar representações latentes robustas e clusters bem definidos, mesmo em cenários desafiadores como o conjunto de dados MNIST, destacando a eficácia da otimização conjunta.

A abordagem iterativa, que alternou entre o ajuste das representações latentes e a atualização dos centróides do K-Means, permitiu melhorar progressivamente a qualidade das representações no espaço comprimido e a coerência dos clusters. O balanceamento entre as perdas de reconstrução e de clustering, controlado pelos hiperparâmetros α e β, revelou-se crucial para alinhar os objetivos complementares de preservar características relevantes dos dados e formar agrupamentos interpretáveis.

Nos experimentos realizados, a configuração do espaço latente desempenhou um papel crucial no desempenho do modelo. A escolha inicial de uma dimensão latente reduzida, como latent_dim=2, mostrou-se particularmente útil para a visualização direta do espaço latente, permitindo uma interpretação clara e intuitiva da estrutura dos clusters. Contudo, essa simplicidade teve um custo: a qualidade das reconstruções foi visivelmente comprometida.

Por outro lado, configurações mais expressivas, como latent_dim=16, proporcionaram um equilíbrio superior entre a capacidade de reconstrução e a formação de clusters. Esses resultados foram especialmente evidentes quando o treinamento foi estendido para 50 épocas, demonstrando uma convergência mais estável e a captura mais detalhada das características intrínsecas dos dados.

O aprimoramento do modelo ao longo do processo foi significativamente impulsionado por uma série de ajustes. A adaptação dos códigos, a calibração cuidadosa dos pesos da função de perda e a exploração de diferentes dimensões latentes revelaram-se fundamentais para alcançar uma maior estabilidade e um desempenho otimizado. Essas modificações, realizadas de maneira iterativa e guiadas por análises, resultaram em um modelo mais robusto e eficiente.

Além disso, a análise iterativa dos clusters e a repetição do processo até a estabilização das atribuições de clusters entre épocas consecutivas demonstraram ser fundamentais para garantir a

convergência do modelo. O uso do K-Means++, que assegurou uma inicialização eficiente dos centróides, também contribuiu significativamente para a qualidade dos resultados.

Entretanto, o projeto apresentou desafios práticos consideráveis. O tempo e o custo computacional, agravados pelas limitações da plataforma Colab Free, representaram obstáculos que exigiram planeamento. A necessidade de recalcular os centróides a cada iteração e o treinamento prolongado do modelo resultaram em tempos de execução substanciais, dificultando a execução de testes mais amplos e iterativos.

Por fim, este estudo confirmou que a otimização conjunta entre autoencoders e K-Means é uma estratégia promissora para aprendizado não supervisionado. O DCN mostrou-se eficaz em capturar representações latentes úteis e formar agrupamentos coesos, reforçando seu potencial para aplicações futuras em análise de dados não rotulados e organização de dados complexos. Apesar das dificuldades computacionais, os resultados obtidos destacam a relevância do DCN como uma ferramenta poderosa em aprendizado profundo e agrupamento.