UNIVERSIDADE ESTADUAL DE CAMPINAS

Faculdade de Engenharia Mecânica

LEONARDO FRANCO DE GODÓI

Proposta de arquitetura de autoencoder convolucional assimétrico para a análise de imagens de vibrações

Campinas

2021

LEONARDO FRANCO DE GODÓI

Proposta de arquitetura de autoencoder convolucional assimétrico para a análise de imagens de vibrações

Dissertação de Mestrado apresentada à Faculdade de Engenharia Mecânica da Universidade Estadual de Campinas como parte dos requisitos exigidos para a obtenção do título de Mestre em Engenharia Mecânica, na Área de Mecatrônica.

Orientador: Prof. Dr. Eurípedes Guilherme de Oliveira Nóbrega

ESTE TRABALHO CORRESPONDE À VERSÃO FINAL DA DISSERTAÇÃO DE MESTRADO DEFENDIDA PELO ALUNO LEONARDO FRANCO DE GODÓI, E ORIENTADA PELO PROF. DR. EURÍPEDES GUILHERME DE OLIVEIRA NÓBREGA.

Campinas

2021

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca da Área de Engenharia e Arquitetura
Rose Meire da Silva - CRB 8/5974

G541p

Godói, Leonardo Franco de, 1995-
Proposta de arquitetura de autoencoder convolucional assimétrico para a análise de imagens de vibrações / Leonardo Franco de Godói. – Campinas, SP : [s.n.], 2021.

Orientador: Eurípedes Guilherme de Oliveira Nóbrega.
Dissertação (mestrado) – Universidade Estadual de Campinas, Faculdade de Engenharia Mecânica.

1. Monitoramento. 2. Maquinas-ferramentas. 3. Inteligência artificial. 4. Aprendizado profundo. 5. Redes neurais (Computação). I. Nóbrega, Eurípedes Guilherme de Oliveira, 1950-. II. Universidade Estadual de Campinas. Faculdade de Engenharia Mecânica. III. Título.

Informações para Biblioteca Digital

Título em outro idioma: Proposal for asymmetric convolutional autoencoder architecture for the analysis of vibration images
Palavras-chave em inglês:
Monitoring
Machines-tools
Artificial intelligence
Deep learning
Neural networks (Computing)
Área de concentração: Mecatrônica
Titulação: Mestre em Engenharia Mecânica
Banca examinadora:
Eurípedes Guilherme de Oliveira Nóbrega [Orientador]
Janito Vaqueiro Ferreira
Luiz Fernando Sommaggio Coletta
Data de defesa: 28-01-2021
Programa de Pós-Graduação: Engenharia Mecânica

Identificação e informações acadêmicas do(a) aluno(a)
- ORCID do autor: https://orcid.org/0000-0003-3736-0387
- Currículo Lattes do autor: http://lattes.cnpq.br/9654474873898930

UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA MECÂNICA

DISSERTAÇÃO DE MESTRADO

Proposta de arquitetura de autoencoder convolucional assimétrico para a análise de imagens de vibrações

Autor: Leonardo Franco de Godói

Orientador: Prof. Dr. Eurípedes Guilherme de Oliveira Nóbrega

A Banca Examinadora composta pelos membros abaixo aprovou esta Dissertação de Mestrado:

Prof. Dr. Eurípedes Guilherme de Oliveira Nóbrega
DMC/FEM/UNICAMP

Prof. Dr. Janito Vaqueiro Ferreira
DMC/FEM/UNICAMP

Prof. Dr. Luiz Fernando Sommaggio Coletta
DEB/FCE/UNESP

A Ata de Defesa com as respectivas assinaturas dos membros encontra-se no SIGA/Sistema de Fluxo de Dissertação/Tese e na Secretaria do Programa da Unidade.

Campinas, 28 de Janeiro de 2021

DEDICATÓRIA

Aos meus pais, Sonia e Walter, ao meu irmão, Luan, e à minha tia, Maria Conceição, por terem feito parte desta fascinante e ainda inacabada jornada.

AGRADECIMENTOS

À minha família pelo apoio incondicional em todos os âmbitos da minha vida.

Ao meu orientador, Prof. Dr. Eurípedes Guilherme de Oliveira Nóbrega, por todo o suporte prestado e pela confiança depositada em meu trabalho.

A todos os professores que fizeram parte das diferentes etapas da minha formação acadêmica e me inspiraram a valorizar a busca pelo conhecimento.

A todos os amigos que, de alguma forma, contribuíram para esta realização.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Código de Financiamento 001.

Tornamos nosso mundo
significativo pela coragem de
nossas perguntas e pela
profundidade de nossas respostas.

Carl Sagan

RESUMO

Nos últimos anos, diversos segmentos da indústria têm se beneficiado das vantagens oferecidas pelo monitoramento de saúde de máquinas rotativas. Diante do ritmo no qual os avanços tecnológicos têm ocorrido e dos ganhos operacionais e econômicos proporcionados pelos métodos computacionais modernos, observa-se uma evidente tendência em se aplicar técnicas de inteligência artificial na manutenção desse tipo de equipamento. Este trabalho contempla o estudo de redes neurais artificiais como métodos aplicados ao monitoramento de condição de máquinas rotativas sob diferentes paradigmas de aprendizado. Uma configuração de rede neural do tipo *denoising convolutional autoencoder* voltada para o diagnóstico de componentes mecânicos é proposta, visando as principais vantagens de diferentes arquiteturas supervisionadas e não supervisionadas para otimização de desempenho na análise de imagens geradas a partir de sinais normalizados de vibrações. O treinamento do modelo é realizado em duas fases distintas, sendo um pré-treinamento por um *autoencoder* assimétrico e um refinamento por uma rede de classificação. Por meio da seleção de hiperparâmetros, experimentos conduzidos com base em dados extraídos de rolamentos ajudam a determinar a estrutura que oferece os resultados mais satisfatórios e a validar a generalidade do método desenvolvido. Uma comparação entre a arquitetura proposta e outras arquiteturas aplicadas para os mesmos fins, incluindo modelos propostos em outros trabalhos publicados, permite avaliar seu desempenho em termos de acurácia e robustez a ruído. O objetivo é avaliar a eficácia do método através do uso de dois conjuntos de dados experimentais públicos produzidos sob diferentes condições operacionais de máquina. Os resultados obtidos por meio dos estudos de caso justificam as opções adotadas na definição do sistema de diagnóstico criado e provam que a configuração proposta oferece maior acurácia na classificação de falhas e maior robustez em comparação às demais arquiteturas abordadas.

Palavras–chave: Máquinas Rotativas, Monitoramento de Condição, Inteligência Artificial, Aprendizado Profundo, Redes Neurais Artificiais

ABSTRACT

In the recent years, several industry segments have benefited from the advantages offered by health monitoring of rotating machines. Given the pace at which technological advances have occurred and the operational and economic gains provided by modern computational methods, there is an evident trend to apply artificial intelligence techniques in the maintenance of such equipment. This work addresses the study of artificial neural networks methods applied to the condition monitoring of rotating machines under different learning paradigms. A denoising convolutional autoencoder neural network configuration for the diagnosis of mechanical components is proposed, aiming the main advantages of different supervised and unsupervised architectures to optimize performance in the analysis of images generated from normalized vibration signals. The training of the model is carried out in two distinct phases, a pre-training by an asymmetric autoencoder and a fine-tuning by a classifier network. Through the selection of hyperparameters, experiments based on data extracted from bearings help to determine the structure that offers the most satisfactory results and to validate the generality of the developed method. A comparison between the proposed architecture and other architectures applied for the same purposes, including models proposed in other published works, allows to evaluate its performance in terms of accuracy and robustness to noise. The objective is to assess the effectiveness of the method through the use of two public experimental datasets produced under different machine operating conditions. The results obtained in both case studies justify the options adopted in the definition of the diagnostic system and prove that the proposed configuration offers greater faults classification accuracy and greater robustness in comparison to the other addressed architectures.

Keywords: Rotating Machines, Condition Monitoring, Artificial Intelligence, Deep Learning, Artificial Neural Networks

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AE	*Autoencoder*
AI	*Artificial Intelligence*
ANN	*Artificial Neural Network*
API	*Application Programming Interface*
BCE	*Binary Cross Entropy*
CAE	*Convolutional Autoencoder*
CCE	*Categorical Cross Entropy*
CM	*Condition Monitoring*
CNN	*Convolutional Neural Network*
CWRU	*Case Western Reserve University*
DBSCAN	*Density-Based Spatial Clustering of Applications with Noise*
DCAE	*Denoising Convolutional Autoencoder*
DL	*Deep Learning*
FP	*False Positive*
GPU	*Graphics Processing Unit*
ML	*Machine Learning*
MLP	*Multilayer Perceptron*
MSE	*Mean Square Error*
PCA	*Principal Component Analysis*
PReLU	*Parametric Rectified Linear Unit*
ReLU	*Rectified Linear Unit*

SNR	*Signal-to-Noise Ratio*
SGD	*Stochastic Gradient Descent*
SVM	*Support Vector Machine*
tanh	Tangente hiperbólica
TN	*True Negative*
TP	*True Positive*
VCM	*Vibration-based Condition Monitoring*

LISTA DE SÍMBOLOS

α	Coeficiente de momento
β	Coeficiente de deslocamento da normalização de lote
γ	Coeficiente de escalonamento da normalização de lote
ρ	Taxa de *dropout*
η	Taxa de aprendizado
θ, $\boldsymbol{\theta}$	Unidade e vetor de parâmetros de uma rede neural
μ	Média
σ, σ^2	Desvio padrão e variância
ϕ	Função de ativação
b, $\mathbf{b}$, $\mathbf{B}$	Unidade, vetor e matriz de vieses
BN	Transformação por normalização em lote
d, $\mathbf{d}$	Unidade e vetor da máscara de *dropout*
e	Número de Euler $= 2,7183$
exp	Função exponencial natural
f_a	Frequência de amostragem
F	Tamanho do filtro convolucional
h, $\mathbf{h}$, $\mathbf{H}$	Unidade, vetor e matriz de ativações em uma camada oculta
L	Perda
LN	Transformação por normalização em camada
M	Tamanho da amostragem
N, $\hat{N}$	Tamanhos dos mapas de atributos de entrada e de saída

P	Probabilidade
Q	Quantidade de amostras em uma célula da matriz de confusão
S	Tamanho do passo em uma convolução
$\mathbf{v}$	vetor de atualização dos parâmetros
w, $\mathbf{W}$	unidade e matriz de pesos
x, $\mathbf{x}$, $\mathbf{X}$	Unidade, vetor e matriz de variáveis de entrada
$\hat{x}$, $\hat{\mathbf{x}}$, $\hat{\mathbf{X}}$	unidade, vetor e matriz de variáveis de entrada reconstruídas
$\tilde{x}$	Variável de entrada normalizada por *z-score*
y, $\mathbf{y}$, $\mathbf{Y}$	Unidade, vetor e matriz de saídas esperadas
$\hat{y}$, $\hat{\mathbf{y}}$, $\hat{\mathbf{Y}}$	Unidade, vetor e matriz de saídas produzidas
z	Pré-ativação de um neurônio
Z	Tamanho do preenchimento em uma convolução

SUMÁRIO

1 INTRODUÇÃO

Máquinas rotativas são consideradas indispensáveis em grande parte dos processos industriais. É fundamental que tais máquinas funcionem de maneira adequada ao longo do tempo sob condições operacionais diversas para que sejam asseguradas a produtividade e a segurança necessária para que se evitem, em certos casos, falhas catastróficas que podem acarretar altos custos de manutenção ou, em situações mais extremas, colocar em risco vidas humanas envolvidas nos processos (Al-Badour *et al.*, 2011).

Em decorrência dos avanços tecnológicos recentes, sistemas mecânicos têm se tornado cada vez mais funcionais e complexos na indústria moderna. Nesse cenário, máquinas rotativas estão entre os equipamentos mais importantes em diversas aplicações industriais, o que torna o diagnóstico de falhas um aspecto crítico no desenvolvimento e na manutenção de sistemas que dependem desse tipo de máquina (Liu *et al.*, 2018). Diante desses fatores, a área de manutenção de máquinas rotativas tem recebido muita atenção por meio de suas diferentes abordagens, basicamente divididas em três tipos: corretiva, preventiva e preditiva (Tchakoua *et al.*, 2014). Para o caso de manutenção preditiva, em que se busca detectar de maneira inteligente o surgimento de falhas que podem afetar o funcionamento do sistema, faz-se necessária a aplicação de alguma técnica de monitoramento de condição (CM, do inglês *Condition Monitoring*) capaz de identificar padrões na operação que indiquem potenciais situações de falha (Stetco *et al.*, 2019).

O monitoramento de condição de máquinas rotativas pode ser realizado através de um vasto conjunto de técnicas invasivas, como análise de óleo e ferrografia, ou não invasivas, como inspeção visual e emissão acústica. Dentre essas e diversas outras técnicas, a análise de vibração, que constitui o monitoramento de condição baseado em vibração (VCM, do inglês *Vibration-based Condition Monitoring*), tem recebido grande destaque devido aos níveis satisfatórios de precisão que proporciona às tarefas de detecção de falhas (Sen *et al.*, 2017).

Elementos rolantes em rolamentos, componentes universais em máquinas rotativas, induzem vibrações específicas de baixa energia, sendo a causa de muitas falhas inesperadas. A posição desses elementos muda continuamente em relação à carga, dificultando a detecção de comportamentos que dependem da velocidade de rotação (Janssens *et al.*, 2016). Tal característica fez com que sinais de vibração extraídos de rolamentos tenham sido amplamente adotados

em sistemas de VCM voltados para detecção e classificação de falhas.

Em métodos tradicionais de monitoramento de condição, características ou atributos de um sinal são extraídos com base em conhecimentos específicos prévios do sistema, o que requer maior tempo de desenvolvimento e aplicação mais intensiva de mão-de-obra especializada (Lei *et al.*, 2016). Por outro lado, métodos orientados a dados, especialmente os baseados em inteligência artificial (AI, do inglês *Artificial Intelligence*), têm sido frequentemente adotados como alternativas, uma vez que não estão sujeitos a desvantagens significativas observadas em abordagens tradicionais, tais como a necessidade de modelagem computacional complexa e a capacidade limitada de generalização (Sun *et al.*, 2018).

O estudo da inteligência artificial revolucionou de maneira expressiva a tecnologia da informação e, consequentemente, várias outras áreas que fazem uso de suas técnicas na solução de problemas, em especial os diversos ramos da engenharia (Zhang *et al.*, 2016). Aplicada na forma de diferentes algoritmos, desde computação evolucionária e lógica nebulosa, até visão computacional e aprendizado de máquina (ML, do inglês *Machine Learning*), a AI tem se mostrado capaz de oferecer ganhos significativos no que diz respeito a aspectos fundamentais de projeto, tais como custo de desenvolvimento e utilização de recursos. No ramo da engenharia mecânica, especificamente, técnicas de AI têm sido empregadas para diversos propósitos, complementando ou substituindo métodos tradicionais.

No que se refere ao monitoramento de condição de máquinas rotativas, muitas pesquisas foram conduzidas na última década envolvendo técnicas de aprendizado profundo (DL, do inglês *Deep Learning*), um ramo avançado de inteligência artificial caracterizado por um extenso conjunto de arquiteturas de redes neurais artificiais (ANN's, do inglês *Artificial Neural Networks*) compostas de múltiplas camadas ocultas (Heo; Lee, 2018). O modelo de perceptron multicamadas (MLP, do inglês *Multilayer Perceptron*) desenvolvido por Zhang *et al.* (2017) realiza o diagnóstico de falhas diretamente de dados brutos coletados de rolamentos. Os trabalhos de Wu *et al.* (2019) e Eren *et al.* (2019) baseiam-se em redes neurais convolucionais (CNN's, do inglês *Convolutional Neural Networks*) unidimensionais para a classificação de falhas a partir de sinais de vibração. As redes neurais convolucionais bidimensionais, por sua vez, também são adotadas como base para o desenvolvimento de sistemas de diagnóstico, tais como os implementados por Chen *et al.* (2019), Xu *et al.* (2019) e Wen *et al.* (2018).

Geralmente, tanto em métodos tradicionais, como em métodos baseados em aprendizado de máquina, a análise de vibração é conduzida a partir de sinais de séries temporais

coletados por acelerômetros em uma ou mais direções. Além das redes neurais inteiramente conectadas, como a MLP, arquiteturas convolucionais têm mostrado alta efetividade em uma variedade de tarefas de reconhecimento e classificação. Enquanto os modelos convolucionais 1D foram desenvolvidos especialmente para dados de natureza sequencial, os modelos 2D exigem uma série de procedimentos para pré-processamento e transformação dos sinais a serem introduzidos como entradas da rede (Jiang *et al.*, 2019).

Métodos de aprendizado de máquina não supervisionados também assumiram um papel importante em diversas aplicações de monitoramento de condição baseadas em inteligência artificial. O modelo proposto por Shao *et al.* (2017) faz uso de uma rede neural do tipo *autoencoder* (AE) para extração robusta de atributos em tarefas de detecção de anomalias. Os trabalhos desenvolvidos por Lu *et al.* (2017), Zhang *et al.* (2019) e Li *et al.* (2018) baseiam-se em variantes de *autoencoder* para pré-treinamento de modelos de classificação voltados para tarefas de CM.

Redes neurais treinadas de forma não supervisionada, tais como os *autoencoders* e suas variantes, podem ser aplicadas para se extrair atributos provenientes dos dados de entrada, caracterizando a etapa de pré-treinamento (Masci *et al.*, 2011). A extração de atributos por meio de inteligência artificial traz evidentes melhorias quanto à qualidade das etapas de refinamento e processamento de dados posteriores (Hasani *et al.*, 2017). Desta forma, a tendência é que modelos de aprendizado profundo não supervisionados, para grande parte das aplicações, sejam cada vez mais adotados em comparação a técnicas tradicionais de análise de atributos baseadas em conceitos mais avançados de processamento de sinais.

Neste trabalho, é proposta uma rede neural do tipo *denoising convolutional autoencoder* (DCAE), cujo objetivo é reunir importantes características de variantes de AE e CNN, de forma a proporcionar maior robustez quanto a aplicações práticas de diagnóstico. A novidade é dada pela implementação de um *autoencoder assimétrico* para pré-treinamento dos parâmetros de um modelo classificador. Para verificação de desempenho do método, são feitas comparações entre a configuração proposta e outras redes neurais construídas especialmente para o diagnóstico de falhas em máquinas rotativas. Todas as implementações foram feitas na linguagem de programação *Python* e as redes foram modeladas com auxílio do pacote *TensorFlow* e sua interface de programação de aplicações (API, do inglês *Application Programming Interface*) de alto nível *Keras*, que disponibiliza os elementos de redes neurais sob o paradigma de programação orientada a objetos.

1.1 Objetivos

Este trabalho propõe uma arquitetura de rede neural do tipo *denoising convolutional autoencoder* para extração de atributos e classificação de falhas em máquinas rotativas a partir de imagens geradas de sinais de vibrações. A seleção dos hiperparâmetros que definem a rede neural é feita através de um conjunto de procedimentos bem definidos e o desempenho do modelo implementado é analisado em termos de acurácia e robustez através da comparação com outros modelos aplicados de maneira similar a tarefas de monitoramento de condição. Essa abordagem permite o estudo de técnicas de aprendizado profundo aplicadas ao monitoramento de condição de máquinas rotativas, de forma a explorar os aspectos que definem a modelagem de redes neurais para fins de diagnóstico de sistemas mecânicos sob diferentes condições operacionais.

1.2 Motivação

A manutenção preditiva de máquinas, além de garantir maior segurança às operações industriais, proporciona ganhos financeiros significativos, uma vez que impede a ocorrência de interrupções e otimiza o uso de equipamentos. Por tais razões, esse tipo de manutenção tem se tornado uma evidente tendência em diferentes ramos da indústria atual, especialmente em atividades críticas cujas falhas podem causar danos irreversíveis.

Com o recente advento da computação, impulsionado principalmente pelo aumento da capacidade de processamento e armazenamento de dados em larga escala, técnicas mais complexas derivadas da ciência da computação, até então inviáveis, passaram a ser aplicadas para diversos fins, incluindo manutenção preditiva. Por meio de métodos de inteligência artificial, o monitoramento de máquinas passou a ser realizado de maneira eficiente e generalista, eliminando, muitas vezes, a necessidade do uso de técnicas tradicionais. Além disso, a AI tornou possível o desenvolvimento de sistema autônomos, menos dependentes da subjetividade humana e suas intervenções. Nesse contexto, inúmeros trabalhos vêm sendo conduzidos visando o desenvolvimento e o aperfeiçoamento de modelos de aprendizado de máquina capazes de ampliar a contribuição de sistemas inteligentes para a solução de problemas de engenharia, promovendo o avanço de diversos setores da indústria.

1.3 Organização da dissertação

Esta dissertação é organizada da seguinte forma:

- Introdução e apresentação da estrutura do texto (Capítulo 1).
- Revisão de conceitos de monitoramento de máquinas rotativas, fundamentos e arquiteturas de redes neurais artificiais, e paradigmas de aprendizado (Capítulo 2).
- Apresentação da metodologia proposta, fluxo de trabalho, descrição do modelo e detalhamento da implementação computacional (Capítulo 3).
- Descrição dos estudos de caso e apresentação dos resultados obtidos através da aplicação da metodologia (Capítulo 4).
- Apresentação das conclusões decorrentes da execução do trabalho e sugestões para possíveis trabalhos futuros (Capítulo 5).

2 FUNDAMENTAÇÃO TEÓRICA

A seguir, são descritos os seguintes temas pertinentes ao desenvolvimento deste trabalho: monitoramento de condição de máquinas rotativas, fundamentos de redes neurais artificiais, aprendizado supervisionado e aprendizado não supervisionado.

2.1 Monitoramento de condição de máquinas rotativas

O monitoramento de condição engloba um conjunto de tarefas que visam garantir o desempenho desejável de uma máquina a curto e longo prazo. Dentre essas tarefas, pode-se citar a análise das condições de operação da máquina, identificação de condições anormais ou falhas em componentes, determinação das causas originais das falhas, avaliação do nível de severidade das falhas e predição da vida útil restante ou tendência de desenvolvimento de condições anormais (Chen *et al.*, 2019).

De forma geral, as tarefas que constituem o CM podem ser divididas em duas categorias: prognóstico e diagnóstico. No prognóstico, busca-se calcular ou prever o futuro como resultado de uma análise racional de dados disponíveis referentes ao sistema monitorado. O diagnóstico, por outro lado, é conduzido para se investigar as causas ou a natureza de uma dada condição, caracterizando-se como o processo de detecção e identificação de falhas associadas a um sistema ou subsistema. A Figura 2.1 ilustra as diferenças entre prognóstico e diagnóstico. O principal fator que distingue as duas abordagens é a dinâmica no tempo, uma vez que o prognóstico se concentra na análise da condição durante o período de degradação para geração de predições, enquanto o diagnóstico se encarrega de determinar os parâmetros que definem uma falha já existente. Entretanto, o diagnóstico de uma condição pode ajudar em seu entendimento e, consequentemente, contribuir para a execução de um prognóstico apurado sobre o sistema no qual ela está inserida (Lee *et al.*, 2014).

Outra forma de se classificar o modo pelo qual o CM é realizado leva em consideração, ao invés da finalidade e da dinâmica no tempo, o impacto físico da aquisição de dados sobre o componente monitorado (Stetco *et al.*, 2019). Neste caso, as técnicas são divididas nas seguintes categorias:

- **Monitoramento invasivo:** análise de vibração, monitoramento de detritos de óleo, métodos de pulso de choque, entre outros.

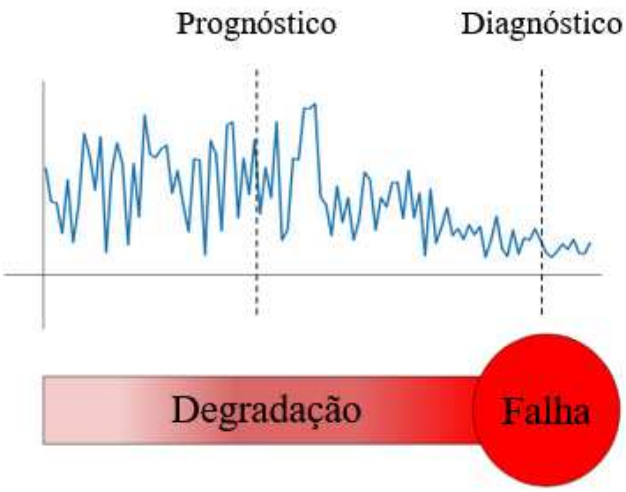

Figura 2.1 – Diferença entre prognóstico e diagnóstico. Adaptado de Lee *et al.* (2014).

- **Monitoramento não invasivo:** teste ultrassônico, inspeção visual, emissão acústica, termografia, entre outros.

Atualmente, sistemas de CM baseados em sinais de vibração são amplamente utilizados em uma grande variedade de máquinas, uma vez que são capazes de oferecer resultados satisfatórios a partir de sensores básicos. Como resultado, diversas técnicas baseadas na análise de vibração nos domínios do tempo e da frequência têm sido propostas para o desenvolvimento de modelos apurados de diagnóstico (Vicuña; Acuña, 2014).

O projeto de um sistema de diagnóstico pode ser realizado conforme três abordagens: baseada em modelos físicos, orientada a dados e híbrida. A análise de vibração baseada em modelos físicos pode ser fortemente afetada por ruídos provenientes de fontes externas ao sistema, fazendo com que sua sensibilidade esteja sujeita a mudanças da posição dos sensores e restrições de espaço em ambientes compactos, ocasionando perda de generalidade (Zhang *et al.*, 2020). A análise orientada a dados, por sua vez, reduz a complexidade da implementação e, apesar de requerer um grande volume de informações, minimiza a necessidade de conhecimento específico e aprofundado dos elementos analisados (Wen *et al.*, 2018). A abordagem de monitoramento híbrida, dada pela combinação das anteriores, é motivada pela necessidade de se tirar proveito dos principais pontos fortes de cada uma, compensando as deficiências que as mesmas apresentam individualmente (Ellefsen *et al.*, 2019). O diagrama da Figura 2.2 apresenta de forma hierárquica as relações existentes entre as principais abordagens. Um sistema se define como híbrido se faz uso de estratégias pertencentes a ambos os ramos concorrentes.

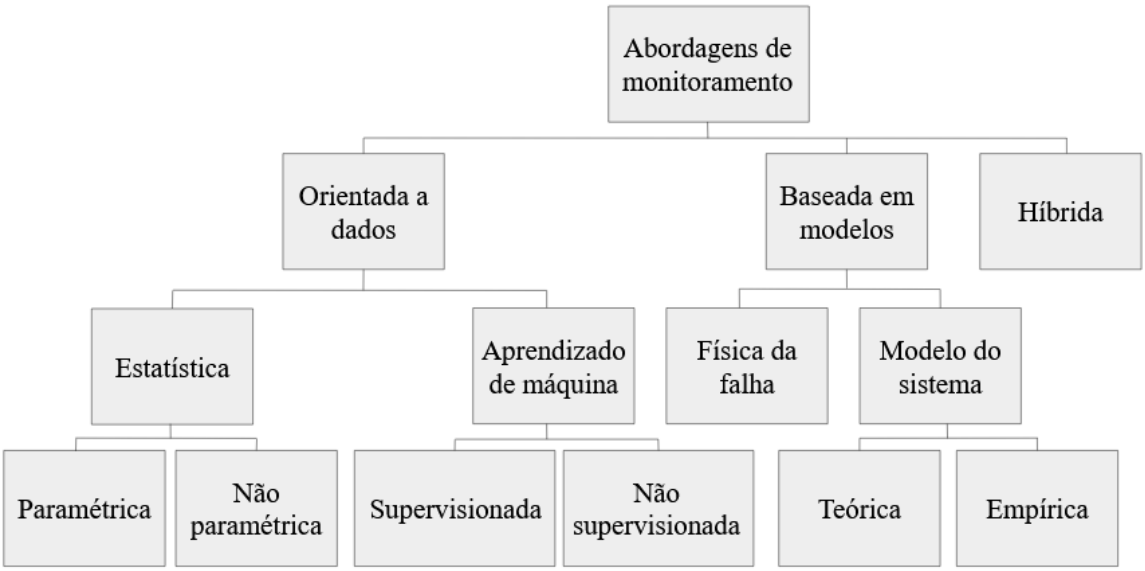

Figura 2.2 – Representação hierárquica das principais abordagens de monitoramento. Adaptado de Ellefsen *et al.* (2019)

Aspectos da modelagem orientada a dados, sobretudo referentes à aplicação de técnicas de aprendizado de máquina sobre as quais se fundamenta o trabalho aqui descrito, são discutidos em subseções posteriores.

2.1.1 Vibração de rolamentos

De acordo com Randall e Antoni (2011), falhas associadas a rolamentos estão entre as razões mais frequentes para quebras de máquina. Diferentemente de engrenagens, cujos sinais são periódicos, rolamentos produzem sinais estocásticos e podem ser modelados, para alguns casos, como cicloestacionários, descritos por funções de autocorrelação periódicas, embora não sejam, de fato, estritamente cicloestacionários.

A Figura 2.3 mostra os principais elementos de um rolamento. A partir dos sinais medidos e pela análise da frequência característica de vibração ou de características estatísticas no domínio do tempo, podem ser observadas falhas provenientes de partes específicas, tais como os anéis interno e externo, a gaiola e os elementos rolantes (Lee *et al.*, 2014).

Dados de vibração de rolamentos são amplamente usados no monitoramento de condição baseado em sinais de vibrações. Para a coleta de dados que compõem a série temporal, é comum a escolha de acelerômetros para a medição da vibração em uma ou mais direções, uma vez que são capazes de oferecer respostas confiáveis a altas frequências e são de fácil uso se comparados a outras alternativas disponíveis no mercado (Chen *et al.*, 2019). No entanto,

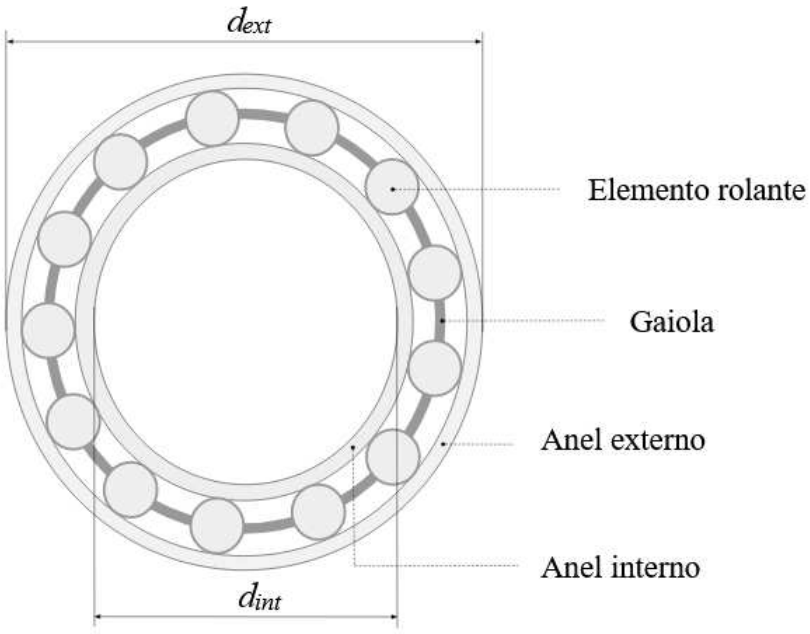

Figura 2.3 – Elementos básicos de um rolamento.

o processamento desses dados, sobretudo para diagnóstico de falhas em máquinas, pode ser complexo, o que fez com que muitos métodos de monitoramento e técnicas de análise de sinais tenham sido explorados ao longo dos últimos anos (Goyal *et al.*, 2018).

2.1.2 Técnicas tradicionais

Métodos tradicionais de monitoramento baseados em dados dependem fortemente da representação dos sinais analisados nos domínios do tempo e da frequência. Apesar de serem também muito utilizadas nas fases de pré-processamento em métodos inteligentes, tais análises desempenham um papel ainda mais crucial em abordagens tradicionais.

A análise tradicional no domínio do tempo compreende um conjunto de técnicas que apresentam de forma imediata a diferença entre as formas de onda analisadas pela comparação direta dos sinais medidos. Em contrapartida, possui capacidade de análise limitada, uma vez que não consegue prover informações suficientes a respeito do comportamento dos sinais e de lidar com sinais não estacionários (Lee *et al.*, 2014). Por outro lado, a análise no domínio da frequência permite representar as componentes de frequência que descrevem um sinal, resultando, porém, na perda da sua informação temporal (Sen *et al.*, 2017). Visando resolver ambas as deficiências, foram desenvolvidos métodos para a análise de sinais nos domínios do tempo e da frequência simultaneamente. No entanto, para todos os casos, obter informações

relevantes dessas análises pode ser uma tarefa de extrema complexidade em muitos casos reais de aplicação.

Apesar de ainda serem muito utilizados em várias áreas de estudo em mecânica, Shao *et al.* (2019) aponta como limitação crucial dos métodos tradicionais de diagnóstico a alta dependência de atributos extraídos manualmente e, portanto, a baixa capacidade de adaptação diante de diferentes cenários. A solução desse problema, muitas vezes, é encontrada no emprego de técnicas baseadas em aprendizado de máquina.

2.1.3 Técnicas baseadas em aprendizado de máquina

Em situações reais, máquinas rotativas usualmente operam em ambientes sob condições variáveis que tornam o mecanismo de falha complexo. Por essa razão, aumenta-se a dificuldade em se extrair as características dos sinais através de métodos tradicionais. Assim sendo, muitos progressos foram alcançados pela aplicação de aprendizado de máquina na identificação de falhas em máquinas (Chen; Li, 2017). Algumas das técnicas de ML mais utilizadas na área de diagnóstico são descritas a seguir:

- **Árvores de decisão:** comumente empregadas em mineração de dados para a criação de sistemas de classificação, as árvores de decisão classificam um conjunto de pontos ou instâncias (população) em vários níveis por meio de nós internos ou de decisão (ramificações) que formam uma árvore invertida contendo um nó inicial (raiz) e dois ou mais nós terminais (folhas). A classificação é então dada pela distribuição das instâncias entre os nós terminais, que representam os diferentes rótulos ou classes. Trata-se de um algoritmo não paramétrico e muito limitado em termos de generalidade e robustez (Song; Lu, 2015).

- **Algoritmos de agrupamento:** voltados para aprendizado não supervisionado, uma vez que lidam com dados que não estão atrelados a classes, esses algoritmos consistem em separar instâncias em grupos de acordo com métricas de similaridade, de forma que instâncias em um mesmo grupo sejam mais similares quanto possível entre si e instâncias em diferentes grupos sejam mais diferentes quanto possível. A dinâmica de agrupamento é definida de acordo com o tipo de algoritmo desejado, tal como o baseado em densidade de grupos, como o agrupamento espacial baseado em densidade de aplicações com ruído (DBSCAN, do inglês *Density-Based Spatial Clustering of Applications with Noise*), e o baseado em centroides, como o *k-means*. A escolha do método de agrupamento de-

pende de um conjunto de características referentes aos dados, como dimensionalidade e escalabilidade (Xu; Tian, 2015).

- **Máquinas de suporte vetorial:** originalmente desenvolvidas para classificação binária, as máquinas de suporte vetorial (SVM's, do inglês *Support Vector Machines*) podem ser adaptadas para realizarem também classificação multiclasse. Tomando como referência o caso mais simples, uma SVM é treinada de modo a determinar um hiperplano capaz de separar os dados entre dois conjuntos, uma classe positiva e uma classe negativa. No caso de os dados não serem linearmente separáveis, eles são transformados e representados em um espaço com maior número de dimensões que possibilita a separação (Jedliński; Jonak, 2015).

- **Redes neurais artificiais:** atualmente, englobam a maior parte das soluções baseadas em inteligência artificial. Em sua forma mais simples, uma rede neural artificial é composta de uma camada de entrada, uma camada intermediária ou oculta e uma camada de saída. A qualidade e a capacidade de processamento da rede podem ser controladas simplesmente pela modificação do número de camadas intermediárias e do número de neurônios nelas existentes. Por causa de sua modelagem paramétrica e de sua alta capacidade de generalização, as ANN's têm sido muito adotadas para tarefas de diagnóstico e prognóstico de máquinas (Liu *et al.*, 2018).

Uma abordagem mais detalhada de redes neurais, bem como de seus principais conceitos e arquiteturas, é realizada mais adiante nas Seções 2.2, 2.3 e 2.4.

2.1.4 Extração de atributos

Um dos principais problemas existentes no diagnóstico de máquinas rotativas se encontra na extração de atributos de sinais de vibração que sejam, ao mesmo tempo, sensíveis à ocorrência de falhas e robustos a ruídos (Chalouli *et al.*, 2017).

Na maioria das vezes, sinais de vibração coletados por sensores sofrem com a contaminação por ruídos a níveis que impossibilitam a aplicação direta de diagnóstico. Portanto, propriedades que descrevem esses sinais podem ser indetectáveis sem a ajuda de técnicas especiais de extração de atributos capazes de aumentar a razão sinal-ruído (SNR, do inglês *Signal-to-Noise Ratio*) e, como resultado, melhorar sua qualidade (Riaz *et al.*, 2017).

Diante dessa necessidade, é comum que um sistema de diagnóstico de falhas atue em duas etapas. Na primeira etapa, o sinal bruto ou resultante de algum processo de pré-processamento é analisado por meio de alguma técnica manual ou automática, de modo que suas características mais significantes sejam extraídas. Na segunda etapa, a identificação da condição do componente, que pode se tratar de um estado saudável ou de falha, é então conduzida a partir dos atributos extraídos (Jia *et al.*, 2018).

Tratando-se de séries temporais, como no caso de sinais de vibração, muitos métodos tais como os descritos na Subseção 2.1.2 permitem a extração manual de atributos. No entanto, além das limitações dos métodos manuais quanto a aspectos ligados à robustez, métodos automáticos têm sido cada vez mais explorados por serem capazes de integrar a etapa de extração de atributos à etapa de identificação de falhas e permitir atualizações em tempo real. Nesse contexto, destacam-se redes neurais como as convolucionais, descritas na Seção 2.3, que extraem atributos por meio da operação de filtros de convolução, e os derivados do *autoencoder*, descritos na Seção 2.4, que executam a extração de atributos por meio de treinamento não supervisionado e permitem a classificação de falhas através de refinamento supervisionado (Lu *et al.*, 2017).

Com o uso de *autoencoders* e outros modelos baseados em aprendizado profundo, pode-se obter de forma muito mais simples três benefícios fundamentais da extração de atributos listados por Chalouli *et al.* (2017): redução de dimensionalidade, análise automática de dados exploratórios e visualização nítida dos dados.

2.2 Fundamentos de redes neurais artificiais

As redes neurais artificiais foram idealizadas com o propósito de se criar inteligência artificial a partir de máquinas cuja arquitetura simula o funcionamento do sistema nervoso humano, por meio de modelos de aprendizado em que unidades computacionais se comportam de maneira similar aos neurônios humanos.

Apesar de terem sido conceitualmente criadas em meados do século XX, as redes neurais ganharam muito impulso no início do século XXI devido ao notável avanço dos recursos computacionais necessários para sua implementação, fazendo com que ressurgissem na forma de aprendizado profundo, com desempenho superior aos métodos de aprendizado de máquina mais simples, na medida em que os modelos são submetidos a um maior número de camadas e a um volume massivo de dados (Aggarwal, 2018). A Figura 2.4 indica a introdução das redes

neurais profundas no vasto campo da inteligência artificial.

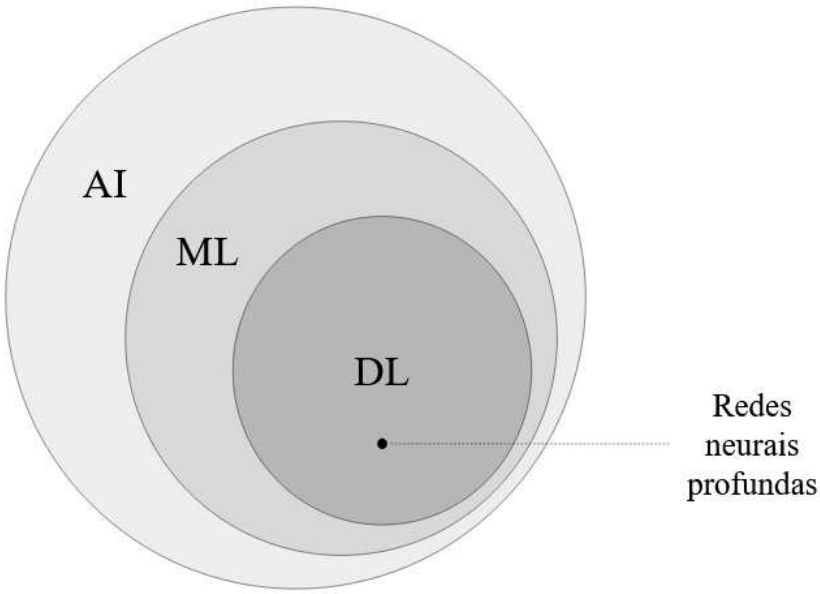

Figura 2.4 – Relação entre inteligência artificial, aprendizado de máquina e aprendizado profundo.

Redes neurais profundas permitem o aprendizado de representações não lineares através de altos níveis de abstração. A habilidade de representar dados e relações complexas de maneira automática confere às ANN's grande valor em várias aplicações de engenharia, sobretudo no campo de diagnóstico de máquinas (Verstraete *et al.*, 2017).

Um modelo de rede neural artificial é definido em função de seus parâmetros, propriedades treináveis otimizadas por meio de aprendizado, e de seus hiperparâmetros, propriedades estabelecidas durante a fase de projeto e que, portanto, não se ajustam de forma automática por meio de treinamento (Skansi, 2018).

A duração do processo de treinamento de uma rede neural é determinada pelo número de épocas, que indica a quantidade de vezes que todo o conjunto de entradas é processado por meio de sucessivas iterações de lotes ou minilotes de dados. O número de épocas e o tamanho de um lote, que define o número de amostras a serem processadas simultaneamente a cada iteração, são exemplos de hiperparâmetros de um modelo. Portanto, o treinamento é geralmente precedido pela análise do número de épocas através de estudos preliminares, para assim se determinar o número de épocas necessário para a convergência. No entanto, um número muito grande de épocas pode ocasionar problemas na generalização do aprendizado.

A partir de conceitos abstraídos da biologia e da matemática, a modelagem de redes neurais conta com um conjunto de fundamentos que visam reproduzir aspectos da inteligência

humana em meios computacionais.

2.2.1 Neurônio artificial

O sistema nervoso humano é composto de inúmeros neurônios, células que conectam-se umas às outras através de sinapses, estruturas formadas pela junção de dendritos e axônios, responsáveis pelo fluxo de informação. Os dendritos são prolongamentos ramificados que capturam os estímulos externos e os conduzem até o corpo da célula. Os axônios, por sua vez, têm a função de transmitir os impulsos nervosos produzidos na célula para outros neurônios. Assim, dendritos podem ser vistos como canais de entrada de informação, enquanto os axônios se encarregam da saída de informação de um neurônio.

O aprendizado em humanos é determinado pela forma que essas conexões se comportam diante dos estímulos externos recebidos. Desse modo, os neurônios artificiais são representados por unidades computacionais conectadas através de pesos, que realizam uma função similar à das sinapses biológicas. A Figura 2.5 apresenta a estrutura básica de um neurônio artificial. Cada entrada $\{x_1, x_2, ..., x_n\}$ de um neurônio tem sua influência sobre a saída $\hat{y}$ controlada pelo valor de seu respectivo peso $\{w_1, w_2, ..., w_n\}$ e pelo viés b associado, que atua como um coeficiente de deslocamento de uma função linear (Aggarwal, 2018).

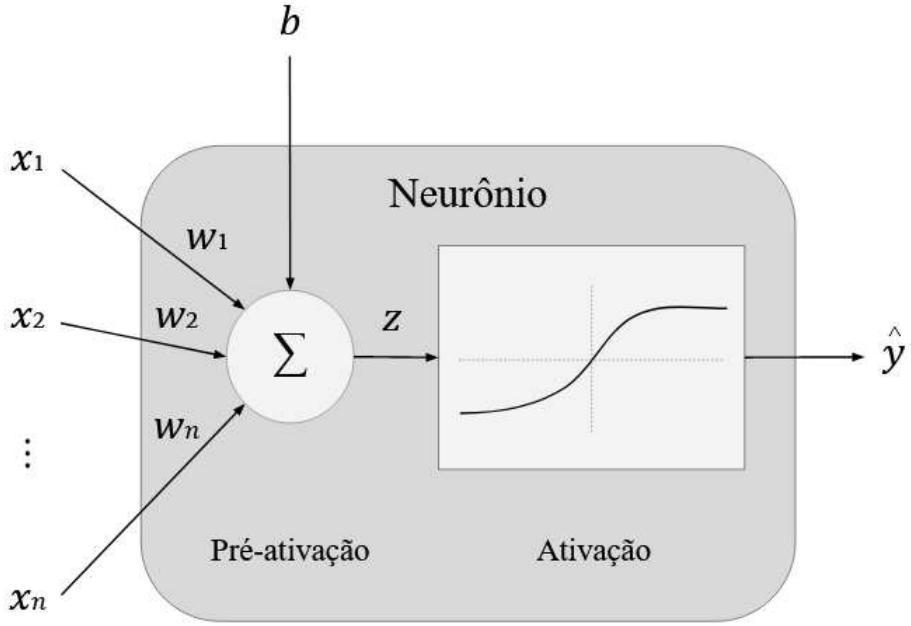

Figura 2.5 – Estrutura de um neurônio artificial.

A saída de um neurônio é condicionada por dois processos executados sucessiva-

mente. Na pré-ativação, é calculado o somatório de todas as entradas e do viés ponderados pelas respectivas conexões, da seguinte forma:

$$z = \sum_{i=1}^{n} w_i x_i + b \tag{2.1}$$

em que z é o valor da pré-ativação, x_i é a i-ésima entrada do neurônio, w_i é o peso da i-ésima conexão do neurônio com a camada anterior e b é o valor do viés.

Na ativação, o valor resultante z desse processo é então submetido a uma função de ativação $\phi(.)$, que o transforma de acordo com o tipo de função não linear escolhida. Dessa forma, a saída $\hat{y}$ do neurônio, que configura um simples perceptron, é dada por:

$$\hat{y} = \phi(z) = \phi\left(\sum_{i=1}^{n} w_i x_i + b\right) \tag{2.2}$$

Para facilitar a compreensão do papel desempenhado pelos pesos, afirma-se que $w_i < 0$ significa inibição e $w_i > 0$ significa excitação da conexão entre dois neurônios.

Os pesos e os vieses compõem o conjunto de parâmetros treináveis de uma rede neural e seu aprendizado, portanto, está associado à atualização desses valores dados o conjunto de entradas, o conjunto de saídas esperadas, os valores de inicialização e as funções de ativação selecionadas (Kubat, 2017).

2.2.2 Funções de ativação

A utilização de funções de ativação em neurônios é motivada pela necessidade de se obter representações não lineares dos dados de entrada, sendo considerada um aspecto crítico na modelagem de uma rede neural. Contudo, não é possível indicar uma função de ativação específica que seja definitivamente melhor ou mais adequada que as demais. A escolha da função de ativação para cada camada da rede deve ser feita com base na natureza da aplicação e no tipo de comportamento que se deseja observar (Aggarwal, 2018).

A atribuição da função de ativação é feita de camada a camada. Neurônios em uma mesma camada devem possuir a mesma função de ativação e, em redes neurais convencionais, uma mesma função de ativação é geralmente atribuída a todas as camadas intermediárias, podendo ou não diferir da ativação selecionada para a saída.

Apesar de haver um amplo conjunto de funções de ativação não lineares disponíveis, são descritas com mais profundidade as funções fundamentais, normalmente mais usadas, das quais muitas outras foram derivadas.

- **Sigmoide:** realiza a transformação não linear do resultado da pré-ativação e, por restringi-lo à faixa de valores $[0, 1]$, pode ser utilizada tanto nas camadas intermediárias como na camada final, no caso de classificação binária, em que valores mais próximos de 1 representam neurônios ativos e valores mais próximos de 0 representam neurônios inativos (Ma *et al.*, 2018). A sigmoide e sua respectiva derivada são dadas pelas seguintes equações:

$$\phi(z) = \frac{1}{1 + e^{-z}} \tag{2.3}$$

$$\frac{\partial \phi(z)}{\partial z} = \phi(z)(1 - \phi(z)) \tag{2.4}$$

Na Figura 2.6, pode-se visualizar as curvas geradas a partir das Equações 2.3 e 2.4.

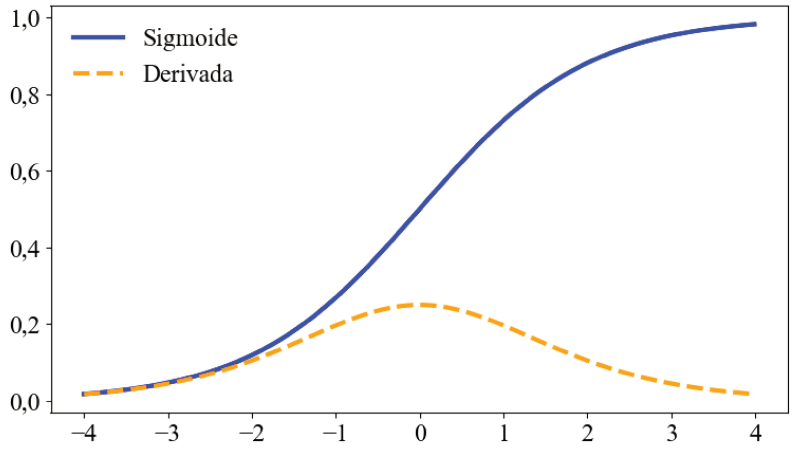

Figura 2.6 – Curvas da função de ativação sigmoide e sua respectiva derivada.

- **Tangente hiperbólica:** outra forma de transformação não linear, a tangente hiperbólica (tanh) possui forma semelhante à da sigmoide, mas abrange também números negativos, com valores restritos à faixa $[-1, 1]$, sendo geralmente utilizada nas camadas intermediárias de uma rede neural (Skansi, 2018). A sua derivada, apesar de também se assemelhar

à da sigmoide, atinge uma amplitude significantemente maior. A função de ativação tanh e sua respectiva derivada são dadas pelas seguintes equações:

$$\phi(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{2.5}$$

$$\frac{\partial \phi(z)}{\partial z} = 1 - (\phi(z))^2 \tag{2.6}$$

A Figura 2.7 apresenta as curvas correspondentes às Equações 2.5 e 2.6.

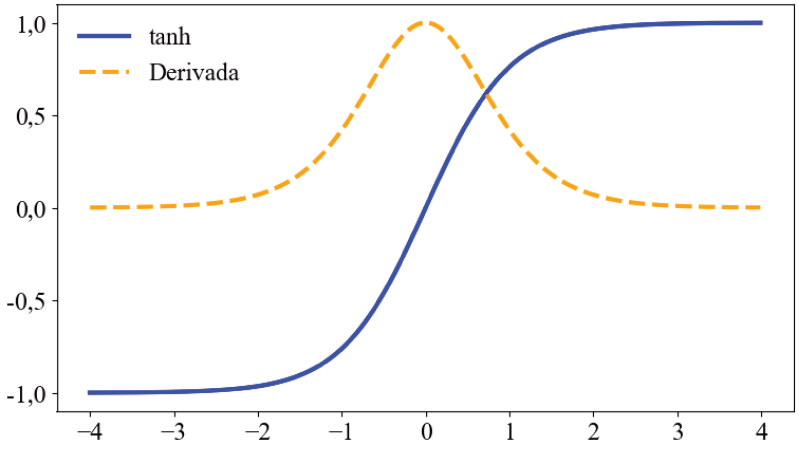

Figura 2.7 – Curvas da função de ativação tangente hiperbólica e sua respectiva derivada.

- **Unidade linear retificada:** comparada às funções anteriores, a unidade linear retificada (ReLU, do inglês *Rectified Linear Unit*) é capaz de aumentar a velocidade do treinamento, além de prevenir o problema de dissipação do gradiente, que impede a atualização dos pesos devido a gradientes muito baixos (Shao *et al.*, 2018). Ao contrário da sigmoide e da tanh, a ReLU não possui limites para os valores mínimo e máximo. Tais características tornam seu uso muito popular principalmente em arquiteturas convolucionais (Liu *et al.*, 2017). A função de ativação ReLU e sua respectiva derivada são dadas pelas seguintes equações:

$$\phi(z) = max(z, 0) = \begin{cases} 0, \text{ se } z < 0 \\ z, \text{ se } z \geq 0 \end{cases} \tag{2.7}$$

$$\frac{\partial \phi(z)}{\partial z} = \begin{cases} 0, \text{ se } z < 0 \\ 1, \text{ se } z \geq 0 \end{cases} \tag{2.8}$$

A Figura 2.8 mostra as curvas referentes às Equações 2.7 e 2.8.

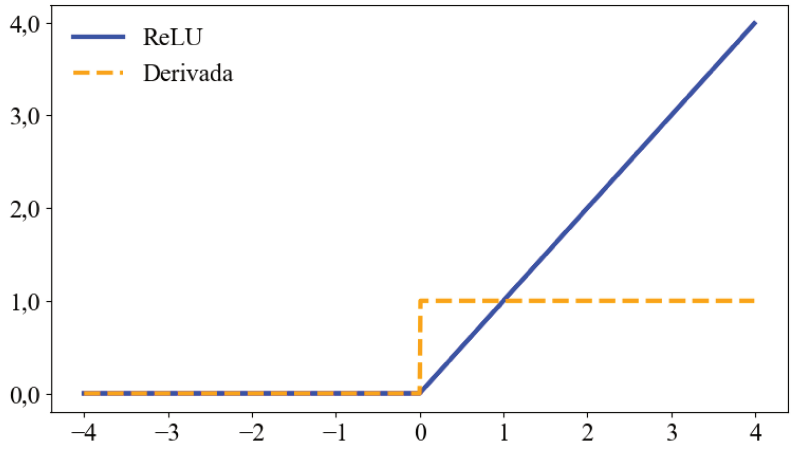

Figura 2.8 – Curvas da função de ativação linear retificada e sua respectiva derivada.

Desde sua concepção, a ReLU tem sido base para a criação de diferentes funções de ativação. Dentre as funções mais populares derivadas da ReLU, está a unidade linear retificada paramétrica (PReLU, do inglês *Parametric Rectified Linear Unit*), que consiste no escalonamento dos valores de $z < 0$ por um coeficiente arbitrário, ao invés de simplesmente igualá-los a zero (Shao *et al.*, 2018).

- ***Softmax*:** comumente utilizada na última camada de uma rede neural, trata-se de uma forma generalizada de regressão logística para classificação multiclasse (Xia *et al.*, 2019). Como resultado da aplicação da *softmax*, obtém-se, para cada vetor de entradas $\mathbf{x}$, sua probabilidade $(0 \geq P \leq 1)$ de pertencer à classe j dentre o conjunto de k classes (Hoang; Kang, 2019), através da equação:

$$\phi(z)_j = P(\hat{y} = j \in k|\mathbf{x}) = \frac{e^{z_j}}{\sum_{i=1}^{k} e^{z_i}}, \quad \text{para } j = 1, 2, ..., k \tag{2.9}$$

Cada neurônio em uma camada com ativação *softmax* corresponde a uma classe e tem como saída sua respectiva probabilidade de atribuição. Logo, a seguinte condição é satisfeita para a camada:

$$\sum_{j=1}^{k} \phi(z)_j = 1 \tag{2.10}$$

O conhecimento da derivada da função de ativação aplicada em uma camada ajuda na compreensão de seu comportamento durante o processo de ajuste dos parâmetros treináveis a ela associados, como será explanado de forma mais detalhada nas próximas subseções.

2.2.3 Funções de erro

Uma função de erro ou perda L permite que os parâmetros de uma rede neural sejam ajustados durante o seu treinamento. Em um modelo de classificação, durante a fase de propagação, em que as ativações de cada camada são calculadas da entrada para a saída, a rede processa os dados de treinamento e gera saídas indicando a probabilidade de uma dada amostra pertencer a cada uma das classes correspondentes. As saídas produzidas pela rede em cada iteração $\hat{\mathbf{y}} = \{\hat{y}_1, \hat{y}_2, ..., \hat{y}_k\}$ são então comparadas às saídas esperadas $\mathbf{y} = \{y_1, y_2, ..., y_k\}$ através da função de erro, que informa o desvio entre os dois conjuntos. Durante a fase de retropropagação, em que ocorre a atualização dos pesos e vieses, a derivada parcial da função de erro é calculada para cada parâmetro treinável da rede neural e o ajuste é caracterizado pela minimização do erro (Ho; Wookey, 2020).

A seguir, são apresentadas algumas das principais funções de erro utilizadas no aprendizado de redes neurais:

- **Erro quadrático médio:** normalmente utilizada para o cálculo do erro de reconstrução, baseia-se na média quadrática da diferença entre k saídas geradas e as respectivas saídas esperadas (Osmani *et al.*, 2019). O Erro quadrático médio (MSE, do inglês *Mean Square Error*) é definido por:

$$L(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{k} \sum_{j=1}^{k} (y_j - \hat{y}_j)^2 \tag{2.11}$$

- **Entropia cruzada binária:** a principal desvantagem do uso do MSE para classificação binária está em sua lenta convergência (Nasr *et al.*, 2002). Nesse caso, torna-se mais

apropriado o uso da entropia cruzada binária (BCE, do inglês *Binary Cross Entropy*), calculada por:

$$L(\mathbf{y}, \hat{\mathbf{y}}) = -\frac{1}{k}\sum_{j=1}^{k}[y_j \cdot log(\hat{y}_j) + (1 - y_j) \cdot log(1 - \hat{y}_j)] \tag{2.12}$$

- **Entropia cruzada categórica:** adequada para classificação multiclasse, em que mais de dois rótulos possíveis podem ser atribuídos a uma amostra, a entropia cruzada categórica (CCE, do inglês *Categorical Cross Entropy*) pode ser aplicada somente quando há o mesmo número de neurônios de saída e de classes (Ho; Wookey, 2020). Assim, o erro é dado por:

$$L(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_{j=1}^{k} y_j log(\hat{y}_j) \tag{2.13}$$

Apesar de haver outras funções de erro, as descritas acima são mais comumente usadas em tarefas de reconstrução de dados, classificação binária e classificação multiclasse, respectivamente.

2.2.4 Algoritmos de otimização

Uma vez que o erro indica o quão distantes as saídas produzidas estão dos valores esperados, a sua minimização induz o modelo a resultados mais próximos dos esperados e, consequentemente, ao aperfeiçoamento do aprendizado sobre os dados de treinamento. Em redes neurais rasas, o processo de aprendizado é relativamente simples, pois o erro pode ser calculado como uma função direta dos parâmetros, o que facilita o cálculo dos gradientes. No entanto, no caso de redes neurais profundas, o erro se caracteriza como uma extensa função composta dependente de parâmetros de todas as camadas (Aggarwal, 2018).

A retroprogação é o método tradicional para viabilizar o aprendizado de redes neurais, uma vez que permite que seja determinado o gradiente do erro em relação a cada peso e viés da rede neural através da aplicação da regra da cadeia de derivação sobre as funções aninhadas no sentido contrário à propagação comum, da saída até a entrada da rede, considerando o fato de que cada parâmetro possui uma contribuição específica para o erro final (Kubat, 2017). É durante a retroprogação que ocorre o aprendizado propriamente dito, quando os pesos e vieses de todas as camadas são atualizados. Nessa fase, os cálculos devem ser realizados para uma

camada por vez, pois os erros associados aos neurônios na camada $l-1$ dependem dos valores obtidos na camada l.

Dessa forma, tornaram-se populares os métodos de otimização baseados no gradiente descendente, que consistem em minimizar a função objetivo $L(\boldsymbol{\theta})$ com relação ao vetor de parâmetros treináveis $\boldsymbol{\theta} = \{\theta_1, \theta_2, ..., \theta_q\}^T$ do modelo pelo ajuste dos mesmos no sentido oposto ao gradiente da função, definido por:

$$\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}) = \begin{Bmatrix} \dfrac{\partial L(\boldsymbol{\theta})}{\partial \theta_1} \\ \dfrac{\partial L(\boldsymbol{\theta})}{\partial \theta_2} \\ \vdots \\ \dfrac{\partial L(\boldsymbol{\theta})}{\partial \theta_q} \end{Bmatrix} \tag{2.14}$$

A Figura 2.9 ilustra um exemplo do princípio de minimização por gradiente descendente. O gradiente de uma função indica a sua direção de subida rumo ao ponto de máximo, portanto, pela utilização do gradiente negativo, a busca é conduzida rumo ao ponto de mínimo da função que descreve o erro (Ruder, 2016).

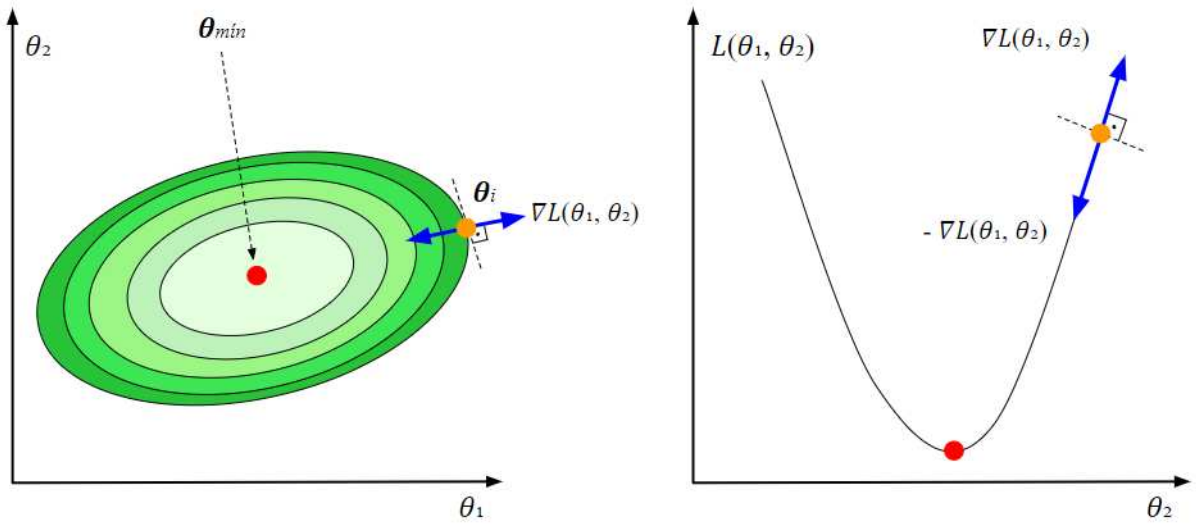

Figura 2.9 – Princípio de minimização do gradiente descendente.

A partir desse conceito, a versão mais simples do algoritmo de otimização por gradiente descendente calcula o gradiente da função de erro para todas as amostras do conjunto de

dados de treinamento, através da seguinte equação:

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \eta \cdot \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}) \tag{2.15}$$

em que η denota a taxa de aprendizado, um hiperparâmetro que representa o tamanho do passo do processo de otimização. Trata-se de uma constante positiva que define a convergência do algoritmo.

A Figura 2.10 ilustra os efeitos da taxa de aprendizado sobre o processo de minimização. Valores muito altos de η aceleram a convergência, mas aumentam o risco de oscilações em torno de regiões planas ou de mínimos locais. Um valor ótimo de η, por outro lado, pode tornar a convergência mais lenta, mas confere maior robustez na busca pelo ponto de mínimo (Ertel, 2017), reduzindo o risco de instabilidade.

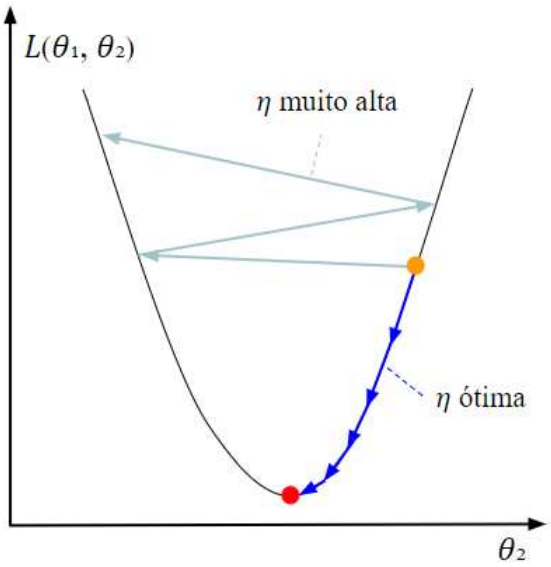

Figura 2.10 – Influência da taxa de aprendizado na busca pelo ponto de ótimo.

Como no gradiente descendente tradicional é necessário que os gradientes para todos os dados de entrada sejam calculados para que se execute uma única atualização, a execução do algoritmo pode ser muito lenta ou, até mesmo, inviável em casos com certa limitação de memória. Uma solução para esse problema é o algoritmo gradiente descendente estocástico (SGD, do inglês *Stochastic Gradient Descent*), que realiza a atualização dos parâmetros para uma única amostra $\mathbf{x}^{(i)}$ com saídas $\mathbf{y}^{(i)}$, selecionada de maneira estocástica (Ruder, 2016).

O SGD é representado pela seguinte equação:

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \eta \cdot \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}; \mathbf{x}^{(i)}; \mathbf{y}^{(i)}) \tag{2.16}$$

Com isso, é esperado que o SGD, além de eliminar redundâncias que podem existir no algoritmo tradicional, acelere o processo de treinamento, possibilitando, em alguns casos, o aprendizado em tempo de execução.

Um certo nível de momento pode ainda ser adicionado ao algoritmo para acelerar a busca em direções relevantes e amortecer oscilações. Concisamente, o momento ajuda na retenção da velocidade do processo de otimização em regiões planas da superfície de erro e evita o problema de mínimo local (Aggarwal, 2018). A sua implementação se dá por meio de um coeficiente de momento α, que define a porção do vetor de atualização $\mathbf{v}$ na iteração $t-1$ a ser usada no cálculo do vetor na iteração t:

$$\begin{aligned} \mathbf{v}_t &= \alpha \cdot \mathbf{v}_{t-1} + \eta \cdot \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}; \mathbf{x}^{(i)}; \mathbf{y}^{(i)}) \\ \boldsymbol{\theta} &= \boldsymbol{\theta} - \mathbf{v}_t \end{aligned} \tag{2.17}$$

Pelo uso do momento, o aprendizado do SGD é acelerado, pois se moverá de maneira consistente na direção que aponta para a solução ótima, permitindo o uso de maiores taxas de aprendizado sem que a qualidade da busca seja afetada. A adição de estocasticidade e do mecanismo de momento conferem ao SGD um desempenho que motivou o surgimento de muitos outros otimizadores dele derivados.

2.2.5 Subajuste e sobreajuste

Em decorrência do treinamento de uma rede neural profunda, podem ser constatados problemas capazes de prejudicar o desempenho do modelo durante a fase de teste, dentre os quais destacam-se o subajuste e o sobreajuste.

O subajuste é observado quando o modelo resultante do treinamento é incapaz de capturar as relações que descrevem o conjunto de dados, tendo pobre desempenho sobre as amostras de treinamento. Nesse caso, soluções relativamente simples podem ser aplicadas, tais como o aumento da quantidade de amostras e o incremento do número de épocas para treinamento (Jabbar; Khan, 2014).

Um problema mais comum e de tratamento mais complexo é o sobreajuste, que ocorre quando um modelo apresenta uma queda significativa no desempenho sobre os dados de teste em comparação ao desempenho obtido durante o processo de treinamento (Mahamad *et al.*, 2010), caracterizando um fenômeno que expõe a baixa capacidade de generalização da rede. Uma forma efetiva de se prevenir o problema de sobreajuste é pela introdução de algum

método de regularização (Hasani *et al.*, 2017). A Figura 2.11 ilustra de maneira simplificada as diferenças entre modelos subajustados e sobreajustados em um espaço de duas variáveis.

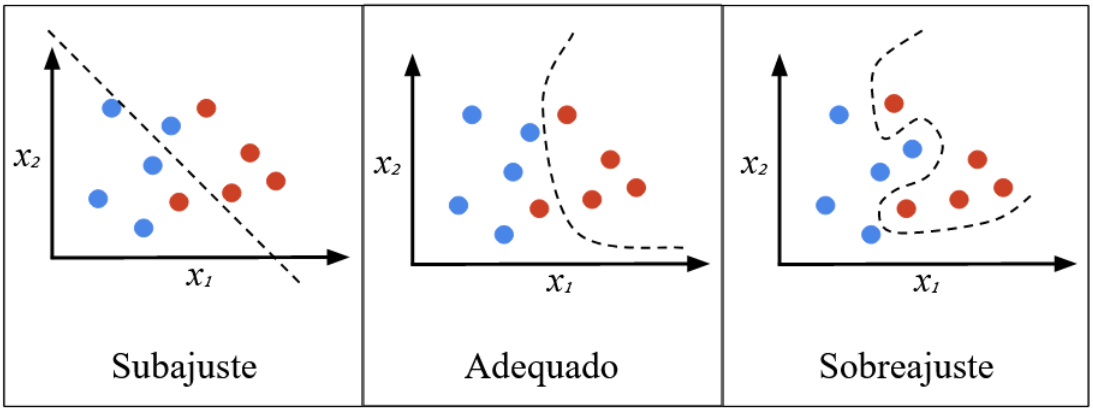

Figura 2.11 – Diferenças entre subajuste e sobreajuste em um modelo de rede neural.

Técnicas tradicionais de regularização consistem no uso de regularizadores baseados em penalidades, que implicam na adição de termos penalizadores à função de erro original, de maneira a restringir a atualização dos parâmetros, reduzindo a complexidade do modelo. Métodos de regularização mais recentes, tais como o *dropout* e a normalização de lote, utilizados neste trabalho, têm recebido grande proeminência graças aos avanços proporcionados.

2.2.6 *Dropout*

O *Dropout* é uma técnica de regularização estocástica amplamente adotada devido à sua simplicidade (Kingma; Ba, 2015), que consiste na inibição de uma porção de neurônios aleatoriamente selecionados em uma camada oculta a cada iteração, baseada na probabilidade definida pelo coeficiente ou taxa ρ. A regularização por *dropout* é comumente utilizada como uma eficiente alternativa aos regularizadores baseados em penalidades e o que torna sua aplicação interessante, além da baixa complexidade de sua implementação, é a sua faixa bem definida de valores de ajuste $[0 \geq \rho \leq 1]$, em que 0 (0%) indica que nenhum neurônio será inibido e 1 (100%) indica que todos os neurônios de uma mesma camada serão inibidos (Srivastava *et al.*, 2014). A Figura 2.12 mostra a atuação do *dropout* entre duas camadas sucessivas.

A implementação do *dropout* é normalmente feita através de uma camada contendo uma máscara binária $\mathbf{d}$ (Khan *et al.*, 2018). Cada elemento $d \in \mathbf{d}$ é independentemente amos-

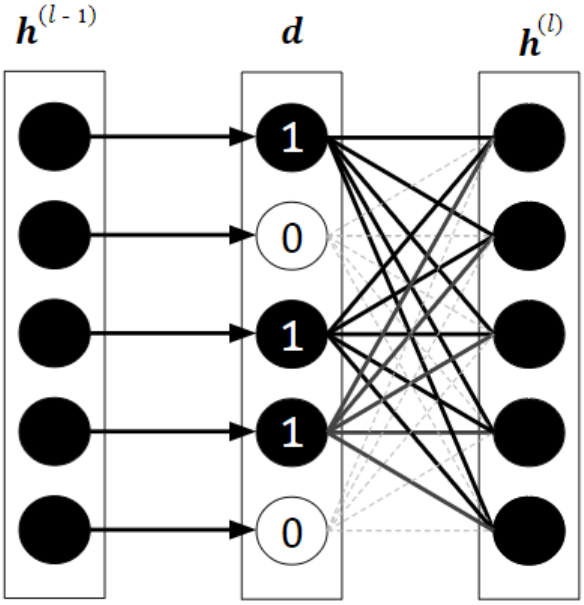

Figura 2.12 – Exemplo de *dropout* entre duas camadas.

trado a partir de uma distribuição de Bernoulli com probabilidade $1 - \rho$ de ser ativado:

$$d \sim Bernoulli(1 - \rho) \tag{2.18}$$

Essa máscara é então aplicada sobre as ativações da camada anterior. O vetor de saídas da camada l, contendo a operação de *dropout*, é dado por:

$$\mathbf{h}^{(l)} = \mathbf{d} \odot \phi(\mathbf{W}^{(l)}\mathbf{h}^{(l-1)} + \mathbf{b}^{(l)}) \tag{2.19}$$

em que $\odot$ denota o produto de Hadamard ou elemento a elemento entre a máscara e as ativações da camada anterior.

2.2.7 Normalização de lote

Durante o treinamento de uma rede neural profunda, a mudança na distribuição das entradas das camadas representa um problema, pois elas precisam se adaptar continuamente à nova distribuição, o que pode retardar o aprendizado e dificultar o treinamento de modelos com não-linearidades saturantes. Tal fenômeno é conhecido como *deslocamento covariável interno* e pode ser atenuado pelo uso de algum tipo normalização sobre as ativações das camadas que, além de acelerar o processo de treinamento, ajuda a evitar o problema de sobreajuste na rede neural (Santurkar *et al.*, 2018). Por esse motivo, em muitos casos, principalmente

nos de redes convolucionais, a normalização das ativações pode até eliminar a necessidade de regularizadores como o *dropout*.

A normalização de lote, baseada no algoritmo proposto por Ioffe e Szegedy (2015), trata-se de uma transformação aplicada sobre cada minilote de ativações. A partir do emprego desse método, é dito que a normalização, até então utilizada somente para fins de pré-processamento, passa a ser parte também da arquitetura interna da rede neural, exercendo o papel de um regularizador capaz de prevenir o sobreajuste e acelerar o processo de aprendizado (Ruder, 2016).

A partir dos valores de média e variância do minilote, cada amostra é submetida a um processo de normalização e, em seguida, aos processos de escalonamento pelo coeficiente γ e deslocamento pelo coeficiente β. Assim, uma camada que receberia x como uma das entradas, passa a receber sua versão transformada $BN_{\gamma,\beta}(x)$.

O algoritmo da normalização de lote é caracterizado pelas seguintes etapas:

(a) Calcula-se a média do minilote:

$$\mu_{\mathcal{B}} = \frac{1}{m}\sum_{i=1}^{m} x_i \tag{2.20}$$

(b) Calcula-se a variância do minilote:

$$\sigma_{\mathcal{B}}^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \tag{2.21}$$

(c) Normaliza-se cada valor de ativação:

$$\tilde{x}_i = \frac{(x_i - \mu_{\mathcal{B}})}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \tag{2.22}$$

(d) Aplica-se escalonamento e deslocamento:

$$y_i = \gamma\tilde{x}_i + \beta \equiv BN_{\gamma,\beta}(x_i) \tag{2.23}$$

Na Equação 2.22, o denominador representa o desvio padrão, que corresponde à raiz quadrada da variância. Nesse caso, o termo ϵ é adicionado para se evitar a possibilidade de divisão por zero.

Existem, ainda, outros métodos de normalização das ativações. A normalização de camada, desenvolvida por Ba *et al.* (2016) diante da dependência observada do método anterior do tamanho do minilote, consiste na versão transposta da normalização de lote. É feita por meio de uma sequência de procedimentos similar à da normalização de lote, porém com a diferença de que a média e a variância são calculadas ao longo das variáveis ou atributos que constituem a entrada da camada de neurônios. Em contrapartida, a normalização de camada é dependente do número de variáveis.

A Figura 2.13 ilustra, de maneira simplificada, as principais diferenças entre a aplicação da normalização de lote e da normalização de camada.

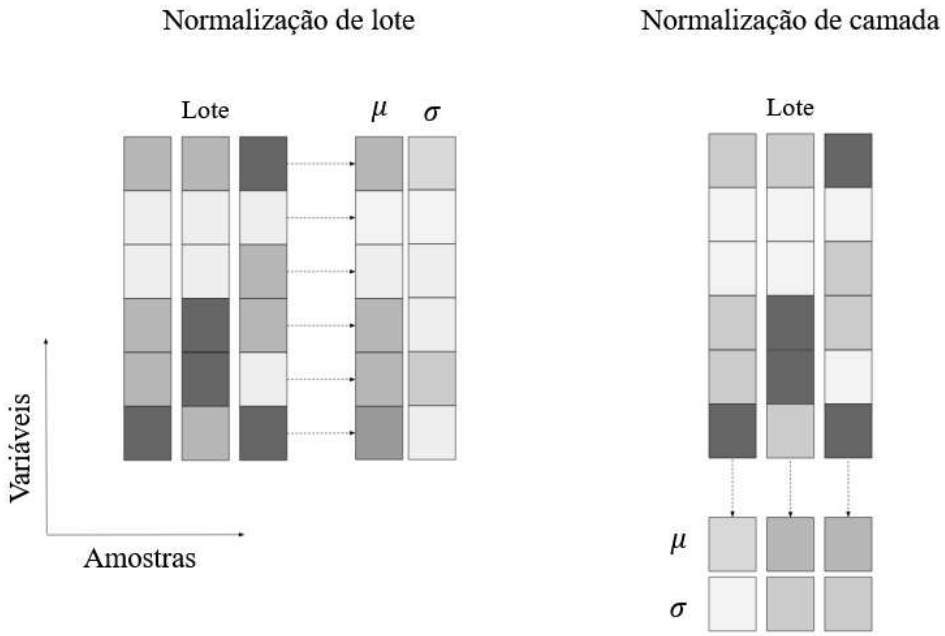

Figura 2.13 – Diferenças entre normalização de lote e normalização de camada.

Apesar de haver diversas alternativas para normalização das ativações, a normalização de lote é utilizada com mais frequência, uma vez que o tamanho do minilote não costuma ser um problema em grande parte das aplicações.

2.2.8 Parada antecipada

Durante o treinamento de uma rede neural, é possível que, em um dado momento, o erro deixe de ser minimizado e comece a aumentar. O crescimento do erro, principalmente sobre os dados de validação, que simulam os dados de teste ao longo do treinamento, pode indicar que

o modelo foi submetido a uma quantidade excessiva de épocas de treinamento, caracterizando uma perda irreversível de generalidade.

Uma alternativa eficaz e de simples implementação para prevenção de sobreajuste é o mecanismo de parada antecipada, que consiste no monitoramento de um indicador específico de desempenho da rede a cada época. Quando o indicador monitorado apresenta uma piora, o processo de treinamento é automaticamente interrompido e o modelo é salvo de acordo com a última época em que se obteve melhoria (Prechelt, 2012). A Figura 2.14 ilustra um exemplo do mecanismo de parada antecipada com base no valor do erro de validação, que é o indicador mais adotado na aplicação desse tipo de algoritmo.

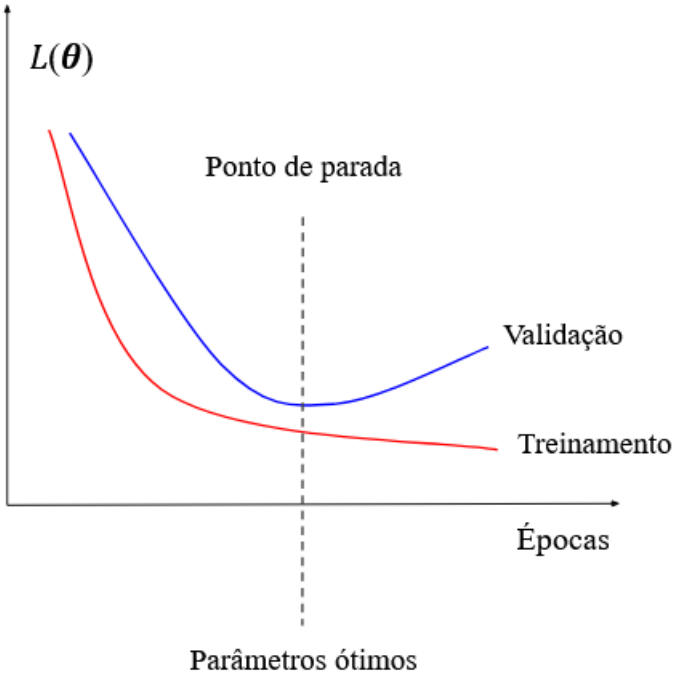

Figura 2.14 – Representação do mecanismo de parada antecipada.

A implementação da parada antecipada leva em consideração duas propriedades básicas: um valor de tolerância a ser considerado no monitoramento da variação do indicador, e o grau de paciência, que define o número de épocas sucessivas que o fenômeno pode ser observado até que o treinamento seja interrompido.

2.2.9 Inicialização dos parâmetros

Para o treinamento de uma rede neural, é necessário que se defina a forma pela qual os parâmetros serão inicializados. Além de favorecer a otimização e possibilitar que sejam atin-

gidos maiores níveis de acurácia, a inicialização adequada dos pesos é um aspecto crucial para a estabilidade do treinamento de redes neurais profundas e a utilização de métodos inadequados pode resultar em problemas como dissipação ou explosão de gradientes (Khan *et al.*, 2018).

Dentre os vários métodos de inicialização de parâmetros existentes, podem ser destacados os pertencentes às três categorias descritas a seguir:

- **Inicialização com zeros:** geralmente aplicada aos vieses, consiste na inicialização de parâmetros com valor zero.
- **Inicialização aleatória:** compreende um vasto conjunto de métodos baseados em distribuições aleatórias para atribuição de valores iniciais aos parâmetros. Os métodos de inicialização aleatória mais populares são o inicializador aleatório gaussiano, o inicializador aleatório uniforme e o inicializador Xavier.
- **Inicialização por pré-treinamento:** consiste na divisão do treinamento em duas etapas sucessivas. No pré-treinamento, a rede é treinada, geralmente, de forma não supervisionada, sendo conduzida a aprender atributos significativos dos dados de entrada. Os parâmetros pré-treinados são então usados para a inicialização do modelo durante a etapa definitiva de treinamento. Trata-se de um método de inicialização capaz de oferecer desempenho superior e maior robustez se comparado a métodos aleatórios convencionais.

2.3 Aprendizado supervisionado

Os diferentes modelos de redes neurais existentes podem ser classificados conforme o tipo de aprendizado ao qual são submetidos. De forma geral, existem três paradigmas de aprendizado muito utilizados atualmente: aprendizado supervisionado, aprendizado não supervisionado e aprendizado por reforço. Apesar de ter recebido crescente atenção nos últimos anos, o aprendizado por reforço não se encontra no escopo do desenvolvimento deste trabalho. Assim sendo, os paradigmas de aprendizado supervisionado e não supervisionado serão abordados com mais detalhes daqui em diante.

No aprendizado supervisionado, a rede é treinada a partir do fornecimento de entradas e padrões de saída correspondentes. Esses pares de entradas e saídas podem ser providos por um professor ou supervisor externo, fazendo com que o rótulo de cada instância seja conhecido e utilizado durante o treinamento. Em síntese, o aprendizado supervisionado se baseia em conjuntos de amostras de classes previamente conhecidas e o aprendizado da rede é então

medido pela sua capacidade de atribuir corretamente um rótulo treinado a uma dada amostra, sendo um paradigma eficiente para se encontrar soluções em diversos problemas lineares e não lineares, tais como classificação, controle de planta e previsão (Sathya; Abraham, 2013).

Dentre as arquiteturas mais populares de aprendizado supervisionado, destacam-se a rede neural perceptron multicamadas e a rede neural convolucional, que serão apresentadas nas próximas subseções.

2.3.1 Perceptrons multicamadas

A unidade básica funcional de uma rede neural do tipo MLP é o perceptron, que calcula a soma ponderada de todas as entradas de um neurônio, somando a elas o valor do viés. O resultado, submetido a uma função de ativação, é obtido pela Equação 2.2, descrita anteriormente nesta seção.

Estendendo-se esse mecanismo, uma MLP profunda é formada pela junção de múltiplas camadas ocultas, capazes de solucionar as restrições de linearidade do perceptron simples (Rababaah *et al.*, 2016). A Figura 2.15 apresenta as diferenças referentes à profundidade de uma rede MLP. Uma MLP rasa é composta de uma única camada oculta, enquanto uma MLP profunda é formada por duas ou mais camadas ocultas (Serwa, 2017).

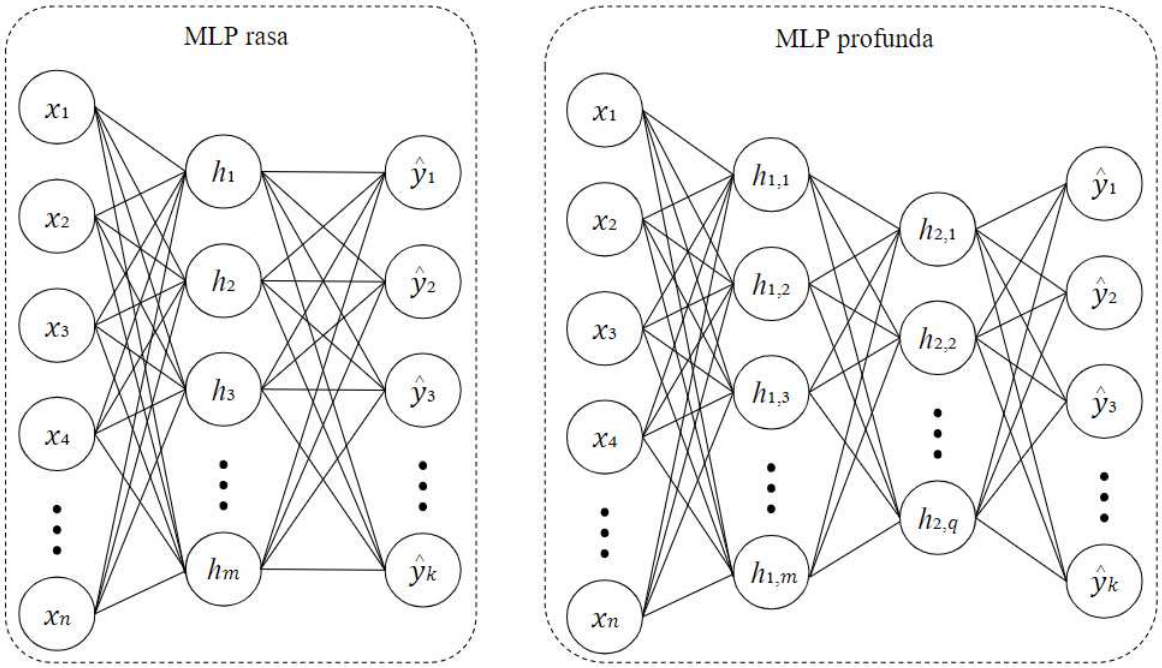

Figura 2.15 – Comparação entre uma MLP rasa e uma MLP profunda.

Nesse caso, não existe uma regra bem definida para a relação entre os tamanhos n,

m e p das camadas, que podem, inclusive, conter o mesmo número de neurônios.

Tendo como base a MLP rasa ilustrada na Figura 2.15, o vetor de saídas é dado pelo cálculo das funções de ativação da camada oculta e da camada de saída aninhadas:

$$\hat{\mathbf{y}} = \phi^{(2)}(\mathbf{W^{(2)}}\phi^{(1)}(\mathbf{W^{(1)}}\mathbf{x} + \mathbf{b}^{(1)}) + \mathbf{b}^{(2)}) \tag{2.24}$$

em que $\phi^{(1)}$, $\mathbf{W}^{(1)}$ e $\mathbf{b}^{(1)}$ são a função de ativação, a matriz de pesos e o vetor de vieses referentes à camada oculta, e $\phi^{(2)}$, $\mathbf{W}^{(2)}$ e $\mathbf{b}^{(2)}$ são os mesmos elementos referentes à camada de saída.

A MLP é um exemplo de rede constituída exclusivamente de camadas inteiramente conectadas, em que todos os neurônios de uma camada são conectados a todos os neurônios da camada anterior (Hoang; Kang, 2019).

2.3.2 Redes neurais convolucionais

A rede neural convolucional é uma classe especial de rede neural artificial que implementa filtros convolucionais, comumente utilizada para a análise de imagens e outras entradas bidimensionais, apesar de possuir versões voltadas para diferentes dimensionalidades. As CNN's são caracterizadas por duas propriedades essenciais: compartilhamento espacial de pesos e amostragem com base em relações espaciais entre os atributos, visando o aprendizado de padrões pela junção de camadas convolucionais e camadas de amostragem, em vez de camadas inteiramente conectadas. Devido à tal característica, as CNN's possuem muito menos conexões e parâmetros se comparadas às redes MLP's (Krizhevsky *et al.*, 2012).

Na Figura 2.16 é mostrado um exemplo de modelo típico de CNN. É importante ressaltar que, apesar da configuração do exemplo ilustrado, não é requerido que as camadas convolucionais e as camadas de amostragem sejam posicionadas de forma intercalada. Existem, portanto, diversos modelos predefinidos de CNN's que realizam a distribuição e a configuração das camadas de diferentes maneiras.

Em uma CNN, as camadas convolucionais executam a convolução de múltiplos filtros com os dados de entrada, gerando atributos locais invariantes à translação. As camadas de amostragem subsequentes são responsáveis pela extração de atributos através do deslizamento de uma janela de tamanho fixo sobre o conjunto de entrada, reduzindo ou aumentando suas dimensões. Para tais propósitos, na programação de uma CNN, a entrada deve ser tratada como um tensor no formato $\{número\ de\ entradas \times largura\ da\ entrada \times altura\ da\ entrada \times profundidade\ da\ entrada\}$.

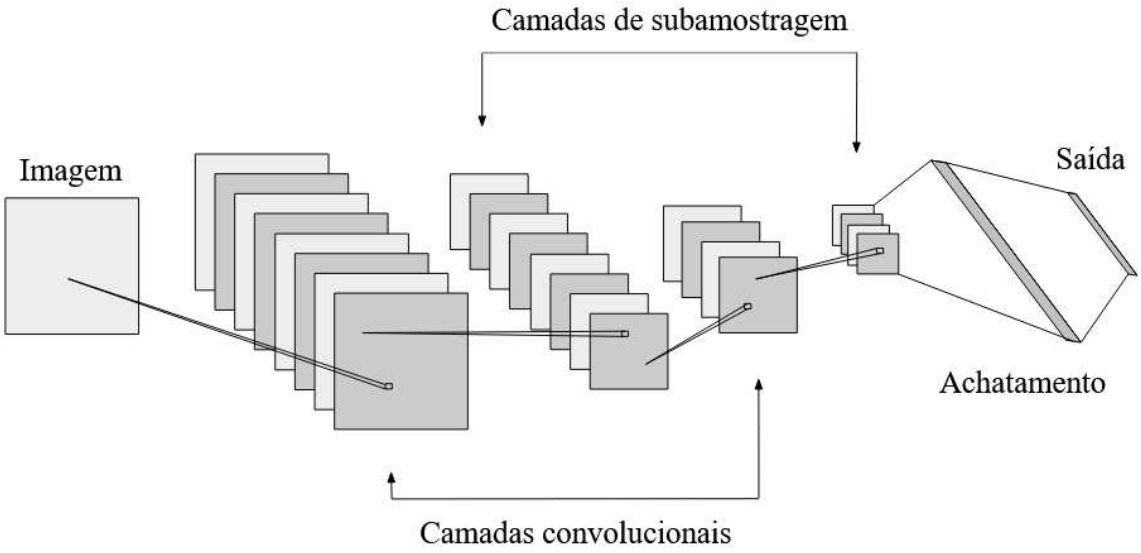

Figura 2.16 – Exemplo de uma rede CNN.

Muito utilizadas na análise de dados sequenciais ou sinais de séries temporais, as CNN's 1D consistem na convolução ao longo de uma única dimensão (largura) de uma dada entrada (Liu *et al.*, 2019). Nas CNN's 2D, os filtros se deslocam ao longo da largura e da altura, correspondentes às duas dimensões que compõem uma imagem (Khan *et al.*, 2018). Nas CNN's 3D, por outro lado, a convolução é realizada ao longo da largura, da altura e da profundidade do objeto analisado, levando em conta, portanto, todas as dimensões do tensor de entrada, de forma a permitir a análise de dados volumétricos e vídeos (Ghadai *et al.*, 2019). Nesse último caso, a dimensão de profundidade se refere à informação temporal, uma vez que um vídeo é composto de imagens ordenadas sequencialmente no eixo do tempo. No entanto, como as principais aplicações das CNN's se encontram associadas ao processamento de imagens, bem como é feito neste trabalho, cada um dos mecanismos que compõem uma rede neural convolucional são detalhados a seguir com base nas CNN's 2D.

A camada convolucional é o elemento mais importante da estrutura de uma CNN. Cada neurônio nessa camada processa uma região específica do sinal de entrada. Os pesos e os vieses, nesse caso, atuam como um conjunto de filtros de convolução. Cada filtro projeta todo o sinal em um novo mapa de atributos, o que significa que o produto escalar é calculado entre o sinal e cada filtro repetidamente (Xie; Zhang, 2017). Os mapas de atributos da camada l resultam da convolução do sinal de entrada ou dos mapas de atributos da camada $l-1$ com

múltiplos filtros contendo o mesmo tamanho, por meio da equação a seguir:

$$\mathbf{H}_j^{(l)} = \phi\left(\sum_{i=1}^{n} \mathbf{W}_{j,i}^{(l)} * \mathbf{H}_i^{(l-1)} + \mathbf{B}_j^{(l)} \right) \tag{2.25}$$

em que $\mathbf{H}_i^{(l-1)}$ é o mapa de atributos i da camada $l-1$, $\mathbf{W}_{j,i}^{(l)}$ é o filtro de convolução entre o mapa de atributo j na camada l e a entrada i na camada $l-1$, e $\mathbf{B}_j^{(l)}$ é a matriz de vieses do mapa de atributos j na camada l.

O processo de geração de um mapa de atributos a partir da convolução de uma entrada com um filtro convolucional é ilustrado na Figura 2.17. Um mapa de atributos na camada $l+1$ resulta da convolução de uma entrada ou mapa de atributos na camada l com um filtro cujas dimensões definem o campo receptivo da extração de atributos e que, em uma rede convolucional, exerce um papel análogo ao de um neurônio.

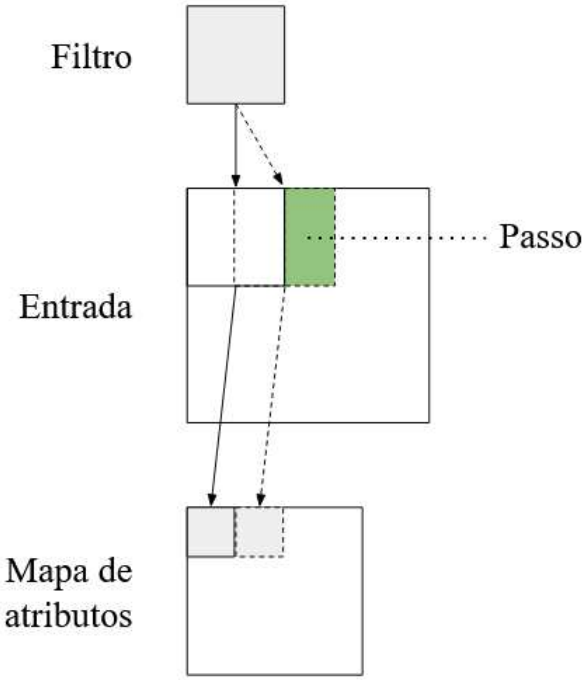

Figura 2.17 – Operação de convolução entre uma entrada e um filtro.

Nota-se que a primeira propriedade que deve ser levada em conta na escolha do filtro é o seu tamanho. As dimensões do filtro, que normalmente possui um formato quadrado, com largura e altura iguais, implicam na resolução da convolução e, consequentemente, no grau de detalhamento da captura de atributos. Outra propriedade importante é o passo da convolução, que determina a forma como o filtro se desloca ao longo da entrada e o nível de sobreposição entre duas convoluções consecutivas.

No entanto, pode haver ainda problemas quanto às dimensões dos novos mapas de atributos. Como pode ser observado na Figura 2.17, se aplicada puramente, a convolução faz com que haja perdas tanto na altura como na largura do mapa resultante em relação à entrada. A solução é o uso do preenchimento que consiste em incluir conjuntos de zeros ao redor da entrada para que, após a convolução, sejam mantidas as dimensões originais. A Figura 2.18 ilustra o mecanismo de preenchimento. À convolução sem preenchimento é dado o nome de *convolução válida* e à convolução com preenchimento é dado o nome de *convolução similar* (Khan *et al.*, 2018).

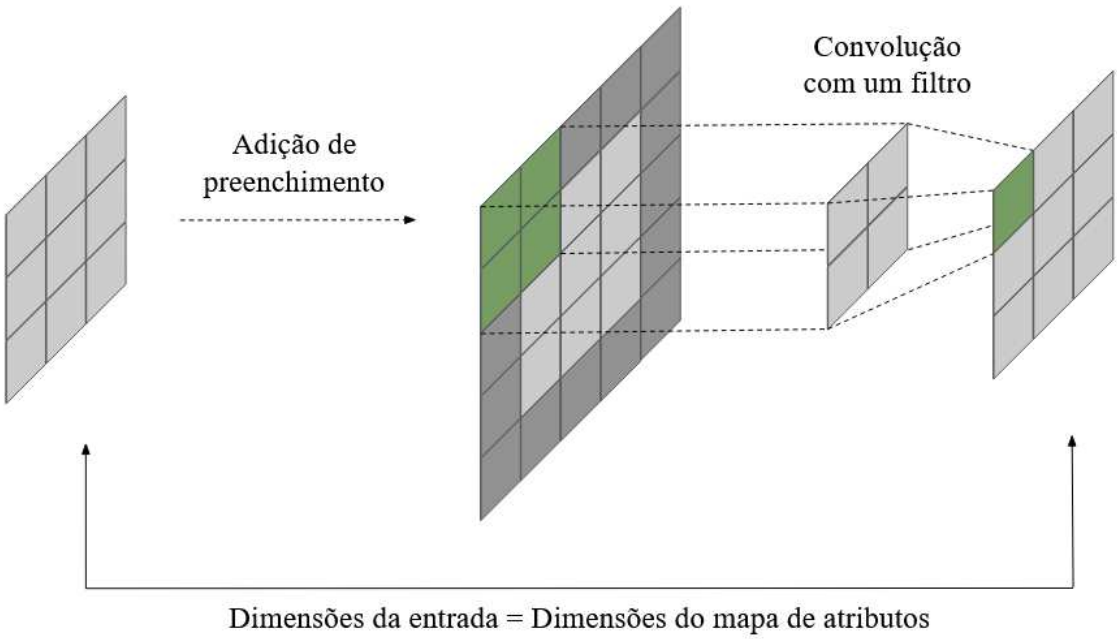

Figura 2.18 – Convolução com preenchimento na entrada.

Sem a utilização de preenchimento, as dimensões (largura ou altura) do mapa de atributos resultantes da convolução são dadas por:

$$\hat{N} = \frac{N - F + S}{S} \tag{2.26}$$

em que N é a largura/altura do mapa de atributos de entrada, F é a largura/altura do filtro e S é o tamanho do passo. Na medida em que a condição $S > 0$ deve ser satisfeita para que o filtro se desloque ao longo das entradas, para qualquer valor de S, considerando um filtro não unitário, isto é, $F > 1$, ocorre uma redução da saída.

Entretanto, é possível adotar um preenchimento de tamanho Z tal que $\hat{N} = N$.

Assim, as dimensões do mapa resultante passam a ser definidas por:

$$\hat{N} = \frac{N - F + S + Z}{S} \tag{2.27}$$

Os atributos extraídos na camada convolucional podem então ser submetidos à camada de amostragem. A operação de subamostragem, mais comum em CNN's, consiste na redução das imagens e, portanto, do número de parâmetros, preservando informações relevantes nelas contidas (Sharma *et al.*, 2018). A sobreamostragem, usualmente aplicada quando se deseja reconstruir uma imagem reduzida, realiza o procedimento inverso, aumentando suas dimensões. A Figura 2.19 apresenta as diferenças entre as operações de subamostragem e de sobreamostragem com dimensões $M \times M$.

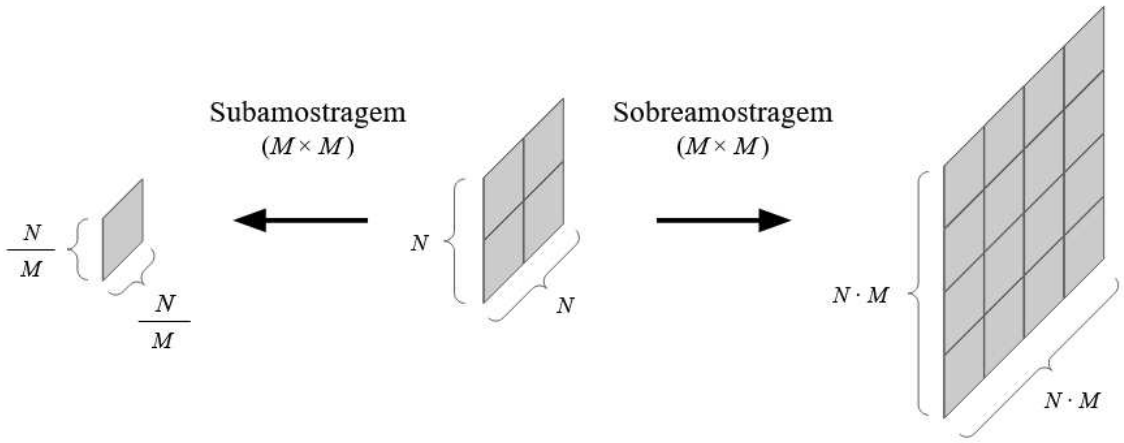

Figura 2.19 – Operações de subamostragem e sobreamostragem.

Assim como as camadas convolucionais, as camadas de subamostragem também estão sujeitas às configurações de passo e de preenchimento. Contudo, para a execução da subamostragem, dois métodos diferentes podem ser escolhidos. Na subamostragem pelo valor máximo, a saída em cada passo equivale ao valor mais alto dentre todos os existentes na região considerada. Na subamostragem pelo valor médio, é calculada a média de todos os valores presentes nessa mesma região (Khan *et al.*, 2018).

De acordo com Zhao *et al.* (2017), CNN's são capazes de automaticamente identificar atributos de imagens de entrada, mostrando-se, ainda, robustas a efeitos de translação, escalonamento e rotação. Para a classificação de dados de séries temporais, os sinais podem ser tratados como imagens e a atribuição de rótulos pode ser realizada em uma camada inteiramente conectada ou, até mesmo, em uma MLP anexada à saída da parte convolucional. Para

isso, é necessário que os mapas bidimensionais sejam antes submetidos ao processo de achatamento, no qual são convertidos para o formato 1D requerido para processamanto em camadas inteiramente conectadas.

2.4 Aprendizado não supervisionado

O paradigma de aprendizado não supervisionado consiste em um vasto conjunto de modelos em que se treina unidades de saída de forma que indiquem padrões relativos aos conjuntos de entradas. Nesse paradigma, o sistema deve descobrir características estatísticas proeminentes da população de entradas. Ao contrário do paradigma de aprendizado supervisionado, não há conhecimento prévio das categorias nas quais as amostras devem ser classificadas. Em vez disso, o sistema deve aprender a desenvolver representações da entrada.

Técnicas não supervisionadas de aprendizado profundo são usadas como alternativas à extração manual de atributos, podendo gerar representações ainda mais eficientes de maneira automática e robusta (Shao *et al.*, 2017).

Em grande parte das aplicações às quais se destinam, especialmente em tarefas de diagnóstico, redes neurais não supervisionadas são utilizadas para o pré-treinamento de modelos que, posteriormente, realizam classificação de amostras por meio de treinamento supervisionado (Lu *et al.*, 2017).

2.4.1 *Autoencoders*

O *Autoencoder* é um tipo de rede neural treinada de forma não supervisionada voltada ao aprendizado de novas representações de um conjunto de dados através da reconstrução da entrada. Para isso, um AE é composto de um *encoder*, que transforma a entrada original $\mathbf{x}$ em uma representação $\mathbf{h}$ em um espaço latente de menor dimensionalidade, e um *decoder*, que gera uma versão reconstruída da entrada $\hat{\mathbf{x}}$ a partir da representação latente (Zhao *et al.*, 2019). A Figura 2.20 ilustra um *autoencoder* raso, contendo uma única camada oculta.

As saídas do *encoder* e do *decoder* de um *autoencoder* raso são dadas respectivamente por:

$$\begin{aligned} \mathbf{h} &= \phi(\mathbf{W}^{(1)}\,\mathbf{x} + \mathbf{b}^{(1)}) \\ \hat{\mathbf{x}} &= \phi(\mathbf{W}^{(2)}\,\mathbf{h} + \mathbf{b}^{(2)}) \end{aligned} \tag{2.28}$$

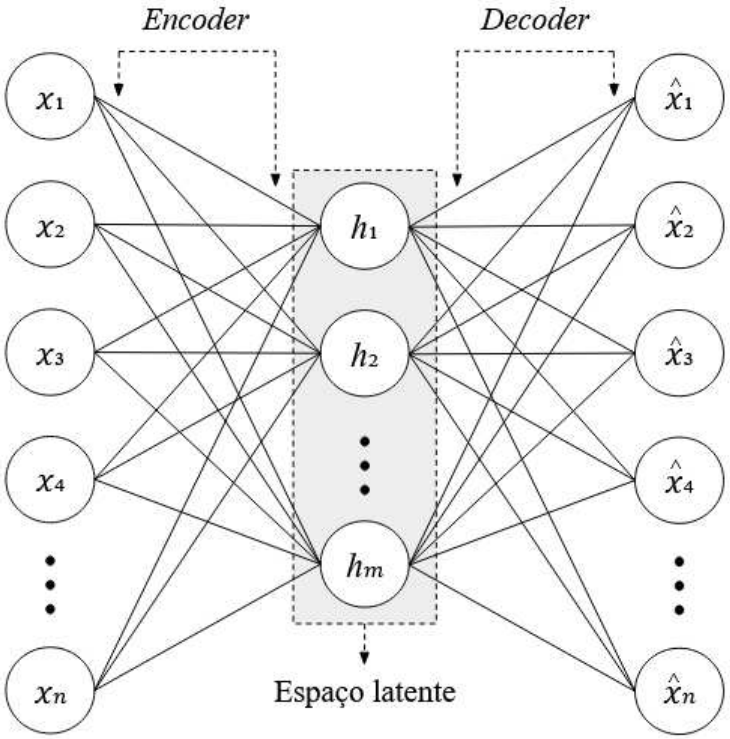

Figura 2.20 – Exemplo de uma rede do tipo AE para pré-treinamento.

em que $\mathbf{W}^{(1)}$ e $\mathbf{b}^{(1)}$ são a matriz de pesos e o vetor de vieses do *encoder*, e $\mathbf{W}^{(2)}$ e $\mathbf{b}^{(2)}$ são a matriz de pesos e o vetor de vieses do *decoder*.

A arquitetura apresentada na Figura 2.20 se trata de um AE *subcompleto*, para o qual é necessário que a condição $m < n$ seja satisfeita. Apesar de menos utilizado, existe também o AE *sobrecompleto*, que, ao contrário do subcompleto, aloca a informação do sinal de entrada em um espaço de maior dimensionalidade.

Quando aplicado a tarefas de monitoramento de condição, o AE permite a extração de atributos relevantes do sistema monitorado por meio do processo de pré-treinamento (Verma *et al.*, 2013). Nesse caso, uma vez que os atributos estão contidos no espaço latente, o *decoder* é descartado e o processo de refinamento é então conduzido a partir dos parâmetros pré-treinados do *encoder*, conectando-o a uma camada de saída com ativação sigmoide ou *softmax*.

A Figura 2.21 apresenta um classificador baseado em AE. Nota-se que a estrutura do modelo na fase de refinamento assemelha-se a uma MLP convencional, com a diferença de que, pela utilização do AE, os pesos e vieses são inicializados com os valores obtidos no pré-treinamento.

Assim como uma MLP, um AE é considerado profundo se contém múltiplas camadas ocultas. A camada central, também denominada *gargalo*, na qual se encontra a repre-

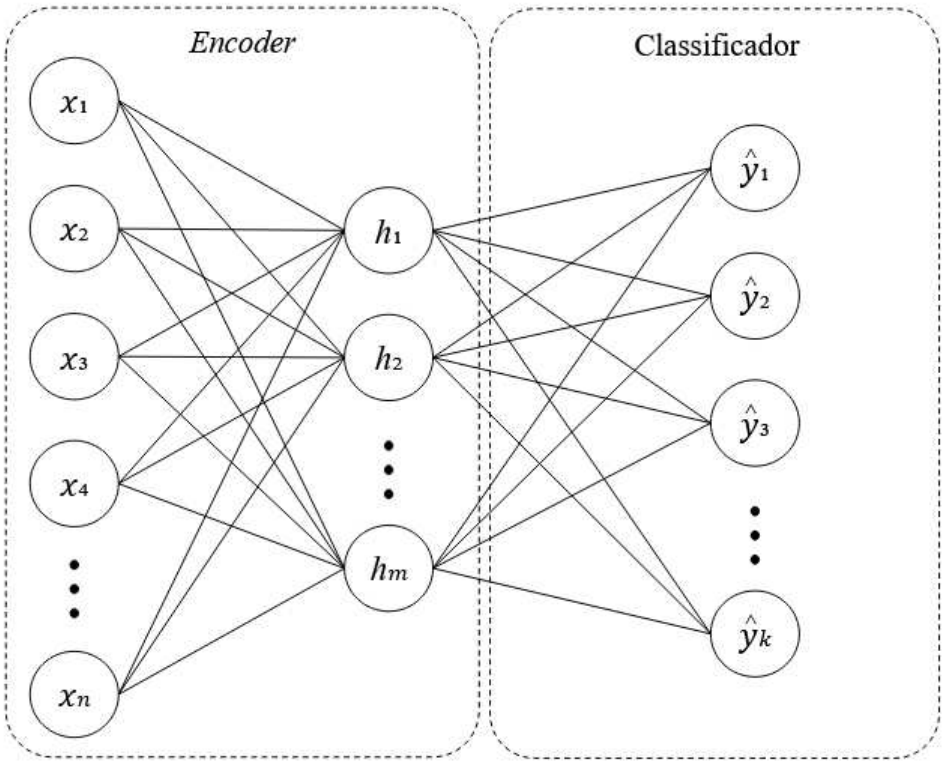

Figura 2.21 – Refinamento a partir do encoder pré-treinado.

sentação latente, contém informações comprimidas referentes à entrada. O resultado, de forma geral, é a redução da dimensionalidade dos dados na saída do *encoder*, em certos casos de forma muito significativa.

Segundo Hinton e Salakhutdinov (2006), um *autoencoder* profundo pode ser visto como uma generalização não linear da análise de componentes principais (PCA, do inglês *Principal Component Analysis*), capaz de solucionar as limitações do mesmo quanto à análise de conjuntos de dados contendo relações não lineares. Além disso, por serem modelados de forma paramétrica, AE's oferecem maior flexibilidade em termos de projeto e manutenção, podendo ser aplicados em casos nos quais se trabalha com quantidades extensas de dados.

Na Figura 2.22 é apresentado um exemplo real do uso de um *autoencoder* para a transformação de dados gerados aleatoriamente no espaço $\mathbb{R}^3$ para o espaço $\mathbb{R}^2$. Apesar de tais resultados terem sido obtidos através de um modelo de AE raso isento de qualquer processo de aperfeiçoamento de arquitetura, eles possibilitam uma visualização clara dos efeitos da aplicação de *autoencoders* para a representação de um mesmo conjunto de dados em espaços de diferentes dimensionalidades. Com o uso do AE, busca-se tornar as classes mais distinguíveis em um espaço reduzido.

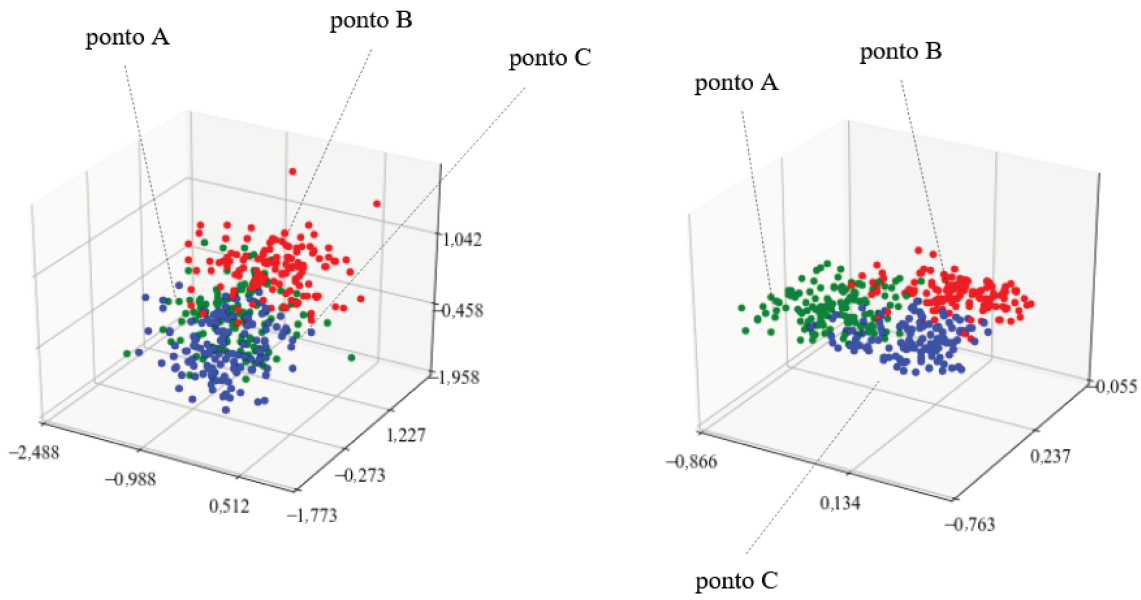

Figura 2.22 – Representação original em três dimensões (à esquerda) e representação comprimida em duas dimensões (à direita).

Visando otimizar e estender sua aplicabilidade, diversas variantes do AE foram desenvolvidas ao longo dos anos (Baldi, 2011), dentre as quais destacam-se três básicas (Zhao *et al.*, 2019):

- ***Sparse autoencoder*:** um termo regularizador é adicionado à função de erro para impedir que o AE aprenda a identidade da entrada. Camadas de *dropout* também podem ser usadas para regularização.

- ***Denoising autoencoder* (DAE):** através da introdução parcial de ruído ou da corrupção do sinal de entrada por meio de *dropout*, a rede é condicionada a aprender representações mais robustas dos dados (Vincent *et al.*, 2008).

- ***Stacked autoencoder*:** *encoders* provenientes de um conjunto de AE's treinados sequencialmente são empilhados, formando uma estrutura profunda em que a representação codificada do *autoencoder* l é usada como entrada do *autoencoder* $l + 1$ (Ma *et al.*, 2018). Trata-se do método de treinamento camada por camada.

2.4.2 *Convolutional autoencoders*

O *convolutional autoencoder* (CAE) é uma variante do AE criada especialmente para o processamento de imagens e outros dados multidimensionais. Assim como um típico

AE, o CAE é formado por um *encoder* e um *decoder*, porém, no lugar de camadas inteiramente conectadas, são usadas camadas de subamostragem, sobreamostragem e convolucionais. Desse modo, o CAE pode ter desempenho superior aos AE's convencionais por incorporar relações espaciais entre *pixels* em uma imagem (Guo *et al.*, 2017). A Figura 2.23 ilustra uma arquitetura CAE típica.

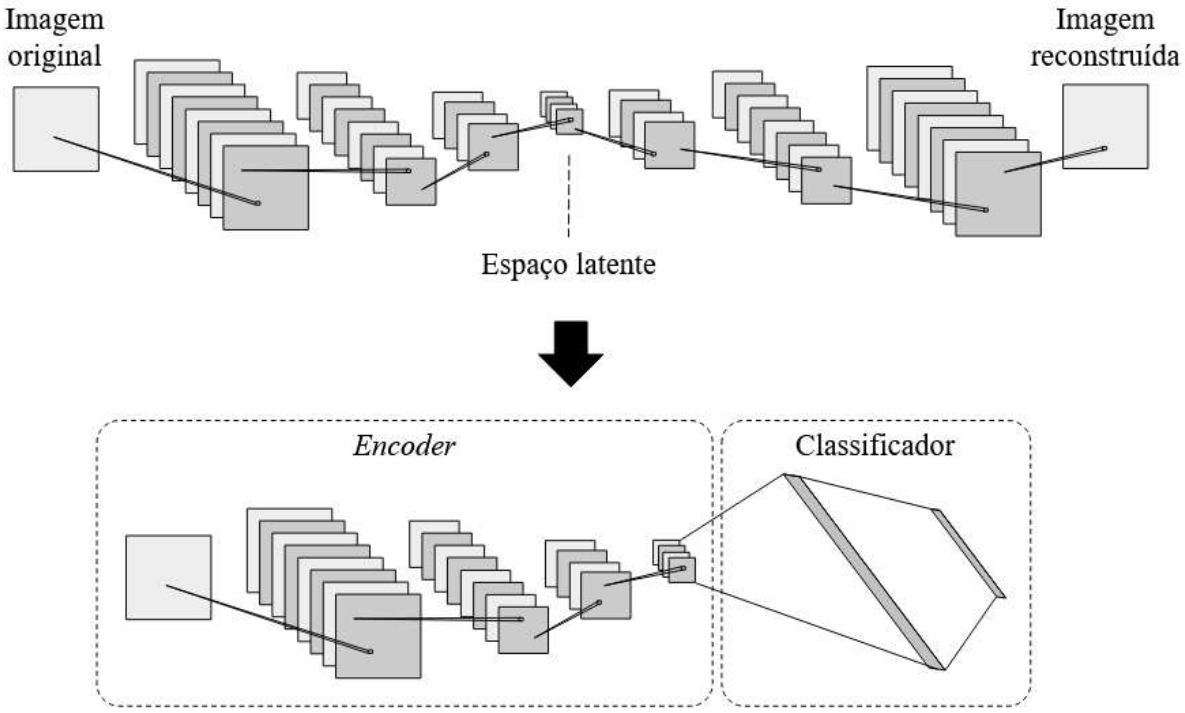

Figura 2.23 – Exemplo de uma rede do tipo CAE.

De maneira similar à classificação com AE, a parte do *encoder* pode ser achatada e processada por uma camada que retorna a probabilidade referente a cada classe. Além disso, é possível notar que, durante a fase de refinamento, a junção do *encoder* e da camada de classificação configura uma topologia CNN típica, porém com parâmetros inicializados de acordo com os valores obtidos no processo de pré-treinamento.

A compressão dos dados de entrada é realizada no *encoder* pelas operações de convolução e de subamostragem. No *decoder*, a descompressão é feita através das operações de convolução ou deconvolução, e de sobreamostragem. A deconvolução pode ainda ser vista como uma convolução transposta, em que se utiliza um filtro resultante da transposição e da inversão do tensor referente ao filtro de convolução original (Aggarwal, 2018).

A exemplo da CNN, existem diversos modos de se estruturar uma rede do tipo CAE. Em algumas arquiteturas, a amostragem não é aplicada logo após cada camada convolucional, e sim somente após múltiplas convoluções. Na maioria dos modelos, o número de filtros nas

camadas convolucionais aumenta conforme se aproxima do gargalo, no entanto, há casos em que o número de filtros diminui entre a entrada da rede e o gargalo, e aumenta entre o gargalo e a saída.

O principal benefício do *convolutional autoencoder* pode ser observado quanto à capacidade de otimização dos pesos e vieses. Os experimentos conduzidos por Masci *et al.* (2011) mostraram que CNN's pré-treinadas com o auxílio de CAE's possuem uma leve, porém consistente vantagem em relação às redes inicializadas com parâmetros aleatórios.

3 METODOLOGIA PROPOSTA

Nas seções a seguir, é descrito em detalhes o sistema desenvolvido para diagnóstico de falhas em máquinas rotativas, desde as etapas de pré-processamento e preparação dos dados de vibração, até a etapa de aplicação e avaliação da rede neural proposta.

3.1 Descrição do sistema

O sistema implementado consiste de duas fases de processamento: *online* e *offline*. O diagrama da Figura 3.1 apresenta a dinâmica do sistema e a relação entre as fases. O processamento *offline*, que ocorre em tempo de projeto, é conduzido para treinamento e definição do modelo. A melhor configuração dentre as avaliadas durante a fase *offline* é definida como o modelo definitivo a ser utilizado no processamento *online*, que ocorre em tempo de execução ou teste do sistema. Durante essa fase, o modelo treinado é aplicado para a classificação de sinais diferentes dos utilizados para treinamento, caracterizando o processo de diagnóstico das amostras de teste.

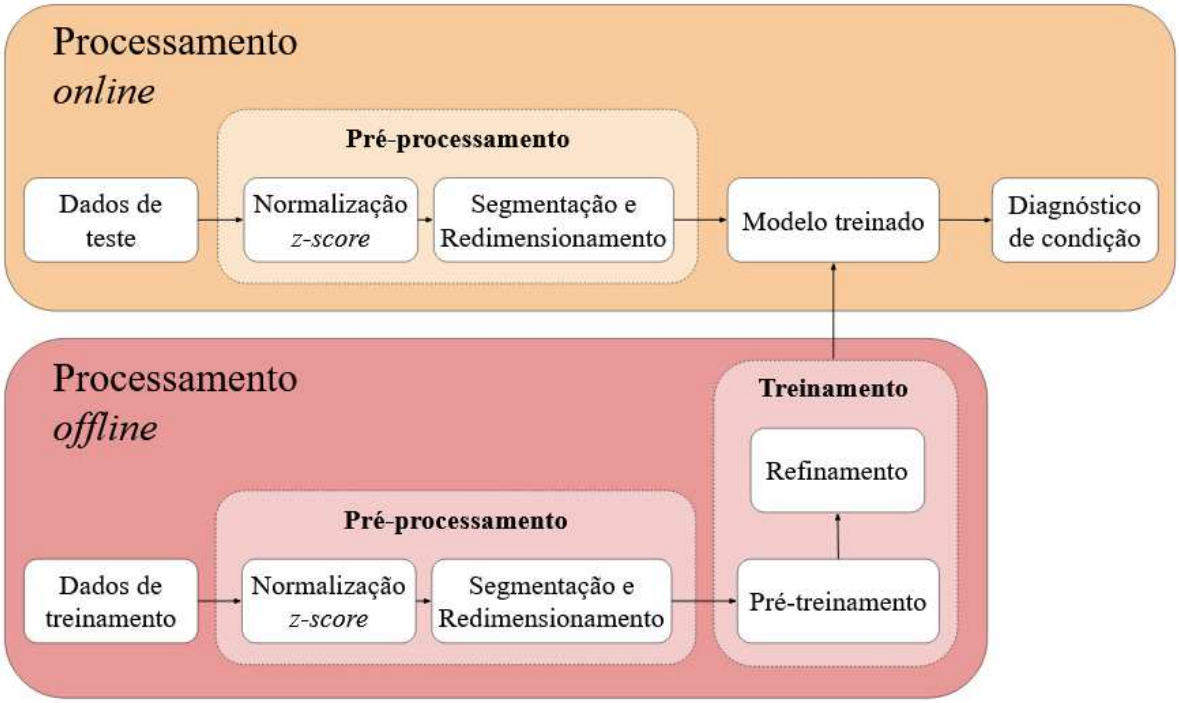

Figura 3.1 – Sistema de diagnóstico proposto.

É importante notar que, tanto na fase *online*, como na fase *offline*, são executados os mesmos procedimentos de pré-processamento dos sinais brutos para que o modelo esteja

habilitado a produzir respostas coerentes. Encerrado o treinamento, o sistema passa a ser definido simplesmente pela estrutura de processamento *online*, e o processamento *offline* é somente acionado quando observada a necessidade de se realizar ajustes no modelo.

Para que seja possível avaliar o desempenho do sistema de forma apropriada, o conjunto de amostras disponível é dividido entre subconjunto de amostras de treinamento e subconjunto de amostras de teste. O uso de dados de teste diferentes dos dados utilizados para o treinamento do modelo permite que seu desempenho seja verificado em relação a dados desconhecidos. Durante o treinamento, uma parte do subconjunto de dados deve ser reservada para validação, que visa simular as amostras de teste ao longo de cada época. Por meio da validação, pode-se prever com exatidão o desempenho da rede neural sobre dados desconhecidos ainda durante seu treinamento.

Nos experimentos realizados, são reservadas 80% das amostras para o subconjunto de treinamento e 20% para o subconjunto de teste. Do subconjunto de treinamento, 10% das amostras são utilizadas para validação, o que corresponde a 8% do total de amostras disponíveis.

3.2 Pré-processamento dos dados

A partir da aquisição dos sinais brutos de vibração, uma série de tratamentos é realizada para que os mesmos sejam devidamente preparados para serem processados pela rede neural. O pré-processamento conduzido neste trabalho consiste na normalização, segmentação e redimensionamento dos dados coletados, respectivamente, conforme os procedimentos descritos a seguir.

3.2.1 Normalização das entradas

Em muitos casos, utilizar dados brutos diretamente como entradas de uma rede neural pode ocasionar problemas que dificultam ou, no pior dos casos, impedem o aprendizado. Tal fenômeno se deve à grande diferença na escala dos valores brutos. No sistema proposto, a normalização *z-score*, como uma forma simples e eficaz de se resolver problemas desse tipo pelo condicionamento do sinal original a uma escala comum com propriedades bem definidas, é aplicada tanto no subconjunto de treinamento como no subconjunto de teste.

A normalização de dados é uma etapa importante em diversos algoritmos de aprendizado de máquina, sendo geralmente aplicada na entrada de redes neurais para limitar as variáveis a uma faixa de valores que as torna comparáveis umas às outras e, consequentemente,

menos vulneráveis a efeitos de instabilidade durante o processo de treinamento (Lee *et al.*, 2018; Roland; Eseosa, 2014).

Através da normalização *z-score*, o sinal original é escalonado com as propriedades de uma distribuição normal pelo cálculo do desvio padrão em relação à média (Du *et al.*, 2017), assegurando um sinal resultante com média $\mu = 0$ e desvio padrão $\sigma = 1$, através da seguinte equação (Heumann *et al.*, 2016):

$$\tilde{x} = \frac{x - \mu}{\sigma} \tag{3.1}$$

Na Figura 3.2, a distribuição do sinal normalizado das primeiras amostras de sinal de vibração de um rolamento é comparada à do sinal bruto por meio de histogramas. Nota-se que o sinal normalizado (histograma à direita) está contido em uma faixa de valores muito mais compacta que a faixa de valores do sinal original (histograma à esquerda). Uma das vantagens mais importantes decorrentes desse efeito é o aumento da velocidade de treinamento.

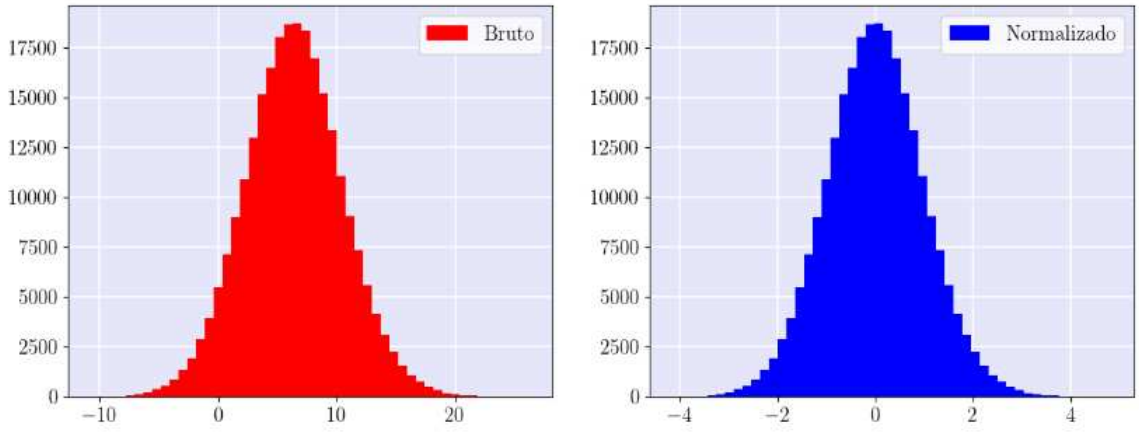

Figura 3.2 – Histogramas do sinal bruto (à esquerda) e do sinal normalizado (à direita) das primeiras 256000 medições de vibração de um rolamento.

A influência da normalização sobre o desempenho do modelo é um dos aspectos analisados ao longo dos experimentos realizados. Uma comparação com o treinamento realizado com sinais brutos é realizada para que se verifique a necessidade de normalizar os dados de vibração para aprendizado e teste.

3.2.2 Segmentação e redimensionamento

Os sinais de vibração considerados neste trabalho caracterizam séries temporais univariáveis, descritas por um único canal de medição. Portanto, devido à inconveniência de

se ter uma rede neural com somente uma variável de entrada, alguns procedimentos devem ser adotados para que a rede possa processar os dados na forma de agrupamentos de múltiplas entradas.

A Figura 3.3 ilustra o processo de preparação dos dados, conduzido em duas etapas. Na primeira etapa, os sinais normalizados são divididos em segmentos contendo o mesmo número de pontos amostrados. Em seguida, cada segmento ou vetor de tamanho N gerado é redimensionado de maneira a ser representado por uma matriz no formato $\sqrt{N} \times \sqrt{N}$, que define uma imagem de vibração. O redimensionamento de um segmento é feito através de um processo simples no qual seus elementos são dispostos na matriz resultante, linha por linha, de acordo com suas respectivas posições no vetor original.

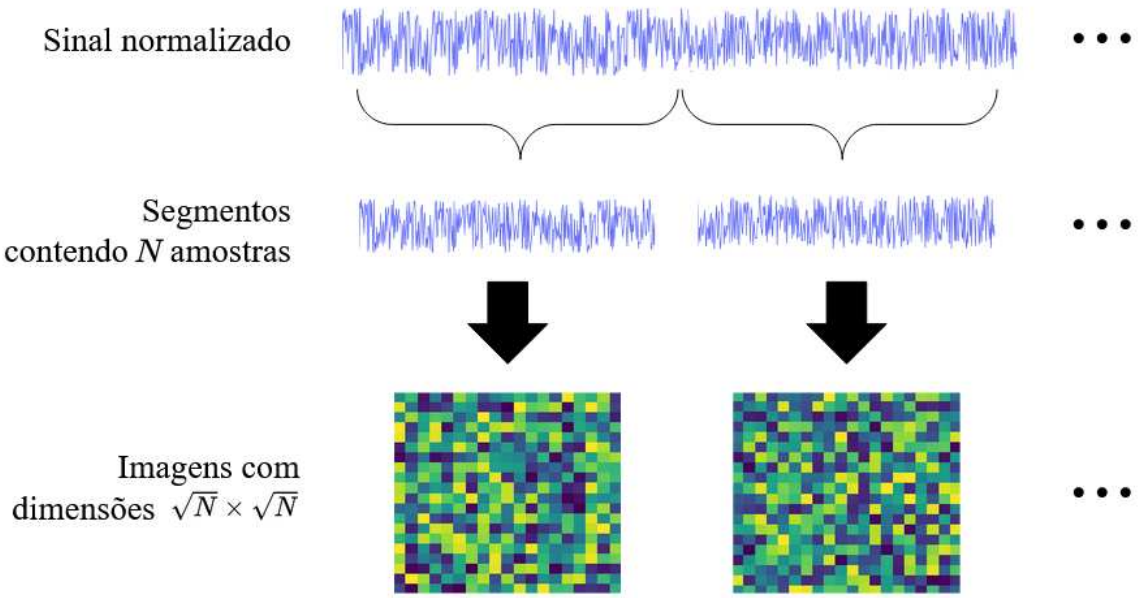

Figura 3.3 – Redimensionamento do sinal para processamento 2D.

A segmentação de dados de séries temporais tem sido comumente adotada para o pré-processamento de sinais de vibrações. Alguns métodos, tais como o proposto por Hoang e Kang (2019), também consistem na geração de imagens de vibrações a partir de tais segmentos. No entanto, em contraste ao procedimento proposto neste trabalho, o algoritmo descrito por Hoang e Kang (2019) se baseia na normalização simples dos sinais brutos, condicionando-os a amplitudes contidas em uma faixa bem definida, de 0 a 255, correspondente à escala de cinza.

Treinar a rede neural com imagens pode conferir importantes vantagens para o processo de extração de atributos. Pela utilização de imagens de segmentos de séries temporais, espera-se que as relações existentes entre os pontos dos sinais de vibração em um mesmo seg-

mento possam ser capturadas de maneira eficiente através de filtros convolucionais.

3.3 *Denoising convolutional autoencoder*

Visando a obtenção de melhorias significativas em relação ao CAE convencional quanto à acurácia e robustez, elementos específicos de outras variantes de AE são adotados na modelagem do *denoising convolutional autoencoder*. A Figura 3.4 ilustra a configuração de DCAE proposta, composta de dois estágios: um *autoencoder* assimétrico para pré-treinamento e uma rede definitiva para classificação. A assimetria é caracterizada pelas diferenças estruturais existentes entre o *encoder* e o *decoder*. Pode-se observar que camadas de *dropout* e normalização de lote são aplicadas somente na parte do *encoder*, de modo a regularizar o processo de compressão da informação de entrada.

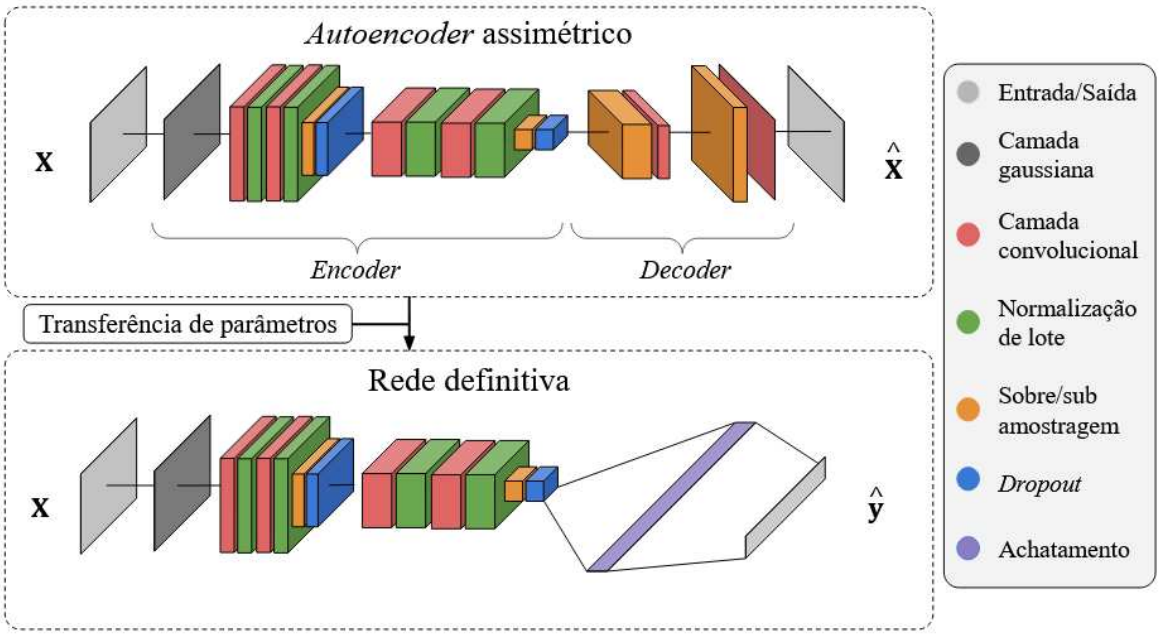

Figura 3.4 – Modelo DCAE proposto para treinamento em duas fases.

O treinamento do modelo é realizado de maneira sequencial em duas fases. Na primeira fase, é conduzido o treinamento não supervisionado do *autoencoder*, que configura a fase de pré-treinamento. Na segunda fase, o modelo é refinado por meio da rede definitiva em um processo de treinamento supervisionado. Ambos os estágios e suas respectivas dinâmicas de treinamento são descritos detalhadamente a seguir.

3.3.1 *Autoencoder* assimétrico

No estágio de *autoencoder* assimétrico, camadas de *dropout* são inseridas após cada camada de subamostragem no *encoder*. A partir do conceito de *denoising autoencoders*, um ruído gaussiano é adicionado ao sinal de entrada para que a rede aprenda a filtrá-lo na reconstrução da entrada original, o que garante melhoria de desempenho. A normalização de lote é efetuada após cada camada convolucional no *encoder* para contribuir na prevenção de sobreajuste e acelerar o treinamento. Na parte do *decoder*, somente camadas convolucionais e de sobreamostragem são utilizadas para que se garanta que as saídas geradas tenham as mesmas dimensões das entradas, sem que haja perda de informação ao longo da propagação.

Nesse estágio, o objetivo é treinar o *autoencoder* assimétrico para que aprenda a produzir reconstruções apuradas das imagens de entrada, comprimindo no gargalo suas respectivas representações em um espaço latente de menor dimensionalidade.

3.3.2 Rede definitiva

Como pode ser visto na Figura 3.4, os parâmetros pré-treinados do *encoder* são transferidos para o estágio de rede definitiva, no qual, na medida em que o *decoder* é descartado, a saída do *encoder* é redimensionada para o formato 1D por meio de um processo de achatamento, e uma camada inteiramente conectada é anexada ao final da rede. Nessa última camada, os atributos extraídos ao longo das estruturas de convolucão e de subamostragem do *encoder* são classificados pela aplicação da função de ativação *softmax*.

Diferentemente do estágio anterior, a rede definitiva é treinada para a tarefa de diagnóstico, que consiste em associar a cada imagem de entrada um rótulo especificando um estado de saúde do componente.

3.3.3 Corrupção da entrada com ruído gaussiano

Em adição às técnicas de regularização aplicadas nas camadas intermediárias, a injeção de ruído gaussiano leva a rede a extrair atributos mais significativos dos dados de vibração. O padrão de ruído gaussiano representa uma aproximação satisfatória de cenários reais (Boyat; Joshi, 2015) e sua função de densidade de probabilidade (PDF) é descrita por:

$$P(x) = \sqrt{\frac{1}{2\pi\sigma^2}} \cdot exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \tag{3.2}$$

Diferentes níveis de ruído são testados para se determinar o valor para o qual a rede obtém os melhores resultados. Com isso, espera-se que o modelo, além de obter melhoria significativa de acurácia, seja capaz de manter desempenho satisfatório mesmo quando sujeito a ruído semelhante ao existente em ambientes reais de operação.

3.3.4 Hiperparâmetros do modelo

Os hiperparâmetros que descrevem o modelo DCAE são divididos em dois grupos de acordo com a maneira pela qual são definidos. Os hiperparâmetros fixos, cujos valores são mantidos ao longo de todos os experimentos para todas as configurações analisadas, são definidos a partir de recomendações de trabalhos utilizados como referências e de convenções bem estabelecidas no campo de redes neurais. Os hiperparâmetros selecionáveis, por sua vez, são aqueles cujos valores são definidos ao longo dos experimentos com base em métricas específicas de desempenho. São adotados como selecionáveis os hiperparâmetros com maior potencial de influência sobre o desempenho do modelo.

A Tabela 3.1 apresenta os principais hiperparâmetros fixos do DCAE, obtidos através de diversos testes preliminares. A taxa de aprendizado é constante, uma vez que nenhum decaimento é aplicado. As dimensões para as operações de subamostragem e sobreamostragem são iguais para que as imagens sejam reconstruídas com o mesmo formato das originais. O tamanho do lote é o mesmo para todas as fases de treinamento, submetidas a diferentes números de épocas. No pré-treinamento, é aplicado o mecanismo de parada antecipada com duas épocas de paciência. No refinamento, o treinamento é executado incondicionalmente até a última época e o modelo salvo é aquele para o qual se obtém a melhor acurácia de validação.

Por se tratar de uma rede do tipo convolucional, os principais fatores que definem sua redução de dimensionalidade são o número de camadas de subamostragem e suas respectivas dimensões. As dimensões de subamostragem utilizadas (2×2) são as mais comumente adotadas em projetos de redes convolucionais, tais como os modelos propostos por Jing *et al.* (2017), Wen *et al.* (2018) e Xu *et al.* (2019). A operação de subamostragem em dois níveis foi definida a partir de testes preliminares realizados durante a elaboração da arquitetura. A inserção de camadas de subamostragem adicionais exigiria que as imagens de entrada tivessem dimensões muito maiores para que não fossem reduzidas a mapas de atributos tão pequenos a ponto de serem incapazes de conter informação suficiente no gargalo.

A Tabela 3.2 lista todos os diferentes valores testados para os parâmetros selecio-

Tabela 3.1 – Parâmetros fixos do modelo

Hiperparâmetros fixos	Configurações
Taxa de aprendizado	0,001
Momento	0,9
Dimensões de amostragem	2×2
Número de filtros convolucionais	32 - 64 - 32 - 1
Épocas para pré-treinamento	60
Épocas para refinamento	20
Tamanho do lote	32
Paciência de parada antecipada	2

náveis a fim de se determinar as melhores opções baseada na acurácia obtida.

Tabela 3.2 – Parâmetros selecionáveis do modelo

Hiperparâmetros selecionáveis	Configurações testadas
Tamanhos dos filtros convolucionais	4 - 2 / 6 - 3 / 8 - 4 / 10 - 5 / 12 - 6
Desvio padrão do ruído	0,1 / 0,2 / 0,3 / 0,4
Taxa de *dropout*	10% / 20% / 30% / 40% / 50%

A exemplo das convenções adotadas em projetos de redes neurais convolucionais, é estabelecido que todos os filtros testados sejam quadrados. De maneira a respeitar as proporções entre as dimensões dos mapas de atributos das camadas extremas e das camadas centrais do *autoencoder*, os tamanhos de filtro são analisados conforme a taxa de redução ou aumento aplicada nas operações de subamostragem ou sobreamostragem. Portanto, a opção $4 - 2$, por exemplo, corresponde a filtros de dimensões 4×4 nas camadas extremas (1ª e 4ª camadas convolucionais) e 2×2 nas camadas centrais (2ª e 3ª camadas convolucionais).

A seleção dos tamanhos dos filtros, do nível de ruído e da taxa de *dropout* é feita no primeiro estudo de caso. Os hiperparâmetros selecionados são então utilizados no segundo estudo de caso, de forma a se verificar a generalidade do modelo. Testes realizados antes e depois da introdução de ruído e regularização permitem avaliar seus respectivos efeitos sobre o desempenho da rede.

3.4 Treinamento

O processo de treinamento do DCAE é dividido em duas fases: pré-treinamento não supervisionado do *autoencoder assimétrico* e refinamento da rede de classificação definitiva. Ambas as fases são descritas a seguir.

3.4.1 Fase de pré-treinamento

O pré-treinamento da rede neural consiste basicamente em prepará-la em termos de seu conjunto de parâmetros para a fase principal de treinamento a ser realizada posteriormente. Durante o pré-treinamento, o *autoencoder* não faz uso dos rótulos associados às amostras do subconjunto de treinamento.

O diagrama da Figura 3.5 apresenta a dinâmica do pré-treinamento. A rede é condicionada a aprender a reconstruir os dados de entrada com o maior grau de eficácia possível. Ao final da propagação, calcula-se o erro baseado na função MSE, descrita pela Equação 2.11, para uma dada entrada $\mathbf{X}$ e sua respectiva versão reconstruída $\hat{\mathbf{X}}$. O ajuste da rede é dado pela atualização dos parâmetros durante a retropropagação do erro, através do algoritmo de otimização SGD, visando diminuir o erro de reconstrução a cada época.

Como resultado do pré-treinamento, é esperado que a rede aprenda a produzir representações robustas e extrair atributos relevantes das entradas processadas. Tal aprendizado é definido pelos valores dos pesos e vieses resultantes da última época dessa fase de treinamento, que são usados na inicialização da rede classificadora.

3.4.2 Fase de refinamento

Após o pré-treinamento, o *decoder*, parte do *autoencoder* responsável pela descompressão ou reconstrução dos dados, é descartado e a informação contida no gargalo é achatada e conectada a uma camada inteiramente conectada com função de ativação *softmax*. Portanto, o refinamento consiste em aprimorar a rede partir de parâmetros pré-treinados, de forma que ela seja capaz de gerar saídas representando probabilidades que correspondam precisamente ao conjunto de saídas esperadas referentes a um dado conjunto de entradas.

A Figura 3.6 ilustra a sistemática de refinamento. Nesse caso, durante o treinamento, a rede tem acesso aos valores de saída esperados e a sua capacidade de associar corretamente um rótulo a uma dada amostra é medido ao final da propagação pela função de erro

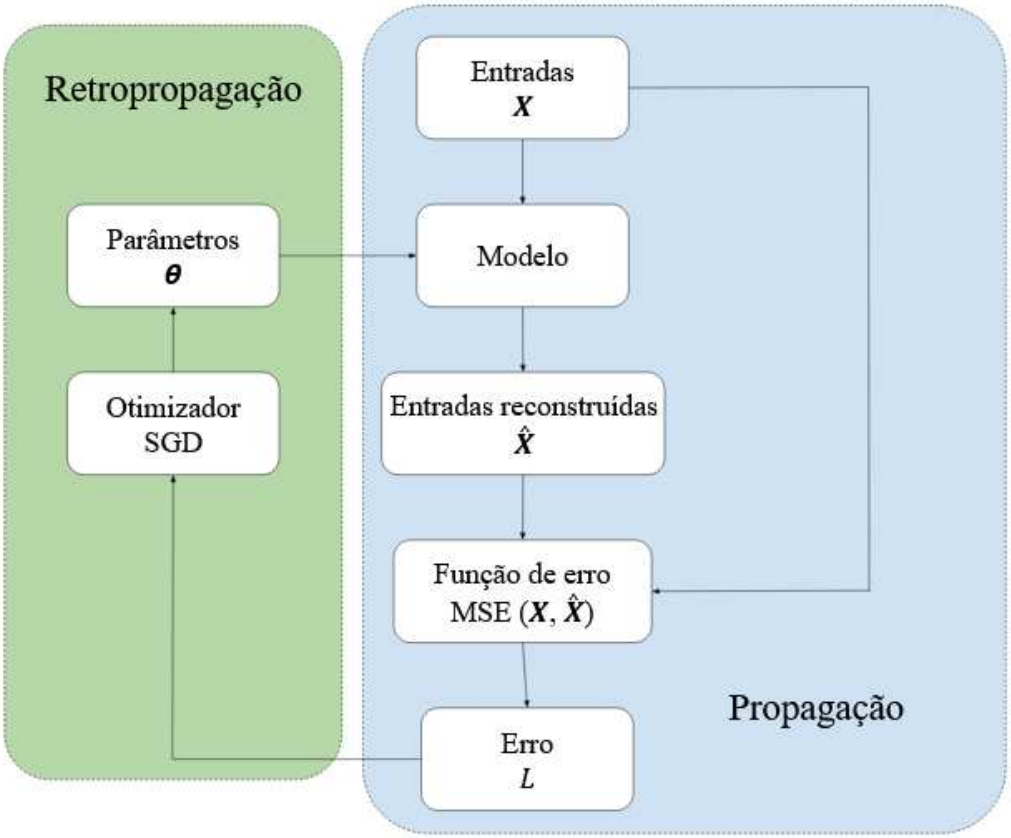

Figura 3.5 – Processo de pré-treinamento.

CCE, definida pela Equação 2.13. De maneira similar à retropropagação conduzida no pré-treinamento, a atualização dos parâmetros é executada por meio do otimizador SGD.

É necessário que um conjunto de dados reservado para diagnóstico ou outros tipos de tarefas baseadas em classificação contenha informações pertinentes aos rótulos ou valores esperados associados a cada entrada. Geralmente, tais rótulos são representados por números inteiros dispostos em vetores colunas, nos quais cada linha indica a classe correspondente a uma amostra. Nesse caso, como um rótulo de classe não possui significado matemático relevante para a rede neural, é preciso que cada saída esperada, originalmente em formato de rótulo, seja submetida ao processo de codificação *one-hot*, por meio do qual é transformada em um vetor binário com número de posições igual ao número de classes possíveis.

Um exemplo de aplicação da codificação *one-hot* para um modelo considerando seis classes de diagnóstico ($k = 6$) é mostrado na Figura 3.7. A posição correspondente à classe esperada recebe o valor 1 e as demais posições são preenchidas com 0. Com o vetor de saídas esperadas codificadas $\mathbf{y}^{(i)} = \{y_0^{(i)}, y_1^{(i)}, ..., y_{k-1}^{(i)}\}$ tendo formato compatível com o vetor de saídas produzidas pela rede $\hat{\mathbf{y}}^{(i)} = \{\hat{y}_0^{(i)}, \hat{y}_1^{(i)}, ..., \hat{y}_{k-1}^{(i)}\}$, ambos referentes à mesma imagem

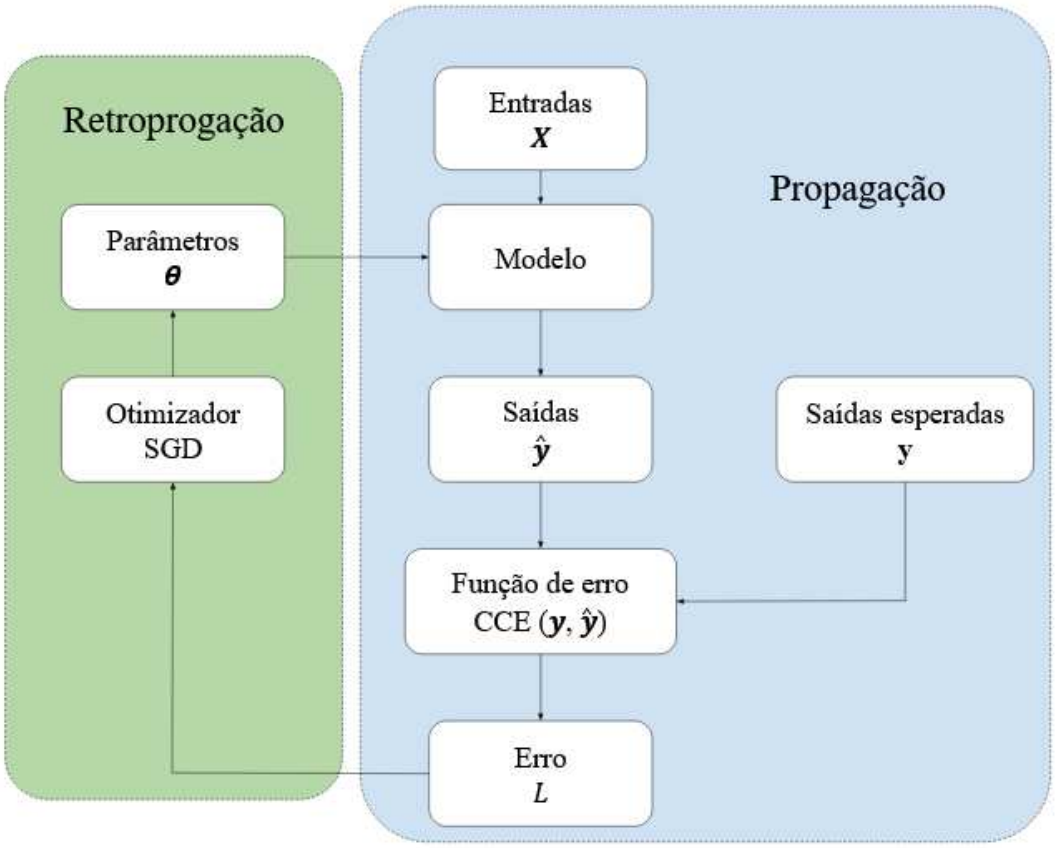

Figura 3.6 – Processo de refinamento.

de entrada $\mathbf{X}^{(i)}$, a entropia cruzada categórica pode então ser calculada pelo erro acumulado calculado em função de cada par de elementos em ambos os vetores.

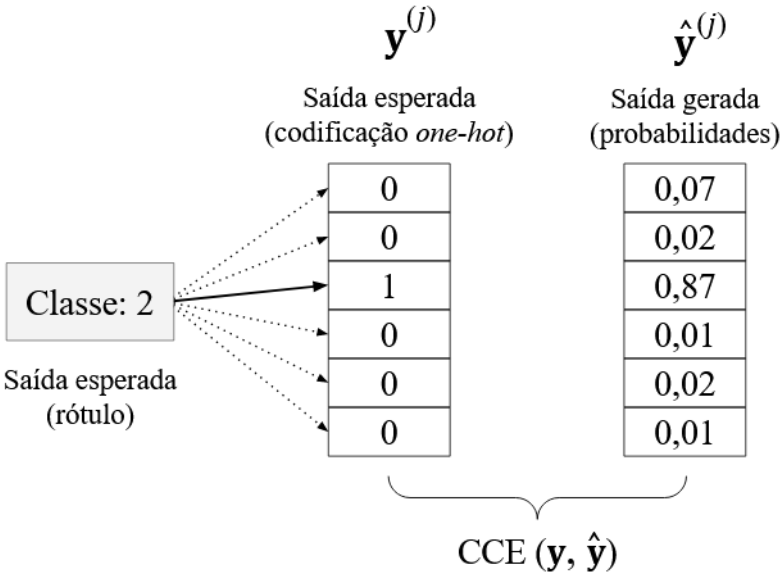

Figura 3.7 – Exemplo de codificação one-hot com seis classes.

No refinamento, considerando a classe com maior probabilidade como rótulo de uma dada amostra, a acurácia parcial da rede é dada pela média da acurácia dos minilotes ao longo de uma época. Portanto, tamanhos de lote muito pequenos podem resultar em valores de acurácia inconsistentes entre épocas sucessivas e, consequentemente, dificultar o monitoramento do desempenho da rede durante o treinamento. Além disso, o efeito do tamanho de lote sobre o cálculo do erro afeta a estabilidade do aprendizado em ambas as fases de treinamento.

3.5 Implementação computacional

O sistema foi inteiramente dessenvolvido em *Python*, uma linguagem de programação de alto nível popularmente utilizada para a criação de aplicações envolvendo aprendizado de máquina e técnicas de ciência de dados em geral.

A linguagem Python oferece uma variedade de pacotes que possibilitam a implementação de todas as funcionalidades necessárias com alto nível de abstração e flexibilidade. Portanto, é uma linguagem apropriada para prototipagem rápida e eficiente de modelos de aprendizado de máquina. Abaixo, estão listados os pacotes que auxiliam nas diferentes etapas de desenvolvimento do sistema.

- ***NumPy*:** útil para a execução de operações algébricas envolvendo vetores e matrizes de muitas dimensões com alto grau de eficiência computacional.

- ***Scikit-learn*:** apresenta diversos recursos para aprendizado de máquina. As classes e funções contidas em seu módulo de pré-processamento são utilizadas para a preparação dos conjuntos de dados. Além disso, o pacote provê funcionalidades que facilitam a análise das métricas de desempenho.

- ***Matplotlib*:** permite a criação de gráficos e visualização de dados. Os resultados obtidos ao longo dos experimentos são analisados com o uso desse pacote.

- ***TensorFlow*:** assim como o *Scikit-learn*, oferece recursos para aprendizado de máquina, porém com foco em redes neurais profundas e na intensa utilização de processamento paralelo em unidades de processamento gráfico (GPU's, do inglês *Graphics Processing Units*).

A API do *TensorFlow*, sobre a qual se baseia a construção dos modelos de rede neurais aqui desenvolvidos, é descrita com mais detalhes a seguir.

3.5.1 API do *TensorFlow* para *Python*

O *TensorFlow* é uma API presente em *Python* e em outras linguagens de programação que fornece um extenso conjunto de recursos para manipulação e operações envolvendo tensores, estruturas multidimensionais contendo um tipo uniforme de dados, que generalizam a noção de escalares, vetores e matrizes.

Um dos principais benefícios de se utilizar o *TensorFlow* para a modelagem de redes neurais é que, além de prover funções para execução de operações de alta complexidade, sua flexibilidade permite que sejam programados elementos tanto de baixo como de alto nível.

A versão atual do pacote, *TensorFlow* 2.0, incorpora como um de seus *frameworks* o *Keras*, uma API de alto nível que encapsula os elementos necessários para o desenvolvimento de redes neurais através de classes padronizadas, além de permitir a criação de classes personalizadas derivadas de classes existentes.

A Figura 3.8 esquematiza de forma hierárquica os recursos disponibilizados no *TensorFlow*. Por meio do módulo *models* do *Keras*, é possível instanciar modelos sequenciais, através da classe *Sequential*, ou flexíveis, através da classe *Model*. A classe de modelagem flexível é indicada quando há a necessidade de se construir redes neurais com arquiteturas ou dinâmicas de treinamento específicas impossíveis de serem representadas pela disposição sequencial de camadas. Ambos os modelos podem ser constituídos pela camadas existentes no módulo *layers*. Todas as camadas que formam o DCAE, apresentadas na Figura 3.4, estão incluídas nesse módulo e podem, portanto, ser aplicadas diretamente na construção do modelo.

Assim como o DCAE, grande parte das arquiteturas de redes neurais podem ser construídas pelo uso dos recursos providos pelo *Keras*. São oferecidas muitas outras camadas além das apresentadas na Figura 3.8, tais como as utilizadas para a modelagem de redes neurais convolucionais de outras dimensões e para a aplicação de outros métodos de regularização ou transformação de tensores.

3.5.2 Processo de modelagem e treinamento

A Figura 3.9 ilustra o processo de modelagem da rede neural com a API do *Keras*. Todas as camadas necessárias para a implementação da rede DCAE, seja em termos do

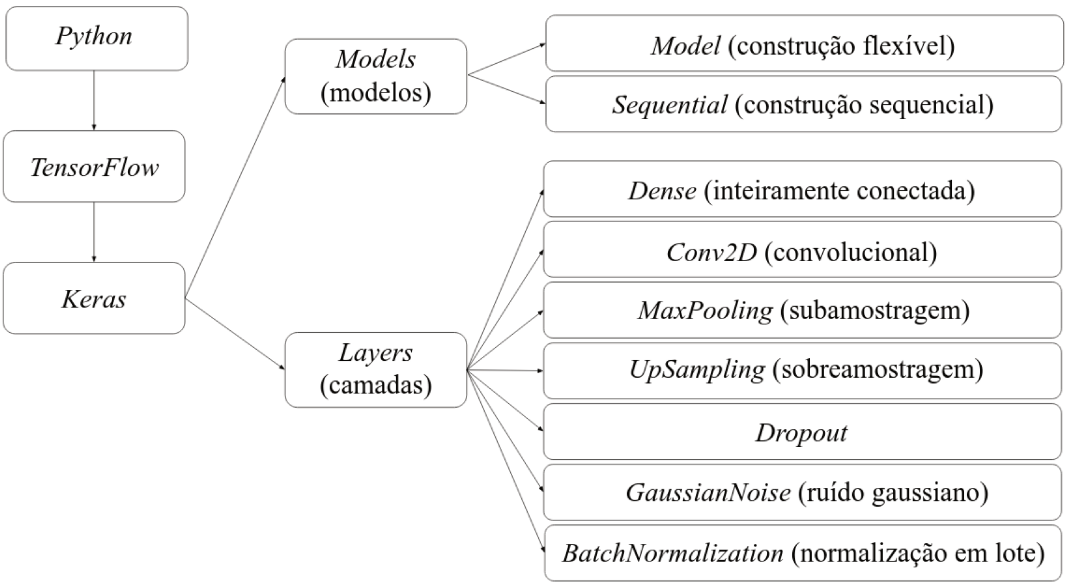

Figura 3.8 – Recursos de modelagem disponibilizados pelo *TensorFlow*.

autoencoder assimétrico ou da rede definitiva, são derivadas do módulo *layers*. O modelo é instanciado da classe *Model*, vista a flexibilidade requerida para garantir a interação entre os diferentes estágios de treinamento.

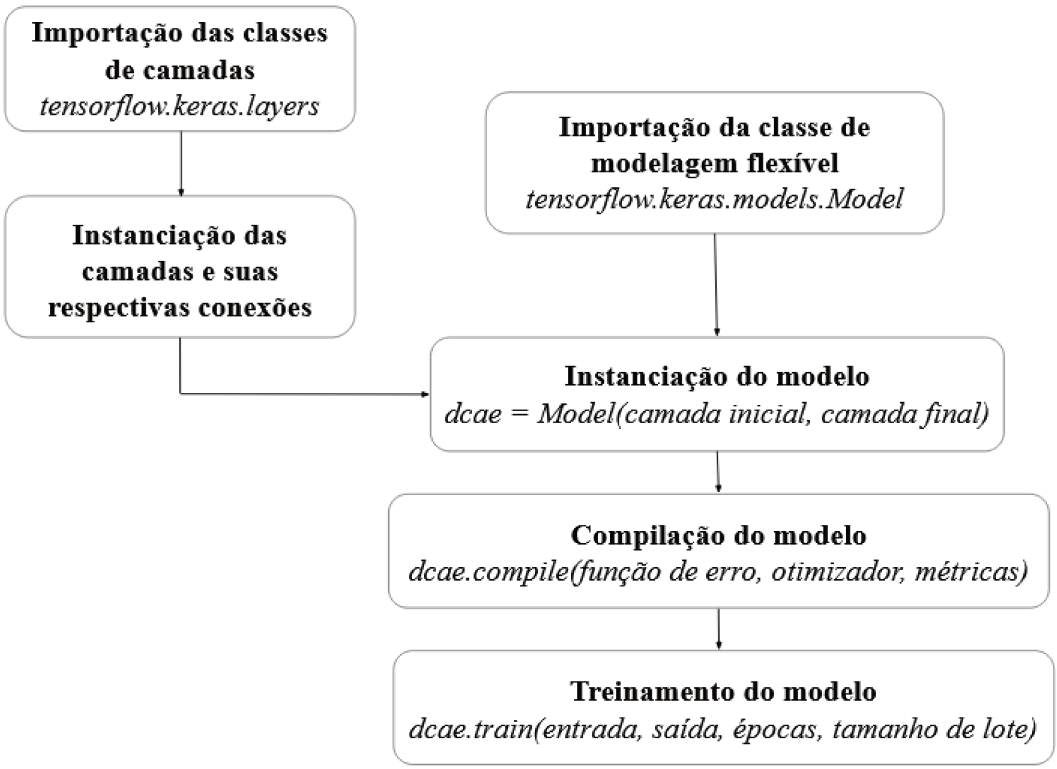

Figura 3.9 – Processo de construção e treinamento do modelo.

Antes do treinamento, o modelo é submetido aos processos de instanciação, no qual define-se a sua estrutura, desde a camada inicial até a camada de saída, e compilação, no qual são fornecidas informações pertinentes ao treinamento, tais como o otimizador, a função de erro e as métricas de avaliação a serem consideradas. Os métodos de compilação e treinamento pertencem à classe de modelagem da qual se instancia o DCAE (*Model.compile* e *Model.train*) e, portanto, podem ser acessadas através do objeto *dcae*.

Como resultado do treinamento do DCAE, tem-se um modelo descrito em função dos hiperparâmetros selecionados e dos parâmetros otimizados, acompanhado de um histórico contendo a evolução do erro e da acurácia de treinamento e validação ao longo das diferentes fases de treinamento.

Um modelo do *TensorFlow* treinado pode ser salvo no formato HDF5 [1], permitindo que seja importado e executado em outras aplicações posteriormente. Ao longo dos experimentos conduzidos, todas as diferentes configurações treinadas são salvas para que possam ser analisadas e comparadas em relação aos dados de teste.

Os códigos implementados para modelagem e treinamento da rede DCAE são disponibilizados nos Apêndices A e B, respectivamente.

3.6 Métricas de avaliação

A seleção dos melhores hiperparâmetros e a análise do desempenho do modelo são realizadas com a ajuda de métricas ou indicadores específicos que visam fornecer informações pertinentes ao comportamento do DCAE em diferentes aspectos.

As métricas de avaliação utilizadas ao longo deste trabalho, baseadas nas definições de Kubat (2017), são descritas a seguir.

- **Matriz de confusão:** tabela que permite visualizar a distribuição dos rótulos atribuídos na classificação dos elementos do subconjunto de teste. A Figura 3.10 apresenta o formato geral de uma matriz de confusão para classificação multiclasse. Cada célula é preenchida com o valor $Q_{(i,j)} \in \mathbb{N}$, tal que Q representa a quantidade de imagens de classe j classificadas como pertencentes à classe i. Portanto, os valores dispostos na diagonal principal, em que $i = j$, indicam o número de acertos referentes a cada classe.

[1] Formato de arquivo binário baseado em estruturação hierárquica, que otimiza o armazenamento de grandes volumes de dados numéricos.

		Classes preditas			
		0	1	...	k-1
Classes esperadas	0	$Q_{(0, 0)}$	$Q_{(0, 1)}$	...	$Q_{(0, k-1)}$
	1	$Q_{(1, 0)}$	$Q_{(1, 1)}$	...	$Q_{(1, k-1)}$
	...	...	...	...	...
	k-1	$Q_{(k-1, 0)}$	$Q_{(k-1, 1)}$	...	$Q_{(k-1, k-1)}$

Figura 3.10 – Estrutura genérica de uma matriz de confusão para para problemas multiclasses.

Assumindo-se, inicialmente, um problema de classificação binária, em que *P* é a classe positiva (de interesse) e *N* a classe negativa, as seguintes propriedades básicas podem ser extraídas:

(a) *True positive (TP):* número total de imagens corretamente classificadas como sendo de rótulo P.

(b) *True negative (TN):* número total de imagens corretamente classificadas como sendo de rótulo N.

(c) *False positive (FP):* número total de imagens incorretamente classificadas como sendo de rótulo P.

(d) *False negative (FN):* número total de imagens incorretamente classificadas como sendo de rótulo N.

Essas propriedades podem ser generalizadas para problemas de classificação multiclasse. Nesse contexto, cada rótulo é analisado individualmente, de forma que o mesmo seja tratado como sendo de classe positiva, enquanto todos os outros são tratados como sendo de classe negativa. Um conjunto de métricas relevantes para a avaliação do desempenho do classificador é definido a partir das propriedades listadas acima:

- **Precisão:** representa a porção de imagens que receberam corretamente um determinado rótulo dentre todas as que o receberam, definida por:

$$\textit{precisão} = \frac{TP}{TP + FP} \tag{3.3}$$

- **Revocação ou *recall*:** indica a porção de imagens que recebem corretamente um dado rótulo dentre o total de imagens que realmente possuem tal rótulo. A revocação é definida por:

$$\textit{revocação} = \frac{TP}{TP + FN} \tag{3.4}$$

- **Pontuação F1:** é a média harmônica entre precisão e revocação. A pontuação F1 surge da necessidade frequente de se combinar as duas métricas anteriores na avaliação do desempenho de um algoritmo. A pontuação F1 deriva de uma métrica de pontuação mais genérica que pode considerar contribuições ponderadas da precisão e da revocação. No entanto, na pontuação aqui utilizada, o mesmo peso é atribuído a cada métrica, resultando na seguinte equação:

$$F1 = 2 \cdot \frac{\textit{precisão} \cdot \textit{revocação}}{\textit{precisão} + \textit{revocação}} \tag{3.5}$$

- **Acurácia:** corresponde à razão entre o número de acertos e o número total de imagens classificadas, definida por:

$$\textit{acurácia} = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.6}$$

- **Erro:** indica o desempenho do processo de otimização. Durante a reconstrução, o erro é calculado pela função MSE e, durante a classificação, pela função CCE. O aprendizado ocorre enquanto o valor de erro diminui durante o treinamento de uma rede neural.

Os códigos apresentados nos Apêndices C e D referem-se à avaliação do desempenho do DCAE através da matriz de confusão e do teste de robustez a ruído.

4 RESULTADOS EXPERIMENTAIS

Diversos conjuntos de dados relativos a componentes de máquinas rotativas estão disponíveis publicamente para fins de estudo e pesquisa. A seguir, são apresentados os dois estudos de caso conduzidos para a validação do método proposto.

4.1 Estudo de caso 1: Conjunto de dados do Instituto Politécnico de Turim

Este estudo de caso se baseou em dados referentes a falhas em rolamentos disponibilizados pelo Instituto Politécnico de Turim (Daga *et al.*, 2019).

As Figuras 4.1 e 4.2 mostram em detalhes os equipamentos utilizados para a coleta dos sinais de vibração. A bancada de teste consiste de um eixo-árvore de alta velocidade para acionamento da rotação de um eixo. O eixo-árvore é fixado a um suporte rígido que se apoia em uma placa de base de aço maciça. A velocidade do eixo-árvore é definida através de um painel de controle de um inversor, embora não possa ser ativamente controlada, uma vez que não há instrumentos para medir sua velocidade ou retorno para o controlador do inversor. Consequentemente, a velocidade real do eixo é sempre menor que a ideal e a diferença aumenta conforme se aumenta a carga aplicada. Uma única placa contém um par de suportes para dois rolamentos idênticos (B1 e B3). Os anéis internos desses rolamentos são conectados a um eixo vazado curto, projetado especificamente para velocidades de até 35000 rpm. Originalmente, esse eixo era parte de uma caixa de engrenagens completa e carregava uma engrenagem de dentes retos responsável pela rotação do eixo. Devido ao torque aplicado, a engrenagem gerava uma força de contato nas direções radial e tangencial sobre o par de rolamentos. No ensaio realizado, a mesma força radial exercida pela engrenagem é substituída pela carga aplicada por um rolamento maior (B2), cujo anel externo é conectado a uma corrediça de precisão com movimento ortogonal ao eixo. Os sinais de vibração, medidos nos pontos mais significativos da estrutura, são coletados a partir de dois acelerômetros triaxiais (A1 e A2), posicionados próximos aos rolamentos B1 e B2, respectivamente.

Informações complementares a respeito dos rolamentos e do acelerômetro utilizados na bancada são apresentadas nas Figuras A.1, A.2 e A.3 do Anexo A.

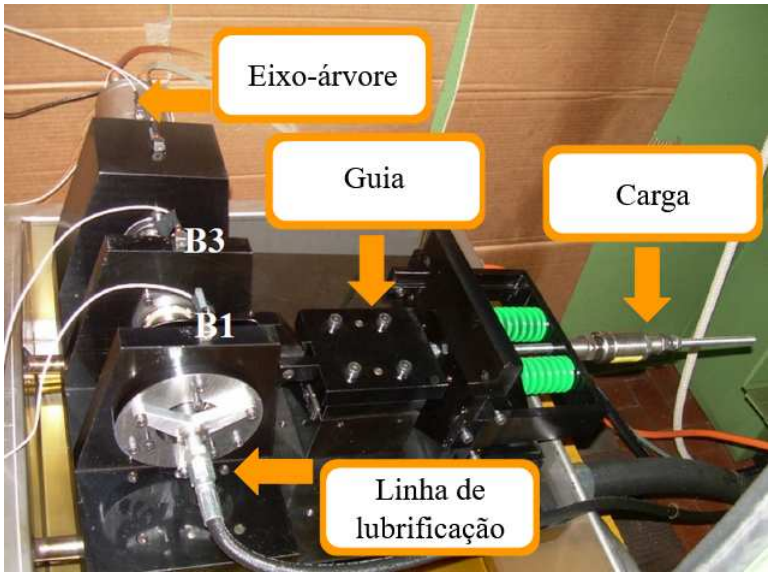

Figura 4.1 – Bancada de teste do Instituto Politécnico de Turim. Adaptado de Daga *et al.* (2019).

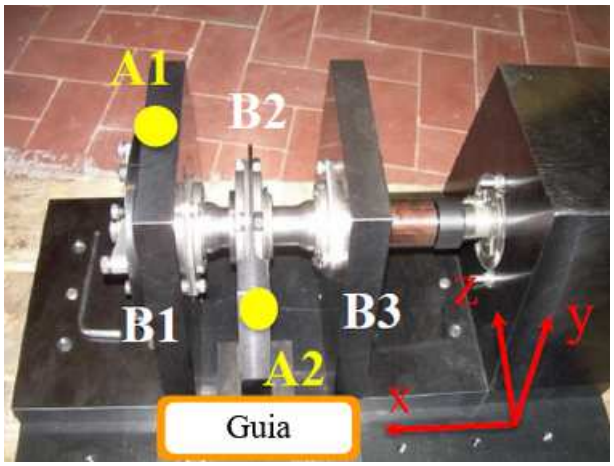

Figura 4.2 – Posição dos acelerômetros e sistema de referência da bancada de teste de rolamentos do Instituto Politécnico de Turim. Adaptado de Daga *et al.* (2019).

4.1.1 Descrição dos dados

A Tabela 4.1 apresenta os canais de medição dos sinais de vibração. Cada acelerômetro realiza a medição de três sinais de aceleração, sendo um axial (x) e dois radiais (y, z). Nesse estudo, foram considerados somente os sinais coletados pelo acelerômetro A1, mais próximo ao rolamento analisado (B1), referentes à aceleração axial na direção x.

A Tabela 4.2 apresenta as condições de operação baseadas na combinação de carga e velocidade nominal de rotação. São disponibilizadas medições para cinco diferentes velocidades de rotação (100, 200, 300, 400 e 500 Hz) sujeitas a cinco diferentes opções de carga (1000, 1400, 1800 N e sem carga). Para os estudos realizados, foram adotadas medições correspon-

Tabela 4.1 – Sinais de aceleração medidos pelos acelerômetros A1 e A2

Sinal	1 (ac. 1)	2 (ac. 1)	3 (ac. 1)	4 (ac. 2)	5 (ac. 2)	6 (ac. 2)
Direção	axial, x	radial, y	radial, z	axial, x	radial, y	radial, z

Fonte: Adaptado de Daga *et al.* (2019).

dentes a todas as opções de carga e velocidade disponíveis.

Tabela 4.2 – Configurações de carga e velocidade

Velocidade nominal (Hz)	Carga nominal (N)			
100	0	1000	1400	1800
200	0	1000	1400	1800
300	0	1000	1400	1800
400	0	1000	1400	-
500	0	1000	-	-

Fonte: Adaptado de Daga *et al.* (2019).

A Tabela 4.3 descreve os sete estados observados no rolamento B1. O sistema de diagnóstico foi implementado levando-se em consideração a existência de um estado saudável e seis estados de falha, correspondentes a três níveis de severidade de indentação no anel interior e em um elemento rolante.

Tabela 4.3 – Condições e rótulos associados ao rolamento B1 no estudo de caso 1

Rótulo	Condições	Diâmetro (μm)
0	saudável	-
1	indentação no anel interno	450
2	indentação no anel interno	250
3	indentação no anel interno	150
4	indentação em um elemento rolante	450
5	indentação em um elemento rolante	250
6	indentação em um elemento rolante	150

Fonte: Adaptado de Daga *et al.* (2019).

Os gráficos da Figura 4.3 permitem visualizar o padrão dos sinais de vibração correspondentes a cada estado de saúde ao longo das primeiras 10 rotações à velocidade de 100 Hz sem carga. Para o período de valores analisado, nota-se que, apesar de se observar numerosos

picos nos sinais de indentação de 450 μm de anel interno e de elemento rolante, é difícil realizar inferências visuais precisas acerca de padrões que descrevem as condições de forma geral.

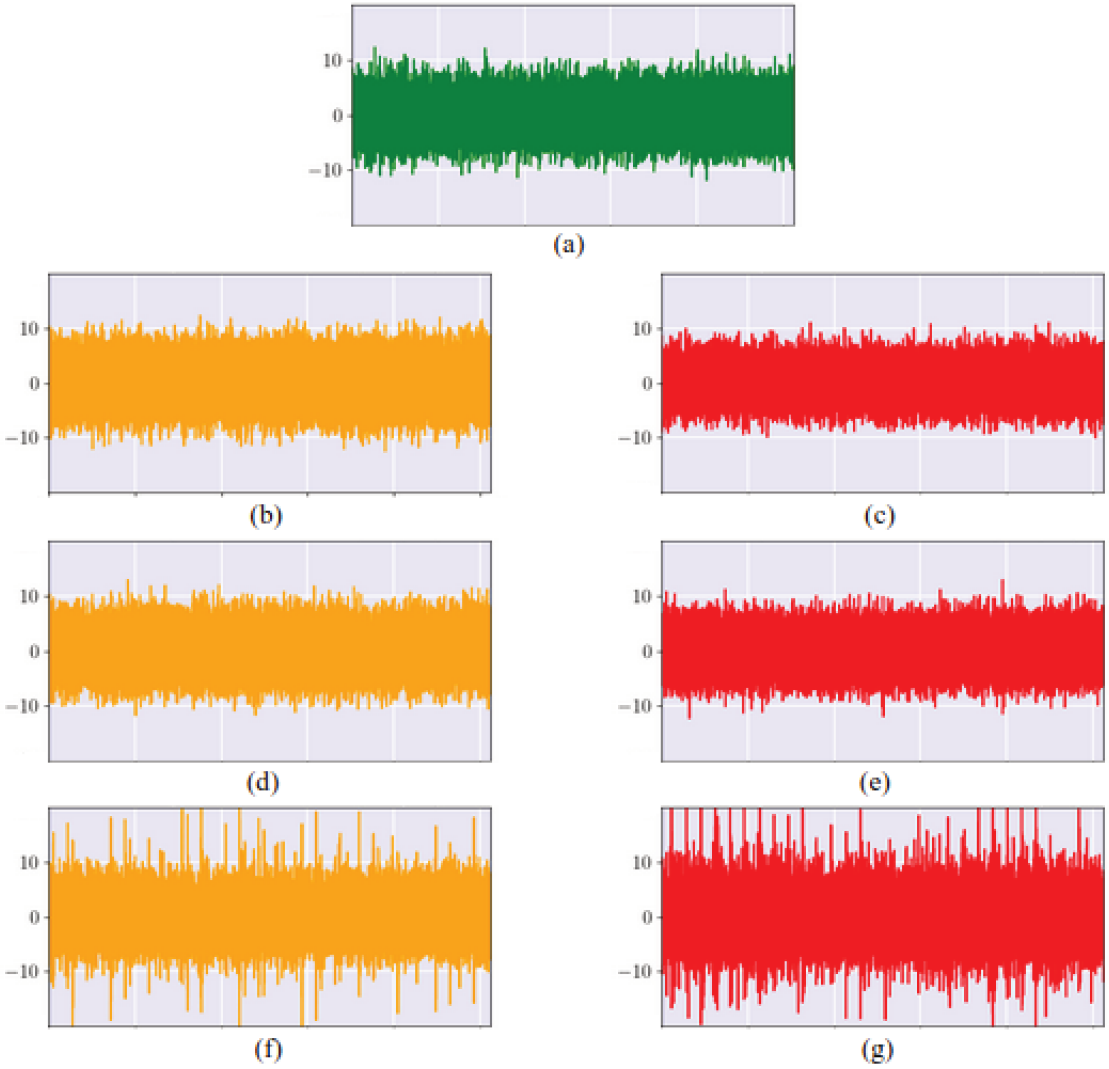

Figura 4.3 – Sinais de vibração de cada estado nas primeiras 10 rotações a 100 Hz sem carga da bancada do Instituto Politécnico de Turim: (a) saudável, indentação no anel interno de (b) 150 μm, (c) 250 μm e (d) 450 μm, e indentação em um elemento rolante de (e) 150 μm, (f) 250 μm e (g) 450 μm.

A Figura 4.4 apresenta imagens de vibrações geradas para cada uma das condições consideradas na Figura 4.3, identificadas pelos respectivos rótulos. Dada a paleta de cores escolhida para a representação das imagens, observa-se que os pontos amarelos correspondem aos valores de maior amplitude nas séries medidas. Assim como nos segmentos unidimensionais, a identificação visual de padrões nas imagens ainda não é trivial, mas a estrutura convolucional do DCAE deve ser capaz de capturar relações espaciais complexas entre os pontos que as compõem.

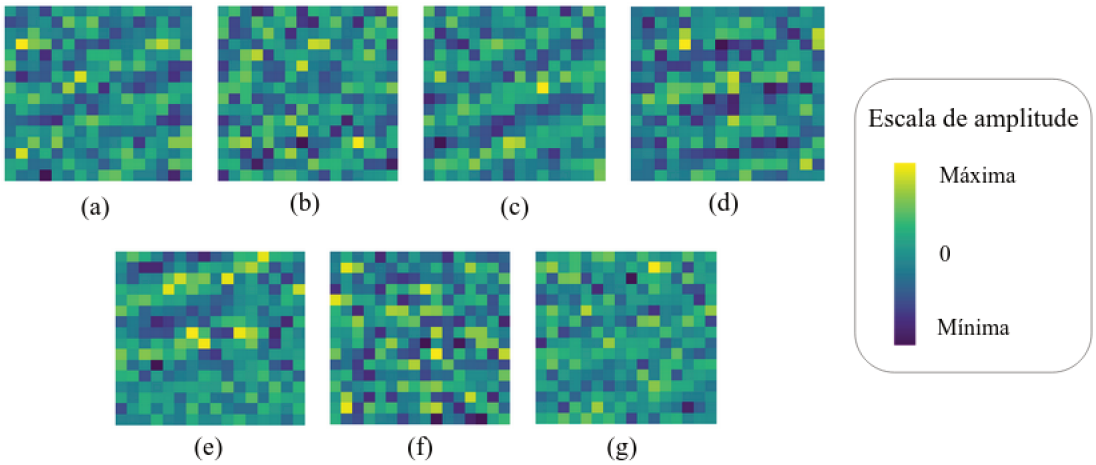

Figura 4.4 – Exemplos de imagens de vibrações geradas para cada uma das sete classes: (a) saudável, indentação no anel interno de (b) 150 μm, (c) 250 μm e (d) 450 μm, e indentação em um elemento rolante de (e) 150 μm, (f) 250 μm e (g) 450 μm.

4.1.2 Preparação dos dados

Cada série temporal foi amostrada com frequência $f_a = 51200$ Hz e duração de 10 segundos, portanto, cada arquivo contém 512000 pontos para cada um dos seis sinais medidos. Dada a frequência de amostragem e a velocidade de rotação selecionada, tem-se 512 pontos amostrados por rotação. Cada sinal referente ao canal escolhido foi então dividido em segmentos com 256 pontos, número que possui significado direto no movimento do eixo (equivalente a $1/2$ rotação) e raiz inteira, o que possibilita a geração de imagens através do redimensionamento. Assim, 2000 segmentos foram obtidos em cada arquivo e o modelo foi treinado com amostras contendo 256 variáveis, redimensionadas de 256×1 para 16×16.

4.1.3 Seleção de hiperparâmetros

Nesse estudo de caso, a definição dos hiperparâmetros selecionáveis foi realizada com base nos efeitos observados sobre o desempenho da rede sobre o subconjunto de validação. O primeiro hiperparâmetro analisado foi o tamanho dos filtros convolucionais. Em seguida, foram também avaliadas as opções de ruído gaussiano e regularização por *dropout*, respectivamente.

- **Tamanhos dos filtros:** considerado um dos fatores mais influentes em uma rede convolucional, o tamanho do filtro define a quantidade de informação espacial a ser extraída a cada passo de convolução. A aplicação de preenchimento nas entradas sobre as quais são

realizadas as convoluções permite que sejam utilizados filtros de grandes dimensões. O gráfico da Figura 4.5 apresenta os resultados da comparação entre diferentes opções de filtros quadrados. Todos os tamanhos testados estão contidos na faixa de 2×2 até 6×6 para as camadas convolucionais centrais, e de 4×4 até 12×12 para as camadas convolucionais das extremidades da rede. Nota-se que o número de parâmetros da rede aumenta exponencialmente conforme se aumenta o tamanho dos filtros. Consequentemente, a utilização de filtros muito grandes resulta em modelos mais complexos e treinamento mais demorado. Do gráfico, pode-se concluir que a melhor acurácia sobre os dados de validação $(93,99\%)$ foi obtida com filtros de tamanhos 12×12 e 6×6 para as camadas extremas e centrais, respectivamente. Usar filtros maiores a partir dessas opções, além de aumentar exponencialmente a complexidade da rede, não garante melhoria significativa de desempenho.

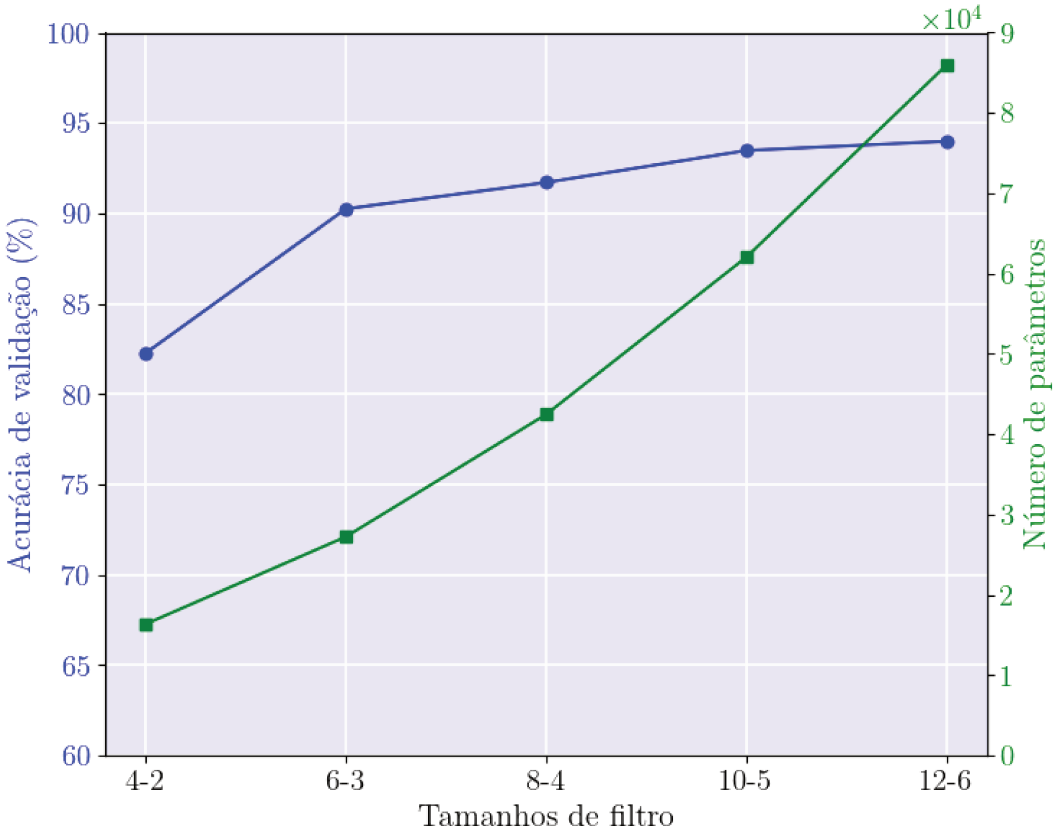

Figura 4.5 – Resultado da comparação de diferentes tamanhos de filtro.

- **Adição de ruído gaussiano:** as imagens de entrada são corrompidas com diferentes níveis de ruído gaussiano. O gráfico da Figura 4.6 apresenta os resultados da comparação entre níveis específicos de ruído com base nos respectivos valores de desvio padrão. A alta

sensibilidade da acurácia a variações do ruído observada no gráfico impõe que um nível adequado de corrupção deve ser introduzido para que se obtenha a melhoria de desempenho e robustez esperada. Considerando a acurácia de validação, o melhor desempenho $(95,57\%)$ foi obtido com desvio padrão $\sigma = 0,2$.

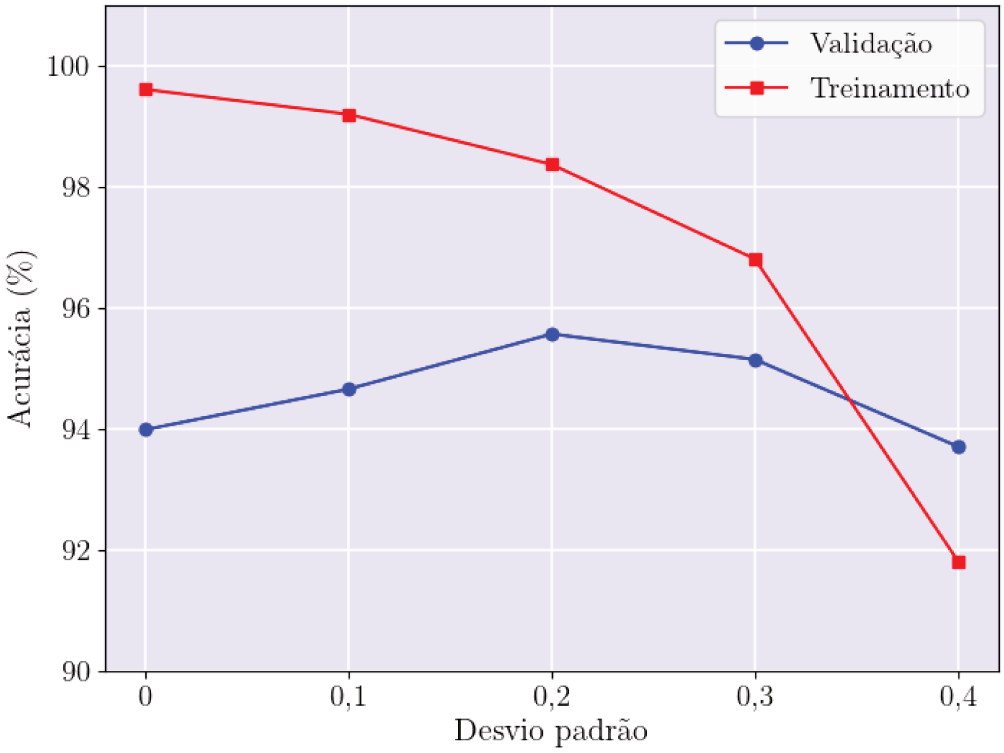

Figura 4.6 – Resultado da comparação de diferentes níveis de ruído na entrada.

- ***Dropout* nas camadas intermediárias:** pela aplicação de *dropout*, foi possível notar um aumento considerável da acurácia de validação na medida em que a acurácia de treinamento diminui. Esse cenário caracteriza um dos principais efeitos da regularização, que se trata do aumento da generalidade do modelo, que passa a apresentar desempenho sobre os dados de validação no mínimo tão satisfatório quanto o obtido sobre os dados de treinamento. A Figura 4.7 mostra os resultados da comparação de diferentes opções de *dropout*. Tomando como referência a melhor acurácia de validação $(97,55\%)$, selecionou-se a taxa $\rho = 20\%$ para todas as camadas de *dropout*.

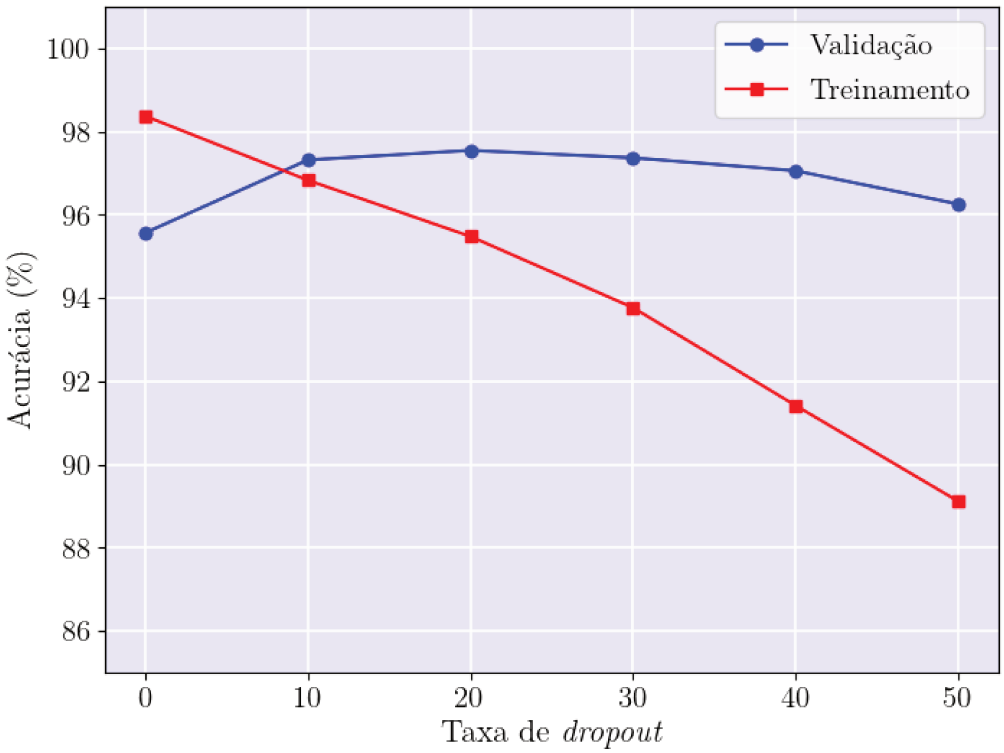

Figura 4.7 – Resultado da comparação de diferentes taxas de *dropout*.

4.1.4 Eficácia do pré-treinamento com *autoencoder* assimétrico

Em adição à seleção dos hiperparâmetros descrita nas subseções anteriores, uma análise foi conduzida para se verificar os efeitos do pré-treinamento sobre o desempenho do treinamento supervisionado. A Figura 4.8 apresenta a minimização do erro de validação nas primeiras 10 épocas do treinamento supervisionado do classificador para dois modelos similares, sendo um inicializado com parâmetros aleatórios, definidos por meio do método de inicialização Xavier, e outro inicializado com parâmetros resultantes do pré-treinamento com o *autoencoder* assimétrico. Os resultados obtidos confirmam a eficácia do *autoencoder* e reforçam as conclusões anunciadas por Masci *et al.* (2011) no que se trata da melhoria oferecida pela inicialização de um modelo convolucional a partir de uma topologia CAE treinada de maneira não supervisionada, uma vez que o pré-treinamento assegurou uma clara vantagem em relação à inicialização aleatória desde a primeira época.

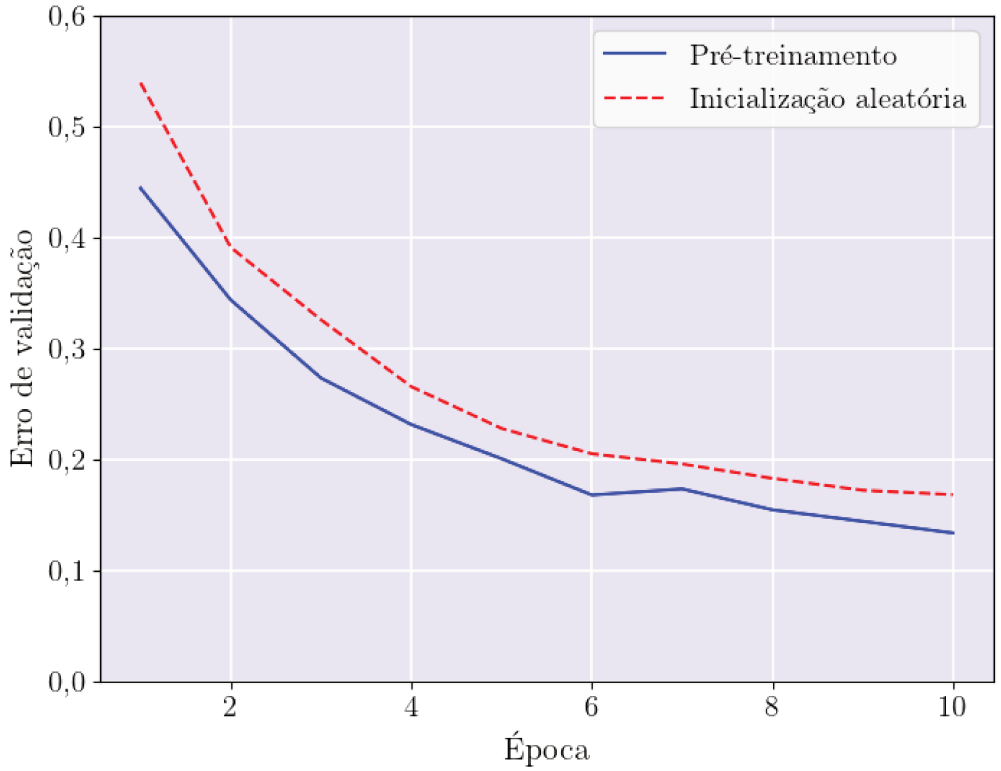

Figura 4.8 – Comparação do erro no treinamento supervisionado a partir de parâmetros pré-treinados e parâmetros inicializados aleatoriamente.

4.1.5 Influência da adição de ruído e regularização

Através dos resultados do treinamento de diferentes configurações do DCAE ao longo da etapa de seleção de hiperparâmetros, foi possível realizar uma análise comparativa entre o desempenho do modelo antes e depois da adição de ruído gaussiano na entrada e *dropout* nas camadas intermediárias.

O gráfico da Figura 4.9 apresenta a evolução da minimização do erro durante a fase de refinamento para ambas as versões do DCAE. O modelo completo, contendo regularização, obteve mais êxito na redução do erro desde as primeiras épocas, além de assegurar um processo de minimização mais suave em comparação ao modelo sem regularização, evitando flutuações e variações bruscas.

4.1.6 Efeitos da normalização

Assim como os elementos pertencentes à arquitetura interna da rede neural, a normalização aplicada na etapa de pré-processamento dos dados também foi avaliada. Embora

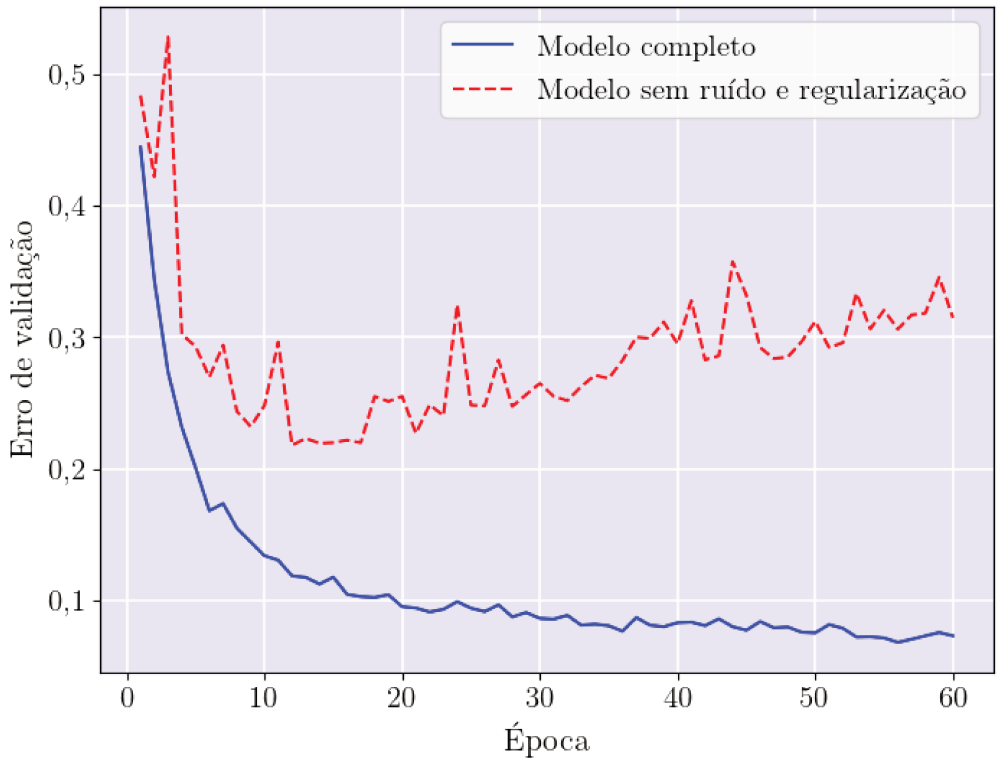

Figura 4.9 – Influência da adição de ruído e regularização no primeiro estudo de caso.

uma das principais contribuições da normalização seja acelerar o treinamento, seu efeito mais relevante para as aplicações aqui abordadas se encontra na melhoria de acurácia esperada com sua aplicação.

O gráfico da Figura 4.10 apresenta a evolução da acurácia de validação do DCAE ao longo da fase de refinamento. Com os dados normalizados, obteve-se uma convergência com acurácia consistentemente superior à obtida com o treinamento usando dados brutos. O impacto da normalização pode ser claramente observado no início do refinamento. Enquanto no treinamento com dados brutos a acurácia obtida na primeira época foi levemente superior a 40%, o treinamento com dados normalizados alcançou uma acurácia superior a 80%. A vantagem do treinamento com dados normalizados, apesar de diminuir ao longo das épocas, manteve-se consistente até o instante de convergência de cada curva de aprendizado.

Outro efeito importante da normalização é observado quanto à estabilidade numérica do processo de minimização do erro. Alguns dos treinamentos realizados sem normalização sofreram com o problema de explosão do gradiente, resultando em valores de erro muito altos que impossibilitaram o aprendizado. Nesses casos, o treinamento precisou ser reiniciado. O

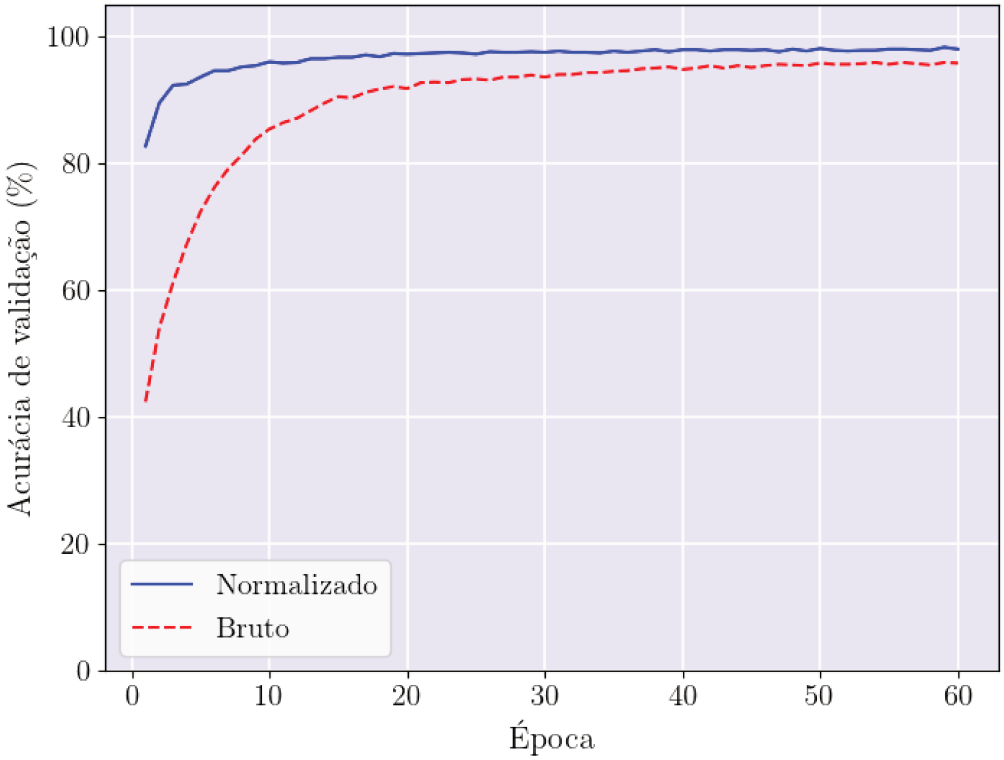

Figura 4.10 – Efeitos da normalização sobre a acurácia de validação no primeiro estudo de caso.

problema de explosão do gradiente não foi constatado durante os treinamentos usando normalização.

4.1.7 Matriz de confusão

O modelo DCAE treinado com os hiperparâmetros otimizados conforme a seleção realizada na Subseção 4.1.3 foi submetido aos subconjuntos de teste para verificação de seu desempenho em relação a dados desconhecidos. Durante o teste, o modelo atribui um rótulo para cada imagem desses subconjuntos. Matrizes de confusão são então usadas para facilitar a visualização da distribuição dos rótulos atribuídos, permitindo entender para quais classes ocorrem as maiores taxas de erro de classificação.

A Figura 4.11 contém a matriz de confusão resultante para este estudo de caso. A acurácia total de $97,62\%$ indica a porção de valores contidos na diagonal principal, correspondentes aos acertos. Embora os demais valores, associados aos erros de classificação, estejam bem distribuídos ao longo da matriz, pode-se notar que não ocorreram confusões entre a classe predita 1 e a classe real 6, referentes a indentações de 450μm de anel interno e 150μm de ele-

mento rolante, respectivamente. É importante ressaltar que confusões observadas nos subconjuntos $\{1, 2, 3\}$ e $\{4, 5, 6\}$ indicam casos em que o DCAE foi capaz de identificar corretamente o tipo de falha, mas não a sua severidade.

		Predito						
		0	1	2	3	4	5	6
Real	0	**6464**	4	23	127	2	26	40
	1	2	**6640**	5	55	35	4	1
	2	22	4	**6570**	66	5	43	58
	3	75	23	76	**6599**	3	24	62
	4	3	23	13	5	**6688**	7	11
	5	24	5	74	53	13	**6616**	35
	6	14	0	40	23	1	4	**6890**
Acurácia total: 97.62%								

Figura 4.11 – Matriz de confusão do primeiro estudo de caso.

A partir dos valores obtidos da matriz de confusão, foi possível realizar a inferência das métricas de precisão, revocação e pontuação F1. A Tabela 4.4 apresenta os resultados obtidos para cada classe. Considerando a pontuação F1, que visa balancear a precisão e a revocação, observa-se que o melhor desempenho de classificação foi obtido para o rótulo 4, correspondente à indentação de 450 μm de elemento rolante. O pior desempenho foi obtido para o rótulo 3, correspondente à indentação de 150 μm de anel interno. Para todas as classes, foram alcançadas pontuação, precisão e revocação acima de 95%.

4.1.8 Comparação dos métodos quanto à robustez a ruído

Para se verificar a robustez da arquitetura proposta, ela foi comparada a outras quatro arquiteturas, todas sujeitas a proporções equivalentes de ruído injetado no subconjunto de dados de teste. Dentre essas arquiteturas, duas se baseiam em outras publicações de aplicações similares e duas resultam de desenvolvimento próprio. Os modelos publicados consistem de uma rede neural CNN 1D, proposta por Ince *et al.* (2016), e de uma rede neural CNN 2D, pro-

Tabela 4.4 – Valores de precisão, revocação e pontuação F1 de cada classe no primeiro estudo de caso

Classe	Precisão (%)	Revocação (%)	Pontuação F1 (%)
0	97,8801	96,6796	97,2761
1	99,1193	98,4871	98,8022
2	96,6034	97,0745	96,8384
3	95,2512	96,1673	95,7070
4	99,1255	99,0815	99,1035
5	98,3938	97,0088	97,6964
6	97,0833	98,8239	97,9458

posta por Hoang e Kang (2019), implementadas conforme suas respectivas especificações. Os modelos MLP e DAE próprios desenvolvidos tiveram seus parâmetros selecionados através de uma sequência de procedimentos bem definidos, similares aos adotados para o DCAE.

Dadas as potências do sinal P_s e do ruído P_r, o nível de ruído introduzido no conjunto de teste é medido através da razão sinal-ruído em escala logarítmica de decibel, definida por:

$$SNR_{dB} = 10\, log_{10}\left(\frac{P_s}{P_r}\right) \tag{4.1}$$

A avaliação de robustez consiste em verificar o desempenho de cada um dos modelos quando submetidos a subconjuntos de teste com diferentes níveis de ruído. Para isso, foram aplicados ruídos com SNR na faixa de 4 até 20 dB. Quanto menor o SNR, maior a proporção de ruído no sinal e, consequentemente, menor é a acurácia esperada.

A Figura 4.12 apresenta os resultados do teste para o primeiro estudo de caso. O DCAE, além manter acurácia superior aos demais modelos ao longo de toda a faixa de valores analisada, foi menos afetado pelo aumento do nível de ruído introduzido. No intervalo de SNR entre 20 e 4 dB, o DCAE perdeu cerca de 9% da acurácia absoluta, enquanto as outras redes convolucionais, CNN 1D e CNN 2D, perderam cerca de 35% e 30%, respectivamente.

4.2 Estudo de caso 2: Conjunto de dados da *Case Western Reserve University*

Os conjuntos de dados utilizados neste estudo de caso, disponibilizados pela Case Western Reserve University (CWRU) com sua respectiva descrição (Loparo, 2012), são com-

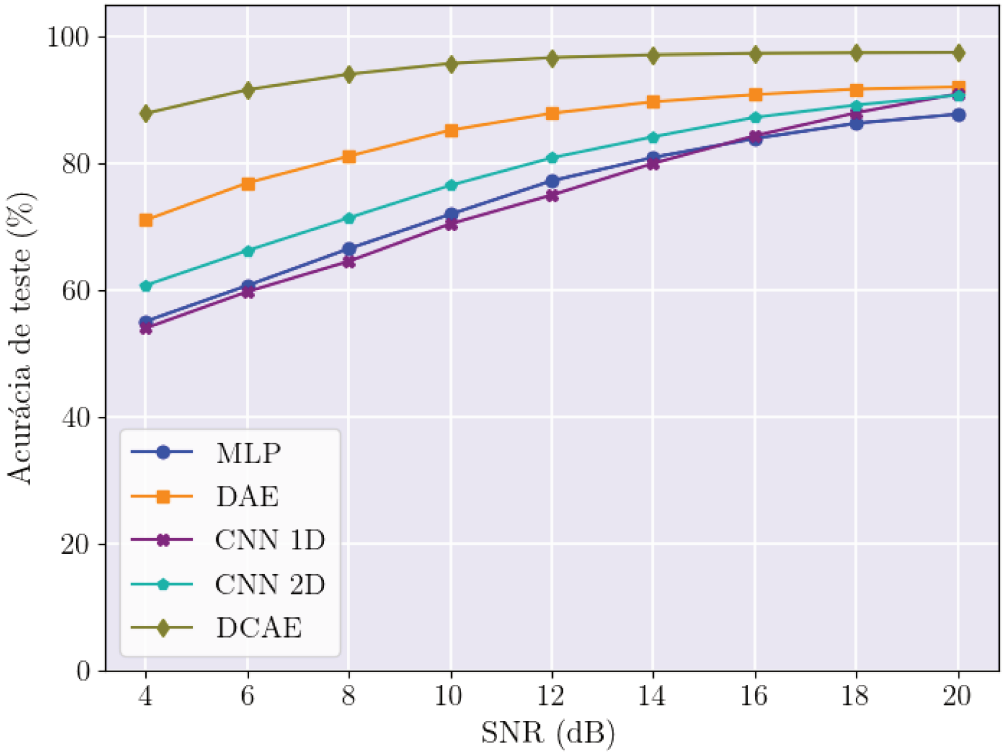

Figura 4.12 – Comparação de robustez a ruído entre o DCAE e outros modelos de redes neurais no primeiro estudo de caso.

postos de séries de vibração geradas por uma bancada de teste de rolamentos.

A Figura 4.13 mostra os equipamentos utilizados para a coleta dos sinais. A bancada consiste de um motor de 2 hp usado para atuação de um eixo acoplado a um rolamento, um dinamômetro e um transdutor de torque para medição e transformação do sinal, amostrado às taxas de 12 e 48 kHz para vibrações normais e com falha.

4.2.1 Descrição dos dados

Para amostragem com $fa = 12$ kHz, adotada neste estudo por conter uma quantidade maior de dados coletados, foram feitas medições para quatro velocidades de rotação, condicionadas pela carga aplicada, e quatro diâmetros de falha. A Tabela 4.5 descreve as diferentes configurações de teste aplicadas à bancada. Falhas específicas são observadas com cada diâmetro de severidade. Visando a capacidade de se diagnosticar a maior variedade possível de estados, além de sinais provenientes da bancada operando sem falha (saudável), foram consideradas medições para todos os diâmetros de falha, exceto para as de $0,028$", uma vez que

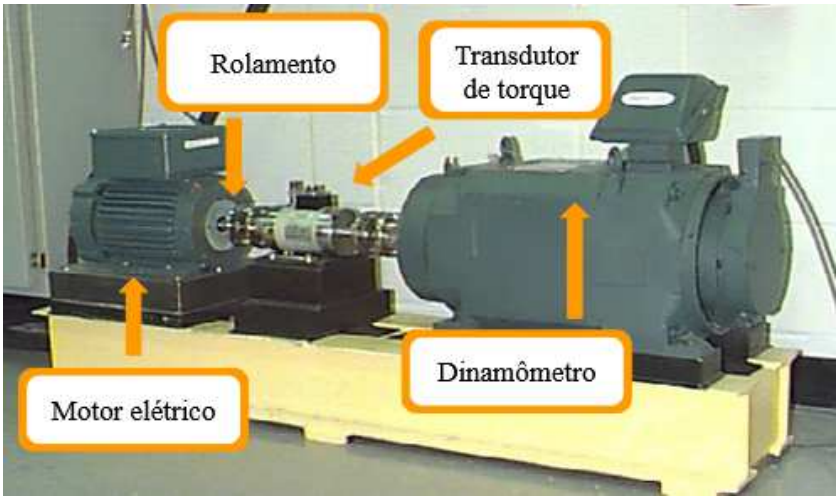

Figura 4.13 – Bancada de teste da *Case Western Reserve University*. Adaptado de Loparo (2012).

não incluem o caso de falha no anel externo. Assim, o diagnóstico levou em conta dez estados possíveis do componente, incluindo um estado saudável e três estados de falha para cada um dos demais diâmetros $0,007$", $0,014$" e $0,021$".

Tabela 4.5 – Configurações da bancada de teste de rolamentos do estudo de caso 2

Diâmetro (in)	Velocidades (rpm)	Falhas
0.007	1797, 1772, 1750, 1730	anel interno, anel externo, elemento rolante
0.014	1797, 1772, 1750, 1730	anel interno, anel externo, elemento rolante
0.021	1797, 1772, 1750, 1730	anel interno, anel externo, elemento rolante
0.028	1797, 1772, 1750, 1730	anel interno, elemento rolante

A Tabela 4.6 lista todos os estados considerados neste experimento. O rótulo 0 corresponde ao estado saudável e o conjunto de rótulos restantes $\{1, 2, ..., 9\}$ representa estados de falha.

Os gráficos da Figura 4.14 apresentam os sinais de vibração coletados nas primeiras 10 rotações a 1797 rpm para cada estado de saúde do rolamento. Em contraste ao estudo de caso anterior, é mais fácil inferir visualmente padrões que definem cada um dos sinais desse conjunto de dados.

A Figura 4.15 apresenta exemplos de imagens geradas correspondentes a cada estado do rolamento. Comparado ao conjunto de dados anterior, é aparentemente mais fácil distinguir algumas classes neste caso. Porém, pode-se afirmar que a simples análise visual ainda é

Tabela 4.6 – Condições e rótulos considerados no estudo de caso 2

Rótulo	Condição	Diâmetro (in)
0	saudável	-
1	falha no anel interno	0,007
2	falha no elemento rolante	0,007
3	falha no anel externo	0,007
4	falha no anel interno	0,014
5	falha no elemento rolante	0,014
6	falha no anel externo	0,014
7	falha no anel interno	0,021
8	falha no elemento rolante	0,021
9	falha no anel externo	0,021

limitada. Assim sendo, os filtros convolucionais do DCAE devem ser capazes de extrair atributos e capturar padrões significativos com maior eficácia.

4.2.2 Preparação dos dados

Os sinais coletados do rolamento com falha no anel externo são originalmente divididos em três categorias de acordo com a posição em que ocorrem (45, 90 ou $180°$). Neste trabalho, somente falhas referentes à posição de $45°$ foram consideradas. Para o caso em que não há carga sobre o motor, dada a frequência de amostragem e a velocidade máxima de 1797 rpm ($29,95$ Hz), tem-se 400.67 pontos por rotação. Cada medição tem pelo menos 121000 pontos e foi dividida em segmentos contendo 400 pontos, que correspondem a aproximadamente uma revolução completa. Assim, a versão redimensionada de cada segmento tem dimensões 20×20.

4.2.3 Influência da adição de ruído e regularização

O gráfico da Figura 4.16 mostra o histórico de minimização do erro durante o refinamento. Nessa aplicação, também é possível notar as vantagens decorrentes da regularização. Apesar de nesse caso o DCAE sem regularização apresentar desempenho muito mais próximo ao DCAE completo do que foi observado no estudo anterior, pode-se notar que a regularização impediu ou amenizou a ocorrência de picos no valor do erro, principalmente nas primeiras

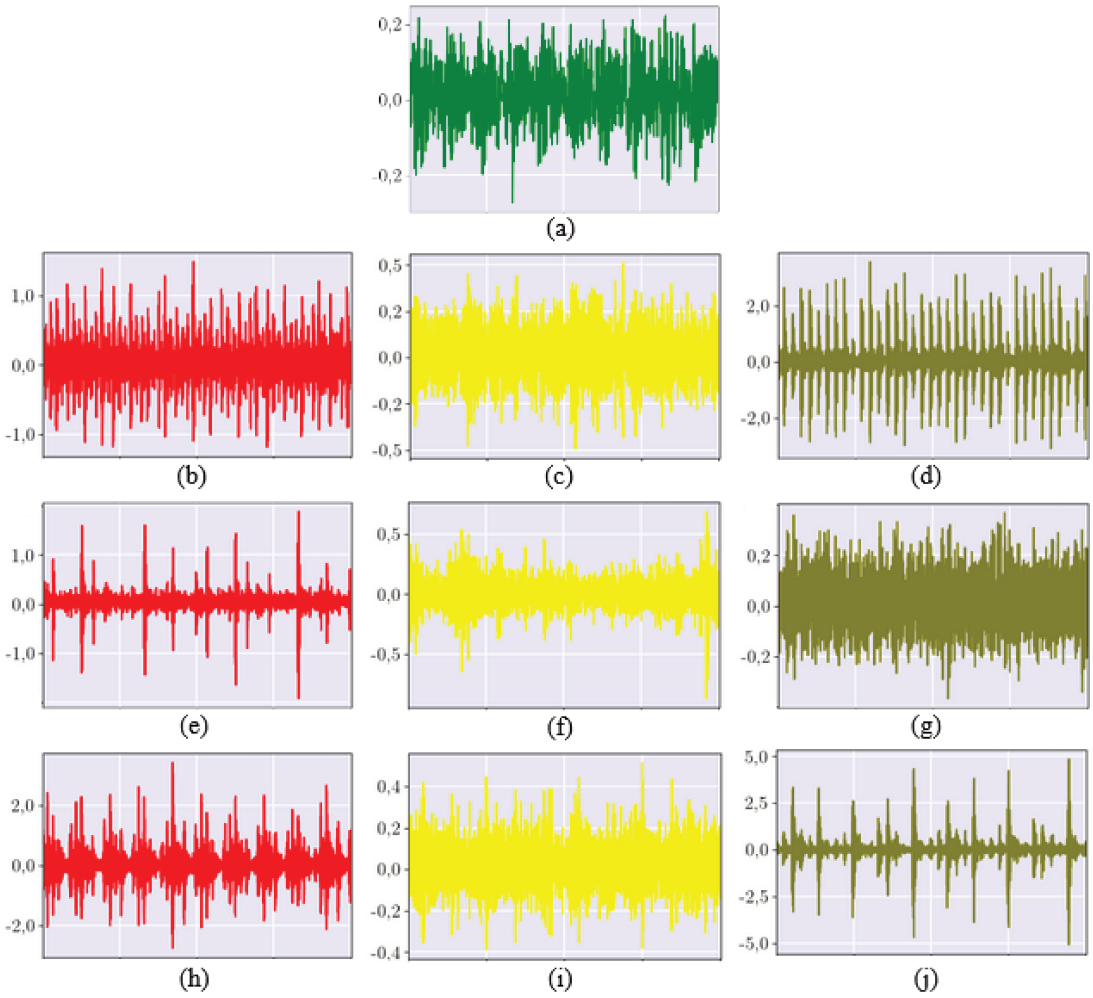

Figura 4.14 – Sinais de vibração de cada estado nas primeiras 10 rotações a 1797 rpm da bancada da *Case Western Reserve University*: (a) saudável, falha de 0,007" de (b) anel interno, (c) elemento rolante e (d) anel externo, falha de 0,014" de (e) anel interno, (f) elemento rolante e (g) anel externo, e falha de 0,021" de (h) anel interno, (i) elemento rolante e (j) anel externo.

épocas do refinamento.

4.2.4 Efeitos da normalização

As vantagens de se normalizar os dados tornaram-se ainda mais evidentes neste estudo de caso. O gráfico da Figura 4.17 mostra a evolução da acurácia ao longo do refinamento sobre os dados da CWRU. Na medida em que a convergência foi alcançada logo nas primeiras épocas com o uso de dados normalizados em uma patamar muito superior de acurácia, próximo a 100%, o treinamento com dados brutos sofreu com oscilações durante todo o processo, com acurácia variando bruscamente entre 60% e 80%.

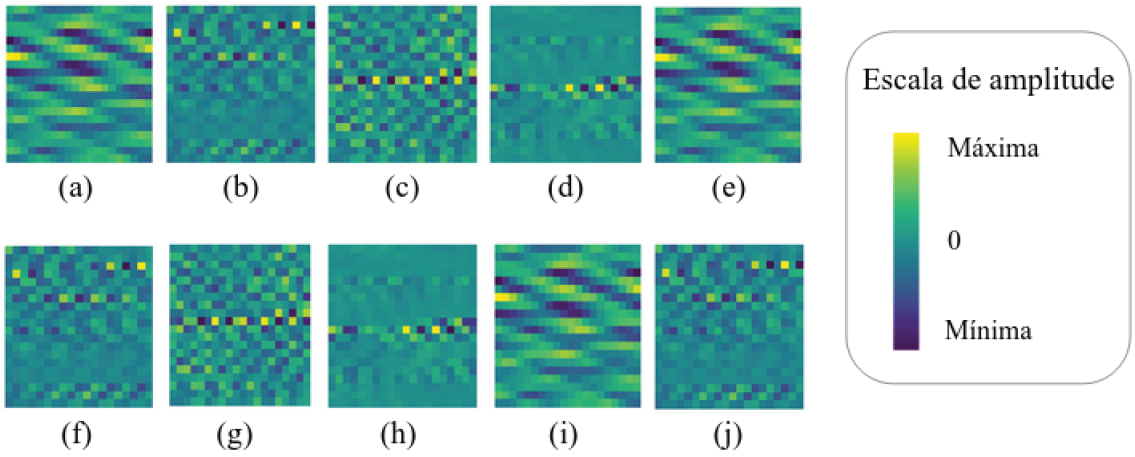

Figura 4.15 – Exemplos de imagens de vibrações geradas para cada uma das dez classes: (a) saudável, falha de 0,007" de (b) anel interno, (c) elemento rolante e (d) anel externo, falha de 0,014" de (e) anel interno, (f) elemento rolante e (g) anel externo, e falha de 0,021" de (h) anel interno, (i) elemento rolante e (j) anel externo.

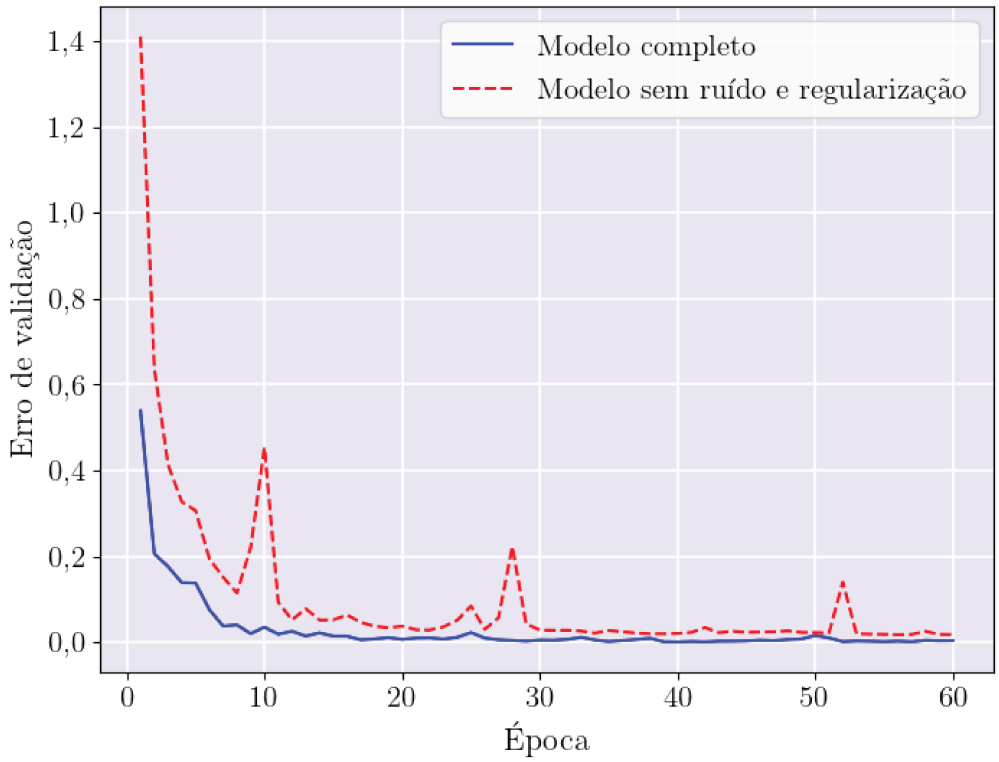

Figura 4.16 – Influência da adição de ruído e regularização no segundo estudo de caso.

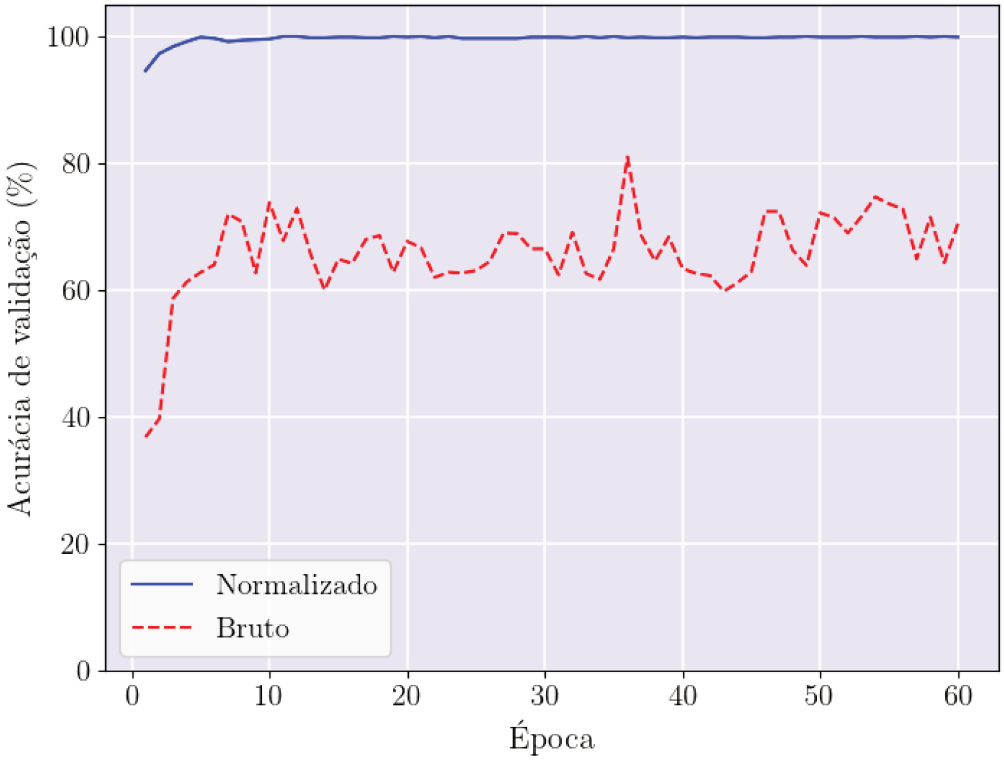

Figura 4.17 – Efeitos de normalização no segundo estudo de caso.

4.2.5 Matriz de confusão

A matriz de confusão apresentada na Figura 4.18 se refere aos resultados do diagnóstico aplicado aos dados da CWRU. O desempenho observado nesse caso foi ainda mais satisfatório que o obtido no experimento anterior. Apenas 3 das 2416 imagens foram classificadas erroneamente, o que corresponde a somente $0,12\%$ do total de imagens testadas. Nota-se que, em todas as confusões, o estado do rolamento foi incorretamente classificado como pertencente à classe 6, correspondente à falha de diâmetro $0,014$" de anel externo.

A Tabela 4.7 contém as métricas da matriz de confusão obtidas para cada classe. Dentre as dez classes consideradas, seis atingiram 100% em pontuação F1. Portanto, dado o subconjunto de teste utilizado, conclui-se que o DCAE foi capaz de executar o diagnóstico mais apurado possível em relação a qualquer uma dessas classes.

		Predito									
		0	1	2	3	4	5	6	7	8	9
Real	0	**249**	0	0	0	0	0	0	0	0	0
	1	0	**252**	0	0	0	0	0	0	0	0
	2	0	0	**233**	0	0	0	0	0	0	0
	3	0	0	0	**236**	0	0	0	0	0	0
	4	0	0	0	0	**222**	0	1	0	0	0
	5	0	0	0	0	0	**239**	1	0	0	0
	6	0	0	0	0	0	0	**235**	0	0	0
	7	0	0	0	0	0	0	0	**253**	0	0
	8	0	0	0	0	0	0	1	0	**261**	0
	9	0	0	0	0	0	0	0	0	0	**233**

Acurácia total = 99,88%

Figura 4.18 – Matriz de confusão do segundo estudo de caso.

Tabela 4.7 – Valores de precisão, revocação e pontuação F1 de cada classe no segundo estudo de caso

Classe	Precisão (%)	Revocação (%)	Pontuação F1 (%)
0	100,0000	100,0000	100,0000
1	100,0000	100,0000	100,0000
2	100,0000	100,0000	100,0000
3	100,000	100,0000	100,0000
4	100,0000	99,5516	99,7753
5	100,0000	99,5833	99,7912
6	98,7395	100,0000	99,3658
7	100,0000	100,0000	100,0000
8	100,0000	99,6183	99,8088
9	100,0000	100,0000	100,0000

4.2.6 Comparação dos métodos quanto à robustez a ruído

A Figura 4.19 apresenta os resultados do teste de robustez para este estudo de caso. Dessa vez, as demais arquiteturas convolucionais conseguiram desempenho similar ao DCAE enquanto sujeitas a baixos níveis de ruído. No entanto, conforme se aumentou o ruído aplicado, o DCAE foi minimamente afetado, com alguma variação observada apenas a partir de 8 dB, na medida em que os demais modelos sofreram quedas bruscas nos valores de acurácia. Os modelos inteiramente conectados (MLP e DAE), apesar de também terem sido pouco afetados pelo aumento do ruído, não atingiram o patamar de acurácia dos modelos convolucionais na faixa de valores de SNR considerada.

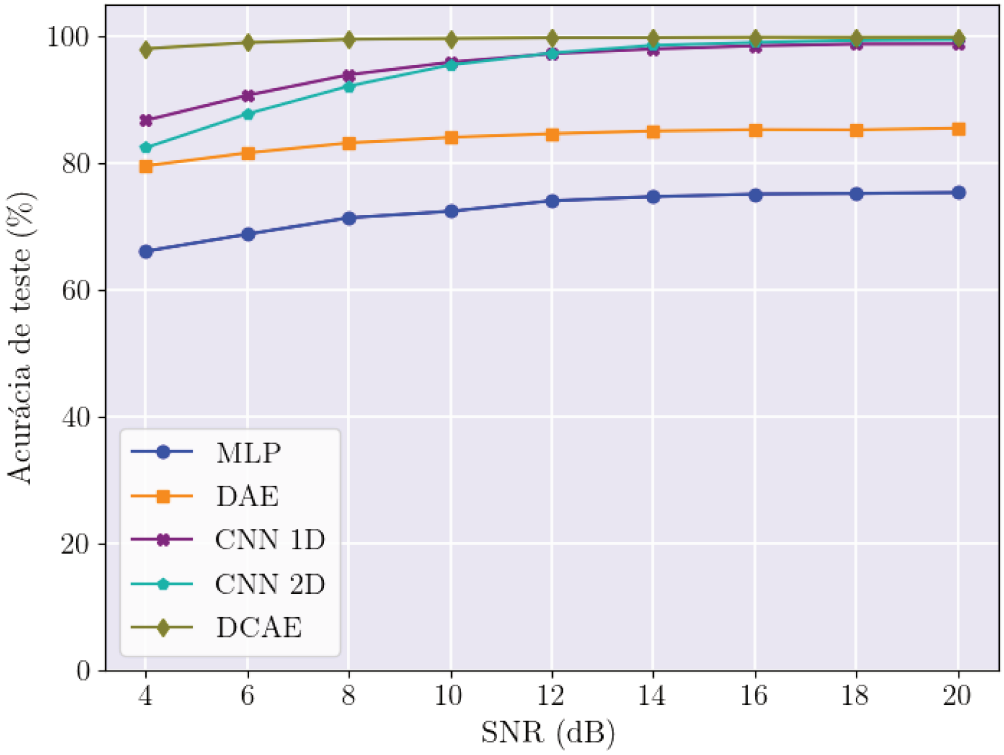

Figura 4.19 – Comparação de robustez a ruído entre o DCAE e outros modelos de redes neurais no segundo estudo de caso.

De maneira similar ao primeiro estudo de caso, observa-se que o DCAE, em comparação com os modelos CNN 1D e CNN 2D, é menos sensível à corrupção dos dados de teste e, portanto, mais robusto. A causa mais provável do comportamento mais satisfatório da rede DCAE em relação às outras redes, considerando os aspectos que as diferenciam, é a introdução de ruído nas entradas durante o treinamento. Dessa forma, afirma-se que a máscara gaussiana

cumpriu com eficácia sua função ao garantir maior robustez ao modelo.

5 CONCLUSÕES

5.1 Considerações do trabalho

A implementação de uma configuração original de *denoising convolutional autoencoder* foi realizada baseada em um *autoencoder* assimétrico para pré-treinamento e uma rede neural convolucional resultante do respectivo *encoder* para classificação de falhas em componentes de máquinas rotativas. O modelo utiliza como entradas imagens geradas a partir de sinais de vibração normalizados e foi avaliado através de sua aplicação sobre dois diferentes conjuntos de dados referentes a falhas em rolamentos. No primeiro estudo de caso, um procedimento para seleção de hiperparâmetros específicos foi realizado, resultando em uma configuração definitiva que foi também testada no segundo estudo de caso.

Apesar das diferenças existentes entre os dois conjuntos de dados, o modelo proposto provou ser eficiente quanto à tarefa de diagnóstico da saúde de componentes em ambos os casos. A acurácia de classificação obtida com o DCAE proposto superou as obtidas com os demais modelos abordados mesmo quando submetidos a altas taxas de ruído. O estudo do desempenho da arquitetura sob a influência de ruído nos dados de teste mostrou que o modelo é mais resistente a possíveis variações que podem vir a ocorrer em ambientes de operação menos controlados. Tal aspecto viabiliza sua aplicação em cenários mais próximos aos reais, sujeitos a condições variáveis de operação.

O erro de validação obtido pelo DCAE foi reduzido através da introdução de uma máscara de ruído gaussiano na entrada da rede e da regularização das camadas intermediárias do *encoder* pelo uso de máscaras de *dropout*. Como consequência da combinação das taxas de ruído e de *dropout* adotadas, a acurácia de validação obtida superou a acurácia de treinamento, resultando na robustez suficiente para se garantir uma maior capacidade de generalização do modelo.

Através da análise dos resultados de validação do treinamento, foi possível observar os benefícios oferecidos pelos procedimentos de pré-processamento adotados. Em ambos os estudos de caso, a normalização *z-score* proporcionou maior precisão de classificação, além de aumentar significativamente a velocidade de convergência e a estabilidade do treinamento. Durante o teste do modelo sobre sinais brutos de vibração, observou-se que a utilização direta de dados coletados dos acelerômetros fez com que a faixa de valores de erro resultantes da propa-

gação dependesse fortemente das características de cada máquina e variasse amplamente entre diferentes condições operacionais, podendo causar efeitos indesejáveis à integridade numérica do procedimento. Tais efeitos são ocasionados principalmente pelo fenômeno de explosão do gradiente, que ocorre quando valores de erro muito altos são obtidos ao final de uma propagação, os quais o computador é incapaz de processar devidamente, e foram evitados com o uso da normalização *z-score*

5.2 Sugestões para trabalhos futuros

Futuros trabalhos baseados nos conceitos aqui discutidos podem se aprofundar no estudo dos efeitos de outros hiperparâmetros sobre o desempenho da rede, tais como o número de filtros e o número de camadas convolucionais. Abordagens interessantes que também podem ser adotadas é a investigação de novos métodos de pré-treinamento e a análise de outras técnicas de regularização, tais como as baseadas em penalidades. Diferentes alternativas para geração de imagens a partir de sinais de vibração também podem ser exploradas. Imagens de espectrogramas geradas através de transformada de Fourier de curto termo ou imagens de escalogramas geradas através de transformada *wavelet* são técnicas capazes de fornecer informações tanto no domínio do tempo, como no domínio da frequência, contribuindo significativamente para o processo de extração de atributos.

Além de modificações na arquitetura ou nos hiperparâmetros da rede, pode também ser investigada a aplicação do método em outros conjuntos de dados, relacionados a diferentes componentes e mecanismos de falha, ou contendo quantidades desbalanceadas de dados disponíveis referentes aos estados de saúde considerados.

REFERÊNCIAS

Aggarwal, C. C. **Neural Networks and Deep Learning: A Textbook**. Cham: Springer International Publishing, 2018.

Al-Badour, F.; Sunar, M.; Cheded, L. Vibration Analysis of Rotating Machinery Using Time–Frequency Analysis and Wavelet Techniques. **Mechanical Systems and Signal Processing**, v. 25, n. 6, p. 2083–2101, aug 2011.

Ba, J. L.; Kiros, J. R.; Hinton, G. E. Layer Normalization. **arXiv preprint arXiv: 1607.06450**, 2016.

Baldi, P. Autoencoders, Unsupervised Learning and Deep Architectures. In: **Proceedings of the 2011 International Conference on Unsupervised and Transfer Learning Workshop**. [S.l.]: JMLR, 2011. p. 37–50.

Boyat, A. K.; Joshi, B. K. A Review Paper: Noise Models in Digital Image Processing. **Signal & Image Processing : An International Journal**, v. 6, n. 2, p. 63–75, 2015.

Chalouli, M.; Berrached, N.-e.; Denai, M. Intelligent Health Monitoring of Machine Bearings Based on Feature Extraction. **Journal of Failure Analysis and Prevention**, v. 17, n. 5, p. 1053–1066, 2017.

Chen, Z.; Gryllias, K.; Li, W. Mechanical Fault Diagnosis Using Convolutional Neural Networks and Extreme Learning Machine. **Mechanical Systems and Signal Processing**, v. 133, p. 106272, 2019.

Chen, Z.; Li, Z. Research on Fault Diagnosis Method of Rotating Machinery Based on Deep Learning. In: **2017 Prognostics and System Health Management Conference (PHM-Harbin)**. [S.l.]: IEEE, 2017. p. 1–4.

Daga, A. P.; Fasana, A.; Marchesiello, S.; Garibaldi, L. The Politecnico di Torino Rolling Bearing Test Rig: Description and Analysis of Open Access Data. **Mechanical Systems and Signal Processing**, v. 120, p. 252–273, 2019.

Du, B.; Xiong, W.; Wu, J.; Zhang, L.; Zhang, L.; Tao, D. Stacked Convolutional Denoising Auto-Encoders for Feature Representation. **IEEE Transactions on Cybernetics**, v. 47, n. 4, p. 1017–1027, 2017.

Ellefsen, A. L.; Æsøy, V.; Ushakov, S.; Zhang, H. A Comprehensive Survey of Prognostics and Health Management Based on Deep Learning for Autonomous Ships. **IEEE Transactions on Reliability**, v. 68, n. 2, p. 720–740, 2019.

Eren, L.; Ince, T.; Kiranyaz, S. A Generic Intelligent Bearing Fault Diagnosis System Using Compact Adaptive 1D CNN Classifier. **Journal of Signal Processing Systems**, v. 91, n. 2, p. 179–189, 2019.

Ertel, W. **Introduction to Artificial Intelligence**. Cham: Springer International Publishing, 2017.

Ghadai, S.; Lee, X. Y.; Balu, A.; Sarkar, S.; Krishnamurthy, A. Multi-Level 3D CNN for Learning Multi-Scale Spatial Features. In: **2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)**. [S.l.: s.n.], 2019. p. 1152–1156.

Goyal, D.; Chaudhary, A.; Dang, R. K.; Pabla, B. S.; Dhami, S. S. Condition Monitoring of Rotating Machines: A Review. **World Scientific News: An International Scientific Journal**, v. 113, n. October, p. 98–108, 2018.

Guo, X.; Liu, X.; Zhu, E.; Yin, J. Deep Clustering with Convolutional Autoencoders. In: **Neural Information Processing**. Cham: Springer, 2017. p. 373–382.

Hasani, R. M.; Wang, G.; Grosu, R. An Automated Auto-Encoder Correlation-Based Health-Monitoring and Prognostic Method for Machine Bearings. **arXiv preprint arXiV: 1703.06272**, 2017.

Heo, S.; Lee, J. H. Fault Detection and Classification Using Artificial Neural Networks. **IFAC-PapersOnLine**, v. 51, n. 18, p. 470–475, 2018.

Heumann, C.; Schomaker, M.; Shalabh. **Introduction to Statistics and Data Analysis**. Cham: Springer International Publishing, 2016.

Hinton, G. E.; Salakhutdinov, R. R. Reducing the Dimensionality of Data with Neural Networks. **Science**, v. 313, n. 5786, p. 504–507, 2006.

Ho, Y.; Wookey, S. The Real-World-Weight Cross-Entropy Loss Function: Modeling the Costs of Mislabeling. **IEEE Access**, v. 8, p. 4806–4813, 2020.

Hoang, D. T.; Kang, H. J. Rolling Element Bearing Fault Diagnosis Using Convolutional Neural Network and Vibration Image. **Cognitive Systems Research**, v. 53, p. 42–50, 2019.

Ince, T.; Kiranyaz, S.; Eren, L.; Askar, M.; Gabbouj, M. Real-Time Motor Fault Detection by 1-D Convolutional Neural Networks. **IEEE Transactions on Industrial Electronics**, v. 63, n. 11, p. 7067–7075, 2016.

Ioffe, S.; Szegedy, C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In: **Proceedings of the 32nd International Conference on International Conference on Machine Learning**. [S.l.: s.n.], 2015. p. 448–456.

Jabbar, H. K.; Khan, R. Z. Methods to Avoid Over-Fitting and Under-Fitting in Supervised Machine Learning (Comparative Study). In: **Computer Science, Communication and Instrumentation Devices**. Singapore: Research Publishing Services, 2014. p. 163–172.

Janssens, O.; Slavkovikj, V.; Vervisch, B.; Stockman, K.; Loccufier, M.; Verstockt, S.; Van de Walle, R.; Van Hoecke, S. Convolutional Neural Network Based Fault Detection for Rotating Machinery. **Journal of Sound and Vibration**, v. 377, p. 331–345, 2016.

Jedliński, Ł.; Jonak, J. Early Fault Detection in Gearboxes Based on Support Vector Machines and Multilayer Perceptron with a Continuous Wavelet Transform. **Applied Soft Computing Journal**, v. 30, p. 636–641, 2015.

Jia, F.; Lei, Y.; Guo, L.; Lin, J.; Xing, S. A Neural Network Constructed by Deep Learning Technique and Its Application to Intelligent Fault Diagnosis of Machines. **Neurocomputing**, v. 272, p. 619–628, 2018.

Jiang, G.; He, H.; Yan, J.; Xie, P. Multiscale Convolutional Neural Networks for Fault Diagnosis of Wind Turbine Gearbox. **IEEE Transactions on Industrial Electronics**, v. 66, n. 4, p. 3196–3207, 2019.

Jing, L.; Zhao, M.; Li, P.; Xu, X. A Convolutional Neural Network Based Feature Learning and Fault Diagnosis Method for the Condition Monitoring of Gearbox. **Measurement**, v. 111, p. 1 – 10, 2017.

Khan, S.; Rahmani, H.; Shah, S. A. A.; Bennamoun, M. **A Guide to Convolutional Neural Networks for Computer Vision**. [S.l.]: Morgan & Claypool Publishers, 2018.

Kingma, D. P.; Ba, J. Adam: A Method for Stochastic Optimization. In: **3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings**. [S.l.: s.n.], 2015. p. 1–15.

Krizhevsky, A.; Sutskever, I.; Hinton, G. E. ImageNet Classification with Deep Convolutional Neural Networks. In: **Proceedings of the 25th International Conference on Neural Information Processing Systems**. Red Hook, NY, USA: Curran Associates Inc., 2012. p. 1097–1105.

Kubat, M. **An Introduction to Machine Learning**. Cham: Springer International Publishing, 2017.

Lee, J.; Wu, F.; Zhao, W.; Ghaffari, M.; Liao, L.; Siegel, D. Prognostics and Health Management Design for Rotary Machinery Systems — Reviews, Methodology and Applications. **Mechanical Systems and Signal Processing**, v. 42, n. 1-2, p. 314–334, 2014.

Lee, K.; Kim, J.-K.; Kim, J.; Hur, K.; Kim, H. Stacked Convolutional Bidirectional LSTM Recurrent Neural Network for Bearing Anomaly Detection in Rotating Machinery Diagnostics. In: **2018 1st IEEE International Conference on Knowledge Innovation and Invention (ICKII)**. [S.l.]: IEEE, 2018. p. 98–101.

Lei, Y.; Jia, F.; Lin, J.; Xing, S.; Ding, S. X. An Intelligent Fault Diagnosis Method Using Unsupervised Feature Learning Towards Mechanical Big Data. **IEEE Transactions on Industrial Electronics**, v. 63, n. 5, p. 3137–3147, 2016.

Li, C.; Zhang, W.; Peng, G.; Liu, S. Bearing Fault Diagnosis Using Fully-Connected Winner-Take-All Autoencoder. **IEEE Access**, v. 6, p. 6103–6115, 2018.

Liu, R.; Meng, G.; Yang, B.; Sun, C.; Chen, X. Dislocated Time Series Convolutional Neural Architecture: An Intelligent Fault Diagnosis Approach for Electric Machine. **IEEE Transactions on Industrial Informatics**, v. 13, n. 3, p. 1310–1320, 2017.

Liu, R.; Yang, B.; Zio, E.; Chen, X. Artificial Intelligence for Fault Diagnosis of Rotating Machinery: A Review. **Mechanical Systems and Signal Processing**, v. 108, p. 33–47, 2018.

Liu, X.; Zhou, Q.; Zhao, J.; Shen, H.; Xiong, X. Fault Diagnosis of Rotating Machinery Under Noisy Environment Conditions Based on a 1-D Convolutional Autoencoder and 1-D Convolutional Neural Network. **Sensors**, v. 19, n. 4, p. 1–19, 2019.

Loparo, K. **Case Western Reserve University Bearing Data Center Website**. 2012. https://csegroups.case.edu/bearingdatacenter/pages/download-data-file. Acesso em: 19 de dez. de 2019.

Lu, C.; Wang, Z.-Y.; Qin, W.-L.; Ma, J. Fault Diagnosis of Rotary Machinery Components Using a Stacked Denoising Autoencoder-Based Health State Identification. **Signal Processing**, v. 130, p. 377–388, 2017.

Ma, J.; Su, H.; Zhao, W.-l.; Liu, B. Predicting the Remaining Useful Life of an Aircraft Engine Using a Stacked Sparse Autoencoder with Multilayer Self-Learning. **Complexity**, v. 2018, p. 1–13, 2018.

Mahamad, A. K.; Saon, S.; Hiyama, T. Predicting Remaining Useful Life of Rotating Machinery Based Artificial Neural Network. **Computers and Mathematics with Applications**, v. 60, n. 4, p. 1078–1087, 2010.

Masci, J.; Meier, U.; CireŞan, D.; Schmidhuber, J. Stacked Convolutional Auto-Encoders for Hierarchical Feature Extraction. In: **Artificial Neural Networks and Machine Learning – ICANN 2011**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011. p. 52–59.

Nasr, G. E.; Badr, E. A.; Joun, C. Cross Entropy Error Function in Neural Networks: Forecasting Gasoline Demand. In: **Proceedings of the Fifteenth International Florida Artificial Intelligence Research Society Conference**. [S.l.]: AAAI Press, 2002. p. 381–384.

Osmani, A.; Hamidi, M.; Bouhouche, S. Monitoring of a Dynamic System Based on Autoencoders. In: **Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19**. [S.l.]: International Joint Conferences on Artificial Intelligence Organization, 2019. p. 1836–1843.

Prechelt, L. Early Stopping - But When? In: Montavon, G.; Orr, G. B.; Müller, K.-R. (Ed.). **Neural Networks: Tricks of the Trade: Second Edition**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012. p. 53–67.

Rababaah, A. R.; Arumala, J.; Dabipi, I. K.; Fotouhi, K.; Hura, G.; Dudi, A. Mechanical System Fault Detection Using Intelligent Digital Signal Processing. **Journal of Machinery Manufacturing and Automation**, v. 5, n. 1, p. 27–39, 2016.

Randall, R. B.; Antoni, J. Rolling Element Bearing Diagnostics — A Tutorial. **Mechanical Systems and Signal Processing**, v. 25, n. 2, p. 485–520, 2011.

Riaz, S.; Elahi, H.; Javaid, K.; Shahzad, T. Vibration Feature Extraction and Analysis for Fault Diagnosis of Rotating Machinery - A Literature Survey. **Asia Pacific Journal of Multidisciplinary Research**, v. 5, n. 1, p. 103–110, 2017.

Roland, U.; Eseosa, O. Artificial Intelligent Techniques in Real-Time Diagnosis of Stator and Rotor Faults in Induction Machines. **International Journal of Scientific & Engineering Research**, v. 5, n. 10, p. 946–954, 2014.

Ruder, S. An Overview of Gradient Descent Optimization Algorithms. **arXiv preprint arXiv: 1609.04747**, p. 1–14, 2016.

Santurkar, S.; Tsipras, D.; Ilyas, A.; Madry, A. How does batch normalization help optimization? **Advances in Neural Information Processing Systems**, v. 2018-December, n. NeurIPS, p. 2483–2493, 2018.

Sathya, R.; Abraham, A. Comparison of Supervised and Unsupervised Learning Algorithms for Pattern Classification. **International Journal of Advanced Research in Artificial Intelligence**, v. 2, n. 2, 2013.

Sen, A.; Majumder, M. C.; Mukhopadhyay, S.; Biswas, R. K. Condition Monitoring of Rotating Equipment Considering the Cause and Effects of Vibration : A Brief Review. **International Journal of Modern Engineering Research**, v. 7, n. 1, p. 36–49, 2017.

Serwa, A. Studying the Effect of Activation Function on Classification Accuracy Using Deep Artificial Neural Networks. **Journal of Remote Sensing & GIS**, v. 06, n. 03, p. 1–6, 2017.

Shao, H.; Jiang, H.; Lin, Y.; Li, X. A novel Method for Intelligent Fault Diagnosis of Rolling Bearings Using Ensemble Deep Auto-Encoders. **Mechanical Systems and Signal Processing**, v. 102, p. 278–297, 2018.

Shao, H.; Jiang, H.; Zhao, H.; Wang, F. A Novel Deep Autoencoder Feature Learning Method for Rotating Machinery Fault Diagnosis. **Mechanical Systems and Signal Processing**, v. 95, p. 187–204, 2017.

Shao, S.; McAleer, S.; Yan, R.; Baldi, P. Highly Accurate Machine Fault Diagnosis Using Deep Transfer Learning. **IEEE Transactions on Industrial Informatics**, v. 15, n. 4, p. 2446–2455, 2019.

Sharma, N.; Jain, V.; Mishra, A. An Analysis of Convolutional Neural Networks for Image Classification. **Procedia Computer Science**, v. 132, p. 377–384, 2018.

Skansi, S. **Introduction to Deep Learning**. Cham: Springer International Publishing, 2018.

Song, Y. Y.; Lu, Y. Decision Tree Methods: Applications for Classification and Prediction. **Shanghai Archives of Psychiatry**, v. 27, n. 2, p. 130–135, 2015.

Srivastava, N.; Hinton, G.; Krizhevsky, A.; Sutskever, I.; Salakhutdinov, R. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. In: **Journal of Machine Learning Research**. [S.l.]: JMLR, 2014. p. 1929–1958.

Stetco, A.; Dinmohammadi, F.; Zhao, X.; Robu, V.; Flynn, D.; Barnes, M.; Keane, J.; Nenadic, G. Machine Learning Methods for Wind Turbine Condition Monitoring: A Review. **Renewable Energy**, v. 133, p. 620–635, 2019.

Sun, J.; Yan, C.; Wen, J. Intelligent Bearing Fault Diagnosis Method Combining Compressed Data Acquisition and Deep Learning. **IEEE Transactions on Instrumentation and Measurement**, v. 67, n. 1, p. 185–195, 2018.

Tchakoua, P.; Wamkeue, R.; Ouhrouche, M.; Slaoui-Hasnaoui, F.; Tameghe, T. A.; Ekemb, G. Wind Turbine Condition Monitoring: State-of-the-Art Review, New Trends, and Future Challenges. **Energies**, v. 7, n. 4, p. 2595–2630, 2014.

Verma, N. K.; Gupta, V. K.; Sharma, M.; Sevakula, R. K. Intelligent Condition Based Monitoring of Rotating Machines Using Sparse Auto-Encoders. In: **2013 IEEE Conference on Prognostics and Health Management (PHM)**. [S.l.]: IEEE, 2013. p. 1–7.

Verstraete, D.; Ferrada, A.; Droguett, E. L.; Meruane, V.; Modarres, M. Deep Learning Enabled Fault Diagnosis Using Time-Frequency Image Analysis of Rolling Element Bearings. **Shock and Vibration**, v. 2017, p. 1–17, 2017.

Vicuña, C. M.; Acuña, D. Q. Cyclostationary Processing of Vibration and Acoustic Emissions for Machine Failure Diagnosis. In: Chaari, F.; Leśkow, J.; Napolitano, A.; Sanchez-Ramirez, A. (Ed.). **Cyclostationarity: Theory and Methods**. Cham: Springer International Publishing, 2014. p. 141–156.

Vincent, P.; Larochelle, H.; Bengio, Y.; Manzagol, P.-A. Extracting and Composing Robust Features with Denoising Autoencoders. In: **Proceedings of the 25th International Conference on Machine Learning**. New York, NY, USA: ACM Press, 2008. p. 1096–1103.

Wen, L.; Li, X.; Gao, L.; Zhang, Y. A New Convolutional Neural Network-Based Data-Driven Fault Diagnosis Method. **IEEE Transactions on Industrial Electronics**, v. 65, n. 7, p. 5990–5998, 2018.

Wu, C.; Jiang, P.; Ding, C.; Feng, F.; Chen, T. Intelligent Fault Diagnosis of Rotating Machinery Based on One-Dimensional Convolutional Neural Network. **Computers in Industry**, v. 108, p. 53–61, 2019.

Xia, M.; Li, T.; Shu, T.; Wan, J.; De Silva, C. W.; Wang, Z. A Two-Stage Approach for the Remaining Useful Life Prediction of Bearings Using Deep Neural Networks. **IEEE Transactions on Industrial Informatics**, v. 15, n. 6, p. 3703–3711, 2019.

Xie, Y.; Zhang, T. Fault Diagnosis for Rotating Machinery Based on Convolutional Neural Network and Empirical Mode Decomposition. **Shock and Vibration**, v. 2017, p. 1–12, 2017.

Xu, D.; Tian, Y. A Comprehensive Survey of Clustering Algorithms. **Annals of Data Science**, v. 2, n. 2, p. 165–193, 2015.

Xu, G.; Liu, M.; Jiang, Z.; Söffker, D.; Shen, W. Bearing Fault Diagnosis Method Based on Deep Convolutional Neural Network and Random Forest Ensemble Learning. **Sensors**, v. 19, n. 5, p. 1088, 2019.

Zhang, R.; Peng, Z.; Wu, L.; Yao, B.; Guan, Y. Fault Diagnosis from Raw Sensor Data Using Deep Neural Networks Considering Temporal Coherence. **Sensors**, v. 17, n. 3, p. 549, 2017.

Zhang, S.; Ye, F.; Wang, B.; Habetler, T. G. Semi-Supervised Learning of Bearing Anomaly Detection via Deep Variational Autoencoders. **arXiv preprint arXiv: 1912.01096**, 2019.

Zhang, S.; Zhang, S.; Wang, B.; Habetler, T. G. Deep Learning Algorithms for Bearing Fault Diagnostics—A Comprehensive Review. **IEEE Access**, v. 8, p. 29857–29881, 2020.

Zhang, Y.; Balochian, S.; Agarwal, P.; Bhatnagar, V.; Housheya, O. J. Artificial Intelligence and Its Applications. **Mathematical Problems in Engineering**, v. 2016, p. 1–6, 2016.

Zhao, B.; Lu, H.; Chen, S.; Liu, J.; Wu, D. Convolutional Neural Networks for Time Series Classification. **Journal of Systems Engineering and Electronics**, v. 28, n. 1, p. 162–169, 2017.

Zhao, R.; Yan, R.; Chen, Z.; Mao, K.; Wang, P.; Gao, R. X. Deep Learning and Its Applications to Machine Health Monitoring. **Mechanical Systems and Signal Processing**, v. 115, p. 213–237, 2019.

Apêndices

APÊNDICE A – CÓDIGO DA MODELAGEM DO DCAE

dcae.py

```
'''

DENOISING CONVOLUTIONAL AUTOENCODER

Autor: Leonardo Franco de Godói

Data: 20/06/2020

'''

################################################################################

# Importando pacotes e módulos
from tensorflow.keras.layers import (
    Input,
    Dense,
    Dropout,
    Conv2D,
    MaxPooling2D,
    UpSampling2D,
    BatchNormalization,
    Flatten,
    Reshape,
    GaussianNoise,
)
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping
import numpy as np
from math import sqrt

# ------------------------------------------------------------------------------

'''

DENOISING CONVOLUTIONAL AUTOENCODER (DCAE)

     Argumentos:
          - X_train = dados entrada de treinamento
          - y_train = rótulos de treinamento
          - filters = número de filtros nas camadas convolucionais
```

```
          - filter_sizes = tamanhos dos filtros convolucionais
          - sampling = Fator de redução na subamostragem
          - dp = nível de dropout (%/100)
          - epochs = número de épocas
          - batch_size = tamanho de lote
          - lr = taxa de aprendizado
          - momentum = momento do SGD
          - stddev = desvio padrão do ruído da entrada
          - pre_training = habilita o pré-treinamento
          - summary = habilita a visualização da arquitetura
          - save = permite salvar modelos e históricos

          *** Deve-se usar entradas (X_train) de tamanho "N",
              tal que "N" seja o quadrado de um número inteiro e
              potência do fator de subamostragem (sampling), para
              que seja possível redimensioná-las e reconstruí-las
              adequadamente.

'''

# Função do DCAE
def dcae(X_train,
         y_train,
         filters=(32, 64),
         filter_sizes=(12, 6),
         sampling=2,
         dp=0.2,
         epochs=(30, 60),
         batch_size=32,
         lr=0.001,
         momentum=0.9,
         stddev=0.2,
         pre_training=True,
         summary=True,
         save=False):

     # Configurações gerais
     act_hidden = 'tanh'
     act_classifier = 'softmax'
     padding='same'
     loss_pt = 'mse'
     loss_ft = 'categorical_crossentropy'
     optimizer = SGD(lr=lr, momentum=momentum)
     val_split = 0.1

     # Opção para salvar o melhor modelo
     cp_pt = ModelCheckpoint('trained_models/dcae_pt.h5',
```

```
                            save_best_only=True,
                            monitor='val_loss',
                            mode='max')

    # Opção para salvar o melhor modelo
    cp_ft = ModelCheckpoint('trained_models/dcae_ft.h5',
                            save_best_only=True,
                            monitor='val_accuracy',
                            mode='max')

    # Parada antecipada
    es = EarlyStopping(monitor='val_loss', patience=2)

    #  Arquitetura do DCAE
    my_input = Input(shape=(X_train.shape[1],))
    reshaped_in = Reshape((int(sqrt(X_train.shape[1])),
                           int(sqrt(X_train.shape[1])), 1))(my_input)
    noise = GaussianNoise(stddev)(reshaped_in)
    conv_1 = Conv2D(filters[0], filter_sizes[0], activation=act_hidden,
                    padding=padding)(noise)
    bn_1 = BatchNormalization()(conv_1)
    down_1 = MaxPooling2D((sampling, sampling), padding=padding)(bn_1)
    dp_1 = Dropout(dp)(down_1)
    conv_2 = Conv2D(filters[1], filter_sizes[1], activation=act_hidden,
                    padding=padding)(dp_1)
    bn_2 = BatchNormalization()(conv_2)
    down_2 = MaxPooling2D((sampling, sampling), padding=padding)(bn_2)
    dp_2 = Dropout(dp)(down_2)
    up_1 = UpSampling2D((sampling, sampling))(dp_2)
    conv_3 = Conv2D(filters[0], filter_sizes[1], activation=act_hidden,
                    padding=padding)(up_1)
    up_2 = UpSampling2D((sampling, sampling))(conv_3)
    conv_4 = Conv2D(1, filter_sizes[0], padding=padding)(up_2)
    reshaped_out = Flatten()(conv_4)

    # Configuração e treinamento do DCAE
    ae = Model(my_input, reshaped_out)
    ae.compile(loss=loss_pt, optimizer=optimizer)
    if pre_training:
        if summary:
            ae.summary()
        dcae_hist_pt = ae.fit(X_train, X_train, epochs=epochs[0],
                              batch_size=batch_size, verbose=summary,
                              validation_split=val_split, shuffle=True,
                              callbacks=[es, cp_pt])

        # Salvando os históricos de pré-treinamento
```

```
136         if save:
137             np.savetxt('hists/dcae_pt_loss_train_.txt',
138                        dcae_hist_pt.history['loss'])
139             np.savetxt('hists/dcae_pt_loss_val.txt',
140                        dcae_hist_pt.history['val_loss'])
141 
142     # Achatamento e adição do classificador
143     fl = Flatten()(dp_2)
144     my_output = Dense(y_train.shape[1], activation=act_classifier)(fl)
145 
146     # Configuração e treinamento do classificador
147     dcae = Model(my_input, my_output)
148     dcae.compile(loss=loss_ft, optimizer=optimizer, metrics=['accuracy'])
149     if summary:
150          dcae.summary()
151     dcae_hist_ft = dcae.fit(X_train, y_train, epochs=epochs[1],
152                             batch_size=batch_size, verbose=summary,
153                             validation_split=val_split, shuffle=True,
154                             callbacks=[cp_ft])
155 
156     # Salvando os históricos de refinamento
157     if save:
158         np.savetxt('hists/dcae_ft_acc_train.txt',
159                    dcae_hist_ft.history['accuracy'])
160         np.savetxt('hists/dcae_ft_acc_val.txt',
161                    dcae_hist_ft.history['val_accuracy'])
162         np.savetxt('hists/dcae_ft_loss_train.txt',
163                    dcae_hist_ft.history['loss'])
164         np.savetxt('hists/dcae_ft_loss_val.txt',
165                    dcae_hist_ft.history['val_loss'])
166 
167     return dcae
```

APÊNDICE B – CÓDIGO DO TREINAMENTO DO DCAE

train.py

```
'''

TREINAMENTO DO DCAE

Autor: Leonardo Franco de Godói

Data: 21/06/2020

'''

################################################################################

# Importando pacotes e módulos
import numpy as np
from dcae import dcae

# Carregando os conjuntos de treinamento
X_train_raw = np.loadtxt('data/X_train_raw.txt')
X_train_zscore = np.loadtxt('data/X_train_zscore.txt')
y_train = np.loadtxt('data/y_train.txt')

# Opções
summary = True
save=True
n_samples = 190400

# Treinando o DCAE
print('\n************ DCAE ************\n')
model = dcae(X_train_zscore[:n_samples],
             y_train[:n_samples],
             epochs=(20, 60),
             summary=summary,
             save=save,
             pre_training=True)
```

APÊNDICE C – CÓDIGO DO TESTE DE ROBUSTEZ A RUÍDO

test_noise.py

```
"""

TESTE DE ROBUSTEZ A RUÍDO

Autor: Leonardo Franco de Godói

Data: 12/07/2020

"""

################################################################################

# Importando os módulos
from tensorflow.keras.models import load_model
import numpy as np
import pandas as pd

# Função para adição de ruído em um sinal
def apply_noise(x, snr):
    x = np.asarray(x)
    SNR_dB = 20
    snr = 10.0**(SNR_dB/10.0)
    p = x.var()
    n = p/snr
    noise = np.sqrt(n)*np.random.randn(x.shape[0])
    x_noisy = x + noise
    return x_noisy

# Carregando os conjuntos de treinamento
X_test_raw = np.loadtxt('data/X_test_raw.txt')
X_test_minmax = np.loadtxt('data/X_test_minmax.txt')
X_test_zscore = np.loadtxt('data/X_test_zscore.txt')
X_test_zsminmax = np.loadtxt('data/X_test_zscoreminmax.txt')
y_test = np.loadtxt('data/y_test.txt')

# Carregando os modelos treinados
mlp = load_model('trained_models/best_models/mlp.h5')
dae = load_model('trained_models/best_models/ae.h5')
cnn1d = load_model('trained_models/best_models/cnn.h5')
cnn2d = load_model('trained_models/best_models/cae.h5')
dcae = load_model('trained_models/best_models/dcae.h5')
```

```
42
43  # Valores de SNR a serem usados na aplicação de ruído
44  snr = (4, 6, 8, 10, 12, 14, 16, 18, 20)
45
46  # Criando o dataframe que armazenará os resultados
47  results = pd.DataFrame(index=['MLP', 'DAE', 'CNN 1D', 'CNN 2D', 'DCAE'],
48                         columns=snr)
49
50  # Número de testes
51  runs = 10
52
53  # Rotina de teste
54  print('\nIniciando os testes...')
55
56  # Para cada nível de ruído
57  for i in range(len(snr)):
58
59      # Visualizando o ruído em andamento
60      print('\nSNR aplicado: ' + str(snr[i]))
61
62      # Variáveis para armazenar a média de precisão de cada modelo
63      acc_mlp = 0
64      acc_dae = 0
65      acc_cnn1d = 0
66      acc_cnn2d = 0
67      acc_dcae = 0
68
69      # Para cada teste
70      for j in range(runs):
71
72          # Visualizando o progresso do teste
73          print('\n\tTeste ' + str(j+1) + '/' + str(runs))
74
75          # Adicionando ruído
76          X_test_zscore_noisy = apply_noise(X_test_zscore)
77          X_test_minmax_noisy = apply_noise(X_test_minmax)
78          X_test_zsminmax_noisy = apply_noise(X_test_zsminmax)
79
80          # Acumulando os valores de precisão dos modelos
81          _, acc = mlp.evaluate(X_test_zscore_noisy, y_test, verbose=0)
82          acc_mlp += acc
83          _, acc = dae.evaluate(X_test_zscore_noisy, y_test, verbose=0)
84          acc_dae += acc
85          _, acc = cnn1d.evaluate(X_test_zsminmax_noisy, y_test, verbose=0)
86          acc_cnn1d += acc
87          _, acc = cnn2d.evaluate(X_test_minmax_noisy, y_test, verbose=0)
88          acc_cnn2d += acc
```

```
89            _, acc = dcae.evaluate(X_test_zscore_noisy, y_test, verbose=0)
90            acc_dcae += acc
91
92        # Calculando a precisão média
93        results.at['MLP', snr[i]] = np.round((acc_mlp/runs)*100, 2)
94        results.at['DAE', snr[i]] = np.round((acc_dae/runs)*100, 2)
95        results.at['CNN 1D', snr[i]] = np.round((acc_cnn1d/runs)*100, 2)
96        results.at['CNN 2D', snr[i]] = np.round((acc_cnn2d/runs)*100, 2)
97        results.at['DCAE', snr[i]] = np.round((acc_dcae/runs)*100, 2)
98
99  # Salvando os resultados no formato CSV
100 results.to_csv(r'tests/robustness_to_noise.csv', index=True, header=True)
```

APÊNDICE D – CÓDIGO DA MATRIZ DE CONFUSÃO

conf_mat.py

```
'''

GERAÇÃO DA MATRIZ DE CONFUSÃO PARA O SUBCONJUNTO DE TESTE

Autor: Leonardo Franco de Godói

Data: 19/07/2020

'''

################################################################################

# Importando os módulos e pacotes
from tensorflow.keras.models import load_model
import numpy as np
import pandas as pd
import seaborn as sn
from sklearn.metrics import confusion_matrix

def conf_mat(actual, predicted,
             samples=100):

     '''

     Argumentos:
          - actual: vetor com rótulos da predição
          - predicted: vetor com rótulos esperados

     '''

     # Matriz de confusão completa
     full_cm = confusion_matrix(actual, predicted)

     # Convertendo as arrays para numpy
     actual = np.asarray(actual[:samples])
     predicted = np.asarray(predicted[:samples])

     # Verificando se as array têm mais de uma dimensão
     # Caso tenham, são redimensionadas
     if actual.ndim > 1:
          actual = np.reshape(actual, (actual.size,))
```

```
42      if predicted.ndim > 1:
43           predicted = np.reshape(predicted, (predicted.size,))
44
45      # Mostrando a matriz de confusão para um conjunto amostras
46      data={'y_actual': actual, 'y_predicted': predicted}
47      df = pd.DataFrame(data, columns=['y_actual', 'y_predicted'])
48      cm = pd.crosstab(df['y_actual'], df['y_predicted'],
49                       rownames=['Actual'], colnames=['Predicted'])
50      sn.heatmap(cm, annot=True, cmap="YlGnBu")
51
52      return full_cm
53
54 # Carregando os conjuntos de teste
55 X_test = np.loadtxt('data/X_test_zscore.txt')
56 y_test = np.loadtxt('data/y_test.txt')
57
58 # Importando o modelo DCAE
59 dcae = load_model('trained_models/best_models/dcae.h5')
60
61 # Fazendo a predição sobre o conjunto de teste
62 y_pred = dcae.predict(X_test)
63
64 # Atribuindo rótulos às amostras
65 y_pred_max = np.argmax(y_pred, axis=-1)
66 y_test_max = np.argmax(y_test, axis=-1)
67
68 # Gerando a matriz de confusão
69 full_cm = conf_mat(y_test_max, y_pred_max, 1000)
```

Anexos

ANEXO A – INFORMAÇÕES ADICIONAIS SOBRE A BANCADA EXPERIMENTAL DO INSTITUTO POLITÉCNICO DE TURIM

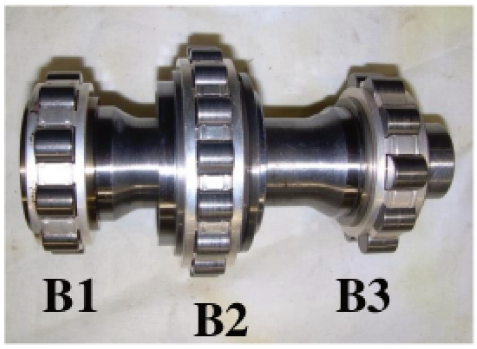

Figura A.1 – Eixo desmontado contendo os três rolamentos. Extraído de (Daga *et al.*, 2019).

	Pitch diameter D (mm)	Rollers diameter d (mm)	Contact angle ϕ (°)	Rolling elements Z
B1 & B3	40.5	9.0	0	10
B2	54.0	8.0	0	16

Figura A.2 – Principais propriedades do rolamentos. Extraído de (Daga *et al.*, 2019).

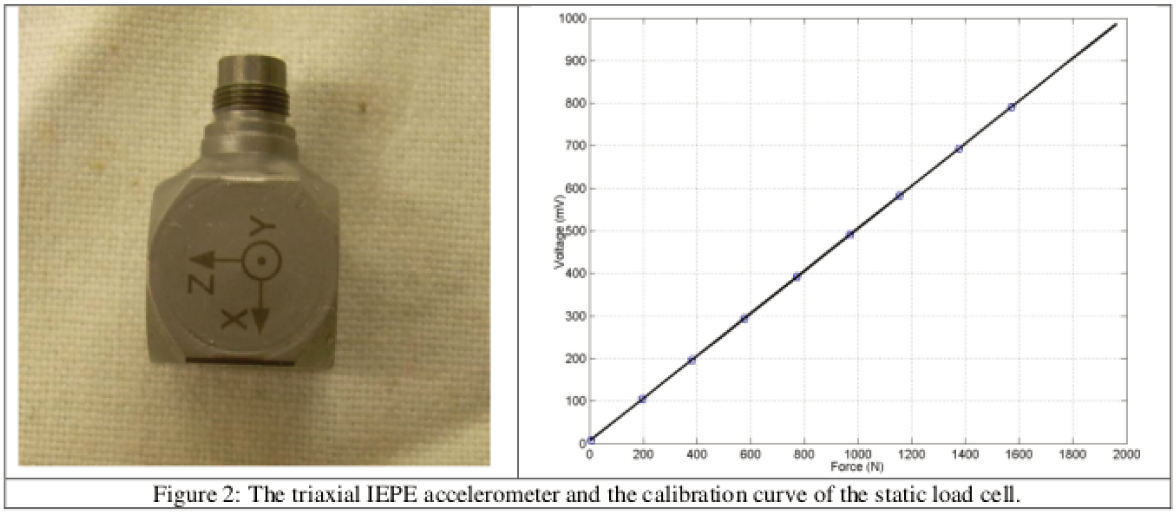

Figure 2: The triaxial IEPE accelerometer and the calibration curve of the static load cell.

Figura A.3 – Acelerômetro triaxial e curva de calibração da célula de carga estática. Extraído de (Daga *et al.*, 2019).