UNIVERSIDADE DE CAXIAS DO SUL
ÁREA DO CONHECIMENTO DE CIÊNCIAS EXATAS E ENGENHARIAS

GUILHERME MENIN STEDILE

DETECÇÃO DE FALHAS EM SISTEMAS FOTOVOLTAICOS A PARTIR DA ANÁLISE DOS SINAIS DE GERAÇÃO

CAXIAS DO SUL

2023

GUILHERME MENIN STEDILE

DETECÇÃO DE FALHAS EM SISTEMAS FOTOVOLTAICOS A PARTIR DA ANÁLISE DOS SINAIS DE GERAÇÃO

Trabalho de Conclusão de Curso apresentado como requisito parcial à obtenção do título de Bacharel em Engenharia de Computação na Área do Conhecimento de Ciências Exatas e Engenharias da Universidade de Caxias do Sul.

Orientador: Prof. Dra. Carine Geltrudes Webber

CAXIAS DO SUL

2023

GUILHERME MENIN STEDILE

DETECÇÃO DE FALHAS EM SISTEMAS FOTOVOLTAICOS A PARTIR DA ANÁLISE DOS SINAIS DE GERAÇÃO

Trabalho de Conclusão de Curso apresentado como requisito parcial à obtenção do título de Bacharel em Engenharia de Computação na Área do Conhecimento de Ciências Exatas e Engenharias da Universidade de Caxias do Sul.

Aprovado(a) em 00/12/2023

BANCA EXAMINADORA

Prof. Dra. Carine Geltrudes Webber
Universidade de Caxias do Sul - UCS

Prof. Dra. Marilda Machado Spindola
Universidade de Caxias do Sul - UCS

Prof. Me. Marcos Eduardo Casa
Universidade de Caxias do Sul - UCS

Este trabalho é dedicado às pessoas que, na infância, desmontavam tudo por curiosidade e, agora adultas, projetam para construir.
(O Autor)

AGRADECIMENTOS

Primeiramente agradeço aos meus pais Ricardo Stedile e Fabiana Andréa Menin Stedile, por me proporcionarem a possibilidade de ter realizado esta graduação, me fornecendo todo apoio necessário desde a escolha do curso até em todas as dificuldades no decorrer do mesmo. Principalmente quando, ao recorrer ao meu pai, o mesmo dizia "eu não entendo nada na sua área mas..."e assim me aconselhava de forma excelente.

A seguir, agradeço a Caroline Bolzan, minha namorada, por seu apoio constante durante os períodos de redação deste trabalho acadêmico. Sua presença foi crucial tanto nos momentos de dedicação à escrita quanto nos momentos de lazer, proporcionando um equilíbrio essencial. Além disso, como estudante do terceiro semestre do curso de Letras, sua contribuição na revisão deste trabalho foi valiosa, fornecendo sugestões gramaticais visando a otimização da redação deste texto.

Posteriormente agradeço a professora Carine Geltrudes Webber, responsável não só pela orientação deste trabalho, como também pela sugestão do tema e, exímio conhecimento na área de AI. Também sou grato aos Professores Marcos Eduardo Casa e Marilda Machado Spindola por fazerem parte da banca avaliadora e proverem ajustes para melhorar a qualidade deste trabalho após avaliação.

Não posso deixar de citar todos os colegas de curso que passam pela minha vida acadêmica e, alguns até seguiram como grandes amigos fora dela. Em especial a todos com os quais tive a oportunidade de aprender e ensinar nos diversos trabalhos em grupos no decorrer da graduação.

“Tudo o que um sonho precisa para ser realizado é alguém que acredite que ele possa ser realizado.”

Roberto Shinyashiki

RESUMO

A produção de energia por meio de painéis fotovoltaicos é uma tecnologia amplamente conhecida e, recentemente, seu custo se reduziu a ponto de ser viável para usuários domésticos instalarem sistemas para uso pessoal em suas casas, sítios, microempresas e afins. No entanto, as instalações ainda apresentam um custo elevado. Por esse motivo, o monitoramento da geração de energia e a realização de manutenções são necessárias para garantir um funcionamento duradouro e otimizado do sistema. Diversos sistemas foram desenvolvidos na tentativa de monitorar o funcionamento e predizer falhas, utilizando várias técnicas e monitorando diversas variáveis. Neste trabalho, observa-se a tentativa de detectar e predizer falhas através de um software de IA baseado no uso de um *AutoEncoder*. Redes neurais são elementos complexos no desenvolvimento de software, e *AutoEncoders* se comportam de maneira específica e diferente se comparados a outros tipos de redes neurais. Portanto, foi apresentado um estudo sobre o funcionamento das redes neurais, como elas "aprendem"e como seus resultados são avaliados. Como o tema deste projeto é *AutoEncoders*, foi dada ênfase maior a eles, conceituando o que são, o que os diferencia das demais redes, como é sua arquitetura e seu funcionamento, além de avaliar artigos na área de detecção de falhas usando *AutoEncoders* em outros tipos de sinais. Adicionalmente, também é apresentado o funcionamento de um sistema de geração de energia fotovoltaico, sua instalação, manutenção, medição da quantidade de energia gerada e a qualidade dessa geração. A importância da manutenção preventiva e preditiva para maximizar a eficiência e a vida útil dos sistemas fotovoltaicos também foi discutida. O projeto do sistema de detecção foi iniciado com a obtenção de um *DataSet* contendo medições do nível de geração juntamente com diversas variáveis sobre o momento desta geração. A segunda etapa constituiu na criação dos conjuntos de treino e teste, bem como no desenvolvimento do *AutoEncoder* e na definição da métrica de avaliação de performance, sendo esta a divergência de Kullback-Leibler (KL). A etapa seguinte concentrou-se no treino do modelo com diferentes números de camadas e configurações. Por fim, as configurações e seus respectivos erros foram exibidos graficamente para facilitar a análise e comparação. Além disso, foi analisada a viabilidade de aplicação em larga escala e as possíveis melhorias futuras no modelo de *AutoEncoder*. Concluímos que os resultados foram satisfatórios ao se avaliar a divergência dos dados após a desconstrução e reconstrução pelo *AutoEncoder*, comparando-os com os dados originais do *dataset*, demonstrando a eficácia do método na detecção de falhas.

Palavras-chave: *AutoEncoders*. Fotovoltaico. Inteligência Artificial. Predição.

ABSTRACT

The production of energy through photovoltaic panels is a well-known technology, and recently its cost has reduced to the point where it is feasible for domestic users to install systems for personal use in their homes, smallholdings, micro-enterprises, and the like. However, installations still present a high cost. For this reason, monitoring energy generation and performing maintenance are necessary to ensure a long-lasting and optimised system operation. Various systems have been developed in an attempt to monitor performance and predict failures, using several techniques and monitoring various variables. This work observes the attempt to detect and predict failures through AI software based on the use of an *AutoEncoder*. Neural networks are complex elements in software development, and *AutoEncoders* behave in a specific and different manner compared to other types of neural networks. Therefore, a study on how neural networks function, how they "learn" and how their results are evaluated was presented. Since the theme of this project is *AutoEncoders*, greater emphasis was given to them, defining what they are, what differentiates them from other networks, their architecture and operation, as well as evaluating articles in the field of fault detection using *AutoEncoders* in other types of signals. Additionally, the operation of a photovoltaic energy generation system, its installation, maintenance, measurement of the amount of energy generated, and the quality of this generation are also presented. The importance of preventive and predictive maintenance to maximise the efficiency and lifespan of photovoltaic systems was also discussed. The detection system project began with obtaining a *DataSet* containing measurements of the generation level along with various variables about the moment of this generation. The second stage involved creating the training and testing sets, as well as developing the *AutoEncoder* and defining the performance evaluation metric, which is the Kullback-Leibler (KL) divergence. The next stage focused on training the model with different numbers of layers and configurations. Finally, the configurations and their respective errors were graphically displayed to facilitate analysis and comparison. Moreover, the feasibility of large-scale application and potential future improvements to the *AutoEncoder* model were analysed. We concluded that the results were satisfactory when evaluating the divergence of the data after the deconstruction and reconstruction by the *AutoEncoder*, comparing them with the original *dataset* data, demonstrating the method's effectiveness in fault detection.

Keywords: *AutoEncoders*, Artificial Intelligence, Photovoltaic, Prediction.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE QUADROS

LISTA DE ALGORITMOS

LISTA DE ABREVIATURAS E SIGLAS

elem.	Elemento
IA.	Inteligência Artificial
AE.	Autoencoders
VAE.	Variational AutoEncoder
DL.	Deep Learning em portugues Aprendizado Profundo
ANN.	Artificial Neural Network ou Rede Neural Artificial em português
FCN.	Fully Connected Network em português Rede Totalmente Conectada
DNN.	Deep Fully Connected Neural Network em português Rede Neural Profunda Totalmente Conectada
CNN.	Convolutional Neural Network em português Rede Neural Convolucional
CSAE.	Analyzes the use of convolutional sparse autoencoder em português autoencoder convolucional esparsamente conectado
GAN.	Generative Adversarial Networks em português Redes Adversárias Generativas em português
PC.	Computador Pessoal do inglês *Personal Computer*
TCC.	Trabalho de Conclusão de Curso
UCS.	Universidade de Caxias do Sul
TanH.	Tangente Hiperbólica
ReLU.	Rectified Linear Unit, em português Unidade Linear Retificada
KL.	Kullback-leibler
GPU.	Graphics Processing Unit em português Unidade de Processamento Gráfico

SUMÁRIO

1 INTRODUÇÃO

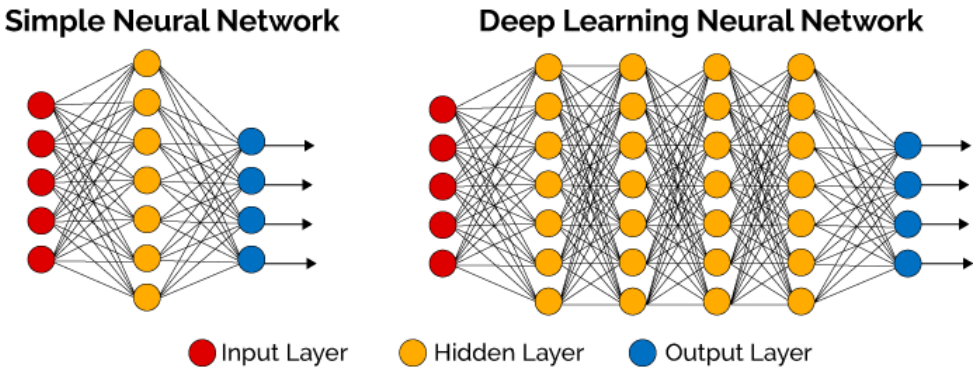
Inteligência artificial (IA) se refere a habilidade de um computador "pensar", isto é, ser capaz de executar tarefas atribuídas a uma inteligência reservada aos humanos, sendo estas exemplificadas por ser lógica, descobrir métodos para obtenção de um resultado e, por fim, aprender com fatos passados ou dados previamente imputados. Trata-se da arte de copiar a inteligência humana baseando-se em ciclos, assim, utilizando de métodos para que a mesma se adéque e obtenha a capacidade de controlar a informação (PADALLAN, 2022). Apesar disto, a IA não se compara a inteligência humana. A IA tem a capacidade apenas de simular a inteligência humana em alguns aspectos, tais como visual e espacial, movimentos corporais, interação com outras pessoas, linguística e principalmente lógico matemática, sendo esta possivelmente a única em que a IA possui um desempenho superior aos humanos (MASSARON, 2021).

Atualmente a arquitetura que mais se assemelha a um cérebro humano trata-se das Redes neurais artificiais (em inglês ANN)(PADALLAN, 2022). As redes neurais são compostas por camadas de neurônios artificiais em sequência. Os neurônios se conectam entre si, por meio de sinapses as quais são atribuídos pesos. Por meio do ajuste dos pesos para que na saída se obtenha o resultado esperado, a rede "aprende"e é capaz de, com certo nível de acurácia, projetar uma saída baseada nas entradas (MASSARON, 2021).

Evoluindo mais um passo, obtemos as redes neurais artificiais profundas, conhecidas pelo nome em inglês *deep learning*. Elas consistem em redes neurais artificiais compostas por mais de três camadas, sendo inclusas a camada de entrada e de saída nestes cálculos. Resumindo, tem-se um modelo *deep learning* quando a rede possuir mais de uma camada oculta (TAULLI, 2019).

Na figura 1, é demonstrado o comparativo de uma rede neural artificial considerada mais baixa ou comum, com uma rede neural que utiliza uma arquitetura de *deep learning* (DL), possuindo assim mais de uma camada oculta.

Figura 1 – Rede Neural Artificial X Rede de aprendizado profundo



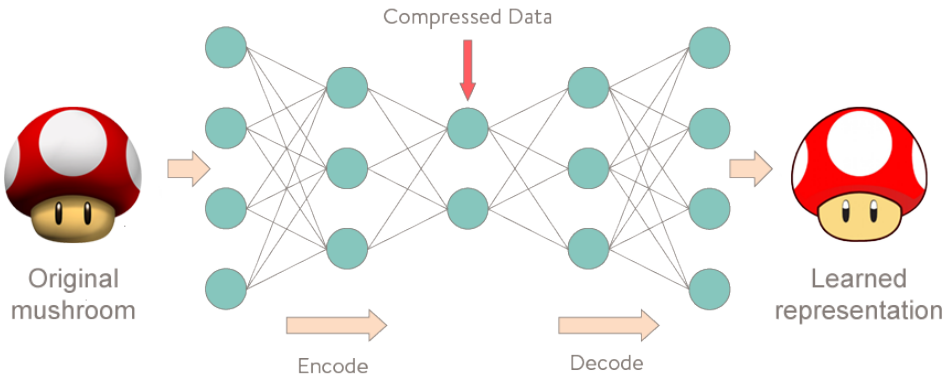
Fonte: Adaptado de www.deeplearningbook.com.br/o-que-sao-redes-neurais-artificiais-profundas/

Se tratando de IA, as redes podem ser classificadas quando ao aprendizado segundo duas categorias principais, aprendizado supervisionado e aprendizado não supervisionado. No primeiro, o algoritmo tem a classificação de cada tipo de dado, como se fossem categorias, por exemplo, temos uma base de dados com múltiplas fotos de carros sendo que, cada foto possui como identificador a marca do veiculo. Já no segundo método, o não supervisionado, os dados não possuem nenhum tipo de classificação, aqui são usados os métodos de DL na tentativa de obter padrões e classifica-los(TAULLI, 2019). Um dos métodos para tal são os AE, técnica que será utilizada ao longo deste trabalho.

Autoencoders são redes neurais artificias cujo objetivo primário é copiar a entrada para a saída, sendo que o mesmo é forçado a não copiar a entrada para a saída de forma 100% sendo assim, a rede é forçada a aprender as principais características, copiando apenas estas e as repassando para a saída, mantendo somente as informações que realmente dão característica aos dados (GOODFELLOW; BENGIO; COURVILLE, 2016).

Na figura 2, é ilustrado o conceito da compressão, estudo de características e descompressão de uma gravura, afim de representar a mesma utilizando apenas o que faz a mesma ser reconhecida como o que ela é, no caso um cogumelo do jogo Mario Bros.

Figura 2 – Exemplo prático de um *Autoencoder*



Fonte: Adaptado de https://www.deeplearningbook.com.br/introducao-aos-Autoencoders/

Variados são os tipos de *Autoencoders*, por exemplo: regularizadores de escassez, anti ruído, regularizadores, estocásticos e eliminadores de ruído (GOODFELLOW; BENGIO; COURVILLE, 2016). Estes podem ser utilizados para múltiplos problemas, como compressão de itens, partindo do princípio que as mesmas mantém a dimensionalidade e a magnitude de cada ponto no espaço latente. Também podemos usar para classificação e detecção de anomalias, utilizando um gabarito de avaliação. Podem ser usados também para a retirada de ruído de imagens ou sinais, bem como para colorir imagens em escala de cinza e podem até mesmo serem usados para gerar novas imagens a partir do nada(LANGR, 2019).

1.1 OBJETIVO GERAL E OBJETIVOS ESPECÍFICOS

O presente trabalho propõe um estudo exploratório sobre a aplicação de modelos baseados em *Autoencoders* para a detecção de falhas em sistemas geradores de energia. Mais especificamente sistemas fotovoltaicos por meio do sinal de geração proporcionado pelos mesmos.

Avaliação de modelos baseados em *Autoencoders* para tarefas de detecção de falhas a partir de dados obtidos através dos sinais de corrente e tensão medidos durante a geração de energia elétrica a partir de painéis fotovoltaicos.

Os objetivos específicos do trabalho são:

- Implementação de um *AutoEnconder* para o problema especificado;
- Efetuar a aquisição de um sinal de geração de energia emitido por um sistema fotovoltaico;
- Por meio de um *AutoEnconder* verificar se é possível antecipar falhas;
- Com o uso de múltiplas métricas, avaliar a performance do *AutoEnconder* em todos os pontos do processo;
- elaborar um modelo para detecção de falhas com base em *AutoEnconder*.

1.2 ORGANIZAÇÃO DO DOCUMENTO

O trabalho de conclusão está organizado em 4 capítulos e 13 subcapítulos e um dos subcapítulos ainda é subdividido em 5. No capítulo 1 temos a introdução. A introdução é responsável por introduzir a área de estudo do tema, a base que levou a criação dos *AutoEnconders* e uma breve citação dos tipos deste sistema de redes neurais. Os subtópicos desta seção são os objetivo geral e objetivos e específicos, onde consta o que este trabalho pretende alcançar e, a seção atual, composta pelo modo como este documento é organizado.

No capítulo 2, constam os conceitos e aplicações do uso de *AutoEnconders*. No corpo do capítulo, temos uma breve introdução sobre *AutoEnconders*, seguindo para o subcapítulo perceptron e a rede multicamadas, onde é explicado o funcionamento de um perceptron (neurônio artificial) e a organização destes perceptrons em rede, visando a criação de redes multicamadas. Em seguida temos o subcapítulo Algoritmo de *Backpropagation*, responsável por explicar e exemplificar como os pesos das redes multicamadas são atualizados segundo o *BackPropagation*. Em seguida, um subcapítulo intitulado conceituação de *AutoEnconders*, capítulo este responsável por explicar o que é um *AutoEnconders* e as principais tarefas realizadas por este tipo de rede neural. Posteriormente é escrito um subcapítulo com os detalhes de arquiteturas possíveis de se assumir para elaborar um *AutoEnconders*, sendo a arquitetura o arranjo dos perceptrons para criar o próprio *AutoEnconders*. Depois, o subcapítulo é responsável por indicar como um *AutoEnconders* funciona em detalhes para assim, facilitar o entendimento do mesmo como um todo. O penúltimo capítulo abre uma breve discussão sobre como os *AutoEnconders* podem ser utilizados na tentativa não só de detectar como prever quando falhas ocorrerão e, para finalizar o capítulo macro sobre *AutoEnconders*, é posto um capítulo com considerações finais.

O Capítulo 3, consiste na implementação, que indica como funciona o projeto no qual fora implementado o sistema composto por *AutoEnconders*. no corpo do capítulo é posta uma introdução geral de tudo que será abordado nos subtópicos do capítulo. O primeiro subcapítulo introduz a energia fotovoltaica, bem com o modo como a energia fotovoltaica é gerada, como é dada a correta manutenção dos sistema em todas as suas partes e, por fim, como é avaliada a quantia e a qualidade da energia gerada. No subcapítulo seguinte, são abordados os materiais utilizados para elaborar e avaliar o sistema proposto, passando por dados, linguagem de programação, bibliotecas, softwares e equipamentos. O terceiro subcapítulo é responsável por indicar qual o método utilizado para determinar como o processo de concepção do software será feito, utilizado e avaliado. Por fim, será exemplificado na minucia, no subcapítulo de percurso metodológico, como exatamente se dará todo o desenvolvimento do projeto de forma detalhada.

Por fim o capitulo 4 consiste em apenas um capítulo, composto pelas considerações finais sobre o trabalho, denotando resultados finais e aprendizados obtidos no decorrer do processo.

2 *AUTOENCODERS: CONCEITOS E APLICAÇÕES*

Para compreender o conceito dos *Autoencoders*, é necessário conhecer sobre os neurônios artificiais e os modelos de redes neurais. Uma rede neural tem por objetivo modelar o processo que ocorre no cérebro humano.

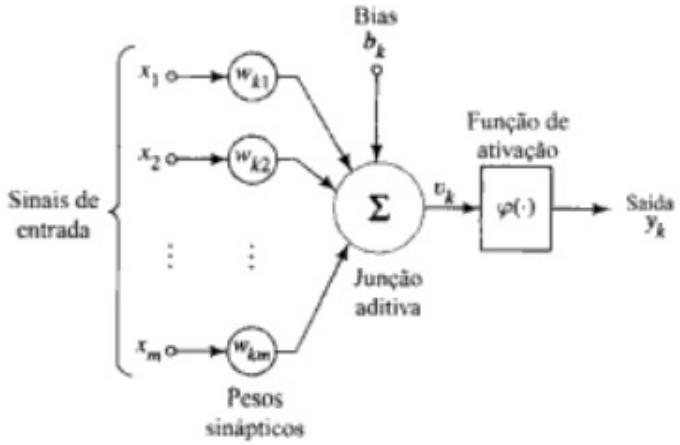
2.1 PERCEPTRON E A REDE MULTICAMADAS

O Perceptron, concebido por McCullon e Pitts (MCCULLOCH; PITTS, 1943), representa um modelo matemático que simula o comportamento de um neurônio humano. De maneira simplificada, o neurônio é inicializado com conexões conhecidas como sinapses, cada uma delas atribuída a um peso. Esse peso regula o sinal proveniente da entrada da sinapse. Todas as sinapses de um determinado neurônio são agregadas através de uma função de soma, a qual pondera os sinais de entrada que percorrem a rede. Os resultados dessa soma são então utilizados como entradas para uma função de ativação, que restringe a amplitude do sinal de saída do neurônio, limitando-o a um valor finito (HAYKIN, 1998).

Mediante a utilização de neurônios artificiais organizados em camadas paralelas, com vários neurônios contidos em cada camada é possível que uma rede artificial adquira conhecimento ao ajustar repetidamente os pesos dos neurônios. A seguir, apresenta-se uma concisa descrição de um neurônio e seus elementos constituintes (PADALLAN, 2022).

Na figura 3 é oferecida uma descrição minuciosa de um neurônio artificial. Um *bias* externo, denotado por b_k, é aplicado, e sua finalidade é ajustar a entrada na função de ativação, aumentando ou diminuindo de acordo com a perspectiva. Os sinais de entrada são $x_1, x_2, \ldots, x_m$, e os pesos das sinapses são $w_{k1}, w_{k2}, \ldots, w_{km}$, os quais variam à medida que os dados evoluem. A combinação linear de saída é expressa como u_k. O termo b_k é associado à representação do *bias*, e, por último, $\varphi(\bullet)$ denota a função de ativação. Após o processamento pelo referido componente, culmina-se no resultado final dos sinais introduzidos no neurônio, identificados como y_k (HAYKIN, 1998).

Figura 3 – Neurônio artificial



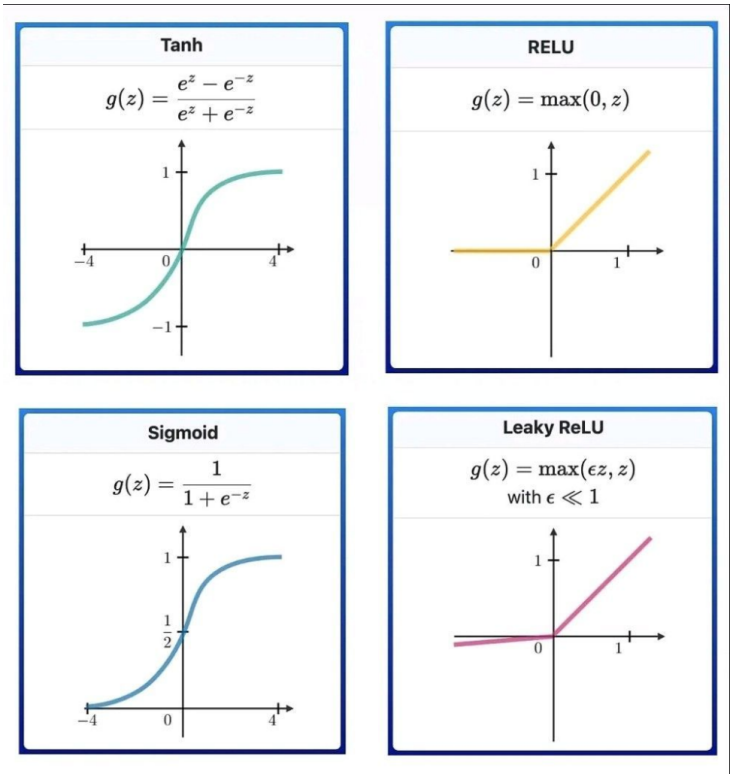
Fonte: retirado de (HAYKIN, 1998)

Matematicamente, o neurônio pode ter suas saídas simplificadas por $v_k = \sum_{j=0}^{m} W_{kj} X_j$ e $Y_k = \varphi(v_k)$(HAYKIN, 1998).

Aos neurônios é submetido um conjunto de entradas que são multiplicadas pelos seus respectivos pesos e por fim, as multiplicações são somadas. Em seguida é adicionado um viés (*bias*), resultando na saída U_k. Posteriormente, esta saída passa por uma função de ativação, modificando U_k e resultando em uma saída Y_k, que representa a saída final do neurônio em consideração.

A figura 4 oferece uma representação visual dos quatro principais tipos de função de ativação amplamente utilizados na atualidade: Tangente Hiperbólica, ReLU (Rectified Linear Unit), Sigmoidal e Leaky ReLU (Rectified Linear Unit com vazamento). Além disso, a figura inclui as respectivas expressões matemáticas que descrevem cada um desses tipos de função. As mesmas tem seus respectivos gráficos ilustrados na figura 3, que se encontra abaixo:

Figura 4 – Principais tipos de função de ativação



Fonte: retirado de https://ai-artificial-intelligence.webyes.com.br/most-used-activation-functions-in-neural-networks/

As funções de ativação têm como objetivo ativar o funcionamento do neurônio, determinando quando a informação emitida pelo neurônio é relevante para ser incorporada à rede. No entanto, a utilização de uma abordagem binária, com 0 para desativado e 1 para ativado, não é suficiente. Essas funções são essenciais para introduzir não-linearidade na rede, representando um componente adicional nos neurônios. Isso possibilita lidar com tarefas mais complexas e obter resultados mais precisos, mesmo em situações menos complexas (RASAMOELINA; ADJAILIA; SINčáK, 2020).

A função tangente hiperbólica (*tanh*) possui a característica de normalizar o valor de saída, de modo que, quando o valor se aproxima de infinito, a função tende a 1, e quando o valor se aproxima de menos infinito, a função tende a -1. Uma das vantagens fundamentais desse modelo de função de ativação é a sua centralização em torno de 0, o que facilita o mapeamento de valores distantes de 0. A *tanh* também é amplamente utilizada para centralizar os dados destinados à próxima camada. Além disso, pode ser derivada para mitigar um dos seus pontos negativos: quando os valores tendem muito a zero, os gradientes ou a influência desse valor na rede começam a desaparecer, resultando na interrupção do processo de aprendizado (RASAMOELINA; ADJAILIA; SINčáK, 2020).

A função sigmoide exibe um comportamento bastante similar à função tangente hiperbólica. Elas compartilham a característica de que, conforme o valor tende a infinito, a saída

se aproxima de 1. No entanto, divergem quando o valor tende a menos infinito: no caso da sigmoide, a saída tende a 0, em contraste com a tendência a -1 na *tanh*. A função sigmoide é frequentemente empregada quando a saída final desejada pode ser interpretada em termos de um percentual, ou quando se busca limitar o valor da saída, especialmente se é desejado evitar saltos abruptos nos valores finais (RASAMOELINA; ADJAILIA; SINčáK, 2020).

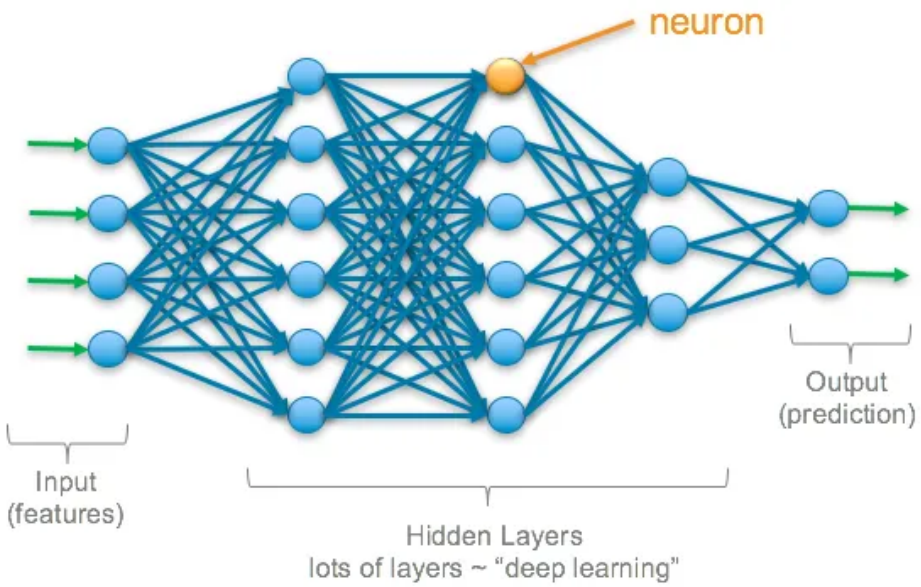
A terceira função de ativação é a função *ReLU* (*Rectified Linear Unit*), que, apesar de sua aparência linear, se destaca por permitir o *backpropagation*. Uma das principais vantagens da função *ReLU* é que nem todos os neurônios são ativados simultaneamente, neurônios com saída menor que 0 tendem a ser desativados. Isso resulta em um processamento mais eficiente, já que neurônios não utilizados são desligados, acelerando a convergência dos gradientes. Quando ativada, a função *ReLU* age de maneira linear, o que contribui para essa eficiência. No entanto, a função de ativação *ReLU* enfrenta um problema conhecido como "morte da ReLU"(*Dying ReLU*). A parte negativa do gráfico retorna um resultado neutro, independentemente do valor, o que significa que, em certos cenários, os pesos e bias dos neurônios não são atualizados durante o *backpropagation*. Isso pode resultar na ativação rara de alguns neurônios (RASAMOELINA; ADJAILIA; SINčáK, 2020).

A problemática do *Dying ReLU* é mitigada com a utilização da função *Leaky ReLU*. Nessa função, na parte negativa do gráfico, é introduzida uma ligeira inclinação na reta. A *Leaky ReLU* mantém as mesmas vantagens de sua contraparte, a função *ReLU*, mas com a capacidade de lidar com o *backpropagation*. No entanto, mesmo com a solução para o problema, quando valores negativos são introduzidos, a função não oferece predições consistentes e demanda um tempo significativo de treinamento na rede nessa região (RASAMOELINA; ADJAILIA; SINčáK, 2020).

Uma rede neural artificial é composta por várias camadas, e cada camada pode conter vários neurônios. A camada de entrada (*input*) é responsável por receber os dados da base de dados e transmiti-los às sinapses dos neurônios artificiais. A camada intermediária, também chamada de oculta, desempenha o papel de processar os dados na busca pelo resultado desejado, ajustando os pesos das conexões entre os neurônios. Por fim, a camada de saída (*output*) transforma a saída da última camada intermediária em uma codificação adequada para o problema de aprendizado em questão. A camada intermediária é crucial para classificar as redes neurais artificiais em dois conjuntos principais: as redes neurais artificiais, que possuem uma camada oculta com um número variável de neurônios, e as redes de aprendizado profundo (*Deep Learning*), que contam duas ou mais camadas ocultas (HAYKIN, 1998).

A figura 5 ilustra uma rede neural do tipo Perceptron Multicamadas *DeepLearning*, com uma indicação das camadas de entrada, intermediárias e saída.

Figura 5 – Rede neural com descrição de camadas



Fonte: retirado de https://srnghn.medium.com/deep-learning-overview-of-neurons-and-activation-functions-1d98286cf1e4

Os limites de uma rede neural nesta configuração incluem a necessidade de os dados de entrada serem numéricos e preparados manualmente por um ser humano, geralmente seguindo um método semi-automático. A codificação das características do problema também é realizada manualmente pelo ser humano. No entanto, para problemas complexos, como detecção de falhas, visão computacional e monitoramento de objetos, as características específicas podem ser númerosas e é desafiador identificar quais delas são realmente relevantes para um modelo de aprendizagem.

Para superar essa questão, foram desenvolvidas e testadas arquiteturas que processam imagens e outros sinais de forma a integrar a extração de características ao processo de aprendizado. Essas abordagens visam automatizar a identificação e utilização das características mais pertinentes para o modelo, permitindo que a rede neural aprenda de maneira mais eficiente e eficaz.

2.2 ALGORITMO BACKPROPAGATION

O *BackPropagation* representa um algoritmo fundamental utilizado no treinamento eficaz de redes neurais artificiais, sendo particularmente relevante no contexto do aprendizado profundo. Sua aplicação visa a otimização dos pesos vinculados às conexões neurais, promovendo uma melhoria gradual na capacidade preditiva da rede por meio da análise e correção dos erros antecipados ((ED.), 1995).

O processo tem início com a apresentação dos conjuntos de treinamento à rede neural. É então conduzida uma *ForwardPropagation*, na qual os dados são passados desde a camada de

entrada até a camada de saída. Durante essa fase, cada neurônio calcula sua saída com base nos estímulos de entrada e nos pesos das conexões correspondentes.

Após a *ForwardPropagation*, o erro ou perda é avaliado comparando as saídas geradas pela rede com os valores desejados, ou seja, os rótulos dos conjuntos de treinamento. Essa métrica de erro quantifica a discrepância entre as predições da rede e o ideal almejado. A etapa subsequente envolve a *BackwardPropagation* ou retro propagação. Seu propósito é calcular as variações mínimas nos pesos das conexões que possam reduzir a perda identificada. Esse cálculo é realizado aplicando a regra da cadeia de derivadas, permitindo determinar o gradiente da função de perda em relação a cada peso ((ED.), 1995).

O gradiente obtido é utilizado para reajustar os pesos das conexões, visando minimizar a perda. Destaca-se o papel crucial da taxa de aprendizado, um parâmetro essencial que regula a magnitude das alterações nos pesos a cada iteração do algoritmo. O ciclo composto por *ForwardPropagation* e *BackwardPropagation* é iterado várias vezes, através do processo conhecido como épocas de treinamento. Ao longo dessas iterações, os pesos são gradualmente ajustados para minimizar a função de perda, otimizando a precisão preditiva da rede neural com base nos estímulos de entrada ((ED.), 1995).

Assim, o *BackPropagation* se estabelece como o pilar fundamental no treinamento de redes neurais, viabilizando a aprendizagem a partir dos dados fornecidos e o aprimoramento contínuo do desempenho da rede ao longo das iterações ((ED.), 1995).

O algorítimo 1 apresenta um código, usando a biblioteca *TensorFlow* bem como bibliotecas auxiliares

Algoritmo 1 – Rede neural usando BackPropagation

```
import tensorflow as tf
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score

# Carregar o conjunto de dados MNIST
digits = load_digits()
X, y = digits.data, digits.target

# Pre-processamento dos dados
scaler = StandardScaler()
X = scaler.fit_transform(X)

# Dividir o conjunto de dados em treino e teste
X_train, X_test, y_train, y_test = train_test_split
(
    X, y, test_size=0.2, random_state=42
)
```

```
20
21 # Parametros da rede neural
22 input_size = X_train.shape[1]
23 hidden_size = 128
24 output_size = 10
25 learning_rate = 0.01
26 epochs = 1000
27
28 # Definir o modelo da rede neural
29 model = tf.keras.Sequential([
30     tf.keras.layers.Dense(hidden_size, activation='sigmoid',
31     input_shape=(input_size,)),
32     tf.keras.layers.Dense(output_size, activation='softmax')
33 ])
34
35 # Compilar o modelo
36 model.compile(optimizer=tf.keras.optimizers.
37                 SGD(learning_rate=learning_rate),
38                 loss='sparse_categorical_crossentropy',
39                 metrics=['accuracy'])
40
41 # Treinar o modelo
42 model.fit
43 (
44     X_train, y_train, epochs=epochs, batch_size=32,
45     validation_split=0.1, verbose=1
46 )
47
48 # Avaliacao do modelo
49 predictions = model.predict(X_test)
50 predicted_labels = np.argmax(predictions, axis=1)
51 accuracy = accuracy_score(y_test, predicted_labels)
52 print("Acuracia no conjunto de teste:", accuracy)
```

Fonte: O Autor (2024)

2.3 CONCEITUAÇÃO DE *AUTOENCODERS*

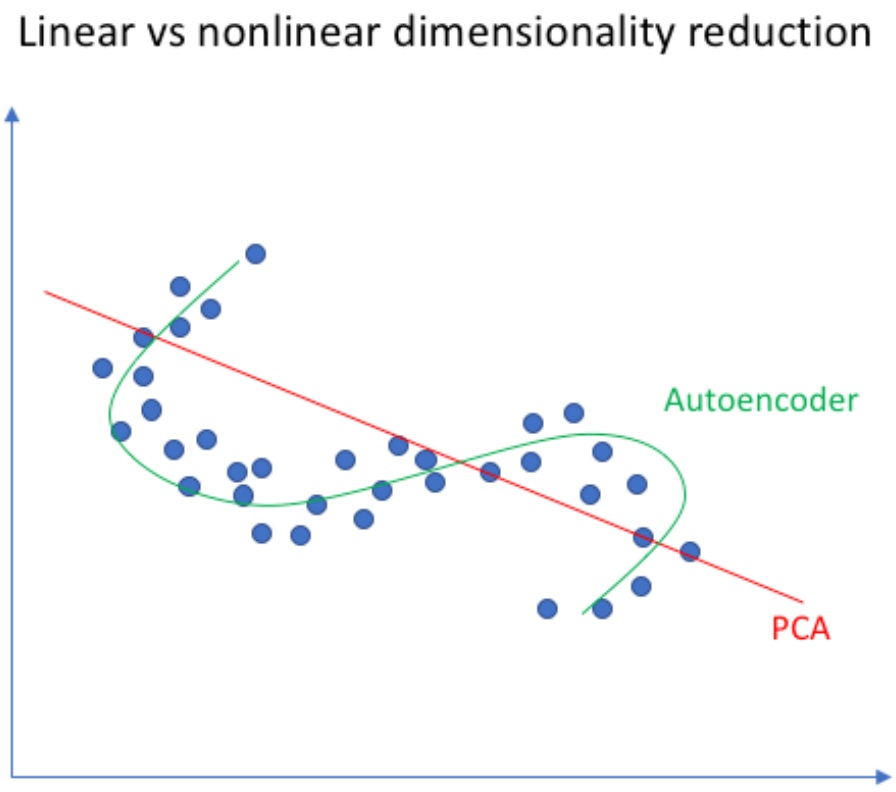
Os modelos de IA, pelo modo como operam, necessitam de um grande número de amostras para cada classe que se busca classificar, especialmente quando há um grande número de *features*, para atribuir uma faixa de valores adequada a cada característica (JOLLIFFE, 2002). Para ilustrar, suponha um conjunto de dados sobre veículos. Quando observa-se o comprimento e a largura de um veículo, nota-se que o número de janelas aumenta, assim como a quantidade de vidro usado na fabricação e possivelmente o tamanho das janelas. No entanto, para identificar o tipo de veículo, não são necessárias todas essas quatro variáveis. Teoricamente, é mais

eficaz para o modelo se pelo menos duas delas forem removidas. Isso resolve tanto o problema da redução de dados desnecessários para o treinamento do modelo quanto a necessidade de um grande número de amostras para a classificação adequada (JOLLIFFE, 2002).

Linearmente, para a obtenção de atributos distintivos, é empregada uma metodologia denominada análise de componentes principais ou, em inglês, *Principal Component Analysis* (PCA). O PCA opera ao identificar as dimensões dos dados que apresentam maior dispersão, identificando, dessa maneira, as características preponderantes que diferenciam substancialmente as classes presentes no conjunto de dados. Estas características são, então, preservadas.

Uma desvantagem do método PCA (Principal Component Analysis) em comparação com os Autoencoders (AEs) é que o PCA é um método linear, o que significa que ele otimiza ao longo de uma linha, enquanto os AEs são capazes de otimizar ao longo de uma curva, proporcionando uma abordagem mais flexível e mais adaptável para a redução de dimensionalidade e extração de características (JOLLIFFE, 2002). A figura 6, ilustra as diferenças dentre os processo de Otimização do PCA e dos AEs.

Figura 6 – Diferença na redução de dimensionalidade de PCAs e AEs.



Fonte: retirado de www.researchgate.net/figure/Comparison-between-PCA-and-Autoencoder-15_fig1_340049776

Autoencoders são derivados de redes neurais artificiais, sendo uma configuração específica destas. Os *Autoencoders* (AEs) foram inicialmente concebidos com o propósito de reduzir a dimensionalidade dos dados, eliminando elementos considerados dispensáveis para representar

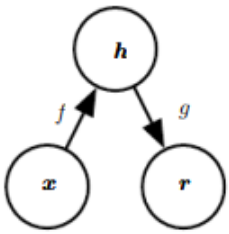
as características essenciais dos mesmos, sem perda significativa de informação. Nesse sentido, os AEs destacam-se por sua capacidade de aprender características inerentes ao conjunto de dados em análise (GOODFELLOW; BENGIO; COURVILLE, 2016). Funcionam através da inserção dos dados de entrada na primeira camada, denominada camada de *encode*, seguida de processamento por múltiplas camadas ocultas até a definição de um modelo dos dados na camada de latência. Posteriormente, esses dados são reconduzidos a uma representação compreensível para os seres humanos, por meio da camada de decodificação (BANK; KOENIGSTEIN; GIRYES, 2023). Em termos simplificados, a missão dos AEs é aprender uma representação de um conceito (algo) com o objetivo de reproduzir a entrada o mais precisamente possível, de maneira autônoma. Por esse motivo, são classificados como uma forma de aprendizado não supervisionado (ATIENZA, 2020).

A redução de dimensionalidade tem como objetivo eliminar o maior número possível de variáveis inseridas em um modelo de treinamento. Em uma base de dados extensa, com centenas de características, é comum encontrar características redundantes, que contribuem pouco ou quase nada para definir um item conforme sua natureza. Dessa forma, busca-se identificar as variáveis principais, aquelas que têm uma relação não linear com outras variáveis. Esse processo é crucial devido à "maldição da dimensionalidade"(BELLMAN, 1957). Um termo que abrange todos os problemas ocasionados pelo excesso de características (*features*).

O *Autoencoder* (AE) pode utilizar diversas fontes de dados, como discursos, texto, imagens, vídeos e uma variedade de sinais, incluindo elétricos e de rádio, para coletar informações. A fim de otimizar a incorporação dos dados no codificador, é fundamental realizar um tratamento que vise a máxima limpeza dos mesmos. Nesse contexto, o AE busca codificar uma entrada em um vetor de baixas dimensões, processo representado pela equação $h = f(x)$, em que $f(x)$ corresponde às características latentes ou ao vetor latente. Este vetor latente constitui uma representação compacta dos dados, quase totalmente comprimida. Em seguida, tem-se a etapa da decodificação, expressa pela equação $g(f(x)) = x$, na qual obtem-se os dados processados que visam ser o mais semelhantes possível aos dados originais. O vetor latente deve possuir uma dimensão suficientemente reduzida para possibilitar a reconstrução dos dados de entrada por meio do decodificador, mas não tão grande a ponto de fazer com que o AE se limite a memorizar e simplesmente reproduzir na saída a mesma informação (ATIENZA, 2020).

A figura 7 ilustra o ciclo feito pelo AE de forma simplificada, sendo x os dados iniciais, f o codificador, h o vetor latente, g o decodificador e, por fim, os dados resultantes representados por r.

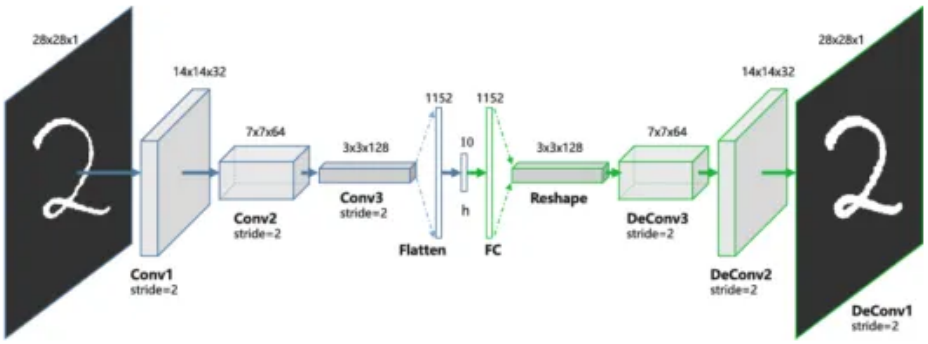
Figura 7 – Fluxograma básico de um *Autoencoder*.



Fonte: retirado de (GOODFELLOW; BENGIO; COURVILLE, 2016)

A figura 8 representa a redução e aumento de dimensionalidade de um *Autoencoder* durante seu processo de aprendizado, usando como exemplo a reconstrução de uma imagem.

Figura 8 – Representação das camadas *Encode* e *Decoder*.



Fonte: retirado de https://poulami98bakshi.medium.com/Autoencoders-367b386fa4e9

Explorando as camadas em maior detalhe, a função primordial da camada de codificação é converter a entrada no vetor latente, efetivamente constituindo uma representação reduzida em dimensionalidade da entrada. Dessa forma, a camada de codificação é orientada a aprender apenas o que julga ser essencial para facilitar uma reconstrução adequada. Para alcançar tal objetivo, o AE emprega diversas arquiteturas, esboçadas a seguir.

2.4 ARQUITETURAS DE *AUTOENCODERS*

Existem várias maneiras de codificar e decodificar informações por meio de *Autoencoders* (AEs). Esses métodos são variados e selecionados de acordo com a natureza específica do problema a ser abordado. Dessa forma, a arquitetura é adaptada para cada caso, levando em consideração não apenas a capacidade computacional necessária, mas também o resultado desejado. Nas seções subsequentes, serão delineadas algumas das principais arquiteturas.

No início, a concepção de um *Autoencoder* (AE) que simplesmente compacta e descompacta um conjunto de dados genérico pode parecer excessivamente elementar. No entanto,

sua aplicação se justifica pelo fato de que o vetor latente deve capturar o máximo de informações possíveis acerca da base de dados em consideração. Esse "máximo"é determinado por uma pessoa, e o AE é responsável por selecionar quais informações são as mais relevantes. O aprendizado desta é apresentado com uma forma simplificada de uma minimização da função de perda, representada pela equação:

$$L(x, g(f(x))) \tag{2.1}$$

Onde L é a função de erro que penaliza a reconstrução pelo decodificador representado por g(f(x)).

Como efeito adicional, o *Autoencoder* (AE) demonstra capacidade de aprender o subespaço primordial das informações presentes no conjunto de treinamento. Em relação aos AEs não-lineares, destaca-se um maior poder computacional quando comparado aos Métodos de Componentes Principais (PCAs), tecnologia para a qual os AEs foram desenvolvidos como uma alternativa otimizada, embora por meio de uma abordagem distinta (GOODFELLOW; BENGIO; COURVILLE, 2016).

Em seguida, foram desenvolvidos *Autoencoders* (AEs) regularizados, na tentativa de garantir que, após determinar a dimensão adequada do espaço latente e a capacidade apropriada para o codificador e o decodificador, o AE não ficasse restrito apenas a copiar a entrada para a saída, mediante ajustes na função de perda. Em vez de limitar a capacidade do codificador, do espaço latente e do decodificador, essa abordagem permite que o AE adquira outras habilidades, além da mera replicação da entrada. Os *Autoencoders* são tão abrangentes que qualquer modelo generativo que envolva variáveis latentes e seja capaz de calcular representações latentes de uma entrada de dados específica pode ser classificado como um AE (GOODFELLOW; BENGIO; COURVILLE, 2016).

Com o intuito de evitar a simples replicação da entrada para a saída, diversos métodos foram propostos para possibilitar algum tipo de aprendizado. Em um desses casos, que será delineado a seguir, emprega-se a penalização por escassez.Esta operação pode ser, de forma simplificada, representada por:

$$L(x, g(f(x))) + \Omega(h) \tag{2.2}$$

Sendo $\Omega(h)$ diretamente aplicado a saída do codificador.

Autoencoders (AEs) desse tipo são frequentemente aplicados em situações em que a detecção de características é necessária, sendo essas características posteriormente utilizadas em outras aplicações. Eles são direcionados para fornecer uma análise estatística dos recursos contidos no conjunto de dados, tornando-se especialmente úteis em cenários que requerem tarefas

de classificação. O termo $\omega(h)$ pode ser interpretado como um fator de normalização, adicionado com o propósito de mitigar o problema mencionado anteriormente, no qual o AE apenas transmite os dados de um ponto a outro sem que algo particularmente útil ocorra (GOODFELLOW; BENGIO; COURVILLE, 2016).

Podemos empregar a alteração da função de custo por meio da penalização da função de custo, mencionada anteriormente como Ω. Essa abordagem visa modificar o erro de reconstrução quando o vetor latente é revertido à sua dimensão original.

Exemplificação da já conhecida penalização da função de custo:

$$L(x, g(f(x))) \tag{2.3}$$

Neste método ela é substituída uma representação simplificada, representada por:

$$L(x, g(f(\tilde{x}))) \tag{2.4}$$

Nesse cenário, $\tilde{x}$ representa nada mais do que uma cópia de x que foi impactada por algum tipo de ruído, geralmente de natureza que afeta diretamente a medição. Devido a essa característica, os *Autoencoders* têm a capacidade de, durante o processo, remover esse ruído, resultando na limpeza dos dados (GOODFELLOW; BENGIO; COURVILLE, 2016).

Outra abordagem propõe a utilização de penalização através de Ω, porém com uma metodologia diferenciada em relação à abordagem convencional. Esta operação pode ser, de forma simplificada, representada por:

$$\Omega(h, x) = \lambda \sum_{i} \parallel \bigtriangledown xhi \parallel^2 \tag{2.5}$$

O propósito dessa abordagem é garantir que, se a função xx sofrer uma pequena variação, a função resultante não varie significativamente. Esse método visa capturar as informações relevantes diretamente da distribuição do conjunto de treinamento, e é conhecido como contrativo, uma variação dos *Autoencoders* que incorpora conexões semidiretas por meio da remoção de ruído (GOODFELLOW; BENGIO; COURVILLE, 2016).

Nos usos mais típicos, os *Autoencoders* são geralmente empregados com apenas uma camada tanto na etapa de codificação quanto na de decodificação. No entanto, assim como diversos outros tipos de redes neurais, eles se beneficiam consideravelmente da adição de múltiplas camadas, resultando nos chamados *Deep Encoders* (DE). Por serem redes do tipo *Feed-Forward*, os *Autoencoders* têm a garantia de poder representar qualquer função, característica essa decorrente do teorema da aproximação universal. Isso é particularmente valioso, pois com

várias camadas, é possível impor restrições arbitrárias (GOODFELLOW; BENGIO; COURVILLE, 2016).

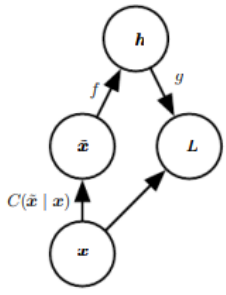
Uma vantagem adicional é a capacidade de reduzir substancialmente o custo computacional ao escolher uma função adequada e usar camadas adicionais. Além disso, se não houver uma exigência de menor capacidade computacional, é viável manter a eficácia do treinamento com uma quantidade menor de dados de treinamento. Frequentemente, é recomendado que, para melhor eficácia, o DE seja treinado com múltiplos AEs, formando assim uma arquitetura composta (GOODFELLOW; BENGIO; COURVILLE, 2016).

Dado que os *Autoencoders* (AEs) são do tipo *FeedForward*, todas as características inerentes a esse tipo de rede são aplicáveis de forma natural aos AEs. Isso inclui a obtenção de uma saída distribuída, representada por $p(y|x)$, cuja minimização é expressa por $-\log p(y|x)$, onde Y é composto por vetores alvo. Cada posição nesses vetores descreve a classe de dado presente no vetor latente. Vale ressaltar que o vetor x, além de servir como dado de entrada, representa o dado que desejamos obter.(GOODFELLOW; BENGIO; COURVILLE, 2016).

Mesmo com essa abordagem, é possível empregar as mesmas arquiteturas mencionadas anteriormente, porém treinadas através da minimização da função definida por $-\log p_{\text{decoder}}(y|x)$, cuja definição principal está relacionada a p_{decoder}. Uma estratégia mais inovadora leva em consideração a generalização da função de codificação $f(x)$, transformando-a em uma codificação distribuída $p_{\text{encoder}}(h|x)$. Nesse contexto, qualquer variável latente do modelo $p_{\text{encoder}}(h|x)$ pode atuar como um codificador estocástico, onde $p_{\text{encoder}}(h|x) = p_{\text{model}}$, e como um decodificador estocástico, onde $p_{\text{decoder}}(h|x) = p_{\text{model}}$ (GOODFELLOW; BENGIO; COURVILLE, 2016).

Outro modelo amplamente empregado são os *Autoencoders* removedores de ruídos (DAE na sigla em inglês). Esses constituem um tipo específico de AE com o propósito de eliminar ruídos em determinado dado, ao tentar prever o dado original a partir da versão ruídosa. Esse processo propaga o dado livre de ruídos para a camada de saída, conforme exemplificado na figura 9.

Figura 9 – Fluxograma básico de um *Denoising Autoencoder*.

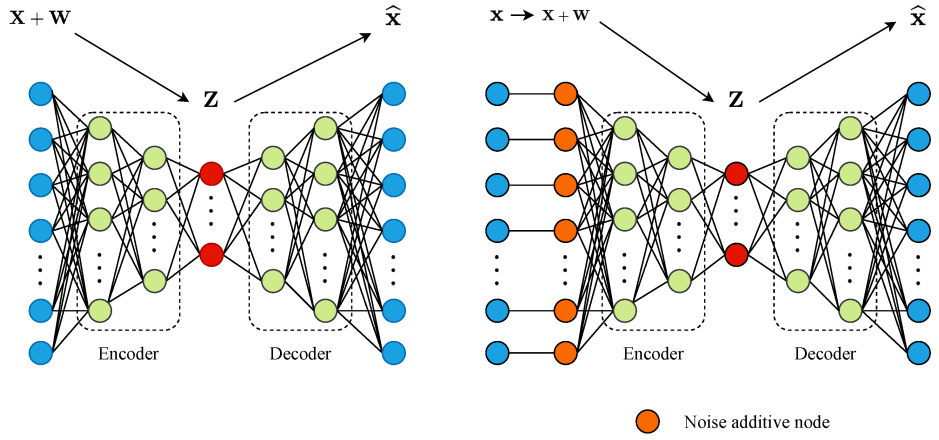


Fonte: retirado de (GOODFELLOW; BENGIO; COURVILLE, 2016)

O DAE usa um processo de corrompimento $C(x|\tilde{x})$, que representa uma distribuição condicional sobre as amostras corrompidas $\tilde{x}$, com base em um conjunto de dados x. O DAE aprende através da reconstrução probabilística $p_{\text{reconstruct}}(x|\tilde{x})$ dos dados imputados e corrompidos $(x, x\,)(x, x\,)$. A abordagem envolve retirar uma amostra x do conjunto de treino, corrompê-la usando $C(x|\tilde{x} = x)$, e, finalmente, estimar a reconstrução distribuída $p_{\text{reconstruct}}(x|\tilde{x}) = p_{\text{decoder}}(x|h)$, onde h é a saída do codificador $f(\tilde{x})$, e p_{decoder} é definido por $g(h)$.

A figura 10 ilustra dois variantes de DAE, onde o primeiro delineia uma configuração de rede submetida a uma base de dados ideal, porém, contendo dados comprometidos. Enquanto isso, a segunda representação à direita ilustra uma base de dados isenta de qualquer tipo de corrupção.

Figura 10 – Variações de um *Denoising Autoencoder*



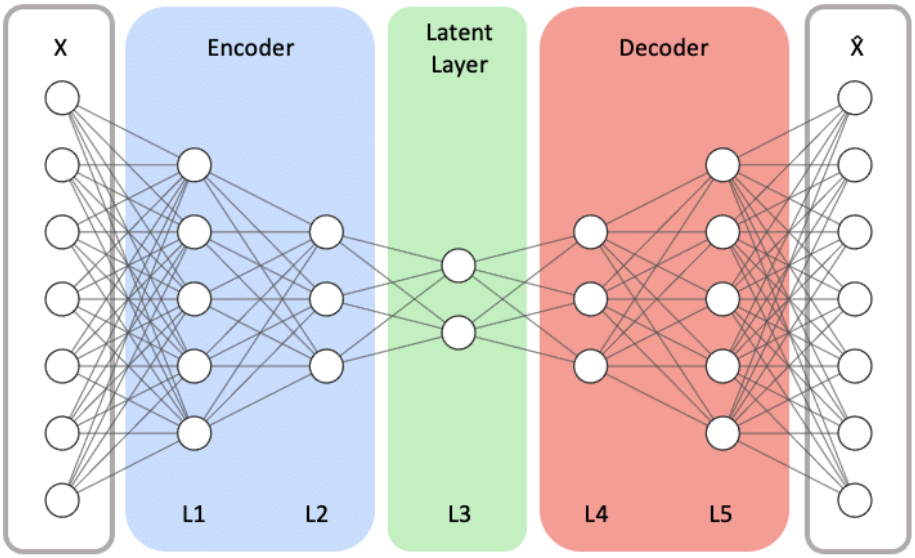
Fonte: retirado de (PISA *et al.*, 2020)

Após a exploração aprofundada do universo dos *Autoencoders*, uma categoria versátil de redes neurais que vai além da mera compressão de dados, tornou-se evidente que essas estruturas têm a capacidade de aprender representações latentes altamente significativas e promissoras. Desde a configuração tradicional do AE até variações mais avançadas, como os *denoising* AEs, percebemos que essas arquiteturas são proficientes em aprender representações latentes de considerável potencial e relevância. Os AEs desempenham um papel fundamental em tarefas de reconstrução e geração, oferecendo *insights* valiosos para uma ampla variedade de aplicações, abrangendo desde a redução de dimensionalidade até a criação de novos conjuntos de dados.

2.5 COMPREENDENDO O FUNCIONAMENTO DO *AUTOENCODERS*

A fim de que seja possível compreender como um modelo baseado em *Autoencoders* é construído, um exemplo aplicado é descrito nesta seção. Para iniciar, é importante compreender que uma arquitetura *Autoencoders* é baseada em camadas sucessivas, conforme ilustrado na figura 11.

Figura 11 – Fluxograma de um *Autoencoder* com legenda.



Fonte: retirado de: https://www.researchgate.net/figure/An-example-Autoencoder-model-architecture-with-symmetrical-encoder-and-decoder-networks_fig2_352703131

As camadas responsáveis pelo *encoder* em um *Autoencoder* são fundamentais para comprimir os dados de entrada em uma representação de menor dimensionalidade, chamada de vetor ou espaço latente. O processo de *encode* se desenrola da maneira descrita a seguir.

Começando com a camada de entrada, ilustrada na figura como X, os dados brutos são inseridos na rede neural, onde cada nó representa uma *feature* dos dados de entrada. Em seguida, seguimos para a camada L1, onde ocorrem transformações lineares por meio de multiplicação de pesos e adição de viés a cada *feature* de entrada. Essa operação é representada como $Z = WX + BZ = WX + B$, onde ZZ é a saída da camada, WW são os pesos, XX são as entradas e BB são os vieses.

Após a transformação linear, temos a camada L2, onde é aplicada uma função de ativação não linear, como ReLU, Sigmoid ou Tanh, introduzindo não linearidades na representação e permitindo a captura de padrões complexos nos dados. As camadas intermediárias subsequentes, que seriam múltiplas camadas tais quais L2, continuam aplicando transformações lineares e funções de ativação não lineares, resultando em uma redução progressiva da dimensionalidade dos dados. Isso comprime as informações, preservando as características mais relevantes.

A última camada do *encoder*, conhecida como camada latente, gera o espaço latente, representado por L3. O espaço latente é a representação de menor dimensionalidade dos dados, contendo as características mais significativas. Durante o treinamento, os parâmetros da rede (pesos e vieses) são otimizados por meio de algoritmos de otimização, como gradiente descendente, para minimizar uma função de custo que compara a saída do *encoder* (o Espaço latente) com a entrada original.

O *encoder* busca aprender uma representação compacta e informativa dos dados, permitindo uma significativa redução na dimensionalidade sem perda de informações cruciais. Essa representação latente pode ser usada para reconstruir os dados originais pelo *decoder* ou para outras tarefas, como *clustering* ou *transfer learning*. Já a segunda camada, a camada de *decoder* em um *Autoencoders* desempenha um papel crucial ao reconstruir os dados originais a partir da representação latente, gerada pelo *encoder*. Examina-se a seguir de forma detalhada como essa camada opera. A entrada, que é o Espaço latente, representado por L3, uma representação compacta de menor dimensionalidade gerada pelo *encoder*. Cada componente desse código latente é multiplicado pelos pesos correspondentes e somado ao viés, uma operação inversa à realizada no *decoder*, representada pela camada L4. Essa operação é expressa como $Z = W'X + B'Z = W'X + B'$, onde ZZ é a saída da camada, $W'W'$ são os pesos inversos e $B'B'$ são os vieses inversos.

Após as transformações lineares, é aplicada uma função de ativação não linear a cada nó da camada, Representada pela camada L5, introduzindo não linearidades na reconstrução e permitindo que o modelo capture padrões complexos e sutilezas presentes no Espaço latente. Aumentando progressivamente a dimensionalidade, cada camada intermediária do *decoder* busca reconstruir as características dos dados originais, expandindo a representação e tornando-a mais similar à entrada original.

A última camada do *decoder* é a camada de saída, esta representada por $\hat{x}$, responsável por gerar a reconstrução final dos dados, mantendo a mesma dimensionalidade que a entrada original. Durante o treinamento, uma função de custo, é calculada comparando a saída do *decoder* com os dados de entrada originais. Essa função penaliza as diferenças, incentivando o modelo a aprender uma reconstrução precisa.

No processo de treinamento, os parâmetros do *decoder*, incluindo pesos e vieses, são otimizados para minimizar a função de custo. Isso é feito utilizando algoritmos de otimização, como o gradiente descendente, para melhorar a qualidade da reconstrução e garantir que o *decoder* recrie os dados originais com precisão.

Assim, o papel essencial do *decoder* é reverter a compactação feita pelo *encoder*, reconstruindo os dados originais a partir do código latente. Isso permite avaliar a eficiência da representação latente aprendida e a qualidade global do modelo de *Autoencoders*.

2.6 APLICAÇÕES NA ÁREA DE DETECÇÃO DE FALHAS

A análise de algoritmos está frequentemente ligada a um problema específico, a uma tarefa para a qual foram desenvolvidos. O mesmo se aplica aos *Autoencoders*. Este capítulo explora suas aplicações em contextos práticos, predominantemente associados à análise e interpretação de sinais.

Uma das partes mais relevantes para tal é a escolha de artigos, para assim avaliar os

mais relevantes, como é utilizado e, por fim, resultados obtidos por estes. Para tal foi utilizada a plataforma Google acadêmico, que indexa artigos, livros, revistas e demais conteúdos oriundos de projetos de pesquisa. A plataforma Google acadêmico permite acesso a publicações disponíveis em portais, tais como Springer [1], ScienceDirect e Cornell University. Assim, a plataforma recomenda publicações revisados por pares. Como palavras chave, foram utilizadas para as pesquisas as seguintes sequências: *Autoencoders Failure*; *Autoencoders Signal*; *Autoencoders for Anomaly Detection*; *Autoencoders Survey*; *Autoencoders Overview*. O período de publicação foi filtrado para conter artigos posteriores a 2018.

O primeiro estudo examinado é o trabalho que deu origem à ideia de implementação apresentada aqui, o qual foi escrito por João Pedro Cardoso Pereira com o título "Unsupervised Anomaly Detection in Time Series Data using Deep Learning". Este trabalho de mestrado concentra-se principalmente na detecção de anomalias em dados de séries temporais sem o uso de supervisão, empregando abordagens de aprendizado profundo. O principal objetivo da dissertação é desenvolver uma estrutura inovadora para identificar anomalias, levando em consideração as relações temporais inerentes a esses tipos de dados. A abordagem proposta deve se destacar por sua capacidade de escalabilidade, eficiência computacional e habilidade de operar sem a necessidade de rótulos prévios para as anomalias. Além disso, a dissertação busca integrar princípios advindos de diversos domínios, como processamento de linguagem natural, reconhecimento de fala e processamento de imagens, em um único conjunto de ferramentas para a detecção de anomalias em séries temporais (PEREIRA, 2018).

A abordagem proposta se vale de uma técnica fundamentada em *Autoencoders* para detectar anomalias em conjuntos de dados de séries temporais. Esse processo se divide em duas etapas distintas: a aprendizagem da representação e a própria detecção das anomalias. O desenvolvimento do modelo para a aprendizagem da representação é baseado em um AE, sendo apresentadas diferentes estratégias para a detecção. O objetivo é criar uma técnica que seja não supervisionada e escalável, capaz de capturar as relações temporais nos dados de séries temporais e identificar anomalias sem depender de rótulos específicos para essa finalidade (PEREIRA, 2018).

O resultado da dissertação foi a concepção de um arcabouço abrangente, versátil e escalável para a detecção de anomalias em dados de séries temporais. A proposta desenvolvida atende plenamente aos critérios estabelecidos e demonstra eficácia na detecção de padrões anômalos em dados provenientes de diversos setores, como energia e saúde. O *framework* faz uso de modelos de aprendizado profundo, incluindo *Autoencoders*, redes neurais recorrentes e mecanismos de atenção, para aprender representações altamente informativas dos dados e identificar observações fora do padrão (PEREIRA, 2018).

O segundo estudo analisado guarda semelhanças com o primeiro, pois se concentra na criação e aplicação de um sistema de detecção e classificação de falhas utilizando um *Auto-*

[1] https://link.springer.com/

encoder convolucional esparsamente conectado (CSAE) para sinais provenientes de múltiplos canais. O objetivo é aprimorar o desempenho no diagnóstico de falhas ao extrair características robustas e informativas dos sinais, viabilizando uma classificação precisa dos diversos tipos de falhas. O artigo aborda a estrutura voltada para a detecção e classificação de falhas, baseada no CSAE, bem como a implementação e desempenho desse método em diversos contextos. Além disso, o autor compara a abordagem CSAE proposta com métodos já existentes e avalia a eficácia dela na classificação de falhas (CHEN; HU; HE, 2018).

A conclusão da avaliação sobre a eficácia do método CSAE na classificação de falhas é que o método proposto alcança taxas de acerto elevadas, quase atingindo 100% para todos os tipos de falhas. O método demonstra uma robustez frente a ruído e erros de medição, evidenciando uma capacidade substancial de aplicação generalizada. Além disso, revela-se ágil e preciso na detecção e classificação de falhas, tornando-o viável para a proteção em tempo real de linhas de transmissão (CHEN; HU; HE, 2018).

O terceiro artigo apresentado adota uma abordagem mais mecânica, concentrando-se no desenvolvimento de um método inteligente para detecção de falhas em rolamentos e monitoramento de suas condições, por meio da utilização de um *Autoencoder* totalmente conectado, combinado com uma técnica conhecida como "winner-take-all", uma técnica em que somente o melhor resultado da etapa segue adiante. A pesquisa explora a aplicabilidade de técnicas de aprendizado profundo, especialmente os AEs, para extrair características e diagnosticar falhas em maquinaria rotativa. O método proposto visa obter um desempenho aprimorado e maior resistência ao ruído em comparação com algoritmos já existentes. A tese também aborda experimentos que avaliam os efeitos de diferentes hiperparâmetros e compara o método proposto com técnicas de ponta. Além disso, proporciona visualizações do modelo proposto para compreender o seu mecanismo interno (LI *et al.*, 2018).

O método adotado para a detecção de falhas em rolamentos e monitoramento de suas condições é um método de diagnóstico inteligente baseado em um *Autoencoder* FC-WTA (Totalmente Conectado com Seleção do Vencedor). Este método faz uso da estrutura simples e da capacidade de aprendizado não supervisionado dos AEs para extrair características valiosas dos dados. O AE FC-WTA é ajustado de forma a extrair características que possuem propriedades desejáveis para o diagnóstico de falhas em rolamentos. Além disso, é aplicado um método de conjunto para aprimorar ainda mais o desempenho do método proposto (LI *et al.*, 2018).

A conclusão do método de diagnóstico inteligente, fundamentado no *Autoencoder* FC-WTA e no método de conjunto adotado, indica que ele superou os algoritmos de referência em desempenho e demonstrou robustez diante de ruídos. O método proposto exibiu uma maior acurácia de classificação e consistência em comparação a outros métodos avançados, como SVM (Support Vector Machine), DNN (Deep Neural Network) e WDCNN (Deep Convolutional Neural Networks with Wide First-layer Kernels). Além disso, o método de conjunto, que aplicou votação suave entre segmentos de sinal, contribuiu para elevar a precisão e a estabilidade do mé-

todo FC-WTA. De modo geral, o método proposto mostrou ser eficaz no diagnóstico de falhas em rolamentos (LI *et al.*, 2018).

Uma aplicação bastante frequente dos *Autoencoders* é a sua utilização em sinais de Eletrocardiogramas (ECGs). Os ECGs, por possuírem um padrão previsível, são altamente úteis para a comparação de diversos modelos, visando a detecção de discrepâncias em relação ao padrão normal. O estudo em questão concentra-se na comparação de vários modelos de aprendizado profundo para a remoção de ruídos em sinais de ECG, incluindo a Rede Totalmente Convolucional (FCN), a Rede Neural Profunda (DNN) e a Rede Neural Convolucional (CNN). O principal objetivo da pesquisa é avaliar o desempenho desses modelos em termos de reconstrução dos sinais e preservação das informações clínicas. Além disso, os pesquisadores investigam o impacto das camadas totalmente conectadas na qualidade da reconstrução dos sinais. Os modelos de aprendizado profundo são treinados de maneira a minimizar o erro quadrático médio entre as formas de onda limpas e as *denoised*. Além disso, a técnica de *dropout* é aplicada nesses modelos para melhorar a eficácia (CHIANG *et al.*, 2019).

A conclusão da comparação entre diferentes modelos de aprendizado profundo para a remoção de ruídos em sinais, apresentada na tese, destaca que a Rede Totalmente Convolucional (FCN) se destaca em relação aos modelos de Rede Neural DNN e CNN. A FCN consegue preservar de maneira mais eficaz a morfologia dos sinais de ECG, mantendo a forma dos complexos QRS (Forma de onda de uma parte especifica do eletrocardiograma) e extraindo as principais características morfológicas dos sinais. Além disso, alcança uma redução de ruído mais eficiente e mantém detalhes essenciais das informações clínicas em comparação com os outros modelos. Enquanto o modelo DNN apresenta uma perda acentuada nas amplitudes dos picos R, o modelo CNN possui distorções menores nos complexos QRS. De modo geral, a abordagem de remoção de ruídos baseada na FCN demonstra um desempenho superior em termos de precisão na remoção de ruídos e preservação da relevância clínica. Dessa forma, fornece orientações sobre o método mais indicado para aplicação no contexto abordado por este trabalho (CHIANG *et al.*, 2019).

Outra aplicação relevante reside na utilização de agrupamento profundo baseado em redes neurais profundas para analisar dados sísmicos. A proposta apresentada na tese engloba uma estratégia de agrupamento em duas etapas, onde os *Autoencoders* totalmente convolucionais sub completos são empregados para extrair características dos dados sísmicos e reduzir sua dimensionalidade. A sintonização fina dos pesos na rede profunda é, então, executada para otimizar a aprendizagem das características e a atribuição de agrupamento. A eficácia desta abordagem é demonstrada pela tese em diversas tarefas, como a determinação da polaridade do primeiro movimento e a discriminação entre eventos sísmicos locais e tele-sísmicos (MOUSAVI *et al.*, 2019).

Na primeira etapa, são empregados *Autoencoders* totalmente convolucionais sub completos para extrair automaticamente características dos dados sísmicos de entrada e reduzir a

dimensionalidade dessas características. Esse procedimento se revela crucial para aprimorar o desempenho do agrupamento, pois reduz a dimensionalidade dos dados, realizando o agrupamento em um espaço de características, em vez do espaço dos dados originais. Na segunda etapa, os pesos da rede neural profunda são refinados de maneira iterativa, otimizando simultaneamente a aprendizagem de características e a atribuição de agrupamento. Este processo assegura que a rede aprenda características eficazes para a tarefa específica de agrupamento. Ao realizar o agrupamento em um espaço de características otimizado com base na atribuição de agrupamento, o método obtém representações de características aprendidas que são altamente eficazes para agrupar os sinais sísmicos. Globalmente, o método combina a capacidade das redes neurais profundas de aprender características complexas com a eficácia dos algoritmos de agrupamento, resultando em uma abordagem robusta para o agrupamento não supervisionado de dados sísmicos (MOUSAVI *et al.*, 2019).

O resultado final do método aplicado para o agrupamento profundo baseado em redes neurais profundas na análise de dados sísmicos é uma acurácia global de 0.97. As métricas de precisão, revocação e pontuação F para polaridades ascendentes são, respectivamente, 0.99, 0.82 e 0.9, enquanto para as polaridades descendentes, os valores correspondentes são 0.72, 0.99 e 0.83. Essa abordagem não supervisionada alcançou níveis de precisão comparáveis aos obtidos por métodos supervisionados, sem a necessidade de dados rotulados, elaboração manual de características e conjuntos de treinamento extensos (MOUSAVI *et al.*, 2019).

Após a leitura, análise e resumo de 5 aplicações correlacionadas com a proposta deste trabalho, seguem abaixo tabelas comparativas de características, métodos e demais partes relevantes dos mesmos. O comparativo tem por finalidade elucidar os pontos principais de cada trabalho apreciado para assim justificar a escolha da arquitetura e base de dados para o proposto neste projeto.

O quadro 1, efetua um comparativo entre todos os artigos.

Quadro 1 – Comparativo entre artigos

Artigo	**Dataset**	**Método de analise**	**Acurácia obtida nos estudos**
Unsupervised Anomaly Detection in Time Series Data using Deep Learning	Sinal de geração vindo de Paineis solares	*Recurrent Neural Network*	98,43%
Detection and Classification of Transmission Line Faults Based on Unsupervised Feature Learning and Convolutional Sparse Autoencoder	Corrente e tensão de linhas de transmissão	*Convolutional Sparse Autoencoder*	99,29%
Bearing Fault Diagnosis Using Fully-Connected Winner-Take-All Autoencoder	Acelerômetros nos rolamentos	*Fully-conected Winner Take All AutoEnconder*	98,53%
ECG Signal Enhancement based on Fully Convolutional Denoising Autoencoder	Sinal de eletrocardiograma (MIT-BIH Arrhythmia)	*Fully Convolutional Network*	25,79%
Unsupervised Clustering of Seismic Signals Using Deep Convolutional Autoencoders	Sinais e ruídos em tempo real em sismógrafos	*Fully Convolutional Network*	98,41%

Fonte: O Autor (2024).

2.7 CONSIDERAÇÕES FINAIS DO CAPÍTULO 2

A eficaz implementação dos *Autoencoders* demanda a consideração de diversos aspectos como: escolha da arquitetura, dimensionamento do espaço latente, uso de algoritmos de regularização e inicialização de pesos, entre outros. No âmago desta tarefa, encontra-se a arquitetura dos *Autoencoders*, determinada por fatores primordiais, tais como o número de camadas, as dimensões do código latente e a configuração dos blocos *encoder* e *decoder*. É prudente realizar experimentações com distintas arquiteturas a fim de identificar a que melhor se adéqua à aplicação em análise.

Lidar com conjuntos de dados extensos representa um desafio adicional em termos de escalabilidade. Expandir para lidar com volumes consideráveis de dados exige recursos computacionais substanciais e o emprego de técnicas específicas. Além disso, a escolha adequada das funções de ativação para cada camada é de extrema importância para assegurar que o modelo aprenda padrões complexos nos dados.

Encontrar a combinação ótima de hiperparâmetros, como a taxa de aprendizagem, o tamanho do *batch* e o número de épocas, constitui um desafio adicional, exigindo iteração e experimentação extensiva. Por fim, a eficiência computacional durante o treinamento e a inferência do *Autoencoders* pode ser problemática, especialmente em cenários com recursos com-

putacionais limitados. O equilíbrio entre o desempenho e o uso eficiente dos recursos representa uma dificuldade adicional a ser superada. Essas complexidades ressaltam a importância de uma compreensão profunda dos princípios e práticas relacionadas à implementação de *Autoencoders*, bem como a necessidade de experimentação iterativa para superar os desafios e obter resultados eficazes.

Em termos de estado da arte, os *Autoencoders* têm desempenhado um papel significativo nas aplicações de aprendizado de máquina e Inteligência Artificial. Sua versatilidade é notável, abrangendo desde a reconstrução de imagens, compressão de dados, geração de dados até o pré-treinamento de redes neurais profundas e aprendizado de representações. Nesse contexto, alguns avanços notáveis incluem os *Variational Autoencoders* (VAEs), conhecidos por sua formulação probabilística que os torna eficazes na geração de imagens e aprendizado de distribuições latentes.

Além disso, destaca-se o surgimento dos *Variational Autoencoders* Beta (β -VAE), que introduzem o hiperparâmetro β para controlar e balancear a fidelidade da reconstrução e a diversidade das representações latentes. Também merecem menção as Redes *Adversarial Autoencoder* (AAE), que combinam elementos dos *Autoencoders* com a estrutura *adversarial* das *Generative Adversarial Networks* (GANs), resultando em representações latentes mais coesas.

No contexto da arquitetura, observa-se uma tendência para modelos mais profundos e complexos, incluindo redes *Autoencoders* profundas e arquiteturas residuais. Além disso, há uma crescente aplicação dos *Autoencoders* em cenários multimodais, integrando diversas modalidades de dados, como texto e imagem, em um único modelo para aprendizado de representações abrangentes.

Para além disso, há um crescente foco na interpretabilidade e robustez dos *Autoencoders*, com pesquisa em técnicas que visam torná-los mais interpretáveis e aplicáveis em domínios onde a explicabilidade do modelo é essencial. Também são notáveis os avanços em transferência de aprendizado com *Autoencoders*, utilizando modelos pré-treinados para melhorar o desempenho em tarefas relacionadas.

Portanto, há mérito substancial neste estudo, dado o potencial impacto positivo que a aplicação de *Autoencoders* na detecção de falhas em painéis fotovoltaicos pode ter. Os sistemas de energia fotovoltaica desempenham um papel fundamental na transição para fontes de energia mais sustentáveis e eficientes. No entanto, a ocorrência de falhas em painéis fotovoltaicos pode prejudicar significativamente a eficiência e a produção de energia desses sistemas. É nesse contexto que a utilização de técnicas avançadas, como os *Autoencoders*, para identificar e diagnosticar falhas em tempo hábil se torna crucial. Ao permitir a detecção precoce e precisa de falhas, os *Autoencoders* podem contribuir para a manutenção proativa e eficaz dos sistemas fotovoltaicos, otimizando a geração de energia e, consequentemente, impulsionando a eficiência e a sustentabilidade dessa fonte energética.

A aplicação dos *Autoencoders* se destaca ao oferecer uma abordagem eficaz para processar os dados gerados pelos painéis fotovoltaicos, capturando padrões sutis que indicam falhas ou degradação do sistema. Com a capacidade de aprender representações latentes complexas dos dados de entrada, os *Autoencoders* podem discernir anomalias que podem passar despercebidas por métodos tradicionais. Além disso, a detecção precoce de falhas em painéis fotovoltaicos é crucial para minimizar custos de manutenção, evitar interrupções não planejadas e prolongar a vida útil dos sistemas. Portanto, este estudo não apenas aborda um tema de relevância tecnológica e ambiental, mas também apresenta um potencial significativo para otimizar o desempenho dos sistemas fotovoltaicos e contribuir para um futuro energético mais sustentável.

3 IMPLEMENTAÇÃO DE MODELO PARA DETECÇÃO DE FALHAS EM SISTEMAS FOTOVOLTAICOS A PARTIR DA ANÁLISE DE SINAL DE GERAÇÃO

Este capítulo trata sobre a implementação de sistemas de geração de energia elétrica por meio do uso de painéis fotovoltaicos passando por toda a estrutura e tecnologia ligada a eles como inversores, conversores, controladores, fios e demais componentes pertinentes. Juntamente a isto, são tratados os *Autoencoders* e como eles podem ser usado para detectar falhas em estruturas de geração que utilizam painéis fotovoltaicos para tal.

Painéis fotovoltaicos, tecnologia de conversão direta da luz solar em eletricidade, têm se destacado como uma fonte para a geração de energia elétrica de maneira ambientalmente responsável, constituindo uma alternativa limpa e sustentável frente aos métodos convencionais. Este tipo de tecnologia exerce um impacto positivo significativo no meio ambiente, particularmente no que concerne à redução da pegada de carbono (Emissão de gases do efeito estufa em comparação com outros meios de geração) e à atenuação dos efeitos das mudanças climáticas. A adoção de painéis fotovoltaicos contribui substancialmente para a redução da dependência de combustíveis fósseis, tais como carvão, petróleo e gás natural, fontes não renováveis de energia que produzem gases de efeito estufa. A transformação direta da luz solar em eletricidade pelos painéis solares resulta na ausência de emissões de gases poluentes durante seu funcionamento, implicando, assim, na redução direta das emissões de dióxido de carbono (CO2) e de outros poluentes prejudiciais ao ambiente. Além disso, a geração de energia por meio de painéis solares minimiza a necessidade de desmatamento para a construção de usinas hidrelétricas ou outras infraestruturas de geração de energia, contribuindo para a preservação de ecossistemas terrestres e aquáticos valiosos. Ressalta-se que a utilização dos painéis fotovoltaicos auxilia na mitigação dos riscos associados à exploração de recursos naturais, como os vazamentos de petróleo e a contaminação de corpos d'água.

É pertinente salientar que, embora os benefícios dos painéis solares para o meio ambiente sejam notáveis, a produção e o descarte desses dispositivos necessitam de cuidados específicos para minimizar possíveis impactos ambientais adversos. Na produção de alguns tipos de painéis são utilizadas substâncias potencialmente tóxicas, como chumbo, cádmio e selênio, que podem representar riscos para a saúde humana e o meio ambiente se não forem manuseados e descartados corretamente. Portanto, descarte impróprio de painéis fotovoltaicos no final de sua vida útil pode resultar em contaminação do solo e da água devido à liberação das substâncias anteriormente citadas, para tal, recomenda-se a reciclagem que, infelizmente ainda não está amplamente estabelecida ou economicamente viável em todas as regiões.

Nesse contexto, a geração de energia por meio de painéis fotovoltaicos não se apresenta

desprovida de desafios, como é o caso de todas as tecnologias. Dentre esses desafios, as condições climáticas, tais como a presença de nebulosidade, chuvas e neve, constituem um fator limitante. A eficiência dos painéis solares está diretamente correlacionada com a quantidade e intensidade da radiação solar disponível. Em regiões com menor incidência solar ou longos períodos de baixa luminosidade, observa-se uma redução na produção de energia. A sazonalidade também exerce influência, manifestando-se com maior geração de energia solar no verão e menor no inverno, impactando a estabilidade do fornecimento energético. Além disso, ao considerar a energia solar como fonte exclusiva, torna-se evidente a intermitência inerente, sendo imperativo recorrer a baterias ou outras formas de armazenamento para manter o fornecimento contínuo. Entretanto, esse recurso implica em um aumento considerável no impacto ambiental devido à necessidade de descarte adequado das baterias, o que representa um desafio adicional.

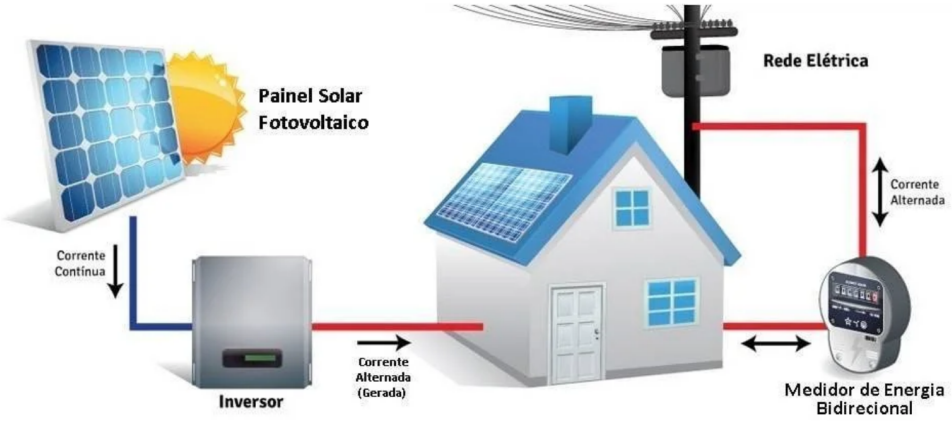
3.1 ENERGIA FOTOVOLTAICA, GERAÇÃO, MANUTENÇÃO E AVALIAÇÃO

A energia fotovoltaica constitui uma forma direta de obtenção de energia elétrica a partir da irradiação solar, valendo-se de células fotovoltaicas, também referidas como células solares, como unidade fundamental. Tais células, compostas primariamente por semicondutores, notadamente o silício, convertem a energia solar em eletricidade por meio da interação dos fótons luminosos com os elétrons do material semicondutor, desencadeando a emissão desses elétrons do estado de repouso. Esse processo origina uma corrente elétrica contínua (CC), decorrente da diferença de potencial gerada pela interação mencionada. Posteriormente, as células fotovoltaicas são agregadas e encapsuladas em módulos solares ou painéis solares, cujo design visa proporcionar uma saída elétrica em termos de corrente e tensão específicas.

Com o intuito de integrar a eletricidade produzida pelos painéis fotovoltaicos nos dispositivos elétricos convencionais, torna-se imperativa a conversão da corrente contínua (CC) em corrente alternada (CA). Este processo é desempenhado por meio de um dispositivo denominado inversor, cuja função primordial é adequar a eletricidade produzida para atender à maioria das demandas das aplicações elétricas tradicionais. Além disso, a eletricidade proveniente dos painéis solares pode ser consumida localmente, direcionada à integração na rede elétrica ou armazenada em baterias para utilização em momentos subsequentes. A aplicação resultante alimenta uma ampla gama de dispositivos, abrangendo desde eletrodomésticos até estações de carregamento para veículos elétricos.

A figura 12, demonstra a nível de componentes, como é feita a geração, conversão e distribuição de energia por meio de painéis fotovoltaicos.

Figura 12 – Instalação macro de um sistema fotovoltaico

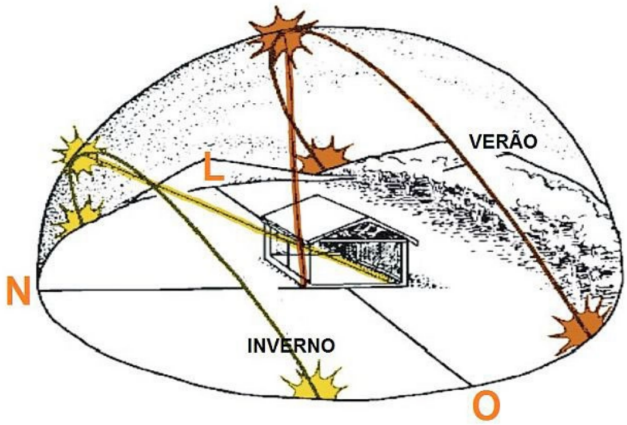


Fonte: retirado de https://mitratech.com.br/como-funciona-energia-solar-fotovoltaica/

A energia fotovoltaica se configura como uma fonte de energia renovável e limpa, desempenhando um papel significativo na mitigação das emissões de gases de efeito estufa e na promoção da sustentabilidade ambiental. A incessante evolução desta tecnologia tem como meta a ampliação da eficiência e a redução dos custos associados, propiciando maior acessibilidade e disseminação global.

A adequada implantação de painéis fotovoltaicos, demanda uma análise cuidadosa da orientação, inclinação e localização geográfica. Estes elementos revestem-se de fundamental importância para otimizar a eficiência e o desempenho do sistema, levando em consideração as particularidades climáticas da região. A orientação dos painéis deve visar a máxima exposição à luz solar. No hemisfério norte, a direção ótima é voltada para o sul; enquanto no hemisfério sul, a orientação voltada para o norte é mais indicada. Essa disposição visa captar ao máximo a luz solar durante o dia, com impacto direto na otimização da produção de energia. A intensidade da radiação solar recebida é diretamente influenciada pela altitude e latitude do local. Regiões em altitudes mais elevadas recebem uma radiação solar mais intensa, o que pode ter um impacto na inclinação ideal dos painéis. Além disso, a latitude do local também afeta o ajuste da inclinação dos painéis, sendo necessário um ajuste compensatório para otimizar a absorção da energia solar. Em locais com alta incidência solar, uma inclinação próxima à latitude pode ser eficaz. Nesse cenário, os painéis ajustáveis têm a capacidade de adaptar a inclinação de acordo com as variações sazonais. A figura 13 ilustra de forma gráfica o modo de avaliação para a correta escolha de posicionamento do mesmo.

Figura 13 – Orientação ótima de um painel solar



Fonte: retirado de https://www.solavistaenergy.com.br/a-posicao-do-painel-solar-no-telhado-pode-influenciar-na-producao-de-energia

A minimização de sombreamento representa aspecto crucial para a eficiência dos painéis. Portanto, é imperativo posicionar os painéis em locais onde não haja sombreamento significativo de vegetação, edificações ou outras estruturas, especialmente durante os períodos de maior incidência de radiação solar. O posicionamento estratégico, levando em consideração a topografia e as possíveis obstruções, assume um papel fundamental. Terrenos mais elevados ou áreas desobstruídas possibilitam a instalação ideal dos painéis, potencializando a captação de luz solar. As condições climáticas locais devem ser consideradas para assegurar a adequada aclimatação dos painéis. Em regiões com temperaturas elevadas, é essencial contemplar estratégias de dissipação de calor dos painéis, visando garantir o desempenho ótimo do sistema. Considera-se, adicionalmente, condições climáticas extremas, tais como tempestades, neve ou ventos intensos. Tal previsão visa garantir a durabilidade e a longevidade do sistema fotovoltaico, além de minimizar possíveis impactos advindos desses eventos.

Os painéis solares possuem uma vida útil estimada de 25 a 30 anos, podendo continuar a gerar energia útil mesmo após esse período, embora possa haver uma leve redução na eficiência ao longo dos anos. Para assegurar um desempenho eficaz e duradouro desses sistemas, a implementação de manutenções adequadas é crucial. A manutenção apropriada dos painéis solares engloba uma série de aspectos essenciais. Em primeiro lugar, é crucial realizar limpezas regulares para garantir a absorção ideal da luz solar, uma vez que impurezas e sujeira podem prejudicar a eficiência do sistema. Adicionalmente, inspeções visuais periódicas são fundamentais para detectar possíveis danos físicos, como arranhões ou rachaduras, os quais podem interferir no desempenho. Caso tais danos sejam observados, intervenções de reparo devem ser executadas prontamente. Outro aspecto significativo é a manutenção das conexões elétricas, que precisam ser checadas e ajustadas regularmente para prevenir falhas no sistema. O monitoramento do desempenho também é essencial, e a implementação de sistemas de monitoramento pode auxiliar no acompanhamento constante do sistema solar, possibilitando a detecção precoce de eventuais problemas. Além disso, os inversores, que desempenham a conversão da corrente contínua para

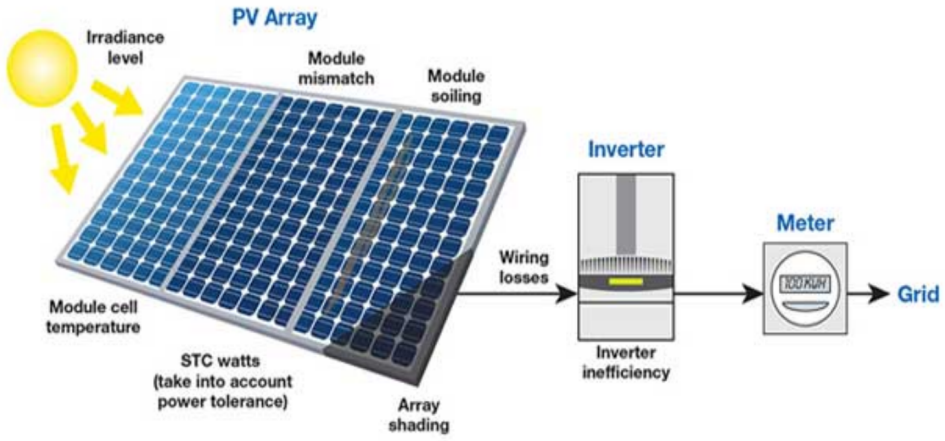
a corrente alternada, necessitam de manutenção periódica para garantir seu funcionamento adequado. A estrutura de montagem dos painéis também requer inspeções frequentes para garantir sua segurança e estabilidade. Em caso de desgaste ou falhas identificadas em qualquer componente, a substituição imediata se configura como uma medida preventiva necessários para o correto funcionamento e, adicionalmente por questões de segurança .

Resumidamente, a implementação de um plano de manutenção abrangente, contemplando desde a limpeza regular até a verificação de componentes e conexões elétricas, revela-se essencial para garantir o funcionamento adequado e duradouro dos painéis solares, otimizando sua vida útil e contribuindo para a eficiência e sustentabilidade do sistema energético.

Já em se tratando de mensuração, a energia produzida por meio de painéis fotovoltaicos se concretiza por intermédio de um instrumento denominado "medidor de energia bidirecional"ou "medidor bidirecional de energia". Este dispositivo é instalado no local onde ocorre a geração de energia solar, geralmente em proximidade aos painéis solares ou no inversor, registrando tanto a energia gerada pelo sistema fotovoltaico quanto a energia consumida da rede elétrica. O medidor bidirecional registra a quantidade de eletricidade produzida pelos painéis solares em termos de kilowatt-hora (kWh), possibilitando a medição da energia utilizada localmente e, em caso de excedente, a energia injetada de volta na rede elétrica. A avaliação da qualidade da energia gerada pelos painéis fotovoltaicos é realizada por meio de parâmetros elétricos e técnicos, garantindo a conformidade da energia com os requisitos de qualidade estabelecidos. Aspectos cruciais para essa avaliação incluem a tensão e frequência, as quais devem estar dentro dos limites aceitáveis conforme as normas locais ou internacionais. A presença de harmônicos na forma de onda da energia gerada deve ser controlada dentro dos parâmetros estabelecidos, visando assegurar um fornecimento de energia de alta qualidade. O fator de potência, próximo de 1, indica uma eficiência adequada na utilização da energia gerada. A forma de onda senoidal é essencial para garantir a compatibilidade e a segurança dos dispositivos elétricos conectados à rede. Adicionalmente, a estabilidade da tensão e flutuações mínimas são necessárias para garantir a confiabilidade do fornecimento de energia, bem como a Distorção Harmônica Total (THD) deve permanecer dentro dos limites aceitáveis para evitar danos nos equipamentos elétricos e eletrônicos.

A figura 14, ilustra múltiplas condições que podem afetar um sistema fotovoltaico em termos de geração/medição. Observamos o meter, foco principal desta etapa, normalmente caracterizado por voltímetros e/ou amperímetros serve para medir a corrente e tensão de geração. Sendo que, quando se torna necessária uma medição mais precisa ou, o acompanhamento da onda de geração como um todo, é usado um Osciloscópio para tal.

Figura 14 – Algumas condições que afetam a geração/medição em sistemas fotovoltaicos



Fonte: retirado de https://www.saurenergy.com/solar-energy-blog/here-is-how-you-can-calculate-the-annual-solar-energy-output-of-a-photovoltaic-system

A avaliação desses parâmetros é conduzida utilizando equipamentos de medição especializados, como analisadores de energia, osciloscópios e medidores de qualidade de energia. As normas e regulamentações específicas de cada país estabelecem os requisitos e padrões para a qualidade da energia gerada por sistemas fotovoltaicos, assegurando, assim, a segurança e eficiência no fornecimento de energia elétrica proveniente da energia solar. Este rigor normativo contribui para a consolidação de um fornecimento de energia solar seguro e eficaz.

3.2 MATERIAIS

A base de dados *Solar Energy Power Generation*, selecionada para este trabalho, pode ser acessada na plataforma *Kaggle* [1]. Tal plataforma online é reconhecida por armazenar dados de competições na área de ciência de dados e aprendizado de máquina, oferecendo um ambiente global que permite a participação de profissionais e entusiastas em desafios práticos. Favore ainda a troca de conhecimentos, o acesso a conjuntos de dados e a colaboração em projetos por meio de ferramentas como *Jupyter Notebook*, hospedados em nuvem e recursos educacionais gratuitos.

Como linguagem de programação, a escolha do *Python* para o desenvolvimento de soluções em *machine learning* é fundamentada nos trabalhos precedentes da área e prevalente na comunidade de ciência de dados e aprendizado de máquina. Python lidera como a linguagem de programação mais prevalente em 2021 para atividades como mineração de dados, aprendizado

[1] https://www.kaggle.com/datasets/stucom/solar-energy-power-generation-dataset

de máquina e aplicativos de aprendizado profundo, sendo amplamente empregado tanto em contextos de pesquisa quanto de produção, abrangendo projetos de diversas escalas(SCHNEIDER, 2022). Existem bibliotecas escritas em linguagem Python para todas etapas do processo de aprendizado de máquina, tornando-a uma candidata para projetos nesta área. Em decorrência destes recursos, ela possui ampla aceitação e à presença de uma comunidade ativa e colaborativa, proporcionando um suporte robusto, documentação acessível e uma riqueza de recursos para os projetos nesse domínio. *Python* oferece uma ampla gama de bibliotecas especializadas, como *TensorFlow*, *Scikit-learn* e *Keras*, *NumPy MatPlotLib* simplificando a implementação eficiente e descomplicada de algoritmos complexos. Sua sintaxe clara, legibilidade e simplicidade aceleram o ciclo de desenvolvimento, permitindo que os desenvolvedores se concentrem no cerne do problema. A flexibilidade e versatilidade do *Python* também são notáveis, viabilizando a integração com outras linguagens e sistemas, aspecto crucial ao incorporar algoritmos de aprendizado de máquina em diversas aplicações, como aquelas destinadas à web, dispositivos móveis e automação. Além disso, o *Python* proporciona bibliotecas robustas de visualização de dados, a exemplo de *Matplotlib*, *Seaborn* e *Plotly*, que auxiliam na análise e interpretação eficaz dos resultados obtidos por meio do aprendizado de máquina. Além de sua relevância no aprendizado de máquina, o *Python* desempenha um papel fundamental na análise de dados, sendo essencial em etapas prévias à concepção de modelos de aprendizado de máquina. Bibliotecas como Pandas, *NumPy* e *SciPy* são notáveis neste contexto. O ecossistema de aprendizado de máquina em *Python* está em contínua expansão, incorporando atualizações e avanços tecnológicos de forma frequente, mantendo-se à vanguarda da inovação. Portanto, a escolha do *Python* como linguagem de programação para o aprendizado de máquina é respaldada pela sua popularidade, pela ampla variedade de bibliotecas especializadas, pela facilidade de uso, pela flexibilidade e pelo sólido suporte colaborativo da comunidade, consolidando-o como uma escolha sólida e confiável para os profissionais que buscam explorar e implementar soluções inovadoras no âmbito do aprendizado de máquina.(PYTHON, 2024)

Dentre as bibliotecas selecionadas para o desenvolvimento do software, a biblioteca *Keras* para a implementação de *Autoencoders* encontra sua fundamentação em diversos fatores que a tornam uma ferramenta altamente adequada para esta finalidade. Primeiramente, *Keras* é uma biblioteca de código aberto amplamente adotada na comunidade de aprendizado de máquina e ciência de dados, conhecida por sua simplicidade e eficiência no desenvolvimento de redes neurais, incluindo arquiteturas de *Autoencoders*. Sua sintaxe intuitiva e modular permite a construção e configuração de redes complexas de maneira descomplicada, o que é especialmente relevante para a elaboração de *Autoencoders*, que necessitam de camadas customizadas e uma arquitetura específica. Além disso, *Keras* possui uma vasta gama de funcionalidades e otimizações, facilitando a implementação de diferentes variantes de *Autoencoders*, como *variational Autoencoders* (VAEs) e *sparse Autoencoders*, ampliando assim as possibilidades de pesquisa e experimentação. Ademais, *Keras* integra-se perfeitamente com outros *frameworks* de aprendizado de máquina, como *TensorFlow* e Theano, proporcionando um ambiente de de-

senvolvimento unificado e robusto. Dessa forma, a seleção da biblioteca *Keras* para a implementação de *Autoencoders* se embasa na sua eficiência, flexibilidade e compatibilidade com outras ferramentas de aprendizado de máquina, culminando na viabilização de pesquisas e aplicações voltadas para a eficaz representação e reconstrução de dados latentes.(KERAS, 2024)

Outra das principais bibliotecas selecionadas, a *Matplotlib* se justifica primariamente pela sua eficiência e versatilidade no âmbito da visualização de dados no contexto da pesquisa em ciência de dados e áreas afins. *Matplotlib* é uma ferramenta de código aberto amplamente reconhecida e consolidada na comunidade científica, possibilitando a criação de uma ampla gama de visualizações gráficas de alta qualidade, incluindo gráficos de dispersão, histogramas, barras, linhas, entre outros. Essa diversidade de representações gráficas é crucial para a análise exploratória dos dados, fornecendo aos pesquisadores a capacidade de apresentar informações complexas de forma clara e compreensível. Além disso, a customização detalhada dos gráficos oferecida pelo *Matplotlib* permite a adequação das visualizações às necessidades específicas de cada projeto, atendendo aos requisitos de apresentação e interpretação de dados de maneira precisa e informativa. Sua integração com outras bibliotecas e *frameworks* amplamente utilizados na comunidade de ciência de dados, como *NumPy* e Pandas, aprimora a eficiência e a praticidade no tratamento e exibição de dados. Portanto, a utilização da biblioteca *Matplotlib* reafirma-se como uma escolha pertinente e valiosa na materialização da análise exploratória de dados, proporcionando a representação gráfica de informações de forma eficaz e instrumental para o avanço do conhecimento.(MATPLOTLIB, 2024)

Por fim, como a última biblioteca julgada como principal e vital para o desenvolvimento, o uso da *NumPy* se justifica como uma escolha fundamental para o desenvolvimento de soluções em aprendizado de máquina e ciência de dados. *NumPy* é uma biblioteca de código aberto em *Python* amplamente reconhecida e adotada, notável pela sua eficiência na manipulação e processamento de arrays multidimensionais e estruturas de dados numéricos. Esta eficiência advém da implementação de operações matemáticas otimizadas, essenciais para a execução de algoritmos complexos, sendo um recurso crucial na construção e análise de modelos de aprendizado de máquina. Além disso, *NumPy* facilita a integração e a interoperabilidade com outras bibliotecas importantes utilizadas na análise de dados, como Pandas, *Scikit-learn* e *Matplotlib*, proporcionando um ecossistema coeso para o desenvolvimento de soluções completas. A sua ampla aceitação na comunidade científica e a vasta documentação disponível enriquecem a base de conhecimento e a compreensão, consolidando-a como uma ferramenta valiosa na realização de análises exploratórias e implementações eficazes em pesquisas científicas. Assim, o uso de *NumPy* se destaca como uma opção essencial para cientistas de dados e pesquisadores, contribuindo significativamente para a excelência e a precisão nas análises e modelagens de dados.(NUMPY, 2024)

A escolha do ambiente de desenvolvimento *Visual Studio* se fundamenta na sua robustez e abrangência de ferramentas que otimizam o processo de codificação. O *Visual Studio*, desen-

volvido pela Microsoft, oferece uma ampla gama de recursos integrados, incluindo editores de código poderosos, depuradores, perfis de desempenho, além de suporte eficiente a diferentes linguagens de programação, como C++, C#, *Python* e muitas outras. Essa versatilidade é fundamental para acomodar as variadas demandas de projetos, possibilitando a escrita de código eficiente e de alta qualidade em diferentes contextos. Além disso, o *Visual Studio* se destaca pela sua interface amigável e intuitiva, que contribui para uma experiência de desenvolvimento fluente e produtiva. A integração com sistemas de controle de versão, a facilidade de colaboração e a disponibilidade de extensões e *plugins* são elementos adicionais que enriquecem o ambiente de desenvolvimento, promovendo maior eficiência e facilitando a manutenção e gestão de projetos. Dessa forma, a escolha do *Visual Studio* se apresenta como uma opção sólida para codificação, proporcionando um ecossistema completo que atende às necessidades de desenvolvedores e favorece o sucesso e a qualidade dos projetos de software.(VISUALSTUDIOCODE, 2024)

A escolha do aplicativo *iTerm* para codificação se baseia na sua qualidade e funcionalidades aprimoradas, contribuindo significativamente para um ambiente de desenvolvimento mais eficiente e produtivo. O *iTerm* é um terminal para *MacOS* que oferece uma interface amigável e poderosa para interação com o sistema operacional e execução de comandos. Sua capacidade de dividir a tela em várias sessões proporciona a execução simultânea de múltiplos processos, o que é particularmente útil para desenvolvedores ao lidar com várias tarefas de codificação ao mesmo tempo. Além disso, o *iTerm* oferece recursos avançados, como busca em histórico, atalhos de teclado personalizáveis, configurações de aparência e suporte a guias, aprimorando a experiência do usuário e tornando o fluxo de trabalho mais flexível e adaptável às necessidades individuais. Adicionalmente, sua integração com *plugins* e extensões, bem como a compatibilidade com uma variedade de *shells* e linguagens de *script*, ampliam suas capacidades e funcionalidades, enriquecendo a experiência de desenvolvimento. Portanto, a utilização do *iTerm* para codificação se justifica pela sua eficiência, versatilidade e contribuição para um ambiente de desenvolvimento mais organizado e produtivo, atendendo às demandas dos profissionais que buscam aprimorar suas práticas de codificação.(ITERM, 2024)

O uso de um MacBook para atividades de desenvolvimento é respaldada por diversos atributos que o tornam uma plataforma altamente adequada e confiável. Os produtos da Apple, conhecidos pela sua qualidade de construção e design inovador, apresentam uma integração única entre hardware e software. No caso do MacBook, o sistema operacional *MacOS* oferece um ambiente robusto e estável para desenvolvedores(APPLEMB, 2024), fornecendo um ecossistema coeso e eficiente para programação. Além disso, a performance e potência dos componentes de hardware presentes nos MacBooks, como processadores avançados e armazenamento de alta velocidade, permitem a execução suave e eficaz de aplicativos e ferramentas de desenvolvimento, acelerando o ciclo de desenvolvimento. A experiência do usuário, marcada pela interface intuitiva, fluidez e segurança, também se traduz em produtividade aprimorada durante o desenvolvimento de projetos. A presença de um ecossistema de desenvolvimento rico, in-

cluindo ferramentas como *Xcode* e Terminal, contribui para um fluxo de trabalho integrado e eficaz, essencial para desenvolvedores que buscam eficiência e alta performance em suas atividades. Portanto, a opção pelo MacBook para o desenvolvimento se baseia em sua combinação única de hardware e software, proporcionando uma experiência confiável e otimizada para profissionais de desenvolvimento em busca de excelência e eficiência em suas tarefas.(APPLEOS, 2024)

3.3 MÉTODO

A pesquisa adota uma abordagem de natureza exploratória. A pesquisa exploratória representa uma etapa preliminar essencial no início de um processo de investigação científica. Tem por escopo promover uma compreensão inicial e ampla de um determinado tema ou fenômeno, visando identificar lacunas de conhecimento, padrões, tendências e variáveis relevantes que demandam uma análise mais aprofundada. Este estágio é de suma importância para delimitar o escopo e a trajetória da pesquisa subsequente. A utilização de métodos como revisão bibliográfica, entrevistas e observação, aliada à análise de dados preexistentes, contribui para o levantamento de informações e *insights* fundamentais que direcionarão a formulação de hipóteses e o delineamento da pesquisa de forma mais detalhada e conclusiva. (GIL *et al.*, 2002)

Em seguida foi realizada a revisão sistemática, tratando-se de uma metodologia rigorosa utilizada no âmbito da pesquisa acadêmica, com o objetivo de sintetizar e analisar minuciosamente as evidências já existentes relacionadas a uma questão de pesquisa específica. Essa modalidade de revisão segue um protocolo predefinido, incorporando critérios de inclusão e exclusão criteriosamente delineados, estratégias de busca pormenorizadas, avaliação criteriosa da qualidade dos estudos incluídos e análise estatística ou qualitativa dos dados. O intuito é agregar e consolidar as descobertas provenientes de múltiplas fontes de pesquisa, proporcionando uma visão holística e imparcial do tema em análise. A revisão sistemática se mostra de extrema relevância para embasar decisões fundamentadas e impulsionar o avanço do conhecimento em uma determinada área, ao integrar de maneira objetiva o estado atual da evidência científica disponível.

Ao se falar de modelos de inteligência artificial (IAs) múltiplas métricas podem ser utilizadas na avaliação da performance. As principais consistem em: acurácia, precisão, erro médio absoluto e quadrático. A acurácia é uma métrica amplamente usada para avaliar proporção de instâncias corretamente classificadas em relação ao número total de instâncias. A precisão mede a proporção de previsões positivas corretas em relação ao número total de previsões positivas. Essas métricas são calculadas a partir da matriz de confusão, que permite visualizar os verdadeiros positivos, falsos positivos, verdadeiros negativos e falsos negativos, proporcionando uma visão do desempenho do modelo classificador. O erro médio absoluto percentual (Mean Absolute Percentage Error - MAPE) é uma métrica de precisão que calcula a média das diferenças absolutas entre as previsões e os valores reais, expressa como uma porcentagem do valor real.

O erro quadrático médio da raiz (Root Mean Square Error - RMSE) é uma métrica de avaliação de modelos que estima a média das diferenças entre previsões e valores reais, com os erros quadrados sendo somados, extraindo-se a raiz quadrada. Estas métricas fornecem uma medida da dispersão dos erros.

3.4 PERCURSO METODOLÓGICO

Esse trabalho caracteriza-se como uma pesquisa de natureza exploratória, que busca investigar, compreender e aplicar técnicas de *DL* no contexto da detecção de falhas, tais como eventuais diminuições na geração de energia, em painéis fotovoltaicos. Para isso, o trabalho seguiu um método composto pelas seguintes cinco etapas:

- Etapa 1: Identificação de *dataset*
- Etapa 2: Pré-processamento dos dados
- Etapa 3: Aplicação de algoritmos com *Autoencoder*
- Etapa 4: Avaliação dos resultados obtidos
- Etapa 5: Indicação do melhor modelo para o problema de detecção de falhas

As seções a seguir descrevem o desenvolvimento de cada uma destas etapas.

3.4.1 Etapa 1: Identificação de *dataset*

O *dataset* utilizado neste estudo foi selecionada por meio de uma pesquisa realizada no site *Kaggle*, que é uma plataforma conhecida por disponibilizar uma ampla variedade de *datasets* abrangendo uma vasta gama de temas, tópicos e formatos. Este *dataset* é identificado no site sob o título *Solar energy power generation dataset* [2].

O *dataset* em questão contém informações sobre a quantidade de energia gerada por um painel solar em um período de tempo, bem como as condições às quais o painel estava submetido no momento da geração. Essas condições incluem variáveis como temperatura, umidade, velocidade e direção do vento, além de outras características relevantes para a geração de energia solar. O quadro 2 indica as variáveis, o que significam e as principais características dos seus valores.

[2] https://www.kaggle.com/datasets/anikannal/solar-power-generation-data

Quadro 2 – Descrição dos atributos do *dataset*

Atributo	Significado	Valor Min	Valor Max	Média	Desvio padrão
temperature 2 m above gnd	Temperatura medida a dois metros acima do solo	-5,3	34,9	15,6	8,85
relative humidity 2 m above gnd	Umidade relativa dois metros acima do solo	7%	100%	51,36	23,5%
mean sea level pressure MSL	Média de pressão ao nivel do mar	997,5	1046,0	1019,33	7,02
total precipitation sfc	Precipitação média	0	3,2	0,03	0,17
snowfall amount sfc	Quantidade de neve	0	1,8	0,002	0,03
total cloud cover	Cobertura total de nuvens	0%	100%	34,05	42,84%
high cloud cover high cld lay	Cobertura total de nuvens em alta altitude	0%	100%	14,45	30,71%
Medium cloud cover high cld lay	Cobertura total de nuvens em media altitude	0%	100%	0,02	36,38%
Low cloud cover high cld lay	Cobertura total de nuvens em baixa altitude	0%	100%	21,37	38,01%
shortwave radiation backwards	Radiação de ondas curtas inversa	0	952,3	387,75	278,45
wind speed 10 m above gnd	Velocidade do vento dez metros acima do solo	0	61,18	16,22	9,87
wind direction 10 m above gnd	Direção do vento dez metros acima do solo	0,54	360	195,07	106,62
wind gust 10 m above gnd	Velocidade das rajadas de vento dez metros acima do solo	0,72	84,96	20,58	12,64
wind speed 80 m above gnd	Velocidade do vento oitenta metros acima do solo	0	66,88	18,97	11,99
wind direction 80 m above gnd	Direção do vento oitenta metros acima do solo	1,12	360	91,16	108,76
wind speed 900 m above gnd	Velocidade do vento novecentos metros acima do solo	0	61,11	16,36	9,88
wind direction 900 m above gnd	Direção do vento novecentos metros acima do solo	1,12	360	192,44	106,51
angle of incidence	Angulo de Incidência dos raios solares	3,75	121,63	50,83	26,63
zenith	zênite solar	17,72	128,41	59,98	19,85
azimuth	azimute solar	54,37	289,04	169,16	64,56
generated power kw	potencia gerada em Kw	0,000595	3056,79	1134,34	937,95

Fonte: O Autor (2024).

A fase inicial do processo consistiu na inicialização dos dados, uma etapa conduzida por meio da utilização da biblioteca Pandas [3]. Esta biblioteca desempenha uma função essencial ao receber um arquivo no formato .CSV e importá-lo para o ambiente de análise. Através

[3] PANDAS URL

dessa operação, o conjunto de dados é carregado e disponibilizado para manipulação e exploração dentro do ambiente de programação. Para simplificar o processo, a biblioteca é comumente inicializada sob o apelido de "pd", possibilitando a execução de comandos relacionados à biblioteca de forma mais concisa na figura 15.

A inicialização dos dados é realizada através do comando *read_csv*, que carrega o arquivo CSV para uma variável denominada "data". Esta variável é utilizada posteriormente para realizar diversas operações de manipulação e análise dos dados. A figura 16 apresenta a instrução de carga do *dataset*.

Figura 15 – Importação da biblioteca Pandas por meio do apelido PD

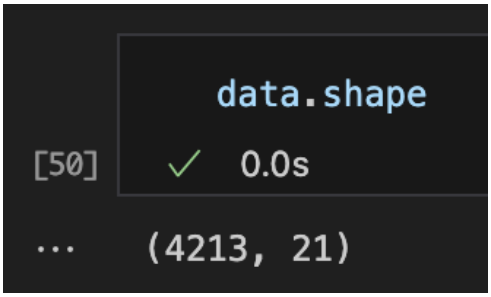
Fonte: O autor (2024)

Figura 16 – Carregamento para a variável data em formato *dataframe*

Fonte: O autor (2024)

Em seguida, foi utilizado o comando *shape*, apenas para fins de confirmação de que a base de dados possui 4231 linhas (número de instâncias) e 21 colunas (número de atributos). Na figura 17 é possível observar a execução do comando e o resultado obtido.

Figura 17 – Uso do comando *Shape* para confirmar o tamanho da matriz



Fonte: O autor (2024)

Prosseguindo com as verificações iniciais, utilizou-se o comando *columns*. Este comando é empregado para identificar os rótulos de cada coluna, caso estejam presentes. A execução dessa instrução está ilustrada na figura 18.

Figura 18 – Uso do comando *Columns* para confirmar o tamanho da matriz

```
[51]  data.columns
      ✓ 0.0s

...   Index(['temperature_2_m_above_gnd', 'relative_humidity_2_m_above_gnd',
             'mean_sea_level_pressure_MSL', 'total_precipitation_sfc',
             'snowfall_amount_sfc', 'total_cloud_cover_sfc',
             'high_cloud_cover_high_cld_lay', 'medium_cloud_cover_mid_cld_lay',
             'low_cloud_cover_low_cld_lay', 'shortwave_radiation_backwards_sfc',
             'wind_speed_10_m_above_gnd', 'wind_direction_10_m_above_gnd',
             'wind_speed_80_m_above_gnd', 'wind_direction_80_m_above_gnd',
             'wind_speed_900_mb', 'wind_direction_900_mb',
             'wind_gust_10_m_above_gnd', 'angle_of_incidence', 'zenith', 'azimuth',
             'generated_power_kw'],
            dtype='object')
```

Fonte: O autor (2024)

Após uma primeira análise visual dos dados, procede-se com a utilização de comandos que visam analisar os dados. Nesse contexto, o comando *info* emerge como uma ferramenta útil da biblioteca Pandas para extrair informações sobre as colunas, fornece a contagem de elementos não nulos em cada coluna, bem como o formato dos dados. A figura 19 apresenta o uso desta instrução.

Figura 19 – Uso do comando *info()* para exibir informações mais técnicas sobre a tabela e suas colunas

```
data.info()
```
[52] ✓ 0.0s

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4213 entries, 0 to 4212
Data columns (total 21 columns):
 #   Column                            Non-Null Count  Dtype
---  ------                            --------------  -----
 0   temperature_2_m_above_gnd         4213 non-null   float64
 1   relative_humidity_2_m_above_gnd   4213 non-null   int64
 2   mean_sea_level_pressure_MSL       4213 non-null   float64
 3   total_precipitation_sfc           4213 non-null   float64
 4   snowfall_amount_sfc               4213 non-null   float64
 5   total_cloud_cover_sfc             4213 non-null   float64
 6   high_cloud_cover_high_cld_lay     4213 non-null   int64
 7   medium_cloud_cover_mid_cld_lay    4213 non-null   int64
 8   low_cloud_cover_low_cld_lay       4213 non-null   int64
 9   shortwave_radiation_backwards_sfc 4213 non-null   float64
 10  wind_speed_10_m_above_gnd         4213 non-null   float64
 11  wind_direction_10_m_above_gnd     4213 non-null   float64
 12  wind_speed_80_m_above_gnd         4213 non-null   float64
 13  wind_direction_80_m_above_gnd     4213 non-null   float64
 14  wind_speed_900_mb                 4213 non-null   float64
 15  wind_direction_900_mb             4213 non-null   float64
 16  wind_gust_10_m_above_gnd          4213 non-null   float64
 17  angle_of_incidence                4213 non-null   float64
 18  zenith                            4213 non-null   float64
 19  azimuth                           4213 non-null   float64
 20  generated_power_kw                4213 non-null   float64
dtypes: float64(17), int64(4)
memory usage: 691.3 KB
```

Fonte: O autor (2024)

Prosseguindo, foi utilizado o comando *describe()* para obter uma descrição de cada coluna. Este comando proporciona uma ampla gama de características estatísticas para cada coluna, incluindo o número de elementos, média dos valores, desvio padrão, valor mínimo, percentis 25%, 50% e 75%, bem como o valor máximo. O resultado da execução deste comando pode ser parcialmente visualizado na figura 20.

Figura 20 – Uso do comando *Describe()* para exibir informações mais estatísticas sobre as colunas

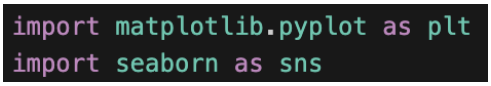
```
# columns basic description
data.describe()
```
[53] ✓ 0.0s Python

	temperature_2_m_above_gnd	relative_humidity_2_m_above_gnd	mean_sea_level_pressure_MSL	total_precipitation_sfc	snowfall_amount_sfc	total_cloud_cover_sfc	high_cloud_cover_high_cld_lay
count	4213.000000	4213.000000	4213.000000	4213.000000	4213.000000	4213.000000	4213.000000
mean	15.068111	51.361025	1019.337812	0.031759	0.002808	34.056990	14.458818
std	8.853677	23.525864	7.022867	0.170212	0.038015	42.843638	30.711707
min	-5.350000	7.000000	997.500000	0.000000	0.000000	0.000000	0.000000
25%	8.390000	32.000000	1014.500000	0.000000	0.000000	0.000000	0.000000
50%	14.750000	48.000000	1018.100000	0.000000	0.000000	8.700000	0.000000
75%	21.290000	70.000000	1023.600000	0.000000	0.000000	100.000000	9.000000
max	34.900000	100.000000	1046.800000	3.200000	1.680000	100.000000	100.000000

Fonte: O autor (2024)

Para gerar representações gráficas dos dados, foram empregadas as bibliotecas *Seaborn* e *Matplotlib*, inicializadas sob o nome de sns e plt, respectivamente (conforme mostrado na figura 21). Essas bibliotecas foram responsáveis por produzir os histogramas e os *scatterplots*. Além disso, para explorar as relações entre todas as variáveis, foram elaborados gráficos nos formatos de *heatmap* e *pairplot*.

Figura 21 – Inicialização da biblioteca *Seaboarn* e *MatPlotLib*



Fonte: O autor (2024)

O primeiro gráfico escolhido para ser plotado e analisado foi o histograma. Um histograma é uma representação gráfica que mostra a distribuição de frequência de um conjunto de dados. Ele é útil para visualizar padrões, identificar *outliers* e compreender a distribuição dos dados. Os picos e vales no histograma indicam onde os dados estão concentrados, tornando-se uma ferramenta essencial em estatística para análise e tomada de decisões.

Para gerar os histogramas para cada variável foi utilizado um laço de repetição do tipo *for*, no qual cada coluna do conjunto de dados foi iterada. O *loop* foi executado com todas as colunas do *dataset*. Para criar os gráficos foi utilizado o comando *histplot* da biblioteca *Seaborn*, e para a plotagem propriamente dita, o comando *figure()*. A implementação desses comandos, com as devidas parametrizações, pode ser visualizada na figura 22.

Figura 22 – Plotagem dos histogramas

Fonte: O autor (2024)

O segundo gráfico escolhido foi um *scatterplot* relacionando os pontos de todas as variáveis com os pontos da variável alvo: potência gerada em Kilowatts/hora. Inicialmente, foi empregado um laço de repetição utilizando as colunas do conjunto de dados como atributo de iteração, e o critério de finalização foi definido como todas as colunas do banco de dados. Em seguida, foi utilizado o comando *figure()* para ajustar o tamanho da figura. Posteriormente, o comando *scatterplot* foi utilizado para gerar o gráfico.

Os comandos subsequentes foram utilizados para adicionar legendas ao gráfico: o comando *title* para inserir a variável comparada com a potência gerada em Kilowatts/hora, os comandos *xlabel* e *ylabel* para definir as descrições nos eixos do gráfico. Por fim, o comando *show* foi empregado para exibir o gráfico na tela. A implementação desses comandos pode ser

visualizada na figura 23, bem como um exemplo do gráfico gerado, considerando dois atributos apenas, na figura 24. Observa-se na visualização que os pontos representam as instâncias e alguns *clusters* mais concentrados se localizam próximos aos eixos.

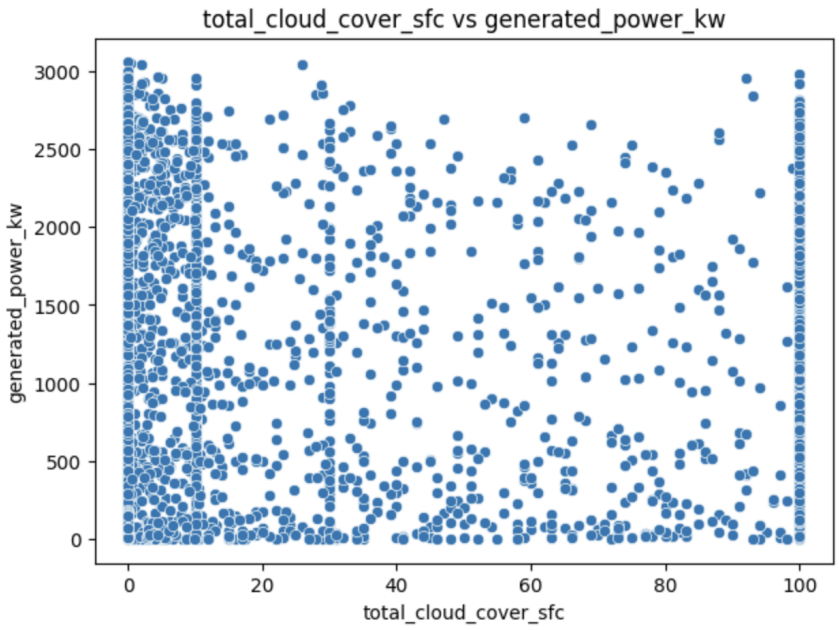
Figura 23 – Plotagem dos *scatterplot*

```
for column in data.columns:
    plt.figure(figsize=(7, 5))
    sns.scatterplot(x=data[column], y= data['generated_power_kw'])
    plt.title(f'{column} vs generated_power_kw')
    plt.xlabel(column)
    plt.ylabel('generated_power_kw')
    plt.show()

✓ 2.3s
```

Fonte: O autor (2024)

Figura 24 – Exemplo de *scatterplot*



Fonte: O autor (2024)

Para detectar correlações entre as variáveis, indicando variáveis que podem ser removidas do treinamento ou gerar algum viés no modelo, foi gerado um *heatmap* de correlação entre todas as variáveis. Para isso, inicialmente foi utilizado o comando *figure* da biblioteca *Matplotlib* com o parâmetro *figsize* para determinar o tamanho da figura. Em seguida, o comando *heatmap* da biblioteca *Seaborn* foi utilizado, tendo como principal parâmetro o comando *.corr()* da biblioteca *Pandas* para calcular a correlação entre todas as variáveis. Por fim, o comando *show* foi empregado para exibir o gráfico na tela.

Tanto o código na IDE, quanto o gráfico gerado, podem ser visualizados nas figuras 25 e 26.

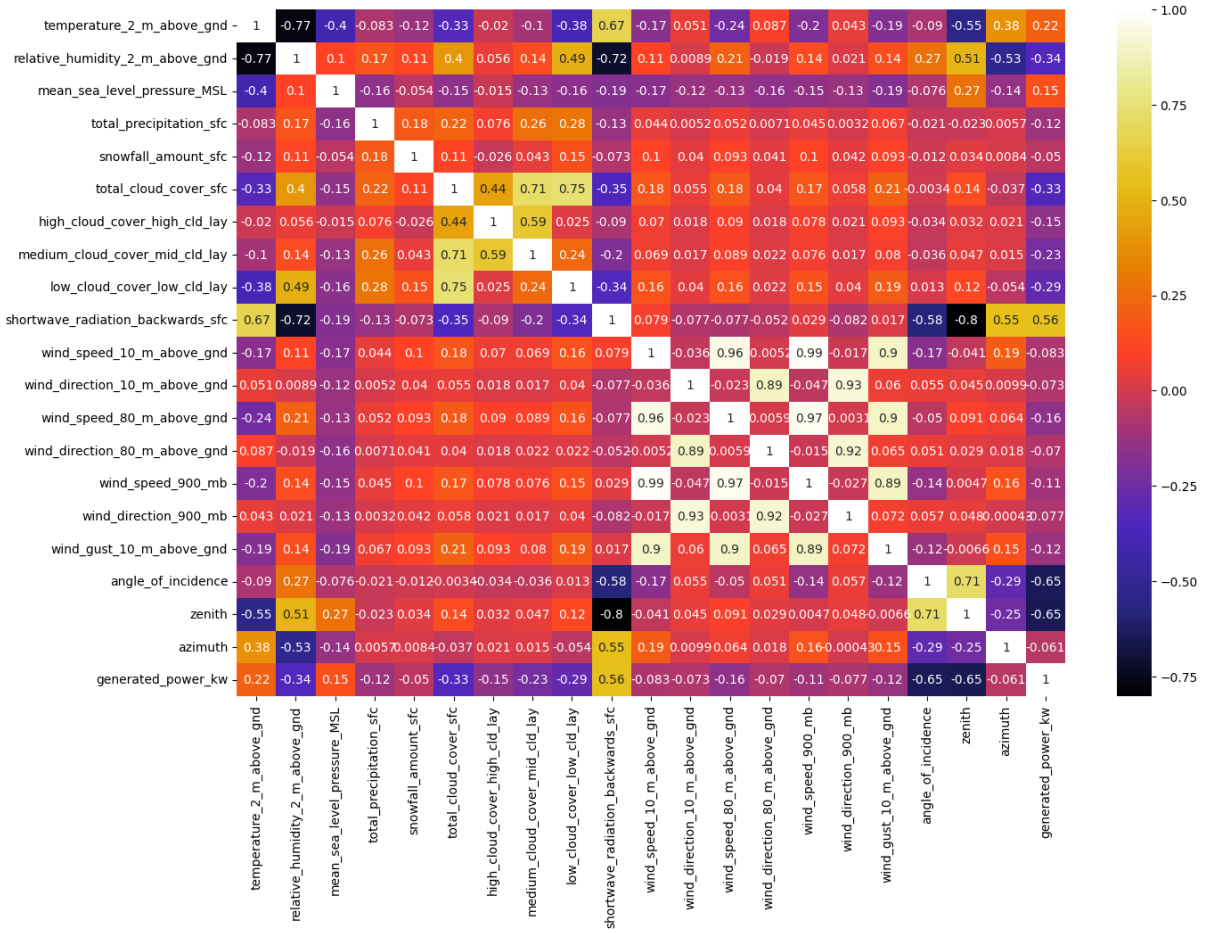
Figura 25 – Código de geração do *heatmap*

```
plt.figure(figsize = (15,10))
sns.heatmap(data.corr(), cmap = 'CMRmap', annot = True)
plt.show()
```

✓ 0.4s

Fonte: O autor (2024)

Figura 26 – *Heatmap*



Fonte: O autor (2024)

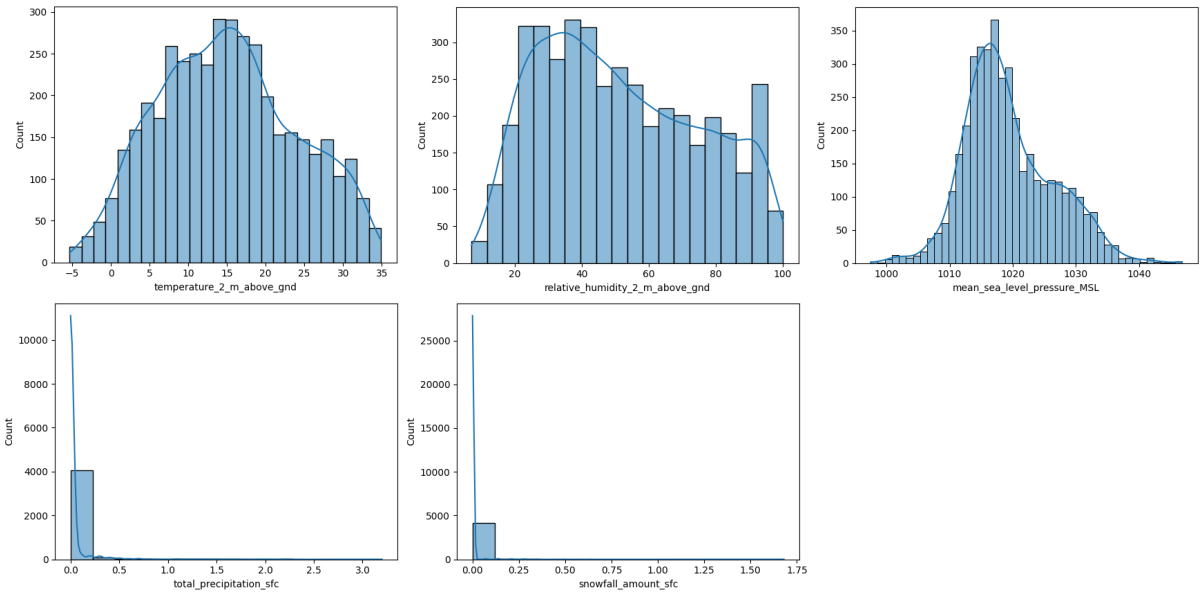
O gráfico de correlação é muito importante na análise exploratória dos dados, pois ele permite uma visão ampla sobre o comportamento dos valores dos atributos.Neste caso, optou-se por realizar uma análise dos atributos separados por categorias. Eles foram agrupados em quatro categorias distintas: clima, nuvens, vento e posição. Para cada categoria são apresentados resumos contendo os valores máximos e mínimos de cada variável, oferecendo uma visão geral das extremidades dos dados. Além disso, para cada categoria, foi compilado um conjunto de histogramas representando a distribuição das variáveis pertencentes a essa categoria. Essa

organização estruturada e detalhada proporciona uma análise mais completa dos dados, permitindo uma compreensão mais profunda de suas características e padrões. Os resumos fornecem *insights* valiosos sobre a dispersão e variação dos dados em cada categoria, facilitando a identificação de tendências e padrões relevantes para a análise e modelagem subsequentes.

Na categoria referente ao clima destaca-se os valores obtidos para cinco características determinantes. Primeiramente, considerando-se a característica da temperatura a dois metros acima do nível do solo, observa-se que seus valores variam entre -5,30 e 34,9 graus Celsius. Outra característica, a umidade relativa a dois metros acima do solo possui variação de 7 a 100%. A média da pressão atmosférica em relação ao nível do mar apresenta uma faixa de variação entre 997 e 1046 milibares. A precipitação total possui valores variando de 0 a 3,2 mm/m2. A última característica destacada é a ocorrência de queda de neve, cujos valores variam entre 0 e 1,68 cm. Por meio do diagrama de correlação 26 observa-se uma correlação moderada a forte (0.77) entre as variáveis temperatura e umidade, evidenciando uma relação significativa entre essas duas características climáticos.

A figura 27 é um agrupamento da distribuição dos valores de todas as variáveis agrupadas na categoria clima.

Figura 27 – Variáveis referentes ao clima



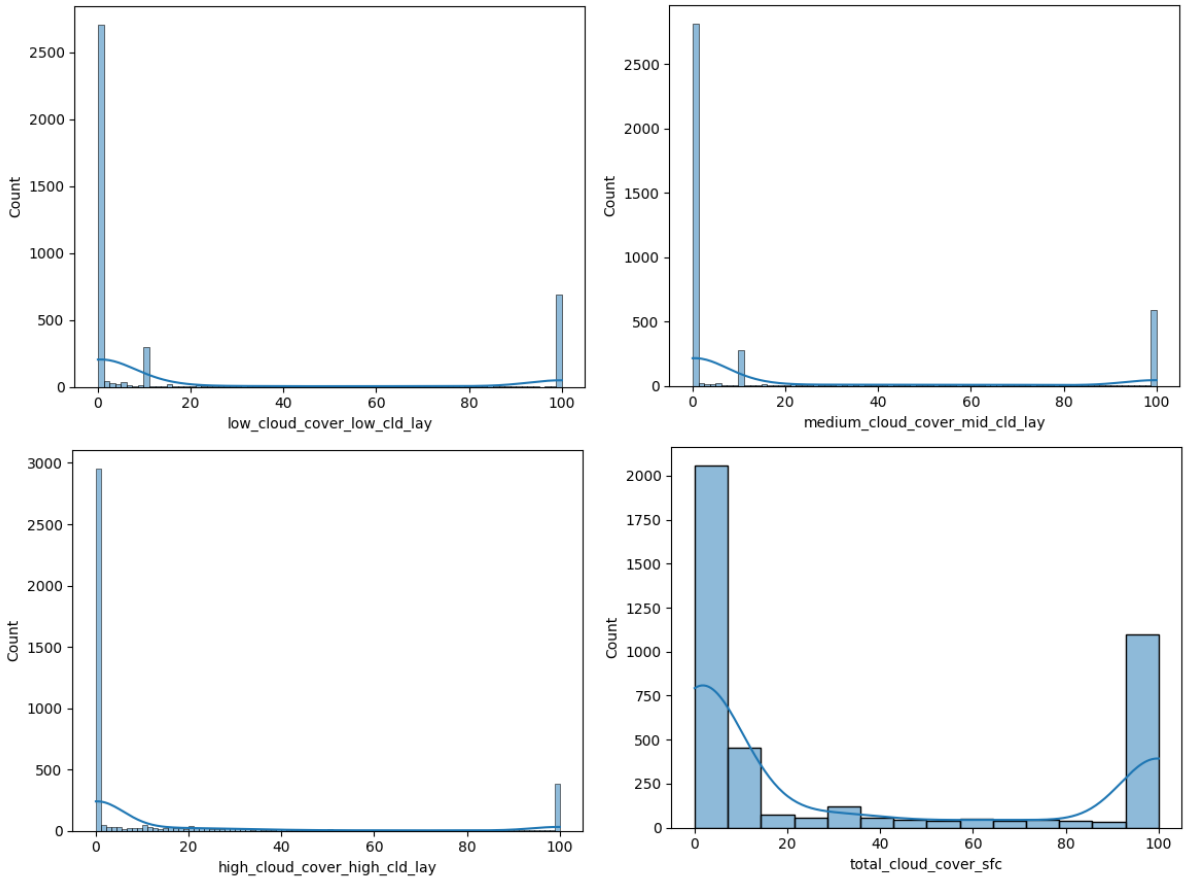
Fonte: O autor (2024)

Na classificação das nuvens, são reconhecidas quatro categorias distintas: cobertura total de nuvens, cobertura de nuvens de alta altitude, cobertura de nuvens de média altitude e cobertura de nuvens de baixa altitude, todas quantificadas em uma escala de 0 a 100%. Uma correlação interessante é observada, uma vez que é natural que a cobertura total de nuvens esteja correlacionada com as nuvens de baixa, média e alta altitude. Os valores de correlação entre a cobertura total de nuvens e cada uma dessas categorias são 0,75, 0,71 e 0,44, respectivamente.

Essa informação sugere que as principais nuvens responsáveis por bloquear a radiação solar na maioria dos dias são as de baixa e média altitude.

A figura 28 é um compilado da distribuição dos valores de todas as variáveis agrupadas na categoria nuvens.

Figura 28 – Variáveis referentes a nuvens



Fonte: O autor (2024)

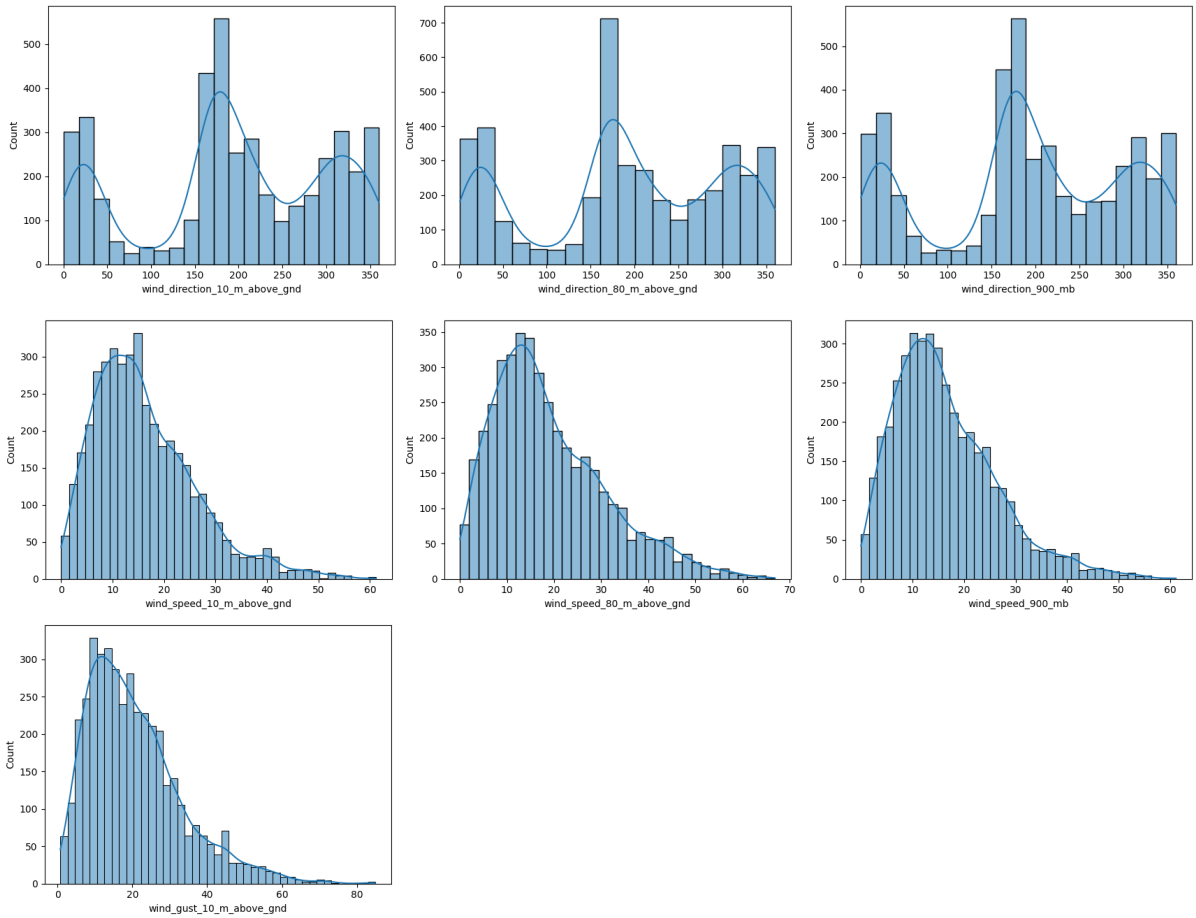
Na categoria dedicada ao vento, que abrange o maior número de categorias, totalizando sete, são contempladas todas as características relacionadas à incidência do vento. Essas características incluem: a direção do vento a uma altura de dez metros acima do nível do solo, variando de 0,53 a 360 graus; a velocidade do vento a dez metros acima do solo, variando de 0 a 61,18 km/h; a velocidade das rajadas de vento a uma altura de dez metros acima do solo, oscilando de 0,72 a 84,96 km/h; a direção do vento a uma altura de oitenta metros acima do nível do solo, variando de 1,12 a 360 graus; a velocidade do vento a uma altura de oitenta metros acima do nível do solo, variando de 0 a 66,8 km/h; a direção do vento a uma altura de novecentos metros acima do nível do solo, com variação de 1,12 a 360 graus; e a velocidade do vento a uma altura de novecentos metros acima do nível do solo, variando de 0 a 61,11 km/h.

Neste contexto, observa-se uma correlação forte da velocidade em todos os três níveis de vento, representada pela média de correlação de 0,9. Além disso, há uma correlação forte,

representada pela média de valores de 0,91, na direção do vento nos três níveis.

A figura 29 é um compilado da distribuição dos valores de todas as variáveis agrupadas na categoria vento.

Figura 29 – Variáveis referentes ao vento



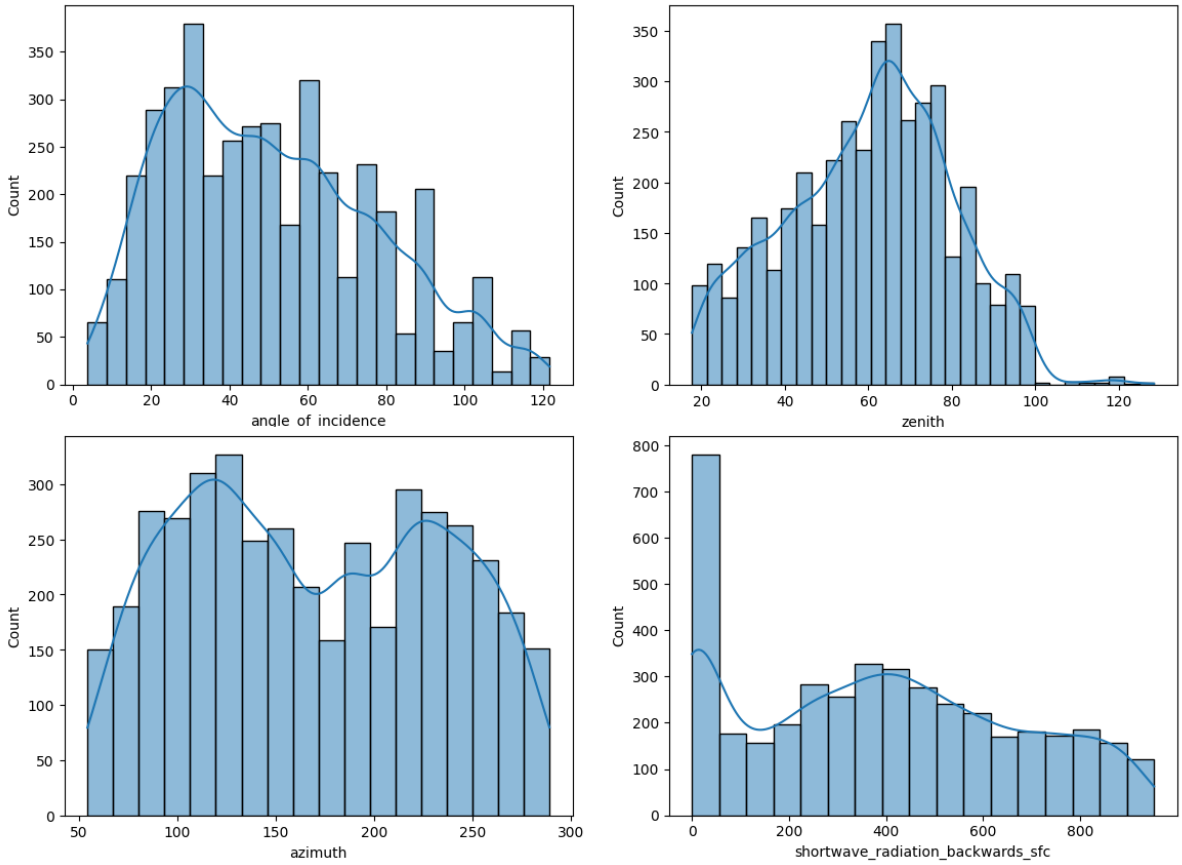
Fonte: O autor (2024)

A classificação final deste estudo aborda a posição dos painéis solares, compreendendo quatro categorias que avaliam tanto a orientação dos painéis quanto sua posição em relação ao globo terrestre. Esta análise inicia-se com a mensuração do ângulo de incidência solar, variando de 3,75 a 121,63 graus. Em seguida, é considerado o zênite solar, que representa o ângulo vertical entre a direção do Sol e a linha vertical local, registrando variação entre 17,72 e 128,41 graus em relação ao horizonte. O azimute solar, indicando a direção horizontal do Sol em relação ao norte geográfico, oscila entre 54,37 e 289,04 graus. Por fim, a análise inclui a reflexão da radiação de ondas curtas, cujos valores variam de 0 a 952,30.

Nesta categoria, observa-se uma correlação moderada a forte entre o ângulo de incidência e o zênite, representada pelo valor de 0,71.

A figura 30 é um compilado da distribuição dos valores de todas as variáveis agrupadas na categoria posição.

Figura 30 – Variáveis referentes a posição

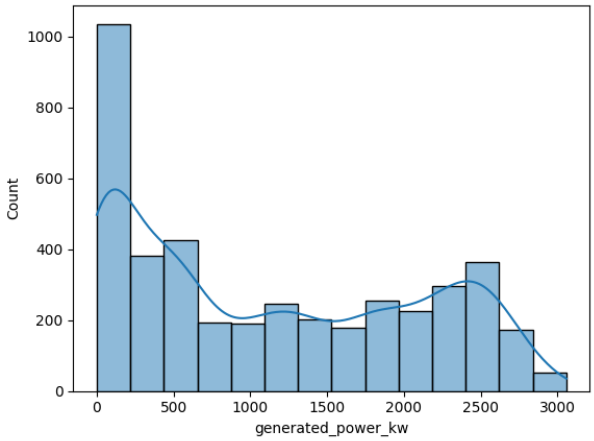


Fonte: O autor (2024)

Por fim, o dado principal deste estudo é a geração proveniente do painel solar, apresentando uma variação de 0.000595 até 3056.79 kilowatt-hora. Este é o campo que almejamos monitorar, avaliando sua eficiência, qualidade e padrão. Esta variável apresenta uma correlação máxima com outras variáveis, sendo mais notavelmente correlacionada com o ângulo de incidência e o zênite, seguido pela radiação de ondas curtas. Isso sugere que, para uma maior geração de energia solar, as principais variáveis a serem observadas são essas, conforme evidenciado pelo conjunto de dados utilizado para este estudo.

A figura 31 é a distribuição dos valores na categoria kilowatt-hora.

Figura 31 – Variável referente a geração



Fonte: O autor (2024)

3.4.2 Etapa 2: Pré-processamento dos dados

Nesta etapa é importante observar a normalização dos dados. A normalização de dados é um processo estatístico usado para colocar diferentes variáveis em uma escala comum. Isso é feito para garantir que as variáveis tenham uma distribuição similar e não causem viés em análises estatísticas ou algoritmos de aprendizado de máquina. A normalização pode envolver ajustar os valores para que estejam dentro de um intervalo específico, como [0, 1] ou [-1, 1], ou transformá-los para terem uma média zero e um desvio padrão de um. Essa etapa é crucial para garantir a consistência e comparabilidade dos dados em análises subsequentes.

Pra normalizar o *dataset* foi escolhido o método Min-Max que consiste em uma técnica de pré-processamento de dados que transforma os valores das variáveis para um intervalo específico, geralmente entre 0 e 1. Isso é feito subtraindo o valor mínimo da variável e dividindo pela diferença entre o valor máximo e mínimo. A fórmula básica é:

$$\text{valor\_normalizado} = \frac{\text{valor} - \text{min}}{\text{max} - \text{min}}$$

A normalização é necessária em algoritmos de aprendizado de maquina pois diferentes escalas nas variáveis podem afetar o desempenho dos modelos, garantindo que todas as características contribuam de maneira equilibrada para o treinamento.

a seguir, na figura 32 temos uma demonstração de como este passo foi feito via código.

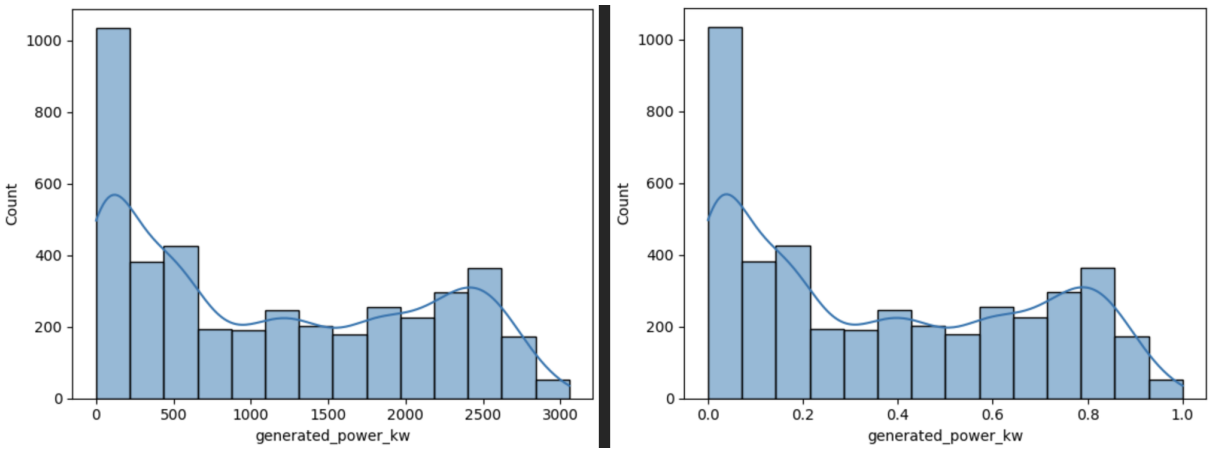
Figura 32 – Código para normalização

```
[179]  normalized_data= (data - data.min()) / (data.max() - data.min())
       ✓  0.0s
```

Fonte: O autor (2024)

uma vez executado este comando, todas as variáveis são normalizadas, para verificação foi escolhida uma variável antes e depois da normalização e seus histogramas foram gerados. os mesmos podem ser vistos na figura 33.

Figura 33 – Variável original x variável normalizada



Fonte: O autor (2024)

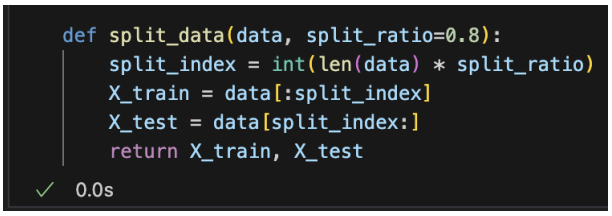
Ao observar o eixo X dos gráficos (gerados através de uma adaptação no código visto na figura 22) é possível notar que o primeiro deles atinge o valor máximo de 3000 enquanto o segundo o valor máximo de 1. Ainda se tratando dos gráficos os mesmos possuem o mesmo tamanho de barra e, nas mesmas posições, indicando assim que todos os valores foram redimensionados se forma igualitária.

O passo seguinte consistiu em separar o conjunto de dados já normalizado em dois subconjuntos, sendo eles treino e teste. Separar conjuntos de dados entre treino e teste é essencial em *machine learning* para avaliar o desempenho de um modelo de forma precisa e evitar *overfitting*. Tal processo ocorre quando um modelo se ajusta muito bem aos dados de treinamento, capturando até mesmo o ruído, o que pode resultar em um desempenho pior ao lidar com novos dados, pois não consegue generalizar adequadamente. O conjunto de treinamento é usado para ajustar os parâmetros do modelo, enquanto o conjunto de teste é reservado para avaliar a capacidade do modelo de generalizar para dados não vistos. Isso ajuda a garantir que o modelo não memorize os dados de treinamento, mas sim aprenda padrões que podem ser aplicados a novos dados, tornando suas previsões mais confiáveis em situações do mundo real.

A função *split data* divide um conjunto de dados em subconjuntos de treinamento e teste com base em uma proporção especificada, com uma proporção padrão de 0,8. Ela calcula o índice de divisão multiplicando o número total de elementos no conjunto de dados *len* pela proporção de divisão e, em seguida, converte este valor para um inteiro. O conjunto de dados é então dividido em *X train*, contendo todos os elementos desde o início até o índice de divisão, e *X test*, contendo todos os elementos do índice de divisão até o final. Finalmente, a função retorna os conjuntos de treinamento e teste, permitindo assim uma partição flexível dos dados.

Como pode ser visto na figura 34.

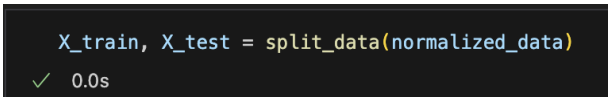
Figura 34 – Código para separar o conjunto de dados em treino e teste



Fonte: O autor (2024)

em seguida, na figura 35 podemos observar a chamada da função, na qual passamos como parâmetro de referência a nossa base de dados normalizada, esta que leva o nome de *normalized data*.

Figura 35 – Código para chamar a função que separa treino e teste



Fonte: O autor (2024)

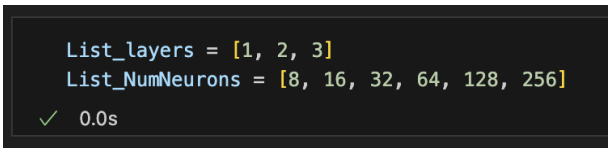
Assim, finaliza-se obtendo o conjunto de dados em duas partes, a parte X train, utilizada para treinar e, que contém 80% dos dados e a base X test contendo os outros 20%.

3.4.3 Etapa 3: Aplicação do algoritmo de *Autoencoder*

Esta seção demonstra, explica e ilustra o desenvolvimento do algoritmo responsável pelos testes de layouts de *Variational Autoencoder* (VAE). Assim como trata também da iteração feita com o objetivo de determinar os melhores hiperparâmetros calculados a partir do menor erro da métrica escolhida para ajustes e comparações dos diferentes layouts.

Inicialmente foram criadas duas listas, que podem ser vistas na figura 36. Essas duas listas definem os parâmetros para uma pesquisa em grade em modelos de rede neural. A lista *List layers* especifica o número de camadas a serem testadas, enquanto *List NumNeurons* determina o número de neurônios considerados em cada camada. Isso permite explorar diversas arquiteturas de rede e encontrar a configuração mais adequada para o problema em questão. Neste exemplo, o total de combinações gera 258 arquiteturas distintas.

Figura 36 – Código para definir as listas de parâmetros para Neurônios e camadas



Fonte: O autor (2024)

Fazendo uso das listas para fins de iteração, determinamos os melhores hiperparametros, isto é, o número de camadas e a quantidade de neurônios em cada camada. Foi desenvolvido um código que itera sobre o número de camadas especificado em *List layers*, gerando todas as possíveis combinações de configurações de camadas e neurônios. Para cada configuração, o código cria, treina e avalia um modelo de *Autoencoder*, registrando o erro de reconstrução. Se o erro de reconstrução for menor do que o menor erro encontrado até o momento, as variáveis são atualizadas para armazenar os detalhes do novo melhor modelo. Após o término da pesquisa, o código imprime a melhor configuração encontrada, juntamente com o erro de reconstrução correspondente. Esse processo permite identificar a arquitetura mais eficaz para o *Autoencoder*, maximizando sua capacidade de reconstrução dos dados de entrada. Esta iteração pode ser vista em forma de código na figura 37.

Figura 37 – Código para definir a melhor configuração

```
for num_layers in gridsearch_layers:
    layer_configs = list(itertools.product(gridsearch_NumNeurons, repeat=num_layers))

    for config in layer_configs:
        print(f"Testing configuration: {num_layers} layers, {config} neurons per layer")
        autoencoder = create_autoencoder(config)
        trained_model = train_autoencoder(autoencoder, X_train, X_test)
        reconstruction_error = evaluate_model(trained_model, X_test)
        print(f"Reconstruction error: {reconstruction_error}")

        if reconstruction_error < best_reconstruction_error:
            best_reconstruction_error = reconstruction_error
            best_model = trained_model
            best_config = (num_layers, config)

print(f"Best configuration: {best_config} with reconstruction error {best_reconstruction_error}")
⊗ 0.0s
```

Fonte: O autor (2024)

Ao percorrer o laço, a primeira função qué foi desenvolvida para compor a repetição foi a *create Autoencoder*. A função começa recebendo uma configuração *config*, que é uma lista de números inteiros representando a quantidade de neurônios em cada camada do *encoder* e do *decoder*. A função define a dimensão de entrada *input dim* com base no número de características dos dados de treinamento *X train.shape 1* e estabelece a dimensão do espaço latente *latent dim* como 2, o que significa que a representação latente será bidimensional.

O próximo passo é a construção do *encoder*. A função cria uma lista chamada *layers list* onde adiciona, iterativamente, camadas densas usando o número de neurônios especificado em *config* e aplicando a ativação *ReLU* em cada camada. Depois de adicionar todas as camadas definidas na configuração, a função adiciona uma camada final densa ao *encoder* para mapear a entrada ao espaço latente de dimensão 2. Todas essas camadas são combinadas em um modelo sequencial, resultando na arquitetura completa do *encoder*.

Para construir o *decoder*, a função cria uma lista chamada *decoder layers*, onde as camadas densas são adicionadas em ordem inversa, usando novamente a ativação *ReLU*. Além disso,

uma camada final densa é adicionada ao *decoder* para reconstruir a entrada original, utilizando a ativação sigmoide para garantir que os valores de saída estejam entre 0 e 1. As camadas do *decoder* são, então, combinadas em um modelo sequencial, similar ao *encoder*.

A função define a camada de entrada com uma forma que corresponde ao número de características dos dados de treinamento. Em seguida, a entrada é passada pelo *encoder* duas vezes para produzir *z mean* e *z log var*, que representam a média e o logaritmo da variância da distribuição latente, respectivamente. Esta separação é necessária para a reparametrização, que é uma técnica usada para permitir a diferenciação através da amostragem estocástica.

A reparametrização é implementada por meio de uma função de amostragem anônima *sampling*. Esta função recebe *z mean* e *z log var* como entradas e gera amostras da distribuição latente. Ela faz isso adicionando uma variável aleatória *epsilon* escalada pela variância, obtida através da exponenciação de *z log var* e multiplicada por 0.5 para ajustar a escala. O resultado é a combinação de *z mean* e o termo estocástico, garantindo que a amostragem seja diferenciável.

A camada Lambda aplica a função de amostragem às saídas do *encoder*, gerando a codificação final latente *encoded* que será usada pelo *decoder*. Em seguida, o *decoder* é aplicado a essa codificação latente para produzir a saída reconstruída.

Finalmente, o *Autoencoder* é construído conectando a entrada original à saída reconstruída pelo *decoder* através de um modelo funcional. Esse modelo é compilado usando o otimizador Adam, com a perda de erro quadrático médio e uma métrica adicional de divergência KL, que regulariza o espaço latente, forçando a distribuição latente a se aproximar de uma distribuição normal padrão.

A função retorna o modelo *Autoencoder* compilado, pronto para ser treinado. Esse modelo possui uma arquitetura simétrica de encoder-decoder, garantindo um processo balanceado de codificação e decodificação dos dados. o código referente aos últimos 5 parágrafos pode ser observado na figura 38.

Figura 38 – Código para criar o AutoEnconder

```
def create_autoencoder(config):
    input_dim = X_train.shape[1]
    latent_dim = 2

    encoder_layers = [layers.Dense(config[0], activation='relu', input_shape=(input_dim,))]
    for num_neurons in config[1:]:
        encoder_layers.append(layers.Dense(num_neurons, activation='relu'))
    encoder_layers.append(layers.Dense(latent_dim))
    encoder = models.Sequential(encoder_layers)

    decoder_layers = [layers.Dense(num_neurons, activation='relu') for num_neurons in reversed(config[1:])] + [layers.Dense(input_dim, activation='sigmoid')]
    decoder = models.Sequential(decoder_layers)

    inputs = layers.Input(shape=(input_dim,))
    z_mean = encoder(inputs)
    z_log_var = encoder(inputs)
    encoded = layers.Lambda(lambda x: x[0] + K.exp(0.5 * x[1]) * tf.random.normal(tf.shape(x[0])))([z_mean, z_log_var])

    decoded = decoder(encoded)

    autoencoder = models.Model(inputs, decoded)

    autoencoder.compile(optimizer='adam', loss='mse', metrics=[kl_divergence_loss])

    return autoencoder
0.0s
```

Fonte: O autor (2024)

A próxima função encontrada na repetição é a *train Autoencoder*. Esta função recebe três parâmetros: o modelo *model*, os dados de treinamento *X train* e os dados de teste *X test*. Durante o treinamento, o método *fit* utiliza tanto as entradas quanto as saídas dos dados de treinamento, já que um *Autoencoder* tenta reconstruir a entrada. O parâmetro *epochs* = 25 define que o modelo verá o conjunto completo de dados de treinamento 25 vezes, enquanto *batch size* = 32 determina que 32 amostras de dados serão processadas antes de atualizar os parâmetros do modelo. Usar tamanhos de lote menores ajuda a suavizar e acelerar o treinamento, permitindo atualizações mais frequentes dos parâmetros. O parâmetro *shuffle* = *True* embaralha os dados de treinamento antes de cada época para melhorar a generalização do modelo.

O método *fit* também usa os dados de teste como dados de validação *validation data* = *X test*, *X test*, permitindo avaliar o desempenho do modelo em dados não vistos durante o treinamento. Isso ajuda a monitorar e ajustar o modelo para evitar *overfitting*. Após o término do treinamento, a função retorna o modelo treinado, que pode ser utilizado para previsões ou análises posteriores. O código referente a esta parte da implementação pode ser visto na figura 39.

Figura 39 – Código para treinar o AutoEnconder

```
def train_autoencoder(model, X_train, X_test):
    model.fit(X_train, X_train,
              epochs=25,
              batch_size=32,
              shuffle=True,
              validation_data=(X_test, X_test))

    return model
✓ 0.0s
```

Fonte: O autor (2024)

A última função definida para ser utilizada na iteração foi a *evaluate model*. Sua tarefa é avaliar o desempenho de um *Autoencoder* usando dados de teste e, calcular o erro de reconstrução. Ela recebe o modelo e os dados de teste como argumentos. Primeiro, os dados de teste são passados pela segunda camada do modelo para obter a representação latente, e em seguida, pela última camada para reconstruir os dados de entrada. O erro de reconstrução é calculado como a média dos erros quadráticos entre os dados de teste originais e os dados reconstruídos. Este erro quantifica a precisão com que o *Autoencoder* consegue reproduzir os dados de entrada a partir de sua representação latente.

Além disso, as variáveis *best reconstruction error*, *best model* e *best config* são inicializadas para rastrear o menor erro de reconstrução encontrado, junto com o modelo e a configuração de *hiperparâmetros* correspondentes. O parâmetro *best reconstruction error* é inicializado como infinito para garantir que qualquer erro calculado posteriormente será menor, enquanto *best model* e *best config* são inicializados como *None* para armazenar o melhor modelo e configuração durante um processo de pesquisa em grade. a função na IDE de desenvolvimento pode ser vista na figura 40.

Figura 40 – Código para avaliar o AutoEnconder

```
def evaluate_model(model, X_test):
    encoded_data = model.layers[1].predict(X_test)
    decoded_data = model.layers[-1].predict(encoded_data)
    reconstruction_error = np.mean(np.square(X_test - decoded_data))
    return reconstruction_error

best_reconstruction_error = float('inf')
best_model = None
best_config = None
✓ 0.0s
```

Fonte: O autor (2024)

A métrica utilizada para a avaliação é a divergência de Kullback-Leibler representada

pela função *kl divergence loss*.Ela implementa o cálculo da divergência KL para um *Variational Autoencoder*. A divergência KL é uma medida de dissimilaridade entre duas distribuições de probabilidade e é essencial no treinamento de um VAE para garantir que a distribuição latente se aproxime o máximo possível de uma distribuição prior desejada, geralmente uma distribuição Gaussiana padrão.

O cálculo da divergência KL começa com a fórmula matemática expressa a seguir:

$$-0.5 \times \sum \left(1 + z_{\text{log\_var}} - z_{\text{mean}}^2 - \exp(z_{\text{log\_var}})\right)$$

A fórmula é composta por quatro termos: a soma de 1 com *z log var*, a subtração do quadrado de *z mean* e a exponenciação de *z log var*. Esses termos são então somados e multiplicados por -0.5 para formar a divergência KL para cada ponto de dados no lote.

Em seguida, a função utiliza a operação *tf.reduce sum* para somar os valores ao longo da última dimensão do tensor resultante, que geralmente representa as *features* dos dados. Esta soma cria uma única perda KL para cada ponto de dados no lote. Finalmente, *tf.reduce mean* é aplicado para calcular a média dessas perdas ao longo de todo o lote, resultando em uma única métrica escalonada que representa a média da divergência KL para o lote de dados. Essa métrica é usada no processo de otimização durante o treinamento do modelo VAE. O código implementado pode ser visto na figura 41.

Figura 41 – Código para criar e parametrizar a divergência KL

```
def kl_divergence_loss(z_mean, z_log_var):
    kl_loss = -0.5 * tf.reduce_sum(1 + z_log_var - tf.square(z_mean) - tf.exp(z_log_var), axis=-1)
    return tf.reduce_mean(kl_loss)
✓ 0.0s
```

Fonte: O autor (2024)

3.4.4 Etapa 4: Avaliação dos resultados obtidos

Após a implementação de todas as etapas para o carregamento, tratamento e treinamento da rede neural, o programa foi executado de modo a determinar a melhor configuração. Para um entendimento mais fácil de qual a melhor configuração, o código foi modificado para inserir um gráfico visual de cada uma das iterações.

Inicialmente foi feita uma modificação na função treino, adicionando a variável *history*, e atribuindo o treino a ela e a retornando junto da variável *model*, como pode ser visto na figura 42.

Figura 42 – Nova função de treino

```
def train_autoencoder(model, X_train, X_test):
    history = model.fit(X_train, X_train,
                        epochs=20,
                        batch_size=32,
                        shuffle=True,
                        validation_data=(X_test, X_test))
    return model, history
✓ 0.0s
```

Fonte: O autor (2024)

A função de avaliação também foi modificada de modo a receber o histórico de treinamento do modelo, assim permitindo uma analise de todas as configurações possíveis tanto de modo escrito como, após convertidas para gráficos, de modo visual. Ela extrai os valores de divergência KL durante a validação *val kl divergence loss*, retornando o menor valor registrado.

O bloco de código inicializa variáveis para rastrear a melhor divergência KL por meio da variável *best kl divergence*, o melhor modelo pela variável *best model* e a melhor configuração de camadas e neurônios pela variável *best config*. Além disso, cria uma lista de nome *results* para armazenar os resultados de todas as configurações testadas. Durante a execução, essas variáveis são atualizadas com base nas avaliações de cada modelo treinado. O código modificado pode ser visto na figura 43.

Figura 43 – Nova função de avaliação

```
def evaluate_model(history):
    kl_divergence_values = history.history['val_kl_divergence_loss']
    return min(kl_divergence_values)

best_kl_divergence = float('inf')
best_model = None
best_config = None
results = []
✓ 0.0s
```

Fonte: O autor (2024)

Por fim, o laço de repetição sofreu modificações de modo a permitir o armazenamento de todas as configurações de camadas, bem como seu resultado KL, para assim reter um histórico passível de *plotagem*. Com relação ao código já visto na figura 37 houve uma inclusão. A configuração atual e sua divergência KL, que anteriormente eram descartadas, agora são armazenadas na lista *results* para referência futura, como pode ser visto na figura 44.

Figura 44 – Novo código para avaliar o AutoEnconder

```
for num_layers in gridsearch_layers:
    layer_configs = list(itertools.product(gridsearch_NumNeurons, repeat=num_layers))

    for config in layer_configs:
        print(f"Testing configuration: {num_layers} layers, {config} neurons per layer")
        autoencoder = create_autoencoder(config)
        trained_model, history = train_autoencoder(autoencoder, X_train, X_test)
        kl_divergence = evaluate_model(history)
        print(f"KL divergence loss: {kl_divergence}")

        if kl_divergence < best_kl_divergence:
            best_kl_divergence = kl_divergence
            best_model = trained_model
            best_config = (num_layers, config)

        results.append((num_layers, config, kl_divergence))

print(f"Best configuration: {best_config} with KL divergence loss {best_kl_divergence}")
```

Fonte: O autor (2024)

Pra plotar o histórico de todas as configurações, facilitando assim a avaliação por meio de um recurso visual, foi criado um código que encontra a menor divergência KL dentre todas as configurações testadas, extraindo-se os valores de KL da lista *results* e aplicando-se a função min para determinar o valor mínimo, que é armazenado em*min kl divergence*. Como próximo passo, normaliza-se os valores de divergência KL dividindo cada um pelo valor mínimo encontrado, criando uma nova lista *normalized results* que contém o número de camadas, a configuração de neurônios e a divergência KL normalizada. Isso facilita a comparação das perdas relativas entre diferentes configurações, com a menor divergência KL normalizada para 1 e os outros valores expressos como múltiplos desse valor mínimo. o código na IDE pode ser visto na figura 45.

Figura 45 – Normalização dos resultados KL

```
min_kl_divergence = min([kl for _, _, kl in results])
normalized_results = [(num_layers, config, kl / min_kl_divergence) for num_layers, config, kl in results]

✓ 0.0s
```

Fonte: O autor (2024)

Subsequentemente são ajustados os valores de divergência KL normalizados, subtraindo 1 de cada valor, para reescalar a escala dos resultados. Esse ajuste é realizado para que a configuração com a menor divergência KL, que foi normalizada para 1, tenha um valor ajustado de 0, facilitando a interpretação visual e destacando as configurações que têm perdas significativamente maiores que a mínima. a implementação pode ser vista na figura 46.

Figura 46 – Ajuste dos resultados KL (menor valor igual a zero na escala)

```
adjusted_results = [(num_layers, config, kl - 1) for num_layers, config, kl in normalized_results]
✓ 0.0s
```

Fonte: O autor (2024)

Por fim, ocorre a plotagem propriamente dita Esse trecho de código visualiza graficamente as perdas de divergência KL ajustadas para diferentes configurações de *Autoencoder*. Primeiramente, são criadas duas listas: uma contendo strings formatadas com informações sobre as configurações de camadas e neurônios, e outra com os valores ajustados de divergência KL. Em seguida, utilizando a biblioteca *Matplotlib*, é gerado um gráfico de barras onde as barras representam as diferentes configurações e suas respectivas perdas ajustadas. O eixo x mostra as configurações de *Autoencoder*, enquanto o eixo y representa a divergência KL ajustada. Uma linha horizontal em vermelho, estilizada como tracejada, é adicionada para indicar a linha de base onde a perda é igual a 1, facilitando a interpretação visual dos resultados. Por fim, o gráfico é exibido, fornecendo uma representação visual clara das relações entre as configurações e suas respectivas perdas de divergência KL. Este pode ser visto na figura 47.

Figura 47 – Plotagem do gráfico de performance das configurações do *Autoencoder*

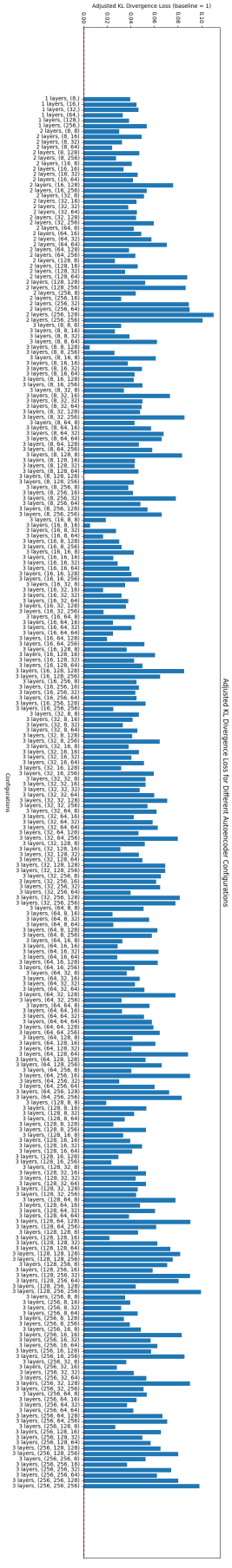
```
configs = [f"{num_layers} layers, {config}" for num_layers, config, _ in adjusted_results]
kl_divergence_values = [kl for _, _, kl in adjusted_results]

plt.figure(figsize=(40, 6))
plt.bar(configs, kl_divergence_values)
plt.xlabel('Configurations')
plt.ylabel('Adjusted KL Divergence Loss (baseline = 1)')
plt.title('Adjusted KL Divergence Loss for Different Autoencoder Configurations')
plt.axhline(0, color='red', linestyle='--')  # Add a horizontal line at y=0
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()
✓ 0.9s
```

Fonte: O autor (2024)

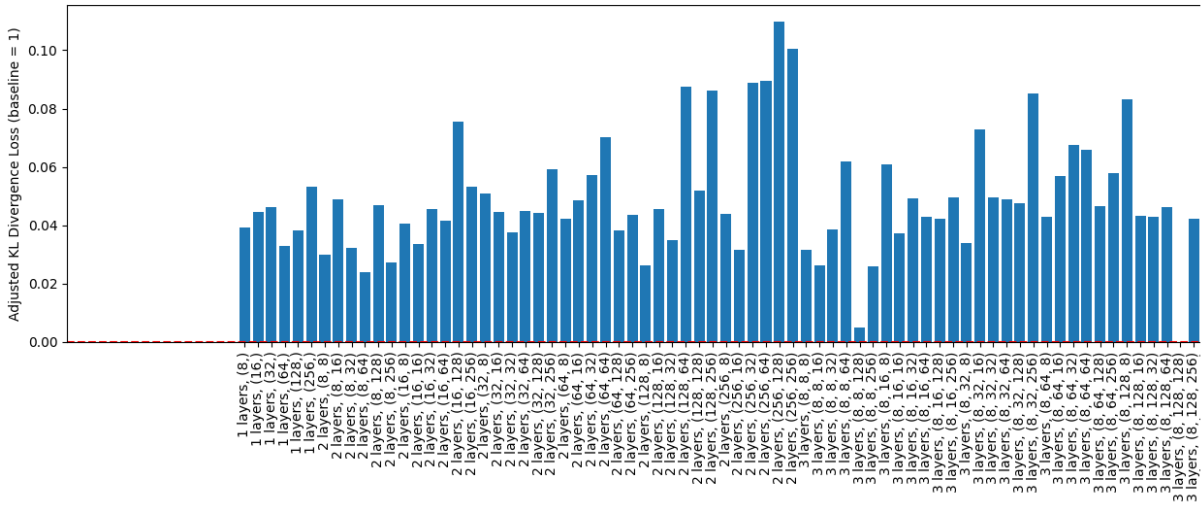
Na figura 48 podemos ver o gráfico gerado e comparar as performances. Adicionalmente a figura 49 faz um recorte na parte inicial do gráfico indo até o menor valor econtrado, que foi utilizado como zero da escala, assim facilitando o entendimento da figura 48.

Figura 48 – Plotagem do gráfico de performance das configurações do *Autoencoder*



Fonte: O autor (2024)

Figura 49 – Plotagem do inicio do gráfico de performance das configurações do *Autoencoder* contendo o eixo Y



Fonte: O autor (2024)

3.4.5 Etapa 5: Indicação do melhor modelo

Foram analisados os resultados de todas as configurações previamente definidas de redes neurais compreendidas entre 1, 2 e 3 camadas, cada uma delas explorando todas as possibilidades de 8, 16, 32, 64, 128 e 256 neurônios por camada. Esse processo foi realizado com o objetivo de identificar a configuração que melhor equilibrasse precisão e eficiência, utilizando a divergência das perdas da variável KL como métrica principal de avaliação.

Recapitulando, a divergência Kullback-Leibler (KL) é uma medida de dissimilaridade entre duas distribuições de probabilidade. Formalmente, ela quantifica a informação perdida ao usar uma distribuição de probabilidade Q para aproximar uma distribuição verdadeira P. Em termos matemáticos, a divergência KL é expressa como

$$D_{KL}(P \parallel Q) = \sum_{i} P(i) \log \frac{P(i)}{Q(i)},$$

onde a soma é sobre todos os possíveis eventos i. A divergência KL é sempre não negativa e é zero somente se P e Q forem idênticas em todas as possíveis realizações. Ela é amplamente utilizada em estatísticas, teoria da informação e aprendizado de máquina, especialmente em contextos onde é necessário comparar modelos probabilísticos.

O resultado da divergência KL é um valor escalar que quantifica a diferença entre as duas distribuições. Valores mais altos de $D_{KL}(P \parallel Q)$ indicam uma maior dissimilaridade entre P e Q, sugerindo que Q é uma má aproximação de P. Em contrapartida, valores baixos indicam que Q é uma boa aproximação de P. Um valor de zero indica que as distribuições são idênticas. É importante destacar que a divergência KL não é simétrica, ou seja, $D_{KL}(P \parallel$

$Q) \neq D_{KL}(Q \parallel P)$, o que significa que a ordem das distribuições importa na medição da dissimilaridade.

Para avaliar o resultado da divergência KL, é necessário considerar o contexto da aplicação. Em muitos casos, é útil comparar os valores de divergência KL obtidos para diferentes modelos Q em relação a uma mesma distribuição verdadeira P. Modelos que produzem menores valores de divergência KL são geralmente preferidos, pois indicam uma melhor correspondência com a distribuição verdadeira. No entanto, a interpretação absoluta do valor de D_{KL} pode ser menos intuitiva sem uma referência ou contexto específico. Em algumas aplicações práticas, é comum definir um limite aceitável de divergência KL para determinar se uma aproximação é suficientemente boa. Além disso, é importante considerar que a divergência KL pode ser sensível a *outliers* e distribuições de cauda pesada, o que deve ser levado em conta ao interpretar os resultados.

A configuração que apresentou a menor divergência nas perdas da variável KL foi a que utilizou 8 neurônios na camada 1, 128 neurônios na camada 2 e também 128 neurônios na camada 3. Esta configuração obteve um resultado de 2.9190, com um tempo de 1 milissegundo (ms) entre os passos de convergência.

A segunda melhor configuração em termos de menor divergência nas perdas da variável KL foi a que utilizou 8 neurônios na camada 1, 8 neurônios na camada 2 e 128 neurônios na camada 3. Esta configuração obteve um resultado de 2.9333, com um tempo de 991 microssegundos (µs) entre os passos de convergência.

A terceira configuração com menor divergência nas perdas da variável KL foi a que utilizou 16 neurônios na camada 1, 8 neurônios na camada 2 e 16 neurônios na camada 3. Esta configuração resultou em um valor de divergência 2.9347, com um tempo de 804 microssegundos (µs) entre os passos de convergência.

A quarta configuração com menor divergência nas perdas da variável KL foi a que utilizou 128 neurônios na camada 1, 8 neurônios na camada 2 e 8 neurônios na camada 3. Esta configuração resultou em um valor de 3.0728, com um tempo de 878 microssegundos (µs) entre os passos de convergência.

No quadro 3 podemos ver um compilado das descrições acima para uma verificação mais rápida.

Quadro 3 – Top 4 configurações de *Autoencoder* ordenado por performance

Ranking da configuração	**Neurônios na camada 1**	**Neurônios na camada 2**	**Neurônios na camada 3**	**Tempo de treino**	**Desvio na reprodução**
Primeiro	8	128	128	1ms	2.9190
Segundo	8	8	128	991μ s	2.9333
Terceiro	16	8	16	804μ s	2.9347
Quarto	128	8	8	878μ s	3.0728

Fonte: O Autor (2024).

3.5 CONSIDERAÇÕES FINAIS DO CAPÍTULO 3

Foi obtido um *dataset* de quantidade de energia gerada versus condições do sistema no momento da geração, dados estes que foram analisados para determinação de sua qualidade e características. Em seguida os dados foram pré processados, sendo normalizados, através do método MinMax para que pudessem ser corretamente carregados em um sistema baseado em *Autoencoder* e, em seguida separados na proporção 80/20 como sendo treino e teste. Para a aplicação do algoritmo, foram parametrizados o número de camada, a quantia de neurônios possível em cada camada e os tipos de função de ativação de cada camada. Além disso, foi definida foi definida a métrica de otimização como sendo a divergência de kullback-leibler e, na parametrização de treino em si, o número de épocas e o tamanho do lote. Posteriormente, foi iterado de modo a modificar o número de camadas e o número de neurônios por camada de modo a passar por todas as configurações possíveis dentre os valores pré selecionados. A avaliação foi feita utilizando a divergência KL sendo selecionada a configuração com o valor mais próximo a zero. Adicionalmente foram avaliados os resultados em todas as configurações possíveis através da geração de um gráfico. Isto posto, concluímos que a hipótese foi validada diante do fato de que todo o treinamento e avaliação de resultados ocorreu de forma a obtermos resultados utilizáveis para o proposto.

Com base no pesquisado e desenvolvido, foi observado que *Autoencoders* são excelentes ferramenta para a detecção de variações no resultado esperado independente do campo de pesquisa. A detecção de falhas é baseada fortemente em um comportamento diferente de algo rotineiro, isto é, uma divergência no que vem ocorrendo em momentos anteriores a divergência em si. No caso de painéis fotovoltaicos, a observação da geração, amostrada a cada X período de tempo, constitui assim um sinal, sinal este que para uma condição C, tende a se manter constante ou, com uma certa margem de erro. Caso seja detectado pelo *Autoencoders* uma divergência além do esperado na curva de geração, pode indicar uma falha ou, em caso de uma sensibilidade alta ou de uma amostragem curta, o incio de uma falha. Pelos motivos citados, a possibilidade de se desenvolver um produto capaz de tentar detectar falhas é factível.

Para tal, deve-se integrar na saída do medidor de geração, um captador do nível de tensão e corrente gerado no momento X. Outros sensores dos mais diversos tipos como termômetros, higrômetros, luxímetros, esfigmomanômetro, dentre outros, devem ser instalados e a cada medição do nível de geração, deve-se fazer uma medida em simultâneo destes componentes, assim gerando uma medida do ambiente em que o painel está inserido. Ao obter uma quantia suficiente de leituras, o modelo pode ser treinado para detectar diferentes comportamentos em condições semelhantes as que ele conhece, indicando assim uma falha parcial ou total do sistema. Um sistema em tempo real pode, teoricamente, até mesmo antecipar uma falha comparando uma divergência com a anterior caso detecte um aumento muito elevado tomando como referencia um comportamento pré aprendido para aquela condição de operação.

O principal desafio deste trabalho foi a base de dados, não a base em si, mas a escolha

de uma base de dados adequada. Uma base de dados que levasse em conta variados tipos de medição, como por exemplo: Medição da densidade de nuvens em diferentes altitudes e,ainda assim possuísse múltiplas instancias de medição se mostrou um desafio impar. Muitas bases encontradas possuíam medições de apenas um local, de usinas de geração domesticas e em casos de grandes companhias, estas que possuem múltiplos pontos de geração, os dados eram quase sempre inacessíveis por se tratarem de dados sensíveis ao *corebusiness* da empresa.

Outro argumento que sustenta a tese de que é possível elaborar um sistema de detecção de falhas em sistemas fotovoltaicos a partir da analise de sinal de geração são os trabalhos correlacionados que levam em conta a aplicação de *Autoencoders* na área de detecção de falhas. Apesar de apenas um deles se basear em sinais de geração de energia, todos tem por base sinais advindos das mais diversas fontes. com exceção do trabalho baseado em eletrocardiogramas, todos obtiveram uma precisão na casa de 98% ou superior, indicando assim que a detecção de anomalias em sinais é não só possível com bem precisa, indicando assim que seu uso leva a resultados no mínimo satisfatórios para uso no campo de sinais.

4 CONCLUSÃO

4.1 SÍNTESE DO TRABALHO

O propósito central deste trabalho foi abordar se um *Autoencoder* seria capaz de detectar falhas em sistemas fotovoltaicos a partir da leitura se um sinal de geração. A escassez de energia é um problema que esta sendo agravado por diversos fatores, tanto no âmbito civil quanto industrial a demanda por energia tem aumentado levando o consumo a atingir picos históricos ano após ano. Desenvolver métodos eficientes e limpos tem sido um desafio cada vez mais importante para a sociedade atual. Mas, não basta o desenvolvimento de tecnologias de geração, elas devem ser operadas e monitoradas para garantir que uma vez desenvolvidas, instaladas e ativadas possam gerar energia no ápice da sua capacidade.

A energia cada vez mais assume um papel de destaque tanto em empresas e industrias quanto em órgãos públicos, por vezes maiores responsáveis tanto pelas plantas geradoras, como por exemplo usinas hidroelétricas e solares, quanto pelo consumo desta energia através de, por exemplo, iluminação publica. Sendo assim, um método de predição e detecção de falha pode ser algo considerado de interesse para a sociedade como um todo pois, os benefícios desta tecnologia podem atingir indiretamente uma grande parcela da população.

A partir dos estudos, levantamentos e analises do estado da arte, bem como o estudo de literatura própria sobre os temas de redes neurais e de geração de energia com enfase na energia fotovoltaica, se chegou aos modelos baseados em *Autoencoders* bem como a variação desta arquitetura considerada mais adequada para o problema proposto, os *Variational Autoencoder*.

Estas pesquisas tiveram como objetivo reunir conhecimentos para a implementação de um algoritmo baseado em *Autoencoders* bem como verificar a variação de método para analise de performance a eles aplicados. Adicionalmente, o intuito é ter uma baliza para verificar se o erro obtido pelo algoritmo implementado é muito discrepante assim indicando que ele não foi capaz de desconstruir e reconstruir com assertividade.

A biblioteca NumPy é essencial em computação científica e análise de dados por fornecer suporte a *arrays* e matrizes multidimensionais, além de uma vasta coleção de funções matemáticas de alto desempenho. Ela permite operações vetorizadas eficientes, o que resulta em um processamento de dados muito mais rápido comparado às listas padrão do *Python*. A *NumPy* também é a base para muitas outras bibliotecas científicas, como *Pandas*, *SciPy* e *scikit-learn*, tornando-se uma ferramenta fundamental para qualquer projeto de análise ou modelagem de dados. Sua integração com C e Fortran permite a execução de cálculos complexos de forma otimizada.

Em termos de recursos, bibliotecas como *NumPy*, *Pandas* e *Scikit Learn*, muito faci-

litam no carregamento e manipulação de *datasets*, fornecendo funçoes prontas para analise e separação tanto do *dataset* em si quanto informações sobre os dados neles contidos. Assim, estas bibliotecas se mostraram praticamente essenciais para a correta utilização do *dataset* e não poderiam ter sua utilização mais recomendada para todo o tipo de trabalho que envolva carregamento de dados e manipulação de matrizes. No caso deste trabalho em especifico, muito uteis foram para o carregamento, identificação dos rótulos das categorias e, descrição simplificada dos rótulos.

Já bibliotecas como a *Seaborn* e a *MatPlotLib* se provaram com um valor impar para exibir gráficos referentes a todos os tipos de visualização dos dados bem como múltiplas analises desde histogramas até diagramas completos de correlação de todas com todas as variáveis. Assim, estas bibliotecas são altamente indicadas para qualquer trabalho que necessite desde coisas mais básicas, como confecção de gráficos com poucos pontos e apenas dois eixos (X e Y), até analises de dados complexas e visualização gráfica de resultados de *outputs* de redes neurais. O uso destas passou pela elaboração de histogramas, *scatterplots* e um *HeatMap* correlacional completo.

Para a implementação de redes neurais completas, as bibliotecas *tensorflow* e, para simplificar o uso desta, *Keras*, abstraem muitas das dificuldades fazendo uso de funções prontas, além disto, permitem o uso de processamento com múltiplos núcleos e com o uso de uma ou mais GPUs que aceleram em muito o treinamento das redes neurais. Adicionalmente permitem uma parametrização impar em relação as configurações de redes neurais dos mais diversos tipos. No caso deste trabalho, foram imperecíveis para a criação, treinamento e avaliação do VAE utilizado na tentativa de detecção de falhas em sistemas fotovoltaicos e, sua utilização pode ser aplicada em diversos projetos que necessitem de inteligencia artificial para seu funcionamento, detecção ou avaliação.

A partir disso foi feita uma avaliação do objetivo geral e dos objetivos específicos, buscando verificar os objetivos alcançados. O objetivo geral consistiu em aplicar um modelo baseado em *Autoencoders* para a detecção de falhas em sistemas geradores de energia, mais especificamente voltado para sistemas fotovoltaicos. O objetivo geral foi dividido em 5 objetivos específicos, estes para possibilitar uma avaliação de múltiplas etapas do processo separadamente.

O primeiro objetivo específico consistiu na possibilidade de implementação de um *Autoencoder* para o problema especificado, este objetivo foi alcançado. O *Autoencoder* foi implementado e, conseguiu iterar sobre dados produzindo uma divergência KL aceitável para determinar se o sinal gerado condizia com a situação de clima, localização e posição de instalação do sistema de geração. Ainda assim, cabe ressaltar que foi obtida uma principal dificuldade. Apesar de termos um melhor valor de divergência KL, sendo este 2.9190, não foi encontrado um trabalho que tenha sido feito sobre o mesmo tema e utilizado a mesma métrica de avaliação para que pudéssemos avaliar se este é um valor satisfatório.

Efetuar a aquisição de um sinal de energia emitido por um sistema necessita que seja a quantia de energia gerada em um dado período de tempo de geração, esta pode ser obtido com o uso de um multímetro, assim como a posição no globo, direção, inclinação, temperatura e umidade do local dentre múltiplas outras características acerca da situação dos painéis pode ser auferida fazendo uso de equipamentos corretos como termômetro, GPS, níveis, bussola dentre outros equipamentos. Mesmo possuindo alguns destes equipamentos de medição carecemos do acesso a uma usina de geração de energia com painéis fotovoltaicos e adicionalmente, caso o acesso fosse concedido teria de se obter um modo de validar se os painéis estavam gerando em situação de normalidade pois o modelo necessita ser treinado utilizando estes casos. Por estes motivos foi utilizada uma base de dados disponível que foi previamente testado obtida em um *WebSite* de nome Kaggle para a construção do modelo. Por tal motivo, este objetivo foi julgado concluído de forma parcial.

Com o uso de múltiplas métricas, avaliar a performance de um *Autoencoder* em todos os pontos do processo também pode ser atingido. Com o uso da métrica KL, conseguimos verificar o quanto o *Autoencoder* conseguiu divergir da medição, assim sendo, foi possível determinar o seu "erro", explorar o nível de perda e a velocidade de treinamento. Adicionalmente, com o uso de bibliotecas conseguimos comparar diferentes configurações além de verificar a distribuição dos valores das características dos atributos (colunas) da base de dados bem como seus valores máximos, mínimos, médio, medianas e nível de correlação de um atributo com outro.

Quanto aos objetivos: por meio de um *Autoencoder* verificar se é possível detectar falhas & elaborar um modelo para detecção de falhas com base em *Autoencoder*, ambos foram concluídos com exito. Após a implementação do modelo, foi verificado que ele é capaz de aprender como são os dados de uma operação normal de geração, dado este fato, o modelo tende a ser capaz de produzir uma divergência relativamente superior a esperada quando alimentado com dados de gerações que apresentem falhas. isto posto, caso as falhas apresentem amplitudes de divergência diferentes entre si, o modelo teoricamente pode ser capaz não só de prever a falha mas também de prever que tipo de falha ocorreu.

Apesar de a base de dados escolhida, após análise, não apresentar nenhuma leitura com falha, não foi possível determinar como gerar uma falha que se assemelhasse a uma falha real. Devido a isso, não foi possível utilizar o modelo treinado em leituras falhas. No entanto, com base na pesquisa teórica e nos experimentos práticos na reconstrução de dados sem falha, é relativamente seguro afirmar que uma leitura com falha tende a, ao ser desconstruída e reconstruída pelo *Autoencoder*, apresentar um nível de divergência maior do que o apresentado por dados corretos.

Para uma comprovação total, recomenda-se o uso de um *dataset* que possua dados com falhas, preferencialmente catalogadas por tipo. O processo ideal inclui treinar um modelo utilizando apenas as leituras de bom funcionamento e, posteriormente, aplicar e medir o erro ao inserir os grupos de dados de cada tipo de falha. Este procedimento tende a gerar, para cada

tipo de falha, uma divergência KL maior do que a obtida com dados corretos, porém, diferentes entre si para cada tipo de falha. Isso permitirá detectar e catalogar as falhas de maneira mais precisa.

4.2 CONTRIBUIÇÕES

Através deste trabalho de conclusão de curso analisamos a efetividade de um sistema de AI baseado em *AutoEcoders* mais especificamente da classe variacional de, a partir de um uma serie de dados, desconstruir e reconstruir uma medição.

Através da decomposição e reconstrução é possível determinar se a quantia de energia gerada por um painel está de acordo com o esperado a ser gerado por painéis dentro de uma condição semelhante a que este painel se encontra. O algoritmo faz isso medindo a divergência KL e comparando com a divergência KL esperada para aquele tipo de *encode* e *decode*.

Portanto o trabalho demonstrou que um VAE (*Variational AutoEconder* é sim capaz de aprender padrões de variáveis que se correlacionam com uma quantia de energia gerada, assim,com o correto refino e dados, podendo ser aplicado em sistemas mais complexos para determinar padrões anormais de geração

4.3 TRABALHOS FUTUROS

Para trabalhos futuros, podemos seguir por dois caminhos principais e distintos. A utilização de um *dataset* de poucos locais diferentes, com múltiplas leituras em cada local, limitando assim as variáveis que se modificam. O outro caminho consiste em aplicar em um *dataset* extraído de uma única usina, esta que já apresentou falhas previamente detectadas.

Em se tratando da utilização de um *dataset* com uma quantidade limitada de lugares, sendo estes lugares caracterizados por possuírem centenas de medições, a abordagem pode transcorrer de forma diferente. Inicialmente, as medições podem ser separadas por local, gerando assim múltiplos *datasets*. Cada um desses *datasets* seria referente a um local específico, onde todas as medições foram realizadas. Essa separação permite um foco mais detalhado e preciso nas condições específicas de cada local, facilitando a análise dos dados e a identificação de padrões particulares a cada ambiente estudado.

Por tal motivo, podemos excluir as variações de posição tanto dos painéis quanto do local dos painéis em relação ao globo terrestre. Dessa forma, concentramos nossa análise apenas nas variações climáticas, eliminando a interferência de fatores geográficos. Essa abordagem torna possível um estudo mais aprofundado dos impactos das condições climáticas sobre os resultados obtidos, permitindo uma compreensão mais clara de como diferentes variáveis climáticas afetam os dados coletados em cada local específico.

No *dataset* que apresenta leituras de apenas uma usina, que já vivenciou falhas ocasio-

nadas preferivelmente por motivos diferentes, o enfoque pode ser direcionado especificamente para as falhas em si. Cada incidente registrado oferece uma oportunidade única de análise, permitindo identificar padrões e anomalias que precedem ou acompanham as falhas. A compreensão detalhada das circunstâncias em que essas falhas ocorreram é essencial para desenvolver estratégias de mitigação e prevenção eficazes. Isso envolve a análise minuciosa dos dados de operação da usina, buscando sinais e indicadores que possam ter contribuído para os eventos de falha.

Compreender a perturbação de sinal que ocorreu no momento da falha ajuda na classificação precisa do incidente. Utilizando *Variational AutoEconders* (VAE), podemos avaliar o comportamento dos sistemas durante as falhas e comparar esses dados com o funcionamento normal. Essa abordagem permite a identificação de discrepâncias e anomalias que caracterizam cada tipo de falha, facilitando a categorização dos incidentes. Classificar as falhas com base nos erros que o VAE exibe proporciona uma visão clara das diferentes causas e efeitos das perturbações, o que é fundamental para a melhoria contínua dos processos operacionais da usina.

Além disso, focar na análise dos sinais pré-falha é uma estratégia preventiva valiosa. Ao estudar os dados anteriores às falhas, podemos identificar padrões que antecedem os incidentes, permitindo a implementação de sistemas de previsão e alerta. A capacidade de predizer quando uma falha ocorrerá é importante para proteger a infraestrutura e evitar danos maiores. Implementar tecnologias de monitoramento e previsão pode salvar componentes de danos significativos, garantir a continuidade das operações e reduzir custos associados a reparos emergenciais.

Os dados contidos neste trabalho podem servir de inspiração para outras pesquisas e artigos em diversas áreas que utilizem *Variational Autoencoders* (VAE). Além disso, esses dados e resultados para estudos que apliquem outras soluções no mesmo campo proposto, ou seja, a detecção de falhas em sistemas fotovoltaicos a partir da análise de sinal de geração.

REFERÊNCIAS

APPLEMB. **MacbookPro Site oficial**. 2024. Disponivel em: <https://www.apple.com/macbook-pro/>. Acesso em: 11 set 2023.

APPLEOS. **MacOS Sonoma Site oficial**. 2024. Disponivel em: <https://www.apple.com/macos/sonoma/>. Acesso em: 11 set 2023.

ATIENZA, R. **Advanced Deep Learning with TensorFlow 2 and Keras: Apply DL, GANs, VAEs, deep RL, unsupervised learning, object detection and segmentation, and more, 2nd Edition**. 2. ed. [S.l.]: Packt Publishing, 2020. ISBN 1838821651,9781838821654.

BANK, D.; KOENIGSTEIN, N.; GIRYES, R. Autoencoders. Springer International Publishing, Cham, p. 353–374, 2023. Disponível em: <https://doi.org/10.1007/978-3-031-24628-9_16>.

BELLMAN, R. **Dynamic Programming**. 1. ed. Princeton, NJ, USA: Princeton University Press, 1957.

CHEN, K.; HU, J.; HE, J. Detection and classification of transmission line faults based on unsupervised feature learning and convolutional sparse autoencoder. **IEEE Transactions on Smart Grid**, v. 9, n. 3, p. 1748–1758, 2018.

CHIANG, H.-T. *et al.* Noise reduction in ecg signals using fully convolutional denoising autoencoders. **IEEE Access**, v. 7, p. 60806–60813, 2019.

(ED.), D. E. R. e. Y. C. **Backpropagation: Theory, Architectures, and Applications**. Lawrence Erlbaum Associates, 1995. (Developments in Connectionist Theory). ISBN 080581258X,9780805812589,0805812598,9780805812596. Disponível em: <http://gen.lib.rus.ec/book/index.php?md5=26668e0c6b7c41586d697f15932880e3>.

GIL, A. C. *et al.* **Como elaborar projetos de pesquisa**. [S.l.]: Atlas São Paulo, 2002. v. 4.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep Learning**. [S.l.]: MIT Press, 2016. <http://www.deeplearningbook.org>.

HAYKIN, S. **Neural Networks: A Comprehensive Foundation**. 2. ed. Prentice Hall, 1998. ISBN 9780132733502,0132733501. Disponível em: <http://gen.lib.rus.ec/book/index.php?md5=23724f3cc14276a0e758dbabf9c9c4d4>.

ITERM. **Iterm Site oficial**. 2024. Disponivel em: <https://iterm2.com/>. Acesso em: 10 set 2023.

JOLLIFFE, I. **Principal Component Analysis**. 2nd. ed. Springer, 2002. ISBN 9780387224404,9780387954424,0387954422. Disponível em: <http://gen.lib.rus.ec/book/index.php?md5=106f5cca2a29459e5b6ba8dcbde044cd>.

KERAS. **Keras Site oficial**. 2024. Disponivel em: <https://keras.io/>. Acesso em: 09 set 2023.

LANGR, V. B. J. **GANs in Action: Deep learning with Generative Adversarial Networks**. 1. ed. [S.l.]: Manning Publications, 2019. ISBN 1617295566,9781617295560.

LI, C. *et al.* Bearing fault diagnosis using fully-connected winner-take-all autoencoder. **IEEE Access**, v. 6, p. 6103–6115, 2018.

MASSARON, J. P. M. L. **Artificial Intelligence For Dummies**. 2. ed. [S.l.]: John Wiley & Sons, 2021. ISBN 1119796768,9781119796763.

MATPLOTLIB. **MatPlotLib Site oficial**. 2024. Disponivel em: <https://matplotlib.org/>. Acesso em: 09 set 2023.

MCCULLOCH, W. S.; PITTS, W. A logical calculus of ideas immanent in nervous activity. **Bulletin of Mathematical Biophysics**, v. 5, 1943. Reprinted in Anderson1988.

MOUSAVI, S. M. *et al.* Unsupervised clustering of seismic signals using deep convolutional autoencoders. **IEEE Geoscience and Remote Sensing Letters**, v. 16, n. 11, p. 1693–1697, 2019.

NUMPY. **NumPy Site oficial**. 2024. Disponivel em: <https://numpy.org/>. Acesso em: 09 set 2023.

PADALLAN, J. O. **Key Concepts in Artificial Intelligence**. Arcler Press, 2022. ISBN 1774691450,9781774691458. Disponível em: <http://gen.lib.rus.ec/book/index.php?md5=F2275B5A22DCF6E8C1AF1AC12A91714A>.

PEREIRA, J. Unsupervised anomaly detection in time series data using deep learning. 2018. Disponível em: <https://api.semanticscholar.org/CorpusID:212412606>.

PISA, I. *et al.* Denoising autoencoders and lstm-based artificial neural networks data processing for its application to internal model control in industrial environments—the wastewater treatment plant control case. **Sensors**, v. 20, n. 13, 2020. ISSN 1424-8220. Disponível em: <https://www.mdpi.com/1424-8220/20/13/3743>.

PYTHON. **Python Site oficial**. 2024. Disponivel em: <https://www.python.org/>. Acesso em: 08 set 2023.

RASAMOELINA, A. D.; ADJAILIA, F.; SINčáK, P. A review of activation function for artificial neural network. In: **2020 IEEE 18th World Symposium on Applied Machine Intelligence and Informatics (SAMI)**. [S.l.: s.n.], 2020. p. 281–286.

SCHNEIDER, F. X. P. **Anomaly Detection and Complex Event Processing Over IoT Data Streams**. 1. ed. [S.l.]: Elsevier, 2022. ISBN 978-0-12-823818-9.

TAULLI, T. **Artificial Intelligence Basics. A Non-Technical Introduction**. [S.l.]: Apress, 2019. ISBN 978-1-4842-5027-3.

VISUALSTUDIOCODE. **VisualStudioCode Site oficial**. 2024. Disponivel em: <https://code.visualstudio.com/>. Acesso em: 10 set 2023.