Classificação de dados gerados por aprendizado não supervisionado com Aprendizado Máquina Extremo

Eduardo Fonseca Rabelo[1] **and Odemir Martinez Bruno**[2]

[1]*Instituto de Física de São Carlos*
[2]*Universidade de São Paulo*

Este manuscrito foi escrito em 24 de Junho, 2024

Abstract

Neste trabalho, investigamos a capacidade de uma técnica de aprendizado não supervisionado para comprimir e descomprimir imagens utilizando Autoencoders Profundos e Convolucionais. A performance do processo de descompressão é avaliada empregando um classificador Extreme Learning Machine (ELM). Nosso estudo foca em verificar a eficiência e precisão da reconstrução de imagens e nos resultados subsequentes de classificação, demonstrando o potencial dos autoencoders na compressão eficaz de imagens e a robustez do ELM na avaliação da qualidade das imagens descomprimidas. Especificamente, o Autoencoder Profundo atingiu uma acurácia de 97.65%, enquanto o Autoencoder Convolucional obteve uma acurácia de 97, 05%.

Keywords: *Visão Computacional, Redes Neurais, CNN, Autoencoder, ELM*

1. Introdução

A busca pela redução de tempo de programação impulsionou a substituição do trabalho manual pelo computacional. Com isso, técnicas de aprendizado máquina vem sido amplamente buscadas para fins de customização de produtos, solução de problemas e até aprimoramentos em processos de automação.

Por isso, diversas técnicas de Aprendizado máquina vem sido desenvolvidas visando ampliar a qualidade de vida até chegar em níveis altíssimos de rendimento e performance ainda oferecendo um tempo relativamente pequeno de processamento. Uma das grandes áreas que rodeiam essas técnicas possibilitaram e aumentaram a performance em análise de imagem por meio de aprendizado profundo que permite compreender e replicar padrões ao nível de reconstruir imagens a partir desses parâmetros.

Com o aumento de complexidade de modelos computacionais a necessidade de aprimoramento de Hardware é inevitável. Por isso, algumas técnicas necessitam de um tempo de processamento menor, visando uma melhoria de desempenho em relação ao custo computacional, i.e., pode oferecer resultados com menor desempenho ou eficiência, porém com um tempo de resposta muito menor como no caso do *Extreme Learning Machine* discutido em seções posteriores.

2. Aprendizado profundo e CNN

Os modelos de classificação de imagens são problemas de aprendizado supervisionado, por isso trabalhamos com dados rotulados e um banco de dados bem estruturado e de alto volume para garantir um bom aprendizado. A área de classificação de imagens pode ser um pouco traiçoeira em relação aos dados, normalmente é bem comum de trabalhar com dados que apresentem uma boa flutuação estatística considerando os diversos fatores externos que podem influenciar na classificação como o corte da imagem, foco, número de pixels, entre outras coisas.

O método utilizado para classificação de imagens é chamado de *Convolucional Neural Networks* (CNN) e possibilita extrair uma quantidade de informações muito grandes de imagens. Utilizando esse método ao invés de classificarmos uma imagem previamente para determinarmos algumas características como formato ou textura, o método CNN utiliza da imagem sem pré processamento e consegue aprender os features de forma autônoma. Os modelos de CNN recebem um mapa tridimensional de features, correspondendo a largura, comprimento e o último relacionando as cores da imagem. A partir desse mapa é executada um conjunto de operações.

2.1. Aprendizado não supervisionado

As técnicas de aprendizado máquina dependem diretamente do tipo de dados que estamos trabalhando, mais especificamente com o rigor de supervisão utilizado durante o treinamento. Em métodos mais comuns como no caso aprendizado supervisionado, a arquitetura do modelo aprende com base nos rótulos utilizados durante o treinamento e os dados não rotulados são utilizados para testar o modelo em termos de precisão e acurácia. Dentre os métodos de treinamento podemos trabalhar com alguns métodos de aprendizado: O aprendizado supervisionado, aprendizado não supervisionado, semi-supervisionado e aprendizado por reforço [3]. Neste trabalho estamos mais interessados nos métodos não supervisionados de aprendizado.

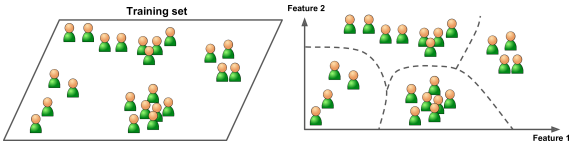

Figure 1. Aprendizado não supervisionado, *clustering* [3].

No aprendizado não supervisionado, os dados de treinamento não são rotulados, e por isso possui um caráter mais auto didata. Um dos métodos mais conhecidos e utilizados é o método de *clustering* que busca principalmente identificar similaridades em grupos de informação e separa-los de acordo com essa proximidade observada como exemplificado na Figura 1. Da mesma forma, os *Autoencoders* também baseia-se no aprendizado não supervisionado, com o foco principal em aprender uma representação compacta dos dados de entrada.

2.2. Autoenconder

Autoenconders são métodos não supervisionados de trabalharmos com redes neurais sem muito parâmetros. A ideia principal do método é promover uma abordagem simples para extrair os *features* do modelo removendo a redundância[4]. Por isso, sua origem está relacionada com a redução de dimensionalidade assim como o método de compressão de imagens JPEG é um método de se trabalhar com essas redundâncias. Esse processo é rodeado pela transformação chamada *Discrete Cosine Transformation* (DCT) [1] que implica em imagens que a maioria das posições são ocupadas por vários zeros e

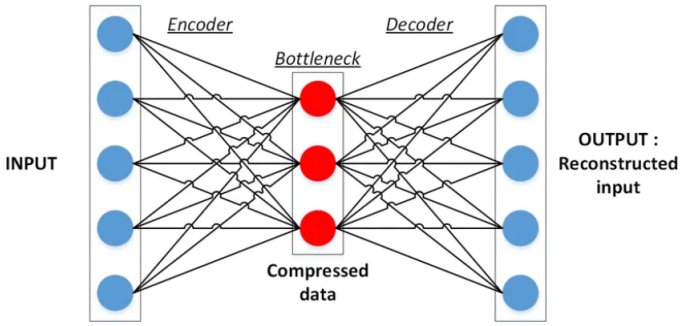

Figure 2. Arquitetura do Autoencoder [8].

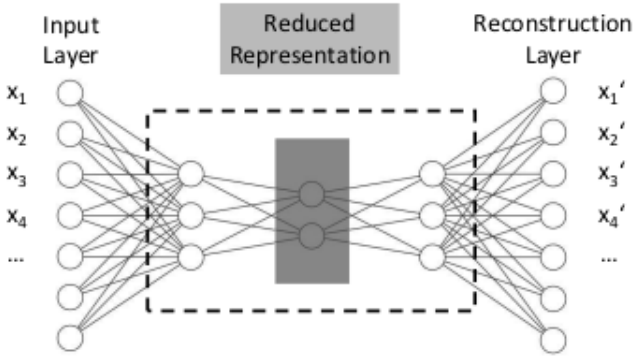

Figure 3. Arquitetura representativa de Autoencoders profundos com três *Hidden Layers* [4].

depois ocupadas por números inteiros. Nesse processo, infelizmente ainda temos a perda de informação uma vez que na reconstrução dessa informação não conseguimos construí-lo de forma completa [8].

Analogamente, o método de *Autoencoding* visa fazer a compressão de dados e depois reconstruir através dos parâmetros aprendidos e por isso, Gallinari & LeCun propuseram a ideia de trabalhar com autoencoders como eliminadores de ruído porém hoje em dia também são amplamente utilizados como geradores de informação [2]. A arquitetura desse método (Figura 2 consiste em tomar a camada de entrada e a camada de saída com a mesma dimensão passando as informações por um codificador e um decodificador.

$$\mathbf{h} = g(\mathbf{x}) \quad \Longrightarrow \quad \mathbf{x'} = f(\mathbf{h})$$

No qual **h** é o codificador e **f** é o decodificador. Esses treinos acontecem visando minimizar o erro no processo de reconstrução usando funções de erro adequada onde trabalhamos com

$$\mathcal{L} = (\mathbf{x}, f(g(\mathbf{x})) = \mathcal{L}(\mathbf{x}, \mathbf{x'}) \tag{1}$$

podendo ser por erro quadrático médio em um tamanho N-dimensional:

$$\mathcal{L}_{MSE}(\mathbf{x}, \mathbf{x'}) = ||\mathbf{x} - \mathbf{x'}||^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - x_i'^2) \tag{2}$$

ou por cross-entropy:

$$\mathcal{L}_{CE}(\mathbf{x}, \mathbf{x'}) = \frac{1}{N}\sum_{i=1}^{N}[-x_i \log x_i' - (1 - x_i)\log 1 - x_i'] \tag{3}$$

Essa arquitetura é amplamente utilizada considerando que na metade do processo é possível acessar camadas com representação reduzida que além de tudo pode ser usada de forma paralela ao modelo decodificador [4]. Nas diferentes arquiteturas possíveis para a criação de Autoencoders, ao longo dos anos esses modelos mais profundos passaram a apresentar uma boa margem de capacidade de representação, representando por uma hierarquia relacionada às camadas intermediárias, representado pela figura 3.

Para trabalhar com imagens de forma mais eficiente na área de *Machine Learning* é comum de se trabalhar com modelos convolucionais, entretanto no caso de Autoencoders Convolucionais pode ser mais vantajoso trabalhar com imagens menores para minimizar o erro na reprodução das imagens. Normalmente, a arquitetura desses modelos são baseados em algumas camadas convolucionais e camadas de "*pooling*" como representado pela figura 4.

Em contrapartida, os modelos convolucionais habituais diminuem proporcionalmente a dimensão da imagem de acordo com a profundidade da arquitetura devido ao número de camadas. Isso devo acontecer principalmente na etapa de codificação, na etapa de decodificação é necessário trabalhar com camadas que atuam de forma contrária na imagem trabalhada, nesse caso são usados camadas como "Convolutional Transpose 2D" ou "Up Sampling 2D". O UpSampling é usado para aumentar a resolução espacial das características extraídas pela rede. Ele "expande" a dimensão espacial dos dados, essencialmente revertendo o processo de "downsampling" realizado pelo codificador através de operações como convoluções com stride maior que um ou pooling [4].

2.3. Extreme Learning Machine (ELM)

Modelos de aprendizado profundo para funcionar demandam de um alto custo computacional, como CPUs e GPUs para otimização de processos. Por isso, ao longo dos últimos anos muitos cientistas vem estudado métodos para substituir os convencionais (*backward-forward algorithm*). Os modelos de *Extreme Learning Machine* são conhecidos por serem uma rede *feed-forward* simples com apenas uma *hidden layer* e por evitarem o treinamento da camada de entrada de dados, uma vez que ela é treinada pelo método chamado matriz pseudo-inversa [5].

Modelos ELM, na maioria das vezes seguem uma estrutura básica. A matriz de *input* tem dimensão M e comprimento N

$$X = [x_1, x_2, \ldots x_N], \quad x \in \mathbb{R} \tag{4}$$

e a matriz de saída pode possuir dimensões extras D com respeito a quantidade de classes no conjunto de dados.

$$Y = \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix} = \begin{bmatrix} y_{11} & \cdots & y_{1D} \\ \vdots & \ddots & \vdots \\ y_{N1} & \cdots & y_{ND} \end{bmatrix}$$

A matriz H é fundamental e representa a saída da camada oculta. Inicialmente, os dados de entrada X são processados pela multiplicação com os pesos w e a adição dos biases b. Esta operação resulta em uma matriz intermediária, H_{in}, que serve como a entrada para a camada oculta. A matriz H é construída a partir de uma função de ativação, podendo ser chamada de matriz de função ativa. Para ser construída é definida uma função h que recebe os valores ativos,

$$h(x) = [h_1, h_2, \ldots, h_L], \quad h(x) = g(x, w, c) \tag{6}$$

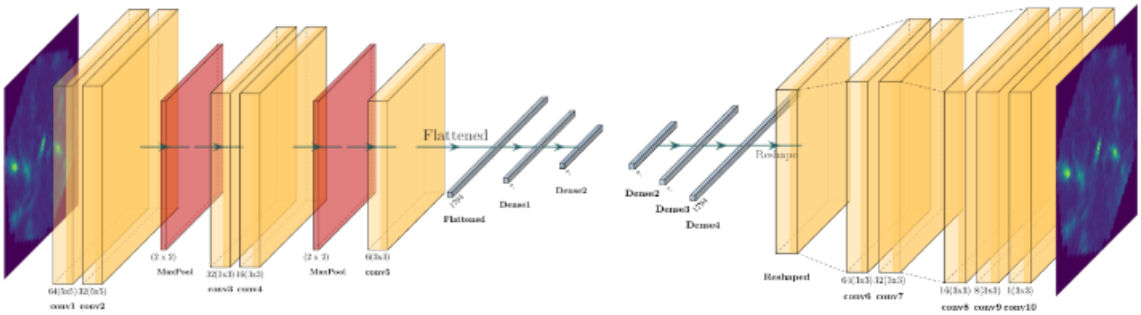

Figure 4. Arquitetura de um modelo de Autoencoder Convolucional utilizado para redução de ruídos em imagens geradas por telescópios de detecção de radiação Cherenvok no Cherenkov Telescope Array (CTA) [9].

$$g(x, w, c) = \frac{1}{1 + \exp^{-(wx+c)}} \tag{7}$$

a qual $g(x, w, c)$ é a função Sigmoid (Figura 5) esboçada abaixo.

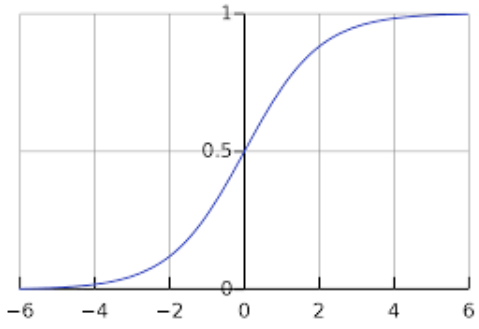

Figure 5. Função Sigmoid: $g(x, w, c)$.

Finalmente, uma matriz de pesos w e a matriz de coeficientes c.

$$w = [w_1, w_2, \dots, w_L] \tag{8}$$

$$c = [c_1, c_2, \dots, c_L] \tag{9}$$

A matriz H, é portanto construída da forma

$$H = \begin{bmatrix} h(x_1) \\ \vdots \\ h(x_N) \end{bmatrix} = \begin{bmatrix} h_1(x_1) & \cdots & h_L(x_1) \\ \vdots & \ddots & \vdots \\ h_1(x_N) & \cdots & h_L(x_N) \end{bmatrix}$$

$$H = \begin{bmatrix} g(w_1 \cdot x_1 + c_1) & \cdots & g(w_L \cdot x_1 + c_L) \\ \vdots & \ddots & \vdots \\ g(w_1 \cdot x_N + c_1) & \cdots & g(w_L \cdot x_N + c_L) \end{bmatrix}_{N \times L}$$

nesse caso, sendo L o número de neurônios na *hidden layer*. Nesse contexto, é importante definir a matriz β porque ela contém os pesos que permitem à rede transformar as ativações da camada oculta nas saídas preditas. Uma vez que

$$\beta = [\beta_1, \beta_2, \dots, \beta_L]^T \tag{10}$$

O próximo passo, portanto é encontrar β de forma que $H\beta$ se aproxime dos valores de saída Y, por isso chegamos no problema a ser resolvido utilizando o método da pseudo-inversa [7].

$$\beta = H^{\dagger} Y \tag{11}$$

e adicionando um fator de regularização no modelo, obtemos finalmente a matriz β da forma:

$$\beta = (H^T H + \frac{I}{C})^{-1} H^T Y \tag{12}$$

3. Métodos computacionais e Estrutura do Modelo

Nesse capítulo, falaremos dos métodos computacionais utilizados que favoreceram a criação de um classificador de imagens reconstruídas utilizando modelos de Autoencoders por meio de ELM. O foco desse projeto foi inicialmente entender e aplicar de forma simples e separada a técnicas de aprendizado não supervisionado discutido anteriormente, os métodos de classificação por modelos "*FeedForward*" e por fim, avaliar métodos de classificação de imagens geradas por diferentes arquiteturas, a primeira por métodos profundos e a segunda por métodos convolucionais.

3.1. MNIST e pré-processamento de dados

Para trabalharmos com redes neurais devemos principalmente trabalhar com segmentações nos bancos de dados para facilitar a etapa de treinamento e validação. Os dados que trabalharemos aqui é segmentado com 60000 imagens no banco de treinamento e outras 10000 para validação. O Data-base utilizado é chamado Instituto Nacional de Padrões e Tecnologia Modificado (MNIST), e é um grande banco de dados de dígitos manuscritos, comumente usado para treinar vários sistemas de processamento de imagens. O banco de dados também é amplamente utilizado para treinamento e teste no campo do aprendizado de máquina [6]. O MNIST fornece imagens de dimensões 28 × 28 pixels, o que oferece uma boa resolução para o escopo desse projeto indicado (Figura 6).

Figure 6. Samples aleatórios do banco de dados MNIST.

De início estamos trabalhando com o TensorFlow, ferramenta bastante utilizada para trabalhos de visão computacional utilizando dados estruturados. Portanto, é importante passar os dados por uma etapa de pré-processamento antes de começarmos a fazer implementações diretas em diferentes arquiteturas.

Para fins de qualidade de resultados, então é importante trabalhar como os dados normalizados (escala de cores de 0 a 1), além de mudar o formato das imagens, que por padrão são oferecidos em níveis de cinza no formato (28,28), uma vez que precisamos trabalhar com pelo menos um canal de cor resultando em uma mudança de formato para (28,28,1).

```
1 # Importa o dos features/labels de treino e
    features/labels de teste do MNIST
```

```
(x_train, y_train), (x_test, y_test) = tf.keras.
    datasets.mnist.load_data()

# Normalizar os dados
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0

# Adicionar uma dimens o de canal
x_train = np.expand_dims(x_train, axis=-1)
x_test = np.expand_dims(x_test, axis=-1)

print(f'x_train shape: {x_train.shape}')
print(f'x_test shape: {x_test.shape}')
```

Code 1. Pŕe-processamento básico de imagens MNIST.

3.2. Autoencoder

Nessa etapa do projeto, a ideia principal foi compreender como diferentes arquiteturas poderiam influenciar na reconstrução do conjunto trabalhado. Por isso, foram criadas duas estruturas, uma profunda e outra convolucional.

3.2.1. Autoencoder Profundo

O autoencoder profundo foi criado um modelo sequencial utilizando o TensorFlow, e como discutido anteriormente na seção 2.2, duas arquiteturas complementares foram utilizadas, a primeira criando um codificador, a segunda criando o decodificador e a conexão de ambas resulta no objetivo final dessa seção.

```
# Encoder
input_layer = layers.Input(shape=x_train_shape)
flattened = layers.Flatten()(input_layer)
hidden = layers.Dense(hidden_size, activation='
    relu')(flattened)
latent = layers.Dense(latent_size, activation='
    relu')(hidden)
encoder = Model(inputs=input_layer, outputs=
    latent, name='encoder')
encoder.summary()

# Decoder
input_layer_decoder = layers.Input(shape=(
    latent_size,))
upsampled = layers.Dense(hidden_size, activation
    ='relu')(input_layer_decoder)
upsampled = layers.Dense(encoder.layers[1].
    output_shape[-1], activation='relu')(
    upsampled)
constructed = layers.Reshape(x_train_shape)(
    upsampled)
decoder = Model(inputs=input_layer_decoder,
    outputs=constructed, name='decoder')
decoder.summary()

# Autoencoder (combina  o do encoder e decoder
    )
autoencoder_input = layers.Input(shape=
    x_train_shape)
encoded_img = encoder(autoencoder_input)
decoded_img = decoder(encoded_img)
autoencoder = Model(inputs=autoencoder_input,
    outputs=decoded_img, name='autoencoder')
autoencoder.summary()
```

Code 2. Códigos em TensorFlow do modelo profundo.

O codificador foi criado em poucas camadas, na primeira camada os dados são comprimidos por uma camada Flatten(), transformando a imagem em um vetor de tamanho 784, que contém todas as informações relacionadas às imagens de entrada, a camada seguinte conta com 100 neurônios, e a camada latente, que é a camada promove a compactação da informação sobre as imagens que estão sendo analisadas e as informações compactadas em 20 neurônios. O codificador pode ser facilmente verificado visualmente quando representado por um fluxograma como abaixo:

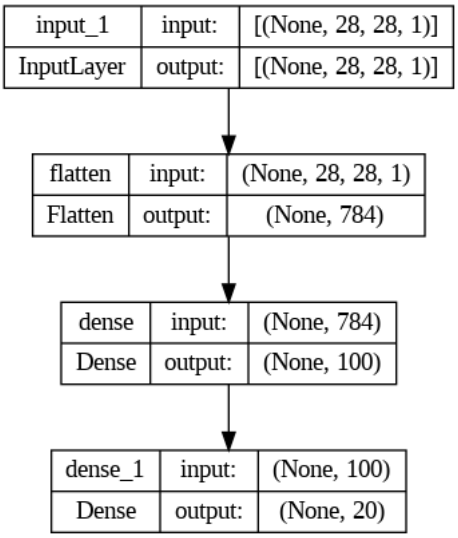

Figure 7. Encoder: estrutura criada para o codificador do modelo profundo.

De forma análoga, o codificador foi criado fazendo o mesmo caminho em direção contrária, no qual os 20 neurônios passam informação para 100 neurônios e finalmente passam por uma camada que ao contrário do *Flatten()* utilizado tem o papel de reordenar as informações novamente no formato inicial de 28×28 pixels. Finalmente, somadas as arquiteturas dos codificadores e decodificadores obtemos o Autoencoder como exemplificado visualmente pela figura **??**.

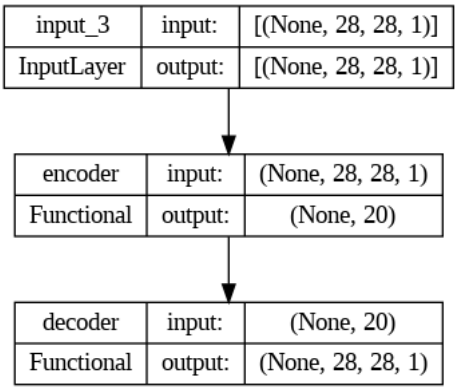

Figure 8. Arquitetura do Autoencoder criado, aplicando a camada de saída do Encoder como camada de entrada do Decoder.

3.2.2. Autoencoder Convolucional

Assim como redes neurais convolucionais mais simples, ainda devemos seguir uma estruturação do modelo para extrair os *feature maps* importantes no processo de aprendizado. As camadas convolucionais são interessantes para extrairmos informações relevantes sobre o conteúdo da imagem, em seguida é aplicada a camada *Pooling* que possui a função de diminuir a imagem e por isso diminuir sua complexidade permitindo aprender novos tipos de parâmetros.

Nesse modelo, para a criação do *Encoder* foram utilizadas duas sequências de convolução e *Pooling* com 32 e 64 filtros respectivamente, finalizando na camada latente com uma convolução bidimensional de 128 filtros e em dimensão 7×7. A construção do *Decoder* possui a mesma fundamentação porém visando reconstruir o formato original da imagem, portanto substituindo as camadas de *Pooling* por camadas do tipo *UpSampling()* com a mesma quantidade de filtros utilizadas

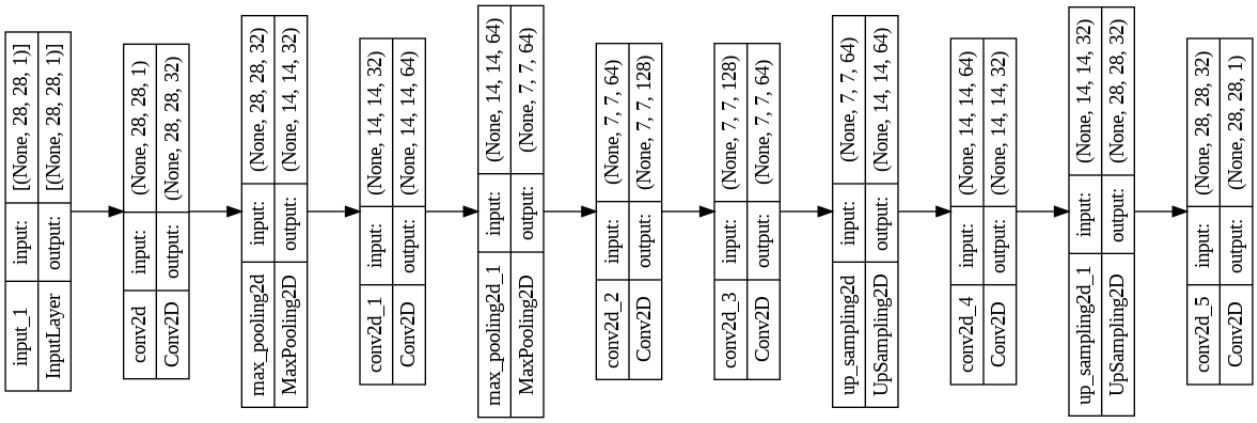

Figure 9. Arquitetura do Autoencoder Convolucional. Encoder + Decoder.

anteriormente. A arquitetura final já conectada pelos blocos de codificação e decodificação podem ser avaliadas visualmente pela Figura 9.

3.3. ELM

Um modelo ELM no geral é bem simples de ser construído quando auxiliado de algumas bibliotecas em python. Como estamos trabalhando como uma única *dense layer* através da reconstrução de imagens foi usado o seguinte bloco de códigos:

```
# One-hot encoding das labels
encoder = OneHotEncoder(sparse=False)
y_train_onehot = encoder.fit_transform(y_train.
    reshape(-1, 1))
y_test_onehot = encoder.transform(y_test.reshape
    (-1, 1))

```

Code 3. Preparação de vetor OneHot.

O código prepara as *labels* de treinamento e teste para serem usadas em modelos de aprendizado de máquina que requerem dados numéricos. O *one-hot encoding* converte as *labels* categóricas em uma representação numérica, onde cada categoria é representada por um vetor binário. Isso permite que o modelo aprenda a relação entre as *features* de entrada e as diferentes classes. Em seguida, devemos entender qual arquitetura utilizaremos para montar o classificador ELM. Nesse processo foi usada apenas uma *hidden layer* com 1000 neurônios para classificação, a dimnesão do *input* está diretamente ligada à dimensão da imagem do MNIST.

```
# Par metros do modelo ELM
input_size = x_train.shape[1]
hidden_size = 1000  # n mero de neur nios na
    camada oculta
output_size = y_train_onehot.shape[1]

```

Code 4. Parâmetros da arquiteura.

Em ELM, temos o processo de seleção aleatória dos pesos das conexões entre a camada de entrada e a camada oculta da rede neural. A ideia principal por trás da aleatoriedade na inicialização dos pesos em um modelo ELM é que a camada de entrada e a função de ativação da camada oculta são tratadas como uma "caixa preta" que mapeia diretamente os dados de entrada para uma representação na camada oculta. Posteriormente, os pesos entre a camada oculta e a camada de saída são determinados de forma analítica, utilizando a pseudo-inversa da matriz de ativação da camada oculta.

```
# Inicializar pesos da camada oculta
    aleatoriamente
W = np.random.randn(input_size, hidden_size)
b = np.random.randn(hidden_size)

```

Code 5. Inicialização de pesos aleatórios.

Dentre as funções de ativação necessárias, podemos usar não apenas a função Sigmoid discutida e demonstrada na seção 2.3. Nesse modelo, utilizamos a função ReLU, e a partir dessa função foi possível construir a matriz H.

```
# Fun   o de ativa   o da camada oculta (ReLU)
def relu(x):
    return np.maximum(0, x)

# Calcular a sa da da camada oculta
H = relu(np.dot(x_train, W) + b)

```

Code 6. Construção da matriz H.

Finalmente, foi possível calcular os pesos na camada de saída utilizando uma função que aplica diretamente os procedimentos da pseudo-inversa em H. E por fim, podemos classificar as imagens a partir de previsões feitas sobre o conjunto de testes e tomar uma acurácia associada ao métod utilizado.

```
# Calcular os pesos da camada de sa da usando
    pseudoinversa
H_pinv = pinv(H)
beta = np.dot(H_pinv, y_train_onehot)

# Fazer previs es no conjunto de teste
H_test = relu(np.dot(x_test, W) + b)
y_pred = np.dot(H_test, beta)

# Converter as previs es para labels
y_pred_labels = np.argmax(y_pred, axis=1)

# Avaliar a acur cia
accuracy = np.mean(y_pred_labels == y_test)
print(f'Test accuracy: {accuracy * 100:.2f}%')

```

Code 7. Código de ajuste de pesos na camada de saída, previsão no conjunto de teste e avaliação da acurácia do calssificador.

Em resumo, o código calcula os pesos da camada de saída do ELM, usa esses pesos para fazer previsões nos dados de teste, converte as previsões em labels e, finalmente, avalia a acurácia do modelo comparando as previsões com as labels verdadeiras.

4. Resultados

Neste capítulo, serão discutidos os resultados obtidos avaliando as implementações em diferentes etapas e condições de sistema. Para verificar a eficiência dos sistemas criados é importante avaliar seu comportamento em diferentes regimes. A avaliação dos Autoencoders aconteceram variando as iterações de treinamento, reconstruindo as imagens e comparando comparando-as com as imagens originais com 5, 10 e 50 epochs, tanto no profundo quanto no convolucional, comparando as imagens reconstruídas com as imagens originais. No caso profundo, representando pela Figura 10, em uma análise superficial é notória como o aprendizado não é inteiramente afetado por pequenas variações da quantidade de iterações. Nesse caso, mesmo que muito pequeno, existe a diferença entre cada imagem reconstruída. O que mostra uma forma bastante eficiente ao trabalhar com autoencoders profundos, uma vez que ainda com um volume muito pequeno de epochs, torna-se possível aprender as características relevantes para a reconstrução de uma imagem. Nesse ponto de aprendizado, as imagens reconstruídas não são tão distantes entre si, entretanto ainda destoam bastante das imagens originais.

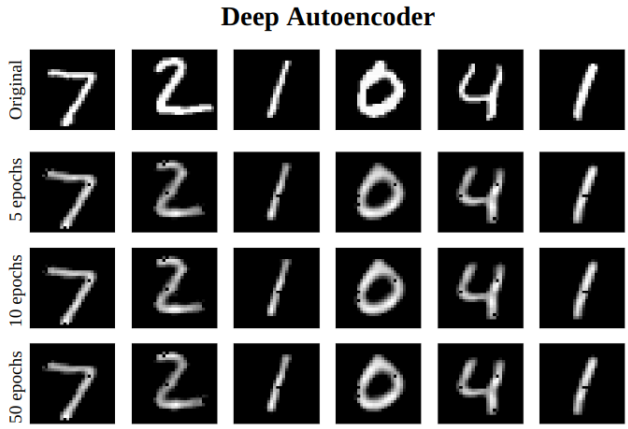

Figure 10. Comparação de imagens reconstruídas por um Autoencoder profundo em 5, 10 e 50 epochs de aprendizagem.

Evidenciado pela Figura 11, os Autoencoders Convolucionais apresentam o mesmo comportamento, logo não apresentam muitas diferenças quando variamos a quantidades de epochs no processo de aprendizado. Nesse caso foi avaliado a reconstrução do modelo para dados aleatórios dentro do data-base MNIST para ver se a reconstrução poderia falhar se tomássemos uma amostra um pouco maior. Obviamente, a quantidade de imagens reconstruídas não traz um enorme significado estatístico devido ao pequeno conjunto extraído, porém ainda é possível dizer que as reconstruções por métodos convolucionais são mais eficientes comparadas às realizadas no método profundo considerando a diferença ínfima entre os dados originais e os reconstruídos.

Além disso, com o interesse de verificar a possibilidade de *Overfit* também foi verificada a *Loss Function* da CNN. O *Overfit* ocorre quando o modelo sobrecarrega o aprendizado conforme os dados de treino, normalmente é resultado de uma alta complexidade do modelo, ou até mesmo da escassez de dados do conjunto de treinamento. No contexto de autoencoders, por estarmos trabalhando com aprendizado não supervisionado, pode parecer irrelevante avaliar o *Overfit* do modelo, entretanto como são usados para redução de dimensionalidade ou detecção de anomalias, um modelo muito complexo pode aprender a reproduzir exatamente os dados de treinamento, incluindo o ruído, em vez de aprender uma representação mais geral.

4.1. Classificação por ELM

O objetivo final do projeto é avaliar a performance de um classificador ELM ao classificar imagens do banco MNIST que foram

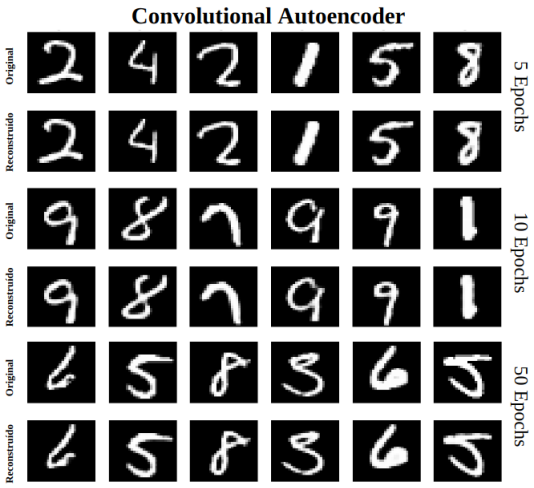

Figure 11. Comparação de imagens reconstruídas por um Autoencoder Convolucional em 5, 10 e 50 epochs de aprendizagem.

reconstruídas ao serem reconstruídos utilizando Codificadores e Decodificadores como forma de compressão e descompressão de uma imagem. Nesse caso, antes de classificar imagens reconstruídas, é importante verificar como o *Extreme Learning Machine* performa sobre os dados originais. Utilizando a estrutura discutida na seção 2.3, foi possível avaliar a precisão do modelo ao classificar cada classe do Data-base. Nesse contexto, utilizando as imagens originais, os testes reportaram uma acurácia de 94.34%. Para verificar como esse modelo é preciso em relação a cada classe, foi criada uma matriz confusão que em sua diagonal principal deve se aproximar de um, indicando uma grande quantidade de Verdadeiros Positivos na classificação dos dados. No caso da Figura 12, é possível observar que os valores aproximam-se sempre ao longo das diagonais de uma unidade, indicando também um bom rendimento do classificador nos dados originais.

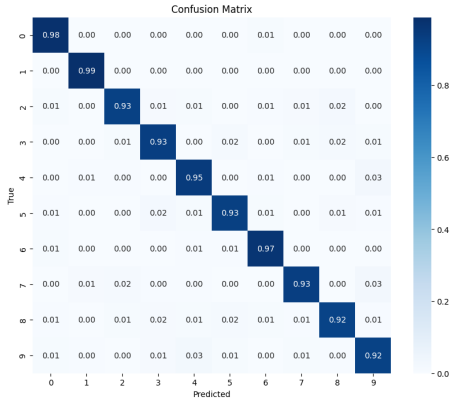

Figure 12. Matriz confusão da classificação por ELM dos dados originais do MNIST.

O mesmo procedimento foi tomado outras duas vezes visando avaliar a classificação em dados reconstruídos. No modelo de classificação de *Autoencoders* profundos, os resultados se mostraram com uma acurácia de classificação ainda maior em relação à classificação de dados originais, reportando nesse caso uma acurácia de 97.65% indicando uma diagonal principal ainda mais próxima de valores unitários como indicado pela 13. Por fim, o mesmo ocorreu no caso do classificador ELM de imagens reconstruídas por CNNs, reportando uma acurácia de 97.05%. É importante pontuar que para fins de comparação mais justas, todas as verficações foram realizadas na mesma seed de geração de pesos pseudoaleatórios na inicialização dos neurônios do ELM.

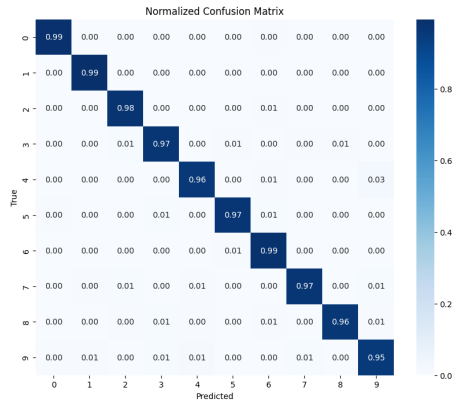

Figure 13. Matriz confusão da classificação por ELM dos dados reconstruídos do MNIST por um *Autoencoder* profundo.

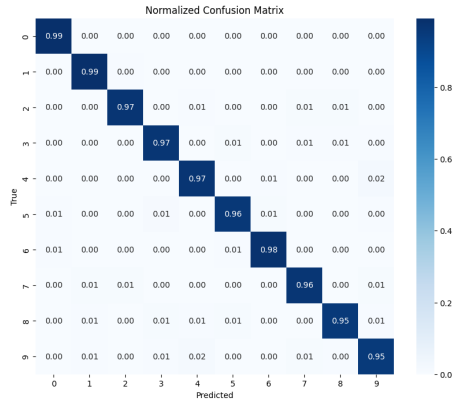

Figure 14. Matriz confusão da classificação por ELM dos dados reconstruídos do MNIST por um *Autoencoder* Convolucional.

5. Conclusões

No que diz respeito à outros métodos de classificação utilizados na área de Visão Computacional, o método ELM mostra-se bastante eficiente, ainda que não represente os melhores resultados dentre os classificadores. Diante de Redes Neurais Convolucionais bem simples para classificação de imagens, o ELM perde em questão de eficiência uma vez que apresenta uma acurácia de 94.34% que pode ser batido facilmente por várias redes com estruturação simples. Entretanto, mesmo que não apresente os melhores resultados, ainda se mostra uma ótima opção visto seu custo computacional muito baixo comparado a outros métodos.

No que tange os conceitos de *Autoencoders*, ambas as técnicas se mostraram eficientes para compressão de dados, uma vez que mostraram que a reconstrução de imagens, no caso de redes profundas, com apenas 20 neurônios consegue carregar informações relevantes o suficientes para reconstruí-las de forma satisfatória que ao passar por um classificador ELM, fornece informação suficiente para ser classificada com uma acurácia de 97.65%. O mesmo ocorre no caso de *Autoencoders* Convolucionais, uma vez que é capaz de trabalhar com uma imagem com dimensão 28×28 pixels e comprimi-la à uma dimensão de 7×7 pixels possibilitando acima disso uma reconstrução quase sem perda de resolução. A reconstrução se mostrou satisfatória ao ponto de atingir uma acurácia de 97.05% por meio do classificador ELM.

References

[1] D. A. Huffman, "The method of maximum entropy spectrum analysis of irregularly sampled data," *IEEE Transactions on Circuits and Systems*, vol. CAS-21, no. 6, pp. 652–654, 1974. DOI: 10.1109/T-C.1974.223784. [Online]. Available: https://doi.org/10.1109/T-C.1974.223784.

[2] P. Gallinari, Y. LeCun, S. Thiria, and F. Fogelman-Soulie, "Memoires associatives distribuees," in *Proceedings of COGNITIVA 87*, Paris, La Villette, 1987.

[3] A. Géron, *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*. O'Reilly Media, 2017, ISBN: 978-1492032649. [Online]. Available: https://www.oreilly.com/library/view/hands-on-machine-learning/9781492032632/.

[4] P. Pang and H. Finkel, *Deep Learning for Physics Research*. Springer, 2020, ISBN: 978-3030505076. [Online]. Available: https://www.springer.com/gp/book/9783030505076.

[5] P. Chomchit. "Extreme learning machine (elm) is the speed-up learning method for artificial neural network." Accessed: 2024-06-26. (2021), [Online]. Available: https://medium.com/@phiphatchomchit/extreme-learning-machine-elm-is-the-speed-up-learning-method-for-artificial-neural-network-98ab00dcf617.

[6] W. Contributors. "Banco de dados mnist." Acesso em: 25 jun. 2024. (2024), [Online]. Available: https://pt.wikipedia.org/wiki/Banco_de_dados_MNIST.

[7] W. contributors, *Moore–penrose inverse — Wikipedia, the free encyclopedia*, [Online; accessed 26-June-2024], 2024. [Online]. Available: https://en.wikipedia.org/wiki/Moore%E2%80%93Penrose_inverse.

[8] Analytics Vidhya, *Autoencoders with tensorflow*, https://medium.com/analytics-vidhya/autoencoders-with-tensorflow-2f0a7315d161, Acessado em 25 jun. 2024. [Online]. Available: https://medium.com/analytics-vidhya/autoencoders-with-tensorflow-2f0a7315d161.

[9] S. F. D. Arturo, "Design of an automatic classification system based in computational intelligence for the cta observatory," Thesis (Bachelor's Degree), Universidad de Granada, Facultad de Ciencias, 2020/2021.