INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE MINAS GERAIS - *CAMPUS* SABARÁ
BACHARELADO EM ENGENHARIA DE CONTROLE E AUTOMAÇÃO

Lucas Rodrigues Coutinho dos Santos

DIAGNÓSTICO AUTOMATIZADO DE FALHAS EM MOTORES DE INDUÇÃO TRIFÁSICOS POR REDES NEURAIS ARTIFICIAIS COM DADOS DO *MOTOR FAULT SIMULATOR* (MFS-UFRJ).

Sabará
2026

LUCAS RODRIGUES COUTINHO DOS SANTOS

DIAGNÓSTICO AUTOMATIZADO DE FALHAS EM MOTORES DE INDUÇÃO TRIFÁSICOS POR REDES NEURAIS ARTIFICIAIS COM DADOS DO *MOTOR FAULT SIMULATOR* (MFS-UFRJ).

Trabalho de Conclusão de Curso apresentado à banca examinadora do curso de Engenharia de Controle e Automação do Instituto Federal de Educação, Ciência e Tecnologia de Minas Gerais *Campus* Sabará, como parte dos requisitos para obtenção do título de Bacharel em Engenharia de Controle e Automação.

Orientador: Prof. Me. Luiz Guilherme Hilel Drumond Silveira

Sabará
2026

Santos, Lucas Rodrigues Coutinho dos

S237i Diagnóstico automatizado de falhas em motores de indução trifásicos por redes neurais artificiais com dados do motor fault simulator (MFS-UFRJ) [manuscrito]. / Lucas Rodrigues Coutinho dos Santos. - 2026.
60 f. : il.

Orientação: Prof. Me. Luiz Guilherme Hilel Drumond Silveira.

Trabalho de Conclusão de Curso (Bacharelado em Engenharia de Controle e Automação) – Instituto Federal de Minas Gerais, *Campus* Sabará.

1. Localização de falhas (Engenharia). – Monografia. 2. Aprendizado do computador. – Monografia. 3. Processamento de Sinais. – Monografia. 4. Automação industrial. – Monografia. 5. Vibração - Medição. – Monografia. 6. Motores elétricos de indução. – Monografia. I. Silveira, Luiz Guilherme Hilel Drumond. II. Instituto Federal de Minas Gerais, *Campus* Sabará. III. Bacharelado em Engenharia de Controle e Automação. IV. Título.

CDU 681.5

César dos Santos Moreira / CRB6-2229
Biblioteca do IFMG *Campus* Sabará

MINISTÉRIO DA EDUCAÇÃO
SECRETARIA DE EDUCAÇÃO PROFISSIONAL E TECNOLÓGICA
INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE MINAS GERAIS
Campus Sabará
Diretoria de Ensino, Pesquisa e Extensão
Conselho de Área - Informática e Comunicação
Rodovia MGC 262, Km 10 - Bairro Sobradinho - CEP 34590-390 - Sabará - MG - www.ifmg.edu.br

ATA DE DEFESA DE TRABALHO DE CONCLUSÃO DE CURSO

Aos nove dias do mês de fevereiro do ano de dois mil e vinte e seis, às dezessete horas, sob a presidência do professor Luiz Guilherme Hilel Drumond, reuniu-se a banca examinadora composta pelos professores abaixo relacionados, para a defesa do Trabalho de Conclusão de Curso (TCC) do discente Lucas Rodrigues Coutinho dos Santos, matrícula nº 0049913, do curso de Engenharia de Controle e Automação, do IFMG *campus* Sabará.

O trabalho intitulado DIAGNÓSTICO AUTOMATIZADO DE FALHAS EM MOTORES DE INDUÇÃO TRIFÁSICOS POR REDES NEURAIS ARTIFICIAIS COM DADOS DO MOTOR FAULT SIMULATOR(MFS-UFRJ) foi apresentado e submetido à apreciação da banca. Após exposição, arguição e deliberação, a banca atribuiu a nota final de 67 pontos (de um total de 80). Somados aos 20 pontos atribuídos ao aluno pelo docente responsável pela disciplina de Projeto II, o aluno ficou com um total de 87 pontos, resultando em **aprovado**, condicionado ao cumprimento das orientações e prazos estabelecidos pelas normas acadêmicas institucionais.

Compuseram a Banca Examinadora:

- Membro 1 (Presidente): Luiz Guilherme Hilel Drumond

- Membro 2: Daniel Neves Rocha

- Membro 3: Moisés Henrique Ramos Pereira

O discente deverá apresentar a versão final do trabalho em formato PDF e depositá-la no repositório institucional até o dia vinte e sete de fevereiro. O não cumprimento dessas exigências implicará na não contabilização das horas referentes aos componentes curriculares de TCC I e TCC II no sistema acadêmico.

A sessão foi encerrada às dezoito horas e quarenta minutos. Para constar, eu, Luiz Guilherme Hilel Drumond, redigi a presente ata que após lida publicamente, foi aprovada e assinada pelos membros da banca examinadora.

Sabará, 09 de fevereiro de 2026.

Documento assinado eletronicamente por **Luiz Guilherme Hilel Drumond Silveira**, **Professor**, em 10/02/2026, às 18:30, conforme Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Daniel Neves Rocha**, **Professor EBTT**, em 10/02/2026, às 20:42, conforme Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Moises Henrique Ramos Pereira**, **Professor**, em 02/03/2026, às 13:03, conforme Decreto nº 10.543, de 13 de novembro de 2020.

A autenticidade do documento pode ser conferida no site https://sei.ifmg.edu.br/consultadocs informando o código verificador **2617960** e o código CRC **7BABCC82**.

23714.001466/2025-48	2617960v1

Dedico este trabalho de conclusão de curso a meu pai, Walter Manoel Rodrigues dos Santos, aquele que sempre apoiou minha educação e que estaria orgulhoso das minhas conquistas.

AGRADECIMENTOS

Agradeço a minha família pela paciência, aos amigos pelo suporte e a todos os docentes do curso de Engenharia de Controle e Automação pela dedicação e compromisso com os alunos. Agradeço ao Prof. Luiz Guilherme Hilel Drumond Silveira pela orientação dedicada.

“O sucesso é a soma de pequenos esforços repetidos dia após dia.”
Robert Collier

RESUMO

A Manutenção Preditiva (PdM) representa uma evolução estratégica na gestão de ativos industriais, utilizando Inteligência Artificial (IA) para antecipar falhas em equipamentos críticos. Este trabalho tem como objetivo desenvolver uma solução computacional para o diagnóstico automatizado de falhas em motores de indução trifásicos, utilizando Redes Neurais Artificiais (RNA). A metodologia proposta baseia-se na análise de sinais de vibração e corrente da base de dados pública do *Motor Fault Simulator* (MFS) da UFRJ. Para lidar com o grande volume de dados, foi implementada uma rotina de processamento em lotes para a extração de características relevantes nos domínios do tempo e da frequência. O principal resultado desta fase do projeto é a construção e validação de um *pipeline* de dados robusto e de uma arquitetura de rede neural.

Palavras-chave: Diagnóstico de Falhas. Aprendizado de Máquina. Processamento de Sinais. Indústria 4.0. Análise de Vibração.

ABSTRACT

Predictive Maintenance (PdM) represents a strategic evolution in industrial asset management, using Artificial Intelligence (AI) to anticipate failures in critical equipment. This work aims to develop a computational solution for the automated fault diagnosis of three-phase induction motors, using Artificial Neural Networks (ANN). The proposed methodology is based on the analysis of vibration and current signals from the public Motor Fault Simulator (MFS) dataset from UFRJ. To handle the large volume of data, a batch processing routine was implemented for the extraction of relevant features in the time and frequency domains. The main result of this project phase is the construction and validation of a robust data pipeline and an MLP-type neural network architecture,

Keywords: Fault Diagnosis. Machine Learning. Signal Processing. Industry 4.0. Vibration Analysis.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ABNT	Associação Brasileira de Normas Técnicas
SEP	Sistema
IFMG	Instituto Federal de Minas Gerais
ANN	- *Artificial Neural Networks* (Redes Neurais Artificiais)
CNN	- *Convolutional Neural Networks* (Redes Neurais Convolucionais)
FFT	- *Fast Fourier Transform* (Transformada Rápida de Fourier)
FTIR	- *Fourier-Transform Infrared Spectroscopy* (Espectroscopia de Infravermelho por Transformada de Fourier)
IA	- Inteligência Artificial
IDE	- *Integrated Development Environment* (Ambiente de Desenvolvimento Integrado)
IoT	- *Internet of Things* (Internet das Coisas)
LAVI	- Laboratório de Engenharia Acústica e Vibrações
MFS	- *Motor Fault Simulator*
ML	- *Machine Learning* (Aprendizado de Máquina)
MLP	- *Multi-Layer Perceptron* (Perceptron de Múltiplas Camadas)
PCA	- *Principal Component Analysis* (Análise de Componentes Principais)
PdM	- Manutenção Preditiva
PNN	- *Probabilistic Neural Networks* (Redes Neurais Probabilísticas)
RNA	- Redes Neurais Artificiais
RNN	- *Recurrent Neural Networks* (Redes Neurais Recorrentes)
RUL	- *Remaining Useful Life* (Vida Útil Remanescente)
SNV	- *Standard Normal Variate*
SVM	- *Support Vector Machines* (Máquinas de Vetores de Suporte)
UFRJ	- Universidade Federal do Rio de Janeiro

SUMÁRIO

1 INTRODUÇÃO

A evolução dos processos industriais, desde a Primeira Revolução Industrial até a era da automação, tem sido marcada por um aumento contínuo na complexidade dos equipamentos e sistemas de produção. Historicamente, a engenharia de confiabilidade ganhou destaque a partir da Segunda Guerra Mundial, quando a necessidade de garantir a eficiência de equipamentos bélicos impulsionou o desenvolvimento de métodos estatísticos para prever falhas (SOUZA; SILVA, 2024). Contudo, por décadas, a abordagem predominante para a manutenção industrial foi a reativa, popularmente conhecida como "quebra-conserta", que aguarda a falha ocorrer para então intervir (ABBAS, 2024). Posteriormente, a manutenção preventiva, baseada em cronogramas fixos, tornou-se padrão, mas frequentemente resulta em trocas de componentes prematuras ou falhas inesperadas entre os intervalos de manutenção (LEE *et al.*, 2019). O advento da Indústria 4.0, com suas tecnologias de Internet das Coisas (IoT), *Big Data* e Inteligência Artificial (IA), proporcionou a transição para uma nova era: a da Manutenção Preditiva (PdM), uma estratégia proativa que visa antecipar falhas por meio da análise contínua de dados (UCAR; KARAKOSE; Kirimça, 2024).

A Manutenção Preditiva (PdM) se baseia na premissa de que é possível prever falhas em equipamentos industriais com base em dados de sensores e históricos de manutenção (SARAN; SARAN; FRANZOTTI, 2024). A proliferação de sensores em ambientes industriais gera um volume massivo de dados (*Big Data*) sobre vibração, temperatura, corrente, pressão, entre outros. A análise e interpretação desses dados complexos e multivariados excedem a capacidade humana, tornando a Inteligência Artificial, e em particular o Aprendizado de Máquina (*Machine Learning - ML*), ferramentas essenciais para implementar a PdM de forma eficaz (OHALETE *et al.*, 2023). A IA permite a criação de modelos robustos capazes de identificar padrões e anomalias que indicam falhas iminentes, possibilitando a passagem de uma abordagem reativa para uma estratégia proativa na gestão de ativos e maximizando a confiabilidade dos sistemas (SOUZA; SILVA, 2024).

A aplicação de modelos de IA, em especial as Redes Neurais Artificiais (RNA), tem-se mostrado extremamente promissora. As RNAs, com sua arquitetura inspirada no cérebro humano, são capazes de aprender com exemplos e modelar relações não lineares complexas entre diferentes variáveis de um sistema, característica essencial para o diagnóstico de falhas em equipamentos dinâmicos (MEOLA, 2005). Estudos demonstram o potencial das Redes Neurais Artificiais (RNAs) em diversas aplicações de Manutenção Preditiva (PdM), desde o uso de Redes Neurais Convolucionais (CNN) e Recorrentes (RNN) para análise de sinais de vibração com alta acurácia (LEE *et al.*, 2019). Além disso, também se destacam as redes neurais de estrutura densa, que são capazes de generalizar padrões a partir de dados de alta dimensão, como os obtidos na análise de óleo (BARBOSA, 2023).

Neste contexto, os motores elétricos de indução trifásicos representam um objeto de estudo de grande relevância, dada sua onipresença e criticidade nos processos industriais. As falhas

nesses equipamentos, frequentemente associadas a defeitos em rolamentos, desbalanceamento de tensão, desalinhamento de eixos ou barras do rotor quebradas, geram sinais complexos e de difícil interpretação. Para investigar esses fenômenos de forma controlada, este trabalho utilizará a base de dados pública do *Motor Fault Simulator* (MFS), desenvolvida pelo Laboratório de Engenharia Acústica e Vibrações (LAVI) da Universidade Federal do Rio de Janeiro (UFRJ). A base de dados, com volume aproximado de 13 GB, é composta por sinais de vibração (obtidos por acelerômetros), corrente trifásica e rotação, registrados em diferentes condições operacionais e sob modos de falha induzidos. Esse conjunto de dados oferece um cenário complexo e representativo para o treinamento e a validação de modelos de inteligência artificial (Laboratório de Engenharia Acústica e Vibrações,). A hipótese central deste trabalho é que a aplicação de Redes Neurais Artificiais aos dados multivariados do MFS é um método que possibilita a classificação e predição de modos de falha em motores de indução, permitindo um diagnóstico automatizado com alta precisão.

A aplicação de técnicas de IA para o diagnóstico de falhas em motores elétricos, como o monitoramento em tempo real de sinais de vibração (MEOLA, 2005), permite uma análise mais precisa e antecipada do estado da máquina. Ao desenvolver um sistema computacional capaz de automatizar esse diagnóstico, busca-se oferecer uma ferramenta de apoio à decisão para as equipes de manutenção, permitindo que intervenções sejam realizadas no momento ideal, evitando falhas catastróficas e otimizando a vida útil do equipamento. Portanto, a relevância deste estudo reside na sua contribuição prática para a solução de um problema real da indústria, ao empregar análise de dados de sensores e modelos de inteligência artificial para apoiar a manutenção preditiva.

1.1 Objetivos

1.1.1 *Objetivo geral*

O objetivo deste trabalho é desenvolver e avaliar um modelo de rede neural artificial para a análise dos dados do MFS/UFRJ e a geração de diagnósticos de manutenção e potenciais falhas em motores elétricos. Utilizando técnicas de monitoramento e análise para prever falhas em equipamentos antes que elas ocorram, permitindo intervenções planejadas e minimizando paradas não programadas.

1.1.2 *Objetivos específicos*

- Definir a estratégia de rotulagem e organização do conjunto de dados MFS-UFRJ para um problema de classificação multiclasse (modo de falha e severidade);
- Implementar uma RNA em Python com apoio das bibliotecas Pandas (versão 2.1.4), NumPy (versão 1.26.3), SciPy (versão 1.11.4), Scikit-learn (versão 1.3.2) e TensorFlow (versão 2.12.0);

- Avaliar o desempenho do modelo com métricas adequadas a cenário multiclasse e potencialmente desbalanceado (acurácia, macro-F1, matriz de confusão e métricas por classe).

1.2 Justificativa

A necessidade de reduzir custos operacionais e paradas de produção não planejadas na indústria moderna impulsiona a busca por métodos de manutenção mais inteligentes e eficazes. Diante deste cenário, este trabalho parte da hipótese de que a aplicação de Redes Neurais Artificiais (RNA) é uma abordagem robusta e precisa para a classificação e predição de modos de falha em motores de indução, a partir da análise de seus sinais operacionais de vibração e corrente. A justificativa para esta hipótese fundamenta-se no sucesso comprovado de metodologias similares em pesquisas recentes e diretamente relacionadas ao tema.

A adoção desta abordagem é justificada por estudos prévios que obtiveram resultados satisfatórios no diagnóstico de falhas em motores de indução com redes neurais e dados de monitoramento. Moura Filho (2023) e Moura Filho et al. (2021) desenvolveram modelos de diagnóstico de falhas em máquinas elétricas rotativas utilizando técnicas de inteligência artificial aplicadas a bases públicas de falhas, demonstrando altas taxas de acerto e reforçando o potencial da IA para PdM em motores de indução. Seus resultados demonstraram alta eficácia na classificação de diferentes condições operacionais, validando o uso de redes neurais para este fim específico. De forma similar, Lee et al. (2019) aplicaram com sucesso arquiteturas de redes neurais para o diagnóstico de defeitos em rolamentos de máquinas rotativas, alcançando uma acurácia de 98% na identificação de falhas e reforçando o alto potencial preditivo da técnica.

Outro ponto importante é a melhoria na segurança. Ao prever falhas antes que elas ocorram, é possível evitar acidentes e garantir um ambiente de trabalho mais seguro para os funcionários. A segurança é uma prioridade em qualquer indústria, e a manutenção preditiva com IA contribui diretamente para isso.

1.3 Organização do Texto

Este trabalho está organizado da seguinte forma: No primeiro capítulo apresento a introdução do tema e a descrição geral do assunto. O segundo capítulo traz a revisão bibliográfica com as referências de outros autores sobre o tema abordado. No terceiro capítulo apresento a metodologia utilizada para o desenvolvimento do assunto. Os resultados obtidos durante o trabalho serão apresentados no quarto capítulo. As conclusões serão discutidas no capítulo 5.

2 REVISÃO BIBLIOGRÁFICA

Este capítulo apresenta a fundamentação teórica para o desenvolvimento de um sistema de manutenção preditiva para motores elétricos utilizando Inteligência Artificial. A revisão da literatura aborda a evolução histórica das estratégias de manutenção, os princípios da manutenção preditiva no contexto da Indústria 4.0, as técnicas de monitoramento e coleta de dados, os algoritmos de Aprendizado de Máquina aplicados ao diagnóstico de falhas e, por fim, sintetiza os achados para contextualizar a contribuição deste trabalho.

A trajetória das práticas de manutenção na indústria reflete uma evolução contínua na busca por eficiência, confiabilidade e otimização de custos. Esta seção aprofunda a análise dessa evolução, dissecando os paradigmas filosóficos e tecnológicos que definem cada era, desde a simples reação a falhas até a predição inteligente de eventos futuros. A compreensão dessa jornada é essencial para contextualizar a relevância e a disrupção causada pela aplicação de Inteligência Artificial na manutenção moderna.

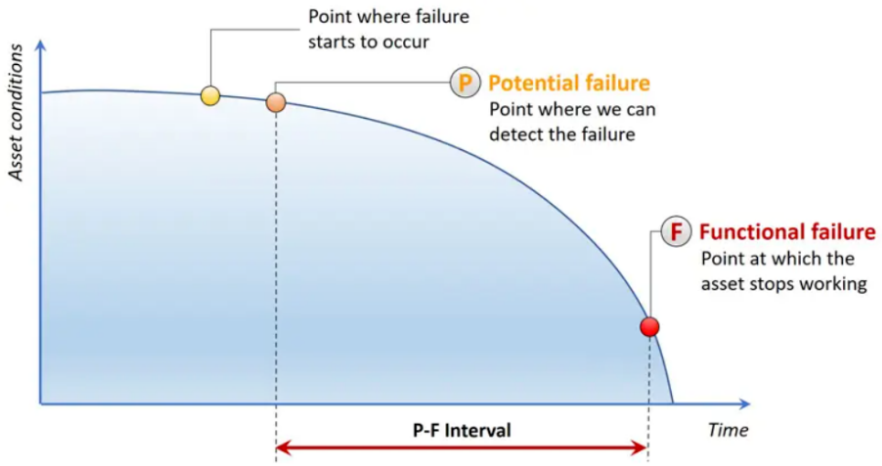
Nos estágios incipientes da industrialização, a manutenção era praticada quase exclusivamente sob o paradigma corretivo, também conhecido como "*run-to-failure*"(operação até a falha). A filosofia subjacente era puramente reativa: um equipamento operava até sua quebra, e somente então as equipes de manutenção eram acionadas para realizar o reparo (ABBAS, 2024; SARAN; SARAN; FRANZOTTI, 2024). Esta abordagem, embora simples, trazia consigo um alto grau de imprevisibilidade, resultando em paradas de produção não planejadas que geravam custos diretos (peças e mão de obra emergencial) e indiretos (perda de produção, atrasos na entrega) extremamente elevados (SARAN; SARAN; FRANZOTTI, 2024). A manutenção corretiva trata o sintoma, a falha funcional, e não a causa raiz da degradação, operando em um ciclo constante de emergências que impedia o planejamento estratégico e a otimização da vida útil dos ativos.

A crescente complexidade dos sistemas produtivos e o custo proibitivo das paradas inesperadas impulsionaram a transição para a manutenção preventiva. Este paradigma representa uma mudança filosófica fundamental: da reação para a proatividade. A manutenção preventiva opera com base em intervenções agendadas, realizadas em intervalos fixos de tempo ou de acordo com um contador de uso (horas de operação, ciclos, quilometragem), independentemente da condição real do componente (ABBAS, 2024). Embora esta abordagem tenha sido um avanço significativo na mitigação de falhas catastróficas, sua eficácia é limitada por uma ineficiência intrínseca. Frequentemente, componentes em perfeito estado de funcionamento são substituídos prematuramente, desperdiçando sua vida útil remanescente, enquanto falhas aleatórias que ocorrem entre os intervalos de manutenção programada não são evitadas, gerando uma falsa sensação de segurança (LEE *et al.*, 2019).

A manutenção preditiva (PdM) surge como a evolução natural da preventiva, substituindo a programação baseada no tempo pela manutenção baseada na condição real do ativo (*Condition-Based Maintenance*). O seu fundamento teórico reside no monitoramento contínuo de parâmetros operacionais — como vibração, temperatura, corrente elétrica e análise de óleo — para detectar

sinais de degradação em estágios iniciais (BARBOSA, 2023). O conceito da curva P-F é central para a PdM, pois descreve a trajetória de um componente desde o ponto em que uma falha potencial (P) se torna detectável até o ponto de sua falha funcional (F). O objetivo da PdM é identificar o ponto P o mais cedo possível, maximizando o intervalo P-F e permitindo que a manutenção seja planejada e executada de forma otimizada, sem interrupções emergenciais e com o máximo aproveitamento da vida útil do componente (BARBOSA, 2023).

Figura 1 – Representação da Curva P-F na Manutenção Preditiva.



Fonte: Adaptado de Barbosa (2023, p. 18).

A implementação da PdM no contexto da Indústria 4.0 é viabilizada por um ecossistema tecnológico que inclui sensores, Internet das Coisas (IoT) e Inteligência Artificial (IA) para a análise automatizada de dados e identificação de padrões associados a falhas. O volume, a variedade e a velocidade dos dados gerados pelo monitoramento contínuo tornam a análise manual impraticável. É nesse ponto que algoritmos de Aprendizado de Máquina, como as Redes Neurais Artificiais (RNA), se tornam indispensáveis, pois são capazes de identificar padrões complexos e não lineares nos sinais que são imperceptíveis a um analista humano, correlacionando-os com modos de falha específicos (FILHO, 2023). O sucesso dessa abordagem é validado por estudos de caso que demonstram retornos sobre o investimento significativos, como a prevenção de falhas catastróficas em equipamentos de mineração que resultaram em economia de centenas de milhares de reais e evitaram dias de parada de produção (SOUZA; SILVA, 2024). A PdM, portanto, não é apenas uma técnica, mas uma filosofia de gestão de ativos que transforma dados em decisões estratégicas, alinhando a manutenção diretamente aos objetivos de negócio da organização.

Além dessas abordagens, Moura Filho et al. (2021) investigaram o diagnóstico de falhas em máquinas elétricas rotativas utilizando técnicas de ensemble learning (Random Forest e Gradient Boosting), aplicadas à base Mafaulda, obtendo acurácia superior a 99

A implementação de um sistema de PdM eficaz depende da coleta de dados de alta qualidade que reflitam o estado de saúde do equipamento. Para motores elétricos e sistemas

rotativos, diversas técnicas de monitoramento são empregadas:

- Análise de Vibração: É uma das técnicas mais poderosas para o diagnóstico de falhas mecânicas como desbalanceamento, desalinhamento, folgas e, principalmente, defeitos em rolamentos. A análise espectral dos sinais de vibração, por meio da Transformada Rápida de *Fourier* (FFT), e a análise de envelope são essenciais para identificar as frequências características de cada modo de falha (MEOLA, 2005). Lee et al. (2019) demonstraram que o uso de dados no domínio da frequência pode levar a uma precisão de diagnóstico superior à da análise no domínio do tempo.
- Análise de Óleo Lubrificante: Em sistemas lubrificados, a análise físico-química do óleo pode revelar o desgaste de componentes internos e a presença de contaminantes. Técnicas como a Espectroscopia de Infravermelho por Transformada de *Fourier* (FTIR) são utilizadas para monitorar a degradação do óleo e o consumo de aditivos (BARBOSA, 2023).
- Monitoramento de Temperatura e Corrente: Sensores de temperatura podem indicar sobreaquecimento por atrito ou problemas de refrigeração. Já a análise da corrente elétrica do motor pode detectar anomalias relacionadas a falhas elétricas ou sobrecargas mecânicas (LEE *et al.*, 2019).

A qualidade e a disponibilidade de dados rotulados, contudo, permanecem como um dos maiores desafios para a pesquisa e aplicação de IA na PdM (KLEIN; BERGMANN, 2019). O pré-processamento dos dados, que inclui limpeza, normalização e extração de características (*feature engineering*), é uma etapa crucial que impacta diretamente o desempenho dos modelos preditivos (ABBAS, 2024).

A aplicação da Inteligência Artificial (IA), e mais especificamente do Aprendizado de Máquina (ML), oferece uma solução robusta para os desafios impostos pela análise de grandes volumes de dados na manutenção preditiva (FILHO, 2023). O Aprendizado de Máquina é uma subárea da IA que se concentra no desenvolvimento de sistemas capazes de aprender diretamente a partir dos dados (FERNANDES, 2024). No contexto do diagnóstico de falhas, um modelo de ML é treinado com um conjunto de dados de um equipamento com estados operacionais já conhecidos, aprendendo a mapear os padrões contidos nos sinais para as condições correspondentes da máquina (FILHO, 2023). Uma vez que o modelo esteja treinado, ele pode ser utilizado para analisar novos dados operacionais e prever o estado atual do equipamento (FERNANDES, 2024). A principal vantagem desta abordagem é a capacidade do ML de identificar padrões sutis nos dados que seriam difíceis de detectar por meio de métodos convencionais (FILHO, 2023). Uma vertente avançada do ML é o Aprendizado Profundo (*Deep Learning*), baseado em redes neurais artificiais e considerado uma tecnologia central na Quarta Revolução Industrial (COSTA; SOUZA; RODRIGUES, 2022). Essa tecnologia se destaca pela sua capacidade de extrair características complexas de forma automatizada para o reconhecimento de padrões (SILVA; SILVA; SANTOS, 2022).

Estudos anteriores demonstram a eficácia de diversas abordagens de ML. Lee et al. (2019) aplicaram *Support Vector Machines* (SVM), Redes Neurais Recorrentes (RNN) e Redes Neurais Convolucionais (CNN) para o monitoramento de sistemas de máquinas-ferramenta, alcançando uma acurácia de 98% na detecção de falhas de rolamento com CNNs. Meola (2005), em um estudo pioneiro com um sistema de motor elétrico, utilizou Redes Neurais Probabilísticas (PNN) e Lógica *Fuzzy* para classificar a qualidade de sinais de vibração, obtendo uma precisão próxima de 100% e detectando com sucesso uma falha na gaiola de um rolamento. A aplicação dessas técnicas em cenários industriais reais tem gerado resultados expressivos. Souza e Silva (2024) relatam um caso na indústria de mineração onde um sistema de IA, analisando dados de vibração e temperatura, previu a falha de um redutor, evitando uma parada de 18 dias e gerando uma economia de aproximadamente R$ 673.000,00. Similarmente, Barbosa (2023) desenvolveu um modelo de ML que, a partir da análise de óleo, classificou a necessidade de manutenção de turbinas eólicas com alta precisão, provando ser uma valiosa ferramenta de apoio à decisão para as equipes de manutenção. Esses resultados evidenciam que, independentemente do setor — seja óleo e gás (OHALETE *et al.*, 2023), mineração ou energia eólica —, a IA se consolidou como uma tecnologia essencial para a otimização da manutenção.

A eficácia de um modelo de Rede Neural Artificial (RNA) depende não apenas de sua arquitetura, mas também da escolha adequada de seus componentes internos e do método de treinamento. Esta seção detalha os conceitos teóricos por trás dos principais elementos utilizados na construção e otimização do modelo proposto neste trabalho, como as funções de ativação, o otimizador e a função de perda.

As funções de ativação são um componente essencial dos neurônios artificiais, responsáveis por introduzir não-linearidade na rede e permitir que ela aprenda relações complexas nos dados (BARBOSA, 2023). Nas camadas ocultas, uma das funções mais utilizadas é a ReLU (*Rectified Linear Unit*). A ReLU é definida matematicamente como f(x) = max(0, x), o que significa que ela retorna o próprio valor de entrada se ele for positivo e zero caso contrário. Essa simplicidade computacional a torna muito eficiente, além de ajudar a mitigar o problema da dissipação de gradiente (*vanishing gradient*) durante o treinamento (FILHO, 2023).

Na camada de saída de um problema de classificação multiclasse, a função de ativação Softmax é a escolha padrão. Sua função é transformar os valores de saída brutos da rede (conhecidos como *logits*) em uma distribuição de probabilidade, onde cada saída representa a probabilidade de a entrada pertencer a uma das classes. O resultado é um vetor de probabilidades cuja soma é igual a 1, facilitando a interpretação do resultado e a seleção da classe mais provável (FILHO, 2023).

O processo de treinamento de uma rede neural consiste em ajustar seus pesos para minimizar uma função de perda, e o algoritmo que guia esse ajuste é chamado de otimizador. O otimizador Adam (*Adaptive Moment Estimation*) é um dos mais utilizados devido à sua eficiência e bom desempenho em uma vasta gama de problemas. Ele é um algoritmo de otimização adaptativo

que combina as vantagens de outros dois otimizadores, o AdaGrad e o RMSProp, ajustando a taxa de aprendizado para cada peso da rede de forma individual e dinâmica ao longo do treinamento (FILHO, 2023).

Por fim, a função de perda (ou função de custo) quantifica o erro do modelo durante o treinamento. Para problemas de classificação multiclasse, a função de perda adequada é a Entropia Cruzada Categórica (*categorical crossentropy*). Ela mede a dissimilaridade entre a distribuição de probabilidade prevista pela camada Softmax e a distribuição de probabilidade real dos dados (onde a classe correta tem probabilidade 1 e as outras, 0). O objetivo do otimizador Adam é, portanto, ajustar os pesos da rede para minimizar o valor desta função de perda (FILHO, 2023).

3 METODOLOGIA

Este capítulo descreve os procedimentos metodológicos adotados para o desenvolvimento da solução computacional de diagnóstico de falhas em motores elétricos. A abordagem detalha a caracterização da pesquisa, a fonte de dados utilizada, as ferramentas de software empregadas e as etapas sequenciais para o tratamento dos dados, construção, treinamento e avaliação do modelo de Inteligência Artificial.

A presente pesquisa classifica-se como de natureza quantitativa e aplicada. É quantitativa, pois se baseia na análise de dados numéricos (sinais de vibração e corrente) para testar uma hipótese por meio de modelos matemáticos e estatísticos. É aplicada, pois visa à geração de um conhecimento prático para a solução de um problema específico da indústria: o diagnóstico automatizado de falhas em motores elétricos. O delineamento da pesquisa pode ser considerado experimental, uma vez que envolve a manipulação de dados em um ambiente computacional controlado para avaliar o desempenho de um modelo de Rede Neural Artificial.

Para a realização dos experimentos, será utilizada a base de dados pública do *Motor Fault Simulator* (MFS), disponibilizada pelo Laboratório de Engenharia Acústica e Vibrações (LAVI) da Universidade Federal do Rio de Janeiro (UFRJ)(Laboratório de Engenharia Acústica e Vibrações,). A escolha desta base de dados se justifica por sua abrangência e adequação aos objetivos do trabalho, sendo uma referência em estudos de diagnóstico de falhas (Laboratório de Engenharia Acústica e Vibrações,).Essa base também foi utilizada em estudos de diagnóstico de falhas em máquinas elétricas rotativas com técnicas de *ensemble learning*, como em Moura Filho et al. (2021), o que reforça sua pertinência para a avaliação de modelos de IA nesta aplicação.

A base de dados inclui sinais de vibração e de corrente elétrica trifásica, coletados sob múltiplos modos de falha induzidos, como desbalanceamento, desalinhamento, defeitos em rolamentos e barras do rotor quebradas. Esta variedade de cenários é fundamental para treinar e validar um modelo de diagnóstico e com capacidade de generalização para diferentes problemas operacionais.

A implementação do código foi desenvolvida integralmente na linguagem Python, versão 3.10. Para garantir a organização e a reprodutibilidade dos experimentos, utilizou-se o ambiente de desenvolvimento integrado (IDE) *Visual Studio Code* em conjunto com notebooks Jupyter. As principais bibliotecas de software empregadas no projeto e suas respectivas versões foram:

- Pandas (versão 2.1.4): Para a estruturação e manipulação dos dados tabulares.
- NumPy (versão 1.26.3): Para operações numéricas e manipulação de arrays, fundamental no processamento dos sinais.
- SciPy (versão 1.11.4): Utilizada para a leitura dos arquivos de dados e para a aplicação da Transformada Rápida de *Fourier* (FFT).

- Scikit-learn (versão 1.3.2): Para a divisão dos conjuntos de dados e para a avaliação das métricas de desempenho do modelo.
- TensorFlow (versão 2.12.0): *Framework* principal para a construção e treinamento do modelo de Rede Neural Artificial.

O desenvolvimento da solução computacional será realizado integralmente na linguagem de programação Python, dada sua consolidada aplicação em projetos de ciência de dados. As rotinas serão implementadas utilizando bibliotecas padrão da área, como Pandas para manipulação de dados estruturados, NumPy para operações numéricas e Scikit-learn para pré-processamento e avaliação (HARRISON, 2021). Para a construção da Rede Neural Artificial, será empregada a biblioteca TensorFlow com sua API de alto nível Keras.

Considerando o grande volume de dados (13 GB), será adotada uma estratégia de processamento em lotes (*batch processing*), na qual os arquivos são lidos e processados individualmente para extração de características(conforme descrito na etapa de feature engineering desta metodologia). Este método evita a sobrecarga de memória e torna o projeto viável em hardware convencional. O processo foi estruturado nas seguintes etapas:

Figura 2 – Fluxograma das etapas do sistema de diagnóstico.

Fonte: Elaborado pelo autor (2025).

A primeira etapa consiste no mapeamento e rotulagem automática dos arquivos da base de dados, atribuindo a cada registro um rótulo de classe correspondente à condição do motor (operação normal ou combinação de modo de falha e severidade). Esses rótulos são

obtidos a partir da nomenclatura do próprio conjunto MFS, presente nos nomes das pastas e subpastas (por exemplo, normal, ballfault-20g e misalignment-0.5mm), e são necessários para viabilizar o treinamento supervisionado do classificador multiclasse e a posterior avaliação por classe. As informações sobre o tipo de falha e severidade, contidas nos nomes das pastas (ex: *misallignament/0,5mm*), serão extraídas e associadas a cada arquivo, criando um *DataFrame* mestre com os caminhos e seus respectivos rótulos. Em seguida, um *loop* iterará sobre este mapa, carregando cada arquivo individualmente com a biblioteca *scipy.io* para a extração dos sinais.

Tabela 1 – Distribuição da quantidade de amostras por classe (rótulo).

Classe (rótulo)	**Quantidade**
imbalance-35g	45
imbalance-25g	47
imbalance-30g	47
imbalance-10g	48
imbalance-15g	48
horizontal-misalignment-2.0mm	49
horizontal-misalignment-1.5mm	49
horizontal-misalignment-1.0mm	49
imbalance-6g	49
imbalance-20g	49
normal	49
horizontal-misalignment-0.5mm	50
vertical-misalignment-1.90mm	50
vertical-misalignment-1.27mm	50
vertical-misalignment-1.40mm	50
vertical-misalignment-1.78mm	50
vertical-misalignment-0.63mm	50
ball_fault-35g	58
vertical-misalignment-0.51mm	51
ball_fault-20g	74
outer_race-35g	78
cage_fault-35g	83
ball_fault-6g	92
cage_fault-6g	97
cage_fault-20g	98
outer_race-6g	98
outer_race-0g	98
cage_fault-0g	98
outer_race-20g	98
ball_fault-0g	99

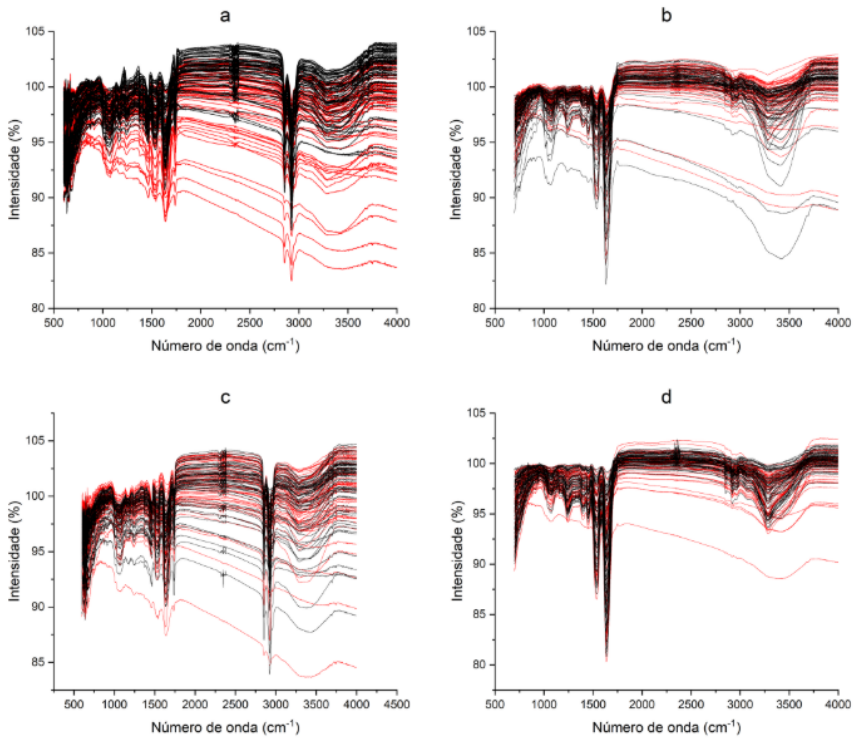
Fonte: Elaborado pelo autor (2026).

As classes consideradas no treinamento correspondem às combinações de modo de falha e severidade presentes na nomenclatura do dataset, totalizando 30 rótulos; a Tabela 1 apresenta a distribuição de amostras por classe utilizada neste estudo.

Após o carregamento, os sinais brutos de vibração e corrente passarão por um processo

de extração de características (*feature engineering*) para alimentar o modelo de RNA. A análise de sinais de corrente do estator, por exemplo, é uma abordagem eficaz para o diagnóstico de falhas em motores de indução, como demonstrado por (FILHO, 2023). Serão extraídas características estatísticas no domínio do tempo (como valor RMS, fator de crista e curtose) e características no domínio da frequência, obtidas a partir da aplicação da Transformada Rápida de Fourier (FFT), que permitem identificar componentes de frequência associadas a falhas específicas conforme abordagem adotada em trabalhos recentes de diagnóstico de falhas em máquinas rotativas, como Moura Filho et al. (2021), que empregam RMS como técnica de síntese de sinais de vibração. Este conjunto de características consolidadas formará a base de dados final para o treinamento. Ao final do processo de extração, cada arquivo (amostra) é representado por um vetor numérico de características; no presente trabalho, esse vetor possui 240 variáveis por amostra, que compõem a matriz de entrada XX utilizada no treinamento da rede neural, enquanto os rótulos correspondentes formam o vetor yy. Na Figura 3 apresenta-se um conjunto de espectros normalizados em diferentes intervalos de número de onda, no qual se observa a sobreposição das curvas em preto e vermelho representando amostras de condições distintas; essa visualização ilustra como a normalização organiza os sinais em faixas consistentes de intensidade, facilitando a identificação de padrões que posteriormente serão convertidos em características de entrada para a rede neural.

Figura 3 – Exemplo de espectros de sinais após normalização.

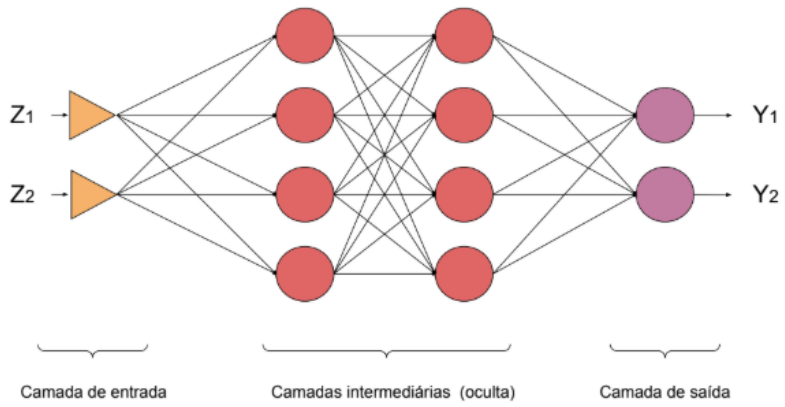


Fonte: Fernandes (2024, p. 15).

Para a tarefa de classificação, será implementado um modelo de Rede Neural Artificial

(RNA) do tipo Perceptron de Múltiplas Camadas (MLP), arquitetura que demonstrou alta eficácia em problemas similares de diagnóstico de falhas em motores de indução (FILHO, 2023).A Figura 4 ilustra de forma esquemática essa arquitetura, composta por uma camada de entrada que recebe o vetor de características extraídas dos sinais (Z1, Z2, ..., Zn), por camadas intermediárias ou ocultas totalmente conectadas, responsáveis por aprender combinações não lineares dessas entradas, e por uma camada de saída que produz as probabilidades associadas a cada classe de falha (Y1, Y2, ..., Ym). Essa representação evidencia o fluxo de informação do modelo proposto, desde os atributos gerados na etapa de pré-processamento até as saídas utilizadas para o diagnóstico automatizado das condições do motor.

Figura 4 – Exemplo de arquitetura de uma Rede Neural do tipo MLP.



Fonte: Adaptado de Moura Filho (2023, p. 25).

O modelo será composto por uma camada de entrada, camadas ocultas com função de ativação *ReLU* e uma camada de saída com função *Softmax*, configuração amplamente utilizada em problemas de classificação multiclasse. A escolha da *ReLU* nas camadas ocultas deve-se à sua simplicidade computacional e à capacidade de introduzir não linearidade sem agravar de forma significativa o problema de dissipação de gradiente, favorecendo o aprendizado de relações complexas entre as características extraídas dos sinais. Já a função *Softmax* na camada de saída é adequada porque transforma os valores produzidos pelos neurônios de saída em uma distribuição de probabilidades somando 1, permitindo interpretar cada componente como a probabilidade do motor pertencer a uma determinada classe de falha.

Para avaliar o impacto de escolhas arquiteturais e de estratégias de treinamento no desempenho do classificador, adotou-se um delineamento experimental comparativo, no qual diferentes configurações de rede neural foram treinadas e avaliadas sob condições controladas. Em todos os experimentos, manteve-se constante o pipeline de preparação dos dados (extração de características e normalização), bem como o particionamento estratificado dos conjuntos de treino e teste, de forma a isolar o efeito das alterações realizadas no modelo.

Nesse contexto, definiu-se um modelo base como referência, por representar a primeira

implementação estável do pipeline e permitir estabelecer um patamar inicial de desempenho. A partir desse *baseline*, foram propostas duas variações: (i) uma configuração intermediária, com ajustes de regularização e de estabilidade de treinamento, e (ii) uma configuração intensificada, com aumento de capacidade do modelo e estratégias de treinamento mais intensivas.

Tabela 2 – Comparação dos parâmetros entre o modelo base, configuração intermediária e configuração intensificada.

Parâmetro	**Modelo base (baseline)**	**Configuração intermediária**	**Configuração agressiva / intensificada**
Entrada	Vetor de 240 características (features) por amostra.	Idem ao modelo base (pipeline mantido constante).	Idem ao modelo base (pipeline mantido constante).
Nº de camadas densas ocultas	3 camadas densas (MLP).	3 camadas densas: 512, 256 e 128 neurônios.	5 camadas densas: 512, 256, 128, 64 e 32 neurônios.
Função de ativação	ReLU.	ReLU.	ReLU.
Camada de saída	Softmax (classificação multiclasse).	Softmax (classificação multiclasse).	Softmax (classificação multiclasse).
Normalização	StandardScaler na entrada (pré-processamento).	StandardScaler na entrada + Batch Normalization após camadas densas.	StandardScaler na entrada + Batch Normalization (todas as camadas densas).
Dropout	Não explicitado no baseline.	Aplicado dropout moderado (regularização).	Dropout nas três primeiras camadas (regularização).
Regularização L2	Não explicitado no baseline.	Penalização L2 nos pesos das camadas densas.	Estratégia descrita como intensificada; mantém uso de mecanismos de regularização (dropout + BN).
Otimizador	Adam.	Adam.	Adam (com controle de taxa de aprendizado).
Função de perda	Categorical crossentropy.	Categorical crossentropy.	Categorical crossentropy.
Controle de treino	Não explicitado no baseline.	Early stopping + ReduceLROnPlateau.	Early stopping + redução gradual da taxa de aprendizado.
Máximo de épocas	~292 (referência).	Não especificado no trecho.	Até 700 épocas (limite superior).
Balanceamento	Sem oversampling agressivo.	Oversampling das classes minoritárias (somente no treino).	Oversampling direcionado às classes com pior F1-score.
Objetivo do teste	Estabelecer um patamar inicial estável para comparação.	Reduzir overfitting e aumentar estabilidade.	Aumentar capacidade do modelo e melhorar macro-F1.

Fonte: Elaborado pelo autor (2026).

A motivação para essas variações foi investigar, de forma incremental, o compromisso

entre desempenho global (acurácia), equilíbrio entre classes (macro-F1) e robustez à distribuição desigual de amostras por classe, aspecto característico de bases de falhas em máquinas rotativas.

Com o objetivo de avaliar o efeito de diferentes estratégias de regularização e estabilização do treinamento sobre o desempenho do classificador, foram definidas configurações comparáveis de MLP, mantendo-se constante o *pipeline* de extração de características e a base de dados utilizada. Além do modelo base, foi implementada uma configuração intermediária, projetada para reduzir *overfitting* e aumentar a estabilidade do aprendizado por meio de mecanismos de regularização e controle de treinamento.

Tabela 3 – Configurações de treinamento por versão do modelo.

Versão	**Épocas (máx.)**	**Critério de parada**	**Batch size**	**Observações**
Base	500	Early Stopping (paciência 50)	64	Configuração de referência com extração de 1024 neurônios inicia.
Intermediária	260	Early Stopping (paciência 24)	96 ou 128	Uso de Dropout (0.10 a 0.15), L2 (5×10^{-5} a 10^{-4}) e Batch Normalization.
Intensificada	700	Early Stopping (paciência 80) + ReduceLROnPlateau	32	Arquitetura mais profunda (5 camadas), com **oversampling focado** para classes minoritárias.

Fonte: Elaborado pelo autor (2026).

Na configuração intermediária, empregou-se uma MLP com três camadas densas de 512, 256 e 128 neurônios, com função de ativação ReLU em todas as camadas ocultas e camada de saída *Softmax* para classificação multiclasse. Para estabilização das distribuições internas durante o treinamento, adicionou-se normalização em lote (*Batch Normalization*) após as camadas densas e aplicou-se *dropout* moderado como técnica de regularização. Adicionalmente, inseriu-se uma camada densa de 64 neurônios imediatamente antes da saída como uma etapa de refinamento da representação aprendida, permitindo que a rede recombine, em menor dimensionalidade, as ativações geradas pelas camadas anteriores antes da aplicação do Softmax. Essa escolha busca aumentar a separabilidade entre classes com assinaturas parcialmente semelhantes (por exemplo, mesmo modo de falha em diferentes severidades), sem elevar excessivamente a complexidade do modelo, uma vez que a camada adicional possui menor número de neurônios do que as camadas anteriores e atua como uma compressão gradual do espaço de características.

Para reforçar a regularização, aplicou-se penalização L2 nos pesos das camadas densas. O treinamento incorporou *Early Stopping*, monitorando a perda no conjunto de validação, com o intuito de interromper o ajuste quando não houvesse melhora consistente e preservar os melhores pesos obtidos. Também foi empregado um agendador de taxa de aprendizado (*ReduceLROnPlateau*), reduzindo a taxa de aprendizado quando a perda de validação atingisse platô, de modo a favorecer convergência mais estável nas etapas finais do treinamento.

O treinamento será realizado utilizando o otimizador Adam em conjunto com a função de perda *categorical_crossentropy*, combinação consolidada na literatura para tarefas de classificação multiclasse com redes neurais. O Adam foi escolhido por ajustar de forma adaptativa a taxa de aprendizado para cada peso da rede ao longo das épocas, o que tende a acelerar a convergência e a tornar o processo mais estável em bases de dados ruidosas e de alta dimensionalidade, como as de sinais de vibração e corrente. A função *categorical_crossentropy*, por sua vez, mede a discrepância entre a distribuição de probabilidades prevista pela camada *Softmax* e os rótulos verdadeiros codificados em *one-hot*, penalizando mais fortemente previsões com baixa probabilidade para a classe correta e, assim, direcionando o treinamento para a maximização da acurácia de classificação entre as diferentes condições de falha. O conjunto de dados será dividido em 80% para treino e 20% para teste, a fim de garantir uma avaliação da capacidade de generalização do modelo.

A divisão do conjunto de dados em 80% para treino e 20% para teste segue uma prática amplamente adotada em problemas de classificação com Redes Neurais Artificiais, pois busca equilibrar a necessidade de disponibilizar a maior parte das amostras para o aprendizado do modelo, preservando ao mesmo tempo uma fração representativa dos dados para avaliar seu desempenho em exemplos não vistos, reduzindo o risco de *overfitting* e permitindo estimar de forma mais confiável a capacidade de generalização (GHOLAMY; KREINOVICH; KOSHELEVA, 2018). Sob essa premissa, o estudo avaliou o desempenho de algoritmos de *ensemble learning* aplicados à base de dados MFS/MAFAULDA.

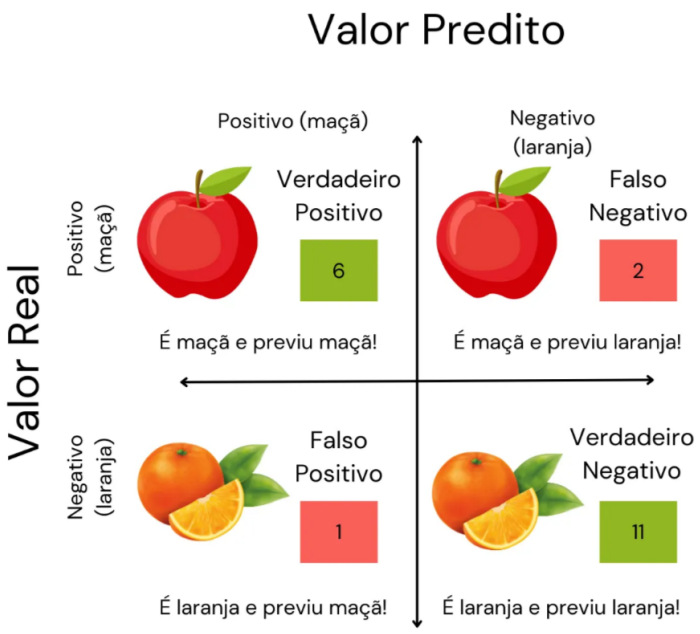
Em problemas de classificação multiclasse aplicados a falhas em máquinas, é comum que a base de dados apresente desbalanceamento de classes, isto é, algumas condições de falha possuírem significativamente mais amostras do que outras, o que tende a induzir o classificador a privilegiar as classes majoritárias e reduzir a sensibilidade para falhas raras. Para mitigar esse efeito, adotou-se uma estratégia de balanceamento no conjunto de treinamento por *oversampling*, definida como a replicação (reamostragem com reposição) de amostras das classes minoritárias até que se alcance uma distribuição mais uniforme entre as classes.

O *oversampling* foi aplicado exclusivamente no conjunto de treino, preservando-se o conjunto de teste (e validação, quando aplicável) como amostras não vistas e com distribuição original, a fim de evitar vazamento de informação e manter a avaliação comparável entre as versões do modelo. Nesta pesquisa, empregou-se *oversampling* uniforme das classes minoritárias, visando reduzir viés do modelo para classes com maior suporte e favorecer uma avaliação mais equilibrada por classe, especialmente ao considerar métricas como macro-F1.

A validação final do modelo será realizada sobre o conjunto de teste, que não participa do treinamento. O desempenho será quantificado por meio de métricas padrão para problemas de classificação, conforme utilizado em trabalhos de referência (FILHO, 2023). As métricas incluem a acurácia geral, a matriz de confusão para análise de erros por classe e os indicadores de precisão, *recall* e *F1-score*, que oferecem uma visão detalhada de desempenho do classificador

para cada tipo de falha. A análise dessas métricas permitirá validar a hipótese do trabalho e aferir a eficácia da solução proposta.

Figura 5 – Exemplo de Matriz de Confusão para avaliação de um classificador.



Fonte: Fernandes (2024, p. 17).

A Figura 5 apresenta um exemplo ilustrativo de matriz de confusão para um classificador binário, ferramenta fundamental para avaliar o desempenho do modelo de RNA no diagnóstico de falhas em motores. Nela, os valores nas células representam a contagem de previsões: os verdes (6 e 11) indicam acertos (Verdadeiro Positivo e Verdadeiro Negativo, respectivamente), enquanto os vermelhos (1 e 2) correspondem aos erros (Falso Positivo e Falso Negativo). A partir dessa estrutura, derivam-se as métricas de avaliação: a precisão (6/(6+1) = TP/(TP+FP)) mede a proporção de previsões positivas corretas entre todas as positivas feitas pelo modelo; o recall (6/(6+2) = TP/(TP+FN)) indica a proporção de casos positivos reais que foram corretamente identificados; e o *F1-score*, média harmônica entre precisão e recall (2×(Precisão×*Recall*)/(Precisão+*Recall*), oferece uma métrica equilibrada especialmente útil em bases desbalanceadas, como as de falhas em máquinas rotativas.

4 RESULTADOS

Neste capítulo são apresentados os resultados obtidos a partir da implementação do pipeline computacional proposto para diagnóstico de falhas em motores de indução, utilizando dados do *Motor Fault Simulator* (MFS-UFRJ). Inicialmente, descrevem-se as etapas de preparação dos dados, desde a organização e rotulagem dos arquivos até a formação do conjunto final de entrada para a rede neural. Em seguida, são apresentados os critérios de separação dos dados para treinamento e teste e as configurações empregadas para iniciar o processo de treinamento do modelo.

A preparação dos dados teve início com a varredura da estrutura de diretórios do conjunto MFS, na qual os sinais previamente disponibilizados em arquivos no formato .csv foram identificados e associados aos respectivos rótulos de condição (normal e modos de falha), conforme a nomenclatura adotada na base. Essa etapa permitiu construir uma base tabular consolidada, reunindo caminho do arquivo, classe e contagem de amostras por categoria, de modo a viabilizar tanto a rastreabilidade do conjunto de dados quanto a análise da distribuição entre classes.

Na sequência, cada arquivo .csv foi carregado individualmente para extração de atributos (*features*) representativos dos sinais, mantendo a estratégia de processamento em lotes descrita na metodologia, com o objetivo de reduzir consumo de memória e garantir reprodutibilidade do processamento. As features foram organizadas em um vetor numérico por amostra e, posteriormente, reunidas em uma matriz de entrada XX, acompanhada do vetor de rótulos yy.

Tabela 4 – Exemplo da matriz de entrada X e do vetor de rótulos y.

ID amostra	**Classe (rótulo)**	**F1 (bruta)**	**F2 (bruta)**	**F3 (bruta)**	**F1 (norm.)**	**F2 (norm.)**	**F3 (norm.)**
T0	normal	-	-	-	-0,11	0,42	-0,05
T1	ballfault-20g	-	-	-	0,73	1,35	0,62
T2	imbalance-35g	-	-	-	-0,69	0,18	-0,54

Fonte: Elaborado pelo autor (2026).

Como etapa de padronização, aplicou-se normalização das variáveis de entrada com *StandardScaler*, reduzindo diferenças de escala entre atributos e favorecendo a estabilidade do treinamento do modelo.

Concluída a etapa de pré-processamento, o conjunto final foi dividido de forma estratificada em dados de treinamento e teste (80%/20%), preservando a proporção entre classes. A partir desse ponto, foram definidos os parâmetros necessários para iniciar o treinamento da Rede Neural Artificial do tipo MLP (arquitetura, função de perda, otimizador e *callbacks*), possibilitando a execução dos experimentos comparativos apresentados nas subseções seguintes, nos quais se analisam o desempenho global e o comportamento do modelo por classe.

Com a base consolidada e padronizada, definiu-se o modelo base como referência inicial para os experimentos. Esse modelo foi implementado a partir da arquitetura MLP descrita no

Capítulo 3 e recebeu como entrada o conjunto de atributos extraídos dos sinais, organizado em uma matriz de características com 240 variáveis por amostra. Nesta etapa, o objetivo foi estabelecer uma configuração estável do *pipeline* (extração, normalização e rotulagem) e gerar um conjunto de dados consistente para a etapa de treinamento e posterior comparação com versões ajustadas do modelo.

A Tabela 1 exemplifica, com três registros, como os dados foram estruturados ao final do pré-processamento: (i) cada amostra é associada a uma classe (rótulo), (ii) os atributos são inicialmente obtidos em valores brutos (antes da padronização) e (iii) na sequência são transformados para valores normalizados, garantindo comparabilidade entre variáveis e favorecendo a convergência do treinamento.

Tabela 5 – Exemplo de registros após extração, normalização e rotulagem.

ID amostra	**Classe (rótulo)**	**F1 (bruta)**	**F2 (bruta)**	**F3 (bruta)**	**F1 (norm.)**	**F2 (norm.)**	**F3 (norm.)**
T0	normal	12,84	0,37	5,19	-0,11	0,42	-0,05
T1	ball_fault-20g	19,02	0,81	7,44	0,73	1,35	0,62
T2	imbalance-35g	8,55	0,29	3,90	-0,69	0,18	-0,54

Fonte: Elaborado pelo autor (2026).

Nota: F1, F2 e F3 representam três atributos dentre o total de 240 variáveis de entrada, apresentados aqui apenas para fins de ilustração do processo de preparação do *dataset* para o treinamento.

Após a construção desse conjunto de dados, realizou-se a codificação dos rótulos e a divisão estratificada em 80% para treinamento e 20% para teste, preservando a distribuição entre classes e permitindo avaliar a capacidade de generalização do modelo em amostras não vistas durante o ajuste dos pesos. A partir desse ponto, o pipeline encontrava-se apto para o treinamento do modelo base, cujos resultados de desempenho são apresentados nas subseções seguintes por meio de curvas de aprendizado, matriz de confusão e métricas de classificação.

Nessa configuração, o modelo atingiu acurácia em torno de 88% no conjunto de teste, com boa capacidade de generalização global, utilizando uma única rede, sem *oversampling agressivo* e com um esquema de validação menos sofisticado do que o adotado nas versões posteriores; esse resultado foi tomado como referência inicial de desempenho.

Todas as versões do modelo seguiram o pipeline metodológico definido, o que garante comparabilidade e aderência às normas de trabalho científico.

A base utilizada foi sempre o *Motor Fault Simulator* (MFS) do LAVI/UFRJ, com sinais de vibração e corrente sob múltiplos modos de falha (desbalanceamento, desalinhamento, defeitos em rolamentos e barras de rotor), em coerência com a escolha metodológica justificada no texto (volume de dados, diversidade de modos de falha e relevância acadêmica).

Na sequência do experimento base, foram executadas as configurações intermediária

e intensificada, conforme delineamento experimental descrito na Metodologia, mantendo-se constantes os dados e o pré-processamento para garantir comparabilidade.

Adicionalmente, foi aplicado balanceamento de classes no conjunto de treino por *oversampling*, conforme descrito na Metodologia.

A acurácia representa a proporção total de classificações corretas no conjunto de teste, sendo adequada para descrever o desempenho global do classificador. Entretanto, em problemas multiclasse com desbalanceamento, a acurácia pode permanecer elevada mesmo quando algumas classes minoritárias apresentam desempenho baixo, pois as classes com maior número de amostras tendem a influenciar mais o resultado agregado. Por esse motivo, o macro-F1 é analisado em conjunto, pois corresponde à média do F1-*score* calculado por classe e, assim, evidencia se o modelo mantém desempenho homogêneo entre condições de falha.

Na comparação entre versões, a configuração intermediária apresenta redução da acurácia em relação ao *baseline*, o que indica aumento de erros globais após a introdução de regularização e mudanças no treinamento, uma vez que mecanismos como *dropout* e penalização L2 reduzem o ajuste excessivo aos padrões dominantes do conjunto de treino, e o *oversampling* altera a distribuição efetiva de exemplos durante o aprendizado, privilegiando classes minoritárias e aumentando o compromisso entre acerto global e desempenho balanceado por classe. Ainda assim, o macro-F1 permanece em patamar consistente para um cenário multiclasse porque essa métrica corresponde à média do F1-*score* calculado por classe, atribuindo o mesmo peso a cada rótulo e, portanto, sendo menos influenciada por classes majoritárias do que a acurácia global. Na configuração intermediária, a introdução de *oversampling* no conjunto de treinamento e de mecanismos de regularização (como *dropout* e penalização L2) tende a reduzir o viés do aprendizado em direção às classes com maior suporte e a estabilizar o ajuste do modelo, preservando simultaneamente precisão e *recall* em parte relevante das classes, o que sustenta o macro-F1 mesmo quando ocorre redução do acerto total. Dessa forma, a queda de acurácia pode refletir mais um aumento de erros em classes dominantes (que afetam fortemente o percentual de acerto) do que uma degradação generalizada do desempenho por classe, o que é coerente com a leitura conjunta de acurácia e macro-F1 adotada neste trabalho. Na versão intensificada, observa-se recuperação da acurácia para patamar próximo ao modelo base e aumento do macro-F1, o que indica melhora simultânea do desempenho global e do equilíbrio entre classes, aspecto desejável em manutenção preditiva por reduzir o risco de degradação do desempenho em falhas menos frequentes.

A versão intensificada representou um passo adicional na direção de uma arquitetura mais profunda e de um treinamento mais prolongado, preservando as mesmas etapas de pré-processamento e a mesma divisão treino/validação/teste do MFS. Nesta pesquisa, o termo “intensificada” refere-se ao aumento deliberado de complexidade e esforço de otimização, caracterizado pela ampliação da rede para cinco camadas densas (512, 256, 128, 64 e 32 neurônios), pela extensão do treinamento para até 700 épocas e pelo ajuste do oversampling apenas no conjunto

Quadro 1 – Comparativo de versões do modelo e leitura dos indicadores de desempenho.

Versão do modelo	**Acurácia (%)**	**Macro-F1**	**Leitura dos indicadores**
Base	88,0	—	Indica bom desempenho global na partição utilizada, servindo como baseline para comparação das alterações posteriores.
Intermediária	81,33	0,80	Mostra queda no desempenho global, porém macro-F1 indica desempenho médio por classe preservado; sugere que ajustes de treinamento alteraram o compromisso entre generalização e acerto global.
Intensificada	87,0	0,86	Recupera acurácia próxima ao baseline e melhora o macro-F1, indicando aumento do desempenho médio por classe e redução de assimetria entre classes.

Fonte: Elaborado pelo autor (2026).

de treinamento priorizando as classes que apresentaram menor F1-*score*, visando reduzir o desequilíbrio de desempenho entre classes. Com essa configuração, buscou-se elevar o macro-F1 e reduzir a quantidade de classes abaixo do limiar de desempenho definido (F1-*score* < 0,8), ao custo potencial de maior risco de sobreajuste, mitigado pelo uso de *Early Stopping* e redução gradual da taxa de aprendizado.

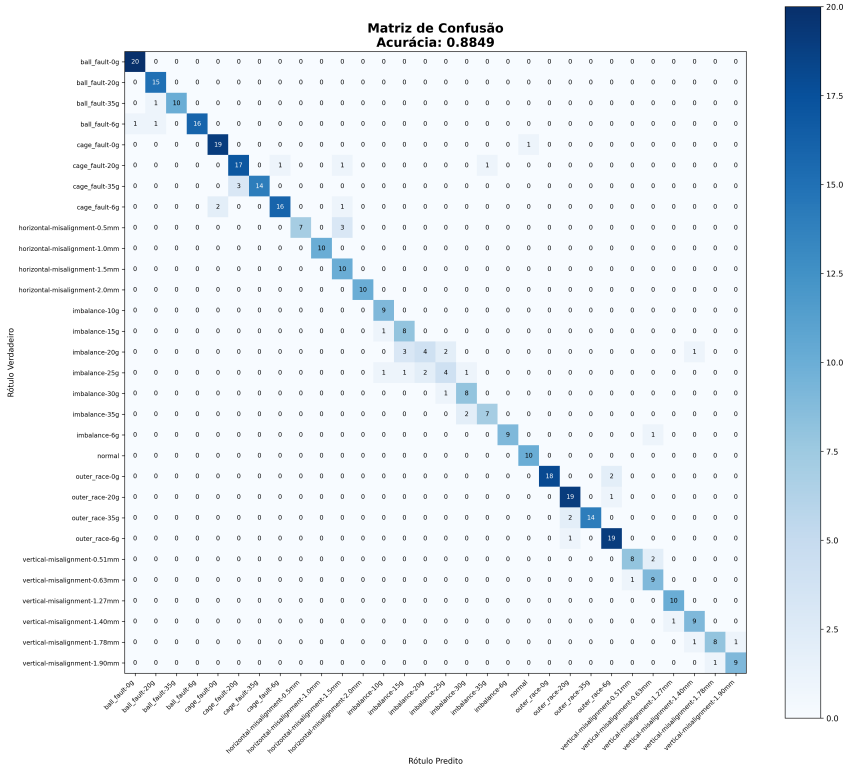
Nesse experimento, a rede foi ampliada para cinco camadas densas (512, 256, 128, 64 e 32 neurônios), todas com ativação ReLU e normalização em lote, com *dropout* aplicado nas três primeiras camadas,isto é, com a desativação aleatória de uma fração dos neurônios durante o treinamento, implementada no Keras pela camada *keras.layers.Dropout(rate)*, sendo *rate* o hiperparâmetro que define a fração de neurônios desativados, como forma de regularização para reduzir sobreajuste e melhorar a generalização do modelo. Assim aumentando a capacidade de modelagem de relações não lineares complexas entre as 30 classes de falha e severidade, uma vez que a rede passa a combinar, em múltiplos níveis, as características extraídas dos sinais para formar representações intermediárias mais discriminativas, necessárias quando diferentes classes apresentam assinaturas parcialmente sobrepostas no espaço de atributos.

O treinamento da versão intensificada foi configurado com limite de 700 épocas, valor maior do que o adotado na versão base e na intermediária, pois o aumento de profundidade da rede e a introdução de mecanismos de regularização (como *dropout* e normalização em lote) tendem a demandar mais iterações para estabilizar a convergência e ajustar os pesos sem comprometer a generalização. Nas versões anteriores, o treinamento encerrou-se em um número menor de épocas (por exemplo, 292 épocas na execução registrada), valor definido automaticamente pelo *Early Stopping* que interrompe o treinamento quando não há melhora na perda de validação dentro da paciência configurada. Isto reforça que a ampliação do limite buscou permitir que o processo de otimização explorasse um horizonte maior de ajuste, especialmente após reduções

de taxa de aprendizado ao longo do treino.

Para evitar que esse limite elevado resultasse em sobreajuste, foi aplicado *Early Stopping* monitorando a perda de validação, encerrando o treinamento quando não há melhora após um intervalo de paciência, e foi utilizada redução gradual da taxa de aprendizado por meio do *callback keras.callbacks.ReduceLROnPlateau*, configurado com *factor*=0.5 (redução multiplicativa da taxa) e *patience*=20, permitindo passos maiores no início (aprendizado rápido) e refinamento progressivo no final (ajustes finos), respeitando ainda o limite inferior min_lr=1e-7. Além disso, o *oversampling* foi direcionado principalmente às classes com pior F1-*score*, com o objetivo de aumentar a exposição do modelo a padrões raros e reduzir o desequilíbrio de desempenho entre classes, uma vez que, em cenários multiclasse com severidades próximas, erros tendem a concentrar-se nas classes minoritárias e/ou mais semelhantes.

Figura 6 – Matriz de Confusão do modelo final aplicado ao dataset MFS/MAFAULDA.



Fonte: Elaborado pelo autor (2026).

O modelo intensificado alcançou acurácia de aproximadamente 87% no conjunto de teste, com macro-F1 em torno de 0,86 e apenas cinco classes com F1-*score* inferior a 0,8, indicando melhora do desempenho médio por classe em relação à versão intermediária e comportamento próximo ao patamar do modelo base.

A Figura 6 apresenta a *matriz de confusão* do modelo final aplicado ao conjunto de *teste*, com acurácia global de 0,8849, e constitui uma ferramenta central para interpretar o desempenho do classificador além da métrica agregada. Em problemas *multiclasse*, como o deste trabalho

(30 classes, combinando modo de falha e severidade), a matriz permite identificar não apenas a taxa total de acertos, mas principalmente quais classes são confundidas entre si e se esses erros se concentram em condições fisicamente correlatas (por exemplo, severidades próximas do mesmo defeito) ou em defeitos distintos, o que teria implicações mais críticas para a Manutenção Preditiva. Assim, a análise da matriz é essencial para avaliar a viabilidade do modelo como ferramenta de apoio à decisão, pois erros diferentes têm impactos diferentes na priorização de inspeções, na escolha de intervenção (alinhamento, balanceamento, troca de rolamento) e na mitigação do risco de falhas severas.

De forma geral, observa-se na Figura 6 uma concentração dominante dos valores na diagonal principal, evidenciando que, para a maior parte das classes, o modelo atribui corretamente o rótulo previsto ao rótulo verdadeiro. Esse padrão é coerente com as métricas por classe apresentadas no relatório de classificação: diversos rótulos atingem valores elevados de precisão, *recall* e *F1-score*, indicando bom equilíbrio entre sensibilidade (capacidade de detectar a classe quando ela ocorre) e especificidade prática (redução de falsas classificações naquela classe). No entanto, a matriz também evidencia que o erro do modelo não é distribuído de forma uniforme: há grupos de classes com maior concentração de confusões, o que revela limitações específicas do espaço de características (*features*) extraído e orienta de forma objetiva onde aprimorar o *pipeline*.

Um primeiro aspecto relevante é que as classes associadas a falhas de rolamento apresentam, em geral, comportamento de classificação robusto, com poucos erros fora da diagonal. No grupo *ball_fault*, os indicadores mostram alto desempenho para diferentes severidades (*F1-score* entre aproximadamente 0,94 e 0,98 e *recall* elevado, chegando a 1,0 em algumas classes), o que é compatível com uma matriz de confusão com baixa dispersão para outras condições. Do ponto de vista do diagnóstico de falhas, isso sugere que as características extraídas conseguiram capturar padrões consistentes associados a defeitos no elemento rolante (*rolling element*), reduzindo o risco de o modelo confundir essa condição com outros modos de falha mecanicamente distintos. Em termos de Manutenção Preditiva, essa evidência é importante porque falhas em rolamentos são típicas em máquinas rotativas e podem evoluir para falhas mais graves; portanto, a alta taxa de acerto nesse grupo favorece a detecção precoce e decisões de intervenção com maior segurança.

Comportamento semelhante aparece no grupo *outer_race*, em que as classes também mantêm *F1-score* alto (em torno de 0,90 a 0,95) e *recall* elevado (0,875 a 0,95), sinalizando que o modelo identifica de forma consistente defeitos no anel externo. A *matriz de confusão* reforça essa interpretação ao apresentar a maior parte das contagens na diagonal para essas classes, com poucas previsões desviando para rótulos de outros defeitos. Essa estabilidade é desejável porque, na prática, confundir falhas do anel externo com desalinhamento ou desbalanceamento poderia alterar totalmente a ação de manutenção recomendada; os resultados indicam que esse tipo de erro entre modos ocorre pouco nesses casos.

No caso de *cage_fault*, embora a diagonal ainda predomine, observa-se desempenho

ligeiramente inferior em algumas severidades (por exemplo, *F1-score* entre aproximadamente 0,85 e 0,93, com *recalls* de 0,8235 a 0,95), o que sugere maior propensão a confusões residuais. Essa diferença é consistente com a interpretação de que falhas na gaiola (*cage*) podem gerar assinaturas que, dependendo da severidade e do ruído, se aproximam de outras falhas de rolamento, elevando a chance de confusão. Ainda assim, a matriz não indica um padrão de erro aleatório: o comportamento permanece concentrado em poucas células fora da diagonal, indicando que o modelo mantém generalização razoável e que os erros tendem a ocorrer em regiões de sobreposição de padrões, e não por instabilidade global do classificador.

Para os desalinhamentos, a matriz permite avaliar um ponto particularmente importante: a distinção entre severidades. No desalinhamento horizontal (*horizontal misalignment*), duas severidades (1,0 mm e 2,0 mm) apresentam elevado desempenho no teste (*recall* e *F1-score* iguais a 1,0), enquanto a severidade mais baixa (0,5 mm) apresenta *recall* de 0,7 e *F1-score* de aproximadamente 0,8235, o que indica que parte das amostras de menor severidade foi confundida com outra condição. Essa assimetria é tecnicamente plausível porque, em níveis iniciais de falha, a energia do defeito no sinal é menor, e a separação entre classes tende a ser mais difícil, especialmente quando o conjunto de atributos resume o sinal por estatísticas e medidas espectrais agregadas. Do ponto de vista de PDM (*Predictive Maintenance*), os resultados indicam elevada eficácia do classificador na identificação de desalinhamentos mais pronunciados, mas pode exigir refinamento para aumentar a sensibilidade a casos incipientes (que são, justamente, o alvo ideal da manutenção preditiva).

No desalinhamento vertical (*vertical misalignment*), o desempenho se mantém elevado e relativamente homogêneo (*F1-score* variando aproximadamente entre 0,82 e 0,95), com pequenas perdas em classes específicas (por exemplo, 0,63 mm e 1,40 mm). A matriz sugere que as confusões, quando ocorrem, permanecem próximas a condições correlatas, o que é compatível com a ideia de que o erro está mais associado à separação entre níveis de severidade do que à confusão entre modos completamente distintos. Em termos práticos, errar a severidade exata dentro do mesmo modo tende a ter impacto menor do que errar o modo de falha, pois a ação recomendada (procedimento de alinhamento) continua sendo coerente, e a severidade pode ser confirmada por inspeção e medição mecânica.

A principal limitação evidenciada pela Figura 6 está concentrada no grupo *imbalance* (desbalanceamento), que reúne as menores métricas por classe e as confusões mais relevantes. Em particular, as classes *imbalance*-20g e *imbalance*-25g apresentam *F1-score* de 0,5, com *recalls* de 0,4 e 0,4444, respectivamente, o que implica que mais da metade das amostras dessas classes foi classificada como outra condição. Ao mesmo tempo, classes como *imbalance*-6g, *imbalance*-10g e *imbalance*-30g apresentam desempenho substancialmente melhor (*F1-score* em torno de 0,80 a 0,95), indicando que o problema não é "o desbalanceamento como um todo", mas a separação de severidades intermediárias (20g e 25g), que provavelmente apresentam assinaturas muito próximas no espaço de características utilizado. Isso reforça a interpretação de que, para

certos níveis, as *features* extraídas podem não ser suficientemente discriminativas para separar severidades adjacentes, o que é coerente com a natureza física do desbalanceamento, em que variações relativamente pequenas de massa podem resultar em padrões espectrais similares, especialmente sob ruído e variabilidade operacional.

Do ponto de vista de aplicação em Manutenção Preditiva, esse padrão de confusão sugere duas implicações objetivas. A primeira é que o modelo é mais confiável para indicar o modo de falha (*failure mode*) (por exemplo, *rolamento vs. desalinhamento*) do que para estimar severidade com alta precisão em todos os casos, especialmente no desbalanceamento intermediário. A segunda é que a matriz orienta melhorias direcionadas, tais como: (i) inclusão de atributos mais sensíveis à componente síncrona associada ao desbalanceamento (por exemplo, amplitude em banda estreita em torno de $1\times$ rotação), (ii) uso de informação de rotação (quando disponível) para normalização espectral e (iii) estratégias de classificação hierárquica, em que o sistema primeiro classifica o modo (*imbalance*) e, em seguida, refina a severidade dentro desse subconjunto.

Portanto, a Figura 6 confirma que o modelo final apresenta boa capacidade de generalização para a maior parte das classes, com erros concentrados em subgrupos fisicamente coerentes e, principalmente, em níveis de severidade próximos. Essa característica é relevante porque indica que, quando ocorrem erros, eles tendem a ser "localizados" (entre classes correlatas), o que reduz a probabilidade de recomendações de manutenção completamente inadequadas e reforça o potencial do modelo como suporte à decisão em um contexto de monitoramento de condição. Em complemento, a Figura 7 (métricas por classe) corrobora a leitura da matriz ao evidenciar que poucas classes concentram os menores *F1-scores*, especialmente no grupo *imbalance*, o que justifica que as ações futuras de melhoria do modelo sejam concentradas nesse conjunto de condições.

Estudos como os de Lee et al. (2019), Meola (2005), Moura Filho et al. (2021) e Barbosa (2023) apontam que, em cenários reais com alta variabilidade de condições operacionais, acurácias entre 80% e 90% são frequentemente consideradas satisfatórias, sobretudo quando acompanhadas de macro-F1 elevados e distribuição equilibrada de desempenho entre as classes, como foi observado na melhor configuração testada. A análise dos erros mostra que as confusões residuais ocorrem predominantemente entre classes do mesmo tipo de falha e com severidades próximas (por exemplo, variações de *"imbalance"* e condições de desalinhamento), o que é esperado quando os padrões extraídos dos sinais se tornam semelhantes à medida que a diferença de severidade diminui. Esse comportamento é relevante do ponto de vista de manutenção preditiva, pois sugere que, mesmo quando há erro, ele tende a ocorrer entre estados fisicamente correlatos (ex.: níveis vizinhos do mesmo defeito), reduzindo a probabilidade de confundir modos de falha estruturalmente distintos (como defeitos de rolamento versus falhas de rotor).

Adicionalmente, a matriz de confusão permite observar que as classes associadas a *imbalance* concentram parte das discrepâncias fora da diagonal, o que é consistente com a identificação dessas condições como mais desafiadoras no conjunto avaliado, sobretudo em níveis

intermediários de severidade. Esse comportamento é corroborado pelas métricas por classe, nas quais *imbalance-20g* e *imbalance-25g* apresentam os menores desempenhos do conjunto (*F1-score* = 0,50 em ambos os casos), com *recall* de 0,40 e 0,4444, respectivamente, indicando que uma parcela significativa das amostras dessas severidades foi predita como outras classes. Em contraste, severidades mais baixas ou mais altas de desbalanceamento apresentaram melhor separabilidade, como *imbalance-6g* (*F1-score* = 0,9474), *imbalance-10g* (*F1-score* = 0,90) e *imbalance-35g* (*F1-score* = 0,8235), sugerindo que a sobreposição de padrões ocorre principalmente entre severidades próximas na faixa intermediária, onde a diferença entre assinaturas pode ser sutil frente à variabilidade dos sinais e ao nível de agregação das *features* extraídas. Do ponto de vista da Manutenção Preditiva, essa evidência é relevante porque indica que o classificador tende a ser mais confiável para identificar o modo de falha "desbalanceamento" do que para discriminar com a mesma precisão todas as severidades intermediárias, recomendando cautela na interpretação desses níveis e reforçando a necessidade de inspeção complementar quando o diagnóstico recair nessas classes.

A diferença de acurácia entre o modelo base (88%), o intermediário (81%) e o agressivo (87%) é explicável tecnicamente por de três fatores principais: arquitetura, regularização e estratégia de balanceamento/validação.

Embora os valores de acurácia observados sejam inferiores aos relatados por técnicas de *ensemble learning* aplicadas à base Mafaulda, como em Moura Filho et al. (2021), que alcançaram acurácia superior a 99% com *Random Forest* e *Gradient Boosting*, os resultados são coerentes considerando o aumento do número de classes e a complexidade dos cenários avaliados neste trabalho

A diferença de desempenho entre o modelo base e a versão intermediária decorre do compromisso entre maximizar a acurácia no *split* de teste e tornar o desempenho mais uniforme entre as 30 classes (modos de falha e severidades). No modelo base, a arquitetura e os hiperparâmetros foram ajustados com regularização mais branda, caracterizada pela aplicação de penalização $L2$ com $\lambda = 1 \times 10^{-4}$ (*kernel_regularizer = keras.regularizers.l2(1e-4)*) e pelo uso de *dropout* com taxas decrescentes (*Dropout(0.4)*, *Dropout(0.3)*, *Dropout(0.2)* e *Dropout(0.1)*), mantendo ainda o controle de treinamento por *Early Stopping* com *patience=50* e a redução dinâmica da taxa de aprendizado via *ReduceLROnPlateau* com *factor=0.5*, *patience=20* e *min_lr=1e-7*. A avaliação foi feita sobre a divisão estratificada 80/20; nessa condição, a rede pode alcançar alta acurácia global ao priorizar padrões dominantes do conjunto de treino e, consequentemente, tende a acertar mais as classes com maior representatividade no *split* (o que "puxa" a acurácia para cima).

Na versão intermediária, foram introduzidos *dropout*, regularização L2, *Early Stopping* e *ReduceLROnPlateau*, além de *oversampling* das classes minoritárias no treino. Esses elementos impõem restrições ao ajuste dos pesos (*dropout*/L2) e alteram a distribuição de exemplos vista durante o treinamento (*oversampling*), reduzindo a chance de o modelo memorizar particularida-

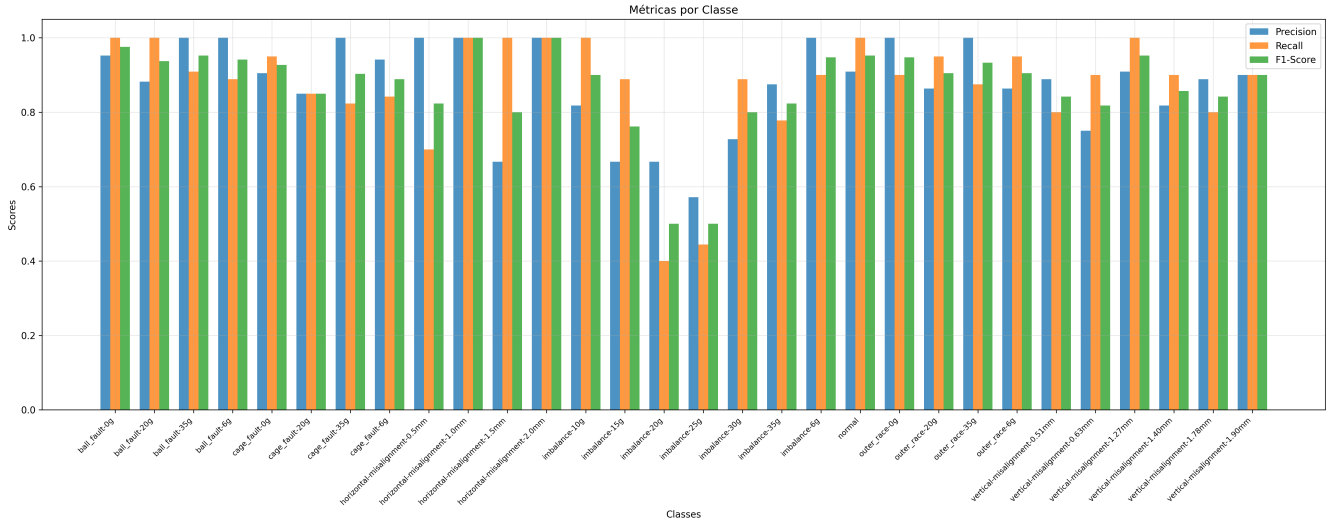
des do conjunto de treino e forçando a rede a aprender padrões mais gerais que se sustentem na validação. Como efeito colateral, é comum que a acurácia global no teste diminua em comparação ao modelo menos regularizado, porque a rede passa a abdicar de parte do ajuste fino que aumentaria o acerto nas classes majoritárias, em troca de melhorar *recall*/precisão em classes raras.

Por isso, mesmo com queda de acurácia (por exemplo, de 88% para 81%), observou-se um indicador mais alinhado ao objetivo de diagnóstico multiclasse: o macro-F1, que dá o mesmo peso a cada classe e, portanto, evidencia se o classificador está consistente também nos modos de falha menos frequentes. Em síntese, o modelo base favoreceu o desempenho global no *split* avaliado, enquanto a versão intermediária foi projetada para reduzir disparidades entre classes (especialmente minoritárias), o que é desejável em manutenção preditiva, onde falhas raras podem ser as mais críticas.

A versão intensificada aumentou a profundidade e a capacidade da rede, introduziu treinamento mais longo e *oversampling* focado; isso permitiu aprimorar acurácia global (de 81% para 87%) e elevar o macro-F1, reduzindo o número de classes com desempenho insatisfatório, mas, ao mesmo tempo, expôs o modelo ao risco de superajuste, isto é, de ajustar-se excessivamente aos padrões específicos do conjunto de treinamento e, com isso, apresentar pior desempenho ao classificar amostras novas (conjunto de teste/validação). As diferenças decorrem também do compromisso entre acurácia global e macro-F1: enquanto o modelo base pode atingir alta acurácia favorecendo as classes majoritárias(isto é, as classes com maior número de amostras na base e, portanto, mais representadas no conjunto de treino/teste, como *ball_fault-0g*,*outer_race-20g*, *outer_race-0g* e *cage_fault-0g*), as versões intermediária e intensificada foram explicitamente ajustadas para tratar melhor classes raras(isto é, as classes com menor número de amostras na base, como *imbalance-35g* e demais classes com quantidades na faixa de 45–49 registros).Esse ajuste foi realizado porque, em um problema multiclasse desbalanceado, o treinamento tende a ser dominado pelas classes com mais exemplos, elevando a acurácia global sem necessariamente garantir bom desempenho nas classes menos representadas; por isso, foram aplicadas estratégias de balanceamento descritas nos Materiais e Métodos (*oversampling* das classes minoritárias no conjunto de treino e, na versão intensificada, reforço adicional nas classes com pior F1-*score*).

Deste modo isso se reflete em macro-F1 mais elevado (0,8695) e em uma matriz de confusão mais homogênea (Figura 6), no sentido de concentrar a maior parte das previsões corretas na diagonal principal e reduzir a dispersão de erros para classes não correlatas. Esse comportamento também é evidenciado pela Figura 7 (Métricas por Classe), na qual o F1-score (barras verdes) permanece alto na maioria das classes, indicando simultaneamente boa precisão e bom recall para diferentes modos de falha e severidades. As exceções ficam concentradas em poucas classes do grupo imbalance (notadamente *imbalance-20g* e *imbalance-25g* com F1 = 0,5, e *imbalance-15g* com F1 = 0,76), o que explica por que, apesar da boa acurácia global, ainda existem classes com desempenho inferior e confusões residuais na matriz.

Figura 7 – Desempenho detalhado do modelo final: Precision, Recall e F1-Score por classe.



Fonte: Elaborado pelo autor (2026).

A Figura 7 (Métricas por classe) apresenta, para cada uma das 30 classes, as métricas calculadas no conjunto de *teste*: *precisão*, *recall* e *F1-score*. A *precisão* indica a confiabilidade das predições do modelo para uma classe específica, ou seja, dentre as amostras previstas como pertencentes àquela classe, qual proporção está correta (menor incidência de falsos positivos). Já o *recall* expressa a capacidade do classificador de identificar corretamente as amostras reais daquela classe, isto é, qual proporção dos exemplos verdadeiros foi recuperada pelo modelo (menor incidência de falsos negativos).

No presente resultado, observa-se que, na maior parte das classes, *precisão* e *recall* permanecem em patamares elevados e relativamente equilibrados, evidenciando que o modelo é, ao mesmo tempo, consistente ao "confirmar" uma classe quando a prediz e sensível para detectá-la quando ela ocorre. Exemplos desse comportamento incluem *ball_fault-0g* (*precisão* 0,9524; *recall* 1,0) e a condição *normal* (*precisão* 0,9091; *recall* 1,0), indicando baixas taxas de erro tanto por alarmes indevidos quanto por falhas não detectadas para esses rótulos.

Em contrapartida, a Figura 7 evidencia que as principais limitações se concentram em poucas classes, sobretudo no grupo *imbalance*, onde ocorre queda significativa de *recall* e, em alguns casos, também de *precisão*. Em *imbalance-20g* e *imbalance-25g*, por exemplo, os *recalls* de 0,40 e 0,4444 mostram que uma parcela relevante das amostras reais dessas severidades não foi identificada corretamente, enquanto as *precisões* de 0,6667 e 0,5714 indicam que parte das predições nessas classes corresponde a amostras de outras condições, refletindo sobreposição de padrões e confusões com classes correlatas. Esse comportamento é consistente com a matriz de confusão (Figura 6), na qual as discrepâncias fora da diagonal se intensificam em severidades intermediárias de desbalanceamento, reforçando que as confusões residuais se concentram em subconjuntos específicos, e não de forma aleatória entre todas as classes.

A partir dessa análise por classe, o *macro-F1* torna-se relevante como síntese do desempenho em um cenário *multiclasse*, pois combina *precisão* e *recall* em uma medida única calculada individualmente para cada classe e, em seguida, promediada com pesos iguais. Assim, o *macro-F1* de 0,8695 reflete o bom desempenho médio do classificador entre as 30 classes, mas também evidencia que quedas localizadas de *recall* e *precisão* em poucas classes (notadamente *imbalance-20g* e *imbalance-25g*) são suficientes para reduzir a média, motivo pelo qual a avaliação do modelo deve considerar simultaneamente acurácia, matriz de confusão e métricas por classe no contexto da Manutenção Preditiva.

5 CONCLUSÃO

A presente pesquisa teve como objetivo desenvolver uma solução computacional baseada em Redes Neurais Artificiais para o diagnóstico automatizado de falhas em motores de indução trifásicos, utilizando como base de dados o *Motor Fault Simulator* (MFS) do Laboratório de Engenharia Acústica e Vibrações da UFRJ. Partindo da fundamentação teórica em Manutenção Preditiva, Inteligência Artificial e monitoramento de condição, foi possível estruturar um *pipeline* completo de aquisição, rotulagem, extração de características e classificação, alinhado às melhores práticas descritas na literatura recente sobre PdM e aplicações de IA em ativos industriais críticos.

Os experimentos conduzidos com o código de referência, responsável por atingir cerca de 88% de acurácia, demonstraram que a metodologia proposta é viável e capaz de produzir classificadores com desempenho elevado em um cenário desafiador, caracterizado pela necessidade de distinguir simultaneamente 30 classes (combinações de modo de falha e severidade) a partir de sinais de vibração e corrente, em que parte das classes apresenta assinaturas parcialmente semelhantes e, portanto, maior propensão a confusões entre severidades próximas.

As versões subsequentes do modelo, nas quais foram introduzidos mecanismos adicionais de regularização (*dropout*, L2), esquemas de *oversampling* e validação mais sofisticados, atingiram acurácias na faixa de 81% a 87%, com macro-F1 acima de 0,80, o que permite concluir, sobre o funcionamento específico do classificador, que (i) o desempenho do modelo é sensível ao compromisso entre especialização no conjunto de treino e capacidade de generalização: ao aumentar a regularização e alterar a distribuição efetiva das amostras via *oversampling*, o modelo tende a reduzir parte dos acertos que inflavam a acurácia global, mas preserva (e, em alguns casos, melhora) o desempenho médio entre classes, refletido no macro-F1; e (ii) o classificador depende diretamente da separabilidade das características extraídas para cada modo de falha e severidade, pois as maiores perdas de desempenho concentram-se em classes com padrões mais semelhantes (por exemplo, diferentes níveis de desbalanceamento), indicando que os erros residuais decorrem mais de sobreposição de assinaturas do que de instabilidade do treinamento.

Do ponto de vista dos objetivos específicos, o trabalho cumpriu as metas propostas: definiu-se uma arquitetura de RNA compatível com o problema de classificação multiclasse do MFS; implementou-se toda a solução em Python 3.10 utilizando bibliotecas amplamente utilizadas (Pandas, NumPy, SciPy, Scikit-learn e TensorFlow/Keras); e validou-se o modelo sobre um conjunto de teste independente, por meio de métricas reconhecidas na área (acurácia, matriz de confusão, precisão, recall e F1-score).

Os resultados obtidos confirmam a hipótese central do estudo: é possível empregar Redes Neurais Artificiais para classificar, com boa precisão, os modos de falha de motores de indução trifásicos a partir da análise de sinais de vibração e corrente, contribuindo diretamente para a implementação de estratégias de Manutenção Preditiva em ambientes industriais.

Do ponto de vista prático, a solução desenvolvida oferece um protótipo funcional de

sistema de apoio à decisão para equipes de manutenção, capaz de indicar automaticamente prováveis modos de falha com base em dados históricos de vibração e corrente.

Esse tipo de ferramenta permite planejar intervenções de forma mais assertiva, reduzindo paradas não programadas, minimizando riscos de falhas catastróficas e aproximando o ambiente industrial dos princípios da Indústria 4.0, em que dados e algoritmos passam a orientar decisões de confiabilidade e disponibilidade de ativos.

Além disso, embora a MLP tenha se mostrado eficaz, outras arquiteturas, como Redes Neurais Convolucionais e Recorrentes, mencionadas na revisão bibliográfica, poderiam potencialmente extrair informações mais ricas dos sinais brutos, o que abre oportunidades para trabalhos futuros.

Como perspectivas de continuidade, sugerem-se três linhas principais: a primeira é o aprofundamento em arquiteturas de Deep Learning, como CNNs e RNNs, diretamente sobre os sinais no domínio do tempo e da frequência, seguindo abordagens semelhantes às de Lee et al. (2019).

A segunda consiste na integração do modelo treinado a um *dashboard* ou sistema supervisório em tempo real, permitindo o monitoramento contínuo de motores em planta, conforme delineado nos objetivos específicos do projeto. A terceira envolve a validação do sistema em bases de dados provenientes de ambientes industriais reais a exemplo dos estudos de Souza e Silva (2024) e Ohalete et al. (2023), consolidando a aplicabilidade da solução no contexto da Engenharia de Confiabilidade.

Em síntese, o trabalho contribui ao demonstrar, de forma prática e fundamentada, que a aplicação de Redes Neurais Artificiais à base MFS-UFRJ é uma abordagem tecnicamente consistente para o diagnóstico de falhas em motores de indução, alinhada às tendências atuais de Manutenção Preditiva e à literatura especializada.

Ainda que haja espaço para incrementos de desempenho, a solução desenvolvida cumpre o objetivo de fornecer uma base sólida, tanto conceitual quanto computacional, para o uso de IA na gestão de ativos industriais, atendendo às exigências acadêmicas e profissionais de um Trabalho de Conclusão de Curso em Engenharia de Controle e Automação.

REFERÊNCIAS

ABBAS, A. AI for Predictive Maintenance in Industrial Systems. **International Journal of Advanced Engineering Technologies and Innovations**, v. 1, n. 1, p. 31–51, 2024. Citado 3 vezes nas páginas 13, 16 e 18.

BARBOSA, J. D. M. **Manutenção preditiva com recurso a Inteligência Artificial**. Dissertação (Dissertação de Mestrado) — Universidade de Coimbra, Coimbra, 2023. Citado 4 vezes nas páginas 13, 17, 18 e 19.

COSTA, R. C. R. d.; SOUZA, R. R. d.; RODRIGUES, I. R. Avaliação de técnicas de redes neurais de aprendizado profundo para segmentação de feridas malignas cutâneas em imagens. In: SANTOS, W. P. d.; MORENO, G. M. M.; YARA, R. (Ed.). **Anais do VI Simpósio de Inovação em Engenharia Biomédica - SABIO 2022**. Recife: UFPE, 2022. p. 70–73. Citado na página 18.

FERNANDES, V. F. Desenvolvimento de rotinas em python para seleção de variáveis em dados espectroscópicos. Trabalho de Conclusão de Curso (Bacharelado em Física). 2024. Citado na página 18.

FILHO, J. d. O. M. **MANUTENÇÃO PREDITIVA APLICADA A MOTORES DE INDUÇÃO TRIFÁSICOS USANDO REDES NEURAIS ARTIFICIAIS**. Dissertação (Dissertação de Mestrado) — Universidade Federal do Ceará, Fortaleza, 2023. Citado 7 vezes nas páginas 17, 18, 19, 20, 24, 25 e 28.

GHOLAMY, A.; KREINOVICH, V.; KOSHELEVA, O. Why 70/30 or 80/20 Relation Between Training and Testing Sets: A Pedagogical Explanation. **Departmental Technical Reports (CS)**, v. 1209, p. 1–6, 2018. Technical Report: UTEP-CS-18-09. Citado na página 28.

HARRISON, M. **Machine Learning Guia de Referência Rápida: Trabalhando com Dados Estruturados em Python**. São Paulo: Novatec Editora, 2021. Citado na página 22.

KLEIN, P.; BERGMANN, R. Generation of Complex Data for AI-based Predictive Maintenance Research with a Physical Factory Model. In: **Proceedings of the 16th International Conference on Informatics in Control, Automation and Robotics**. Praga: SCITEPRESS, 2019. p. 40–50. Citado na página 18.

Laboratório de Engenharia Acústica e Vibrações. **Motor Fault Simulator**. <https://www2.smt.ufrj.br/~offshore/mfs/page_01.html>. [s.d.]. Acesso em: 22 jul. 2025. Citado 2 vezes nas páginas 14 e 21.

LEE, W. J. *et al.* Predictive Maintenance of Machine Tool Systems Using Artificial Intelligence Techniques Applied to Machine Condition Data. **Procedia CIRP**, v. 80, p. 506–511, 2019. Citado 3 vezes nas páginas 13, 16 e 18.

MEOLA, T. **Monitoramento em tempo real da qualidade de sinais de vibrações, utilizando inteligência artificial**. Dissertação (Dissertação de Mestrado) — Universidade Federal de Uberlândia, Uberlândia, 2005. Citado 3 vezes nas páginas 13, 14 e 18.

OHALETE, N. C. *et al.* Advancements in predictive maintenance in the oil and gas industry: A review of AI and data science applications. **World Journal of Advanced Research and Reviews**, v. 20, n. 3, p. 167–181, 2023. Citado 2 vezes nas páginas 13 e 19.

SARAN, D. J.; SARAN, M. C. B.; FRANZOTTI, C. L. O impacto da inteligência artificial na manutenção industrial: benefícios, desafios e tendências. In: **Simpósio de Tecnologia (Sitefa) - Fatec Sertãozinho**. Sertãozinho: [s.n.], 2024. v. 7, n. 1, p. e7114. Citado 2 vezes nas páginas 13 e 16.

SILVA, A. F. d.; SILVA, A. C. G. d.; SANTOS, W. P. d. Apoio ao diagnóstico do câncer de mama utilizando imagens termográficas e redes neurais artificiais. In: SANTOS, W. P. d.; MORENO, G. M. M.; YARA, R. (Ed.). **Anais do VI Simpósio de Inovação em Engenharia Biomédica - SABIO 2022**. Recife: UFPE, 2022. p. 58–61. Citado na página 18.

SOUZA, A. C. A.; SILVA, P. R. N. d. Aplicação da inteligência artificial na engenharia de confiabilidade e manutenção preditiva: um estudo de caso na indústria de mineração. **Revista Ibero-Americana de Humanidades, Ciências e Educação**, São Paulo, v. 10, n. 10, p. 3646–3658, Oct 2024. Citado 2 vezes nas páginas 13 e 17.

UCAR, A.; KARAKOSE, M.; Kirimça, N. Artificial Intelligence for Predictive Maintenance Applications: Key Components, Trustworthiness, and Future Trends. **Applied Sciences**, v. 14, n. 2, p. 898, 2024. Citado na página 13.

APÊNDICE A – CÓDIGO PRINCIPAL

Listing A.1 – Código de treinamento de redes neurais para portas lógicas

```
import os
import random
import numpy as np
import pandas as pd
from scipy.stats import kurtosis, skew
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import confusion_matrix, classification_report,
    f1_score
from sklearn.utils.class_weight import compute_class_weight
import tensorflow as tf
from tensorflow import keras
from tqdm import tqdm
import matplotlib.pyplot as plt
import json
from datetime import datetime

# Configura  es para 95%
ROOT_DIR = 'C:/Users/Lucas/Documents/full'
SEED = 42
EPOCHS = 500
PATIENCE = 50

# Criar diret rio para resultados
RESULTS_DIR = f"resultados_tcc_{datetime.now().strftime('%Y%m%d_%H%M%
    S')}"
os.makedirs(RESULTS_DIR, exist_ok=True)

def set_seeds(seed=SEED):
    """Define seeds para reproducibilidade total"""
    random.seed(seed)
    np.random.seed(seed)
    tf.random.set_seed(seed)
    os.environ['PYTHONHASHSEED'] = str(seed)

def coletar_arquivos_e_rotulos(root_dir):
    """Coleta arquivos CSV e seus r tulos com estat sticas
       detalhadas"""
    arquivos, rotulos = [], []
```

```
37      contagem_classes = {}
38
39      print("[FASE␣1]␣Coletando␣arquivos␣e␣r tulos...")
40      for subdir, _, files in os.walk(root_dir):
41          for f in files:
42              if f.endswith('.csv'):
43                  p = os.path.join(subdir, f)
44                  arquivos.append(p)
45                  partes = os.path.normpath(subdir).split(os.sep)
46                  rotulo = 'normal' if 'normal' in partes else f"{
                        partes[-2]}-{partes[-1]}"
47                  rotulos.append(rotulo)
48
49                  # Contagem para an lise de distribui  o
50                  contagem_classes[rotulo] = contagem_classes.get(
                        rotulo, 0) + 1
51
52      print(f"[FASE␣1]␣Total␣de␣arquivos:␣{len(arquivos)}")
53      print(f"[FASE␣1]␣N mero␣de␣classes:␣{len(contagem_classes)}")
54
55      # Salvar distribui  o de classes
56      dist_df = pd.DataFrame(list(contagem_classes.items()), columns=['
            Classe', 'Quantidade'])
57      dist_df = dist_df.sort_values('Quantidade')
58      dist_df.to_csv(f"{RESULTS_DIR}/distribuicao_classes.csv", index=
            False, encoding='utf-8')
59
60      print("\nDistribui  o␣de␣classes␣(top␣10␣menores):")
61      print(dist_df.head(10).to_string(index=False))
62
63      return arquivos, rotulos, contagem_classes
64
65  def extrair_features_avancadas_tempo_frequencia(p):
66      """
67  ␣␣␣␣Features␣no␣dom nio␣do␣tempo␣E␣frequ ncia␣(simples)␣para␣TCC
68  ␣␣␣␣"""
69      x = pd.read_csv(p, header=None).values.astype(np.float64)
70
71      # DOM NIO DO TEMPO (Features existentes aprimoradas)
72      rms = np.sqrt(np.mean(np.square(x), axis=0))
73      curt = kurtosis(x, axis=0, fisher=True, bias=False)
74      assim = skew(x, axis=0, bias=False)
```

```
    crest = np.max(np.abs(x), axis=0) / np.maximum(rms, 1e-12)
    energia = np.sum(np.square(x), axis=0) / x.shape[0]
    media_abs = np.mean(np.abs(x), axis=0)

    # Entropia com m ltiplos bins
    def entropia_multibins(arr, bins_list=[16, 32, 64]):
        entropias = []
        for bins in bins_list:
            eps = 1e-12
            H = []
            for c in range(arr.shape[1]):
                col = arr[:, c]
                hist, _ = np.histogram(col, bins=bins, density=True)
                p = hist / (np.sum(hist) + eps)
                p = p[p > 0]
                Hc = -np.sum(p * np.log2(p + eps))
                H.append(Hc)
            entropias.extend(H)
        return np.array(entropias)

    entropia_multi = entropia_multibins(x)

    def zero_crossing_rate(arr):
        zc = []
        for c in range(arr.shape[1]):
            col = arr[:, c]
            sgn = np.sign(col)
            sgn[sgn == 0] = 1
            crosses = np.sum(sgn[1:] * sgn[:-1] < 0)
            zc.append(crosses / max(len(col) - 1, 1))
        return np.array(zc)

    zcr = zero_crossing_rate(x)
    forma = rms / np.maximum(media_abs, 1e-12)
    pulse_idx = np.max(np.abs(x), axis=0) / np.maximum(media_abs, 1e
       -12)
    clearance = np.max(np.abs(x), axis=0) / np.maximum(np.square(
       media_abs), 1e-12)

    # Features de percentis expandidas
    percentis = [5, 10, 25, 50, 75, 90, 95]
    features_percentis = []
```

```
115     for pct in percentis:
116         features_percentis.extend(np.percentile(np.abs(x), pct, axis
                =0))
117 
118     # Raz es entre percentis
119     q10 = np.percentile(np.abs(x), 10, axis=0)
120     q90 = np.percentile(np.abs(x), 90, axis=0)
121     q25 = np.percentile(np.abs(x), 25, axis=0)
122     q75 = np.percentile(np.abs(x), 75, axis=0)
123 
124     q90_q10_ratio = q90 / np.maximum(q10, 1e-12)
125     q75_q25_ratio = q75 / np.maximum(q25, 1e-12)
126 
127     # DOM NIO DA FREQU NCIA (Simples para TCC)
128     def features_frequencia_simples(sinal):
129         features_freq = []
130         for canal in range(sinal.shape[1]):
131             # FFT b sica
132             fft = np.fft.fft(sinal[:, canal])
133             fft_magnitude = np.abs(fft)[:len(fft)//2]  # Apenas
                    frequ ncias positivas
134 
135             if len(fft_magnitude) > 0:
136                 # Estat sticas simples no dom nio da frequ ncia
137                 freq_rms = np.sqrt(np.mean(np.square(fft_magnitude)))
138                 freq_max = np.max(fft_magnitude)
139                 freq_median = np.median(fft_magnitude)
140                 freq_std = np.std(fft_magnitude)
141 
142                 # Raz o entre bandas (simples)
143                 mid_point = len(fft_magnitude) // 2
144                 low_band = np.mean(fft_magnitude[:mid_point])
145                 high_band = np.mean(fft_magnitude[mid_point:])
146                 band_ratio = low_band / np.maximum(high_band, 1e-12)
147 
148                 features_canal = [freq_rms, freq_max, freq_median,
                        freq_std, band_ratio]
149             else:
150                 features_canal = [0, 0, 0, 0, 0]
151 
152             features_freq.extend(features_canal)
153 
```

```
154         return np.array(features_freq)
155
156     features_freq = features_frequencia_simples(x)
157
158     # Features de derivada (din mica)
159     dx = np.diff(x, axis=0)
160     if dx.shape[0] > 0:
161         dx_rms = np.sqrt(np.mean(np.square(dx), axis=0))
162         dx_mean_abs = np.mean(np.abs(dx), axis=0)
163     else:
164         dx_rms = np.zeros(x.shape[1])
165         dx_mean_abs = np.zeros(x.shape[1])
166
167     # Features globais entre canais
168     zcr_std = np.std(zcr) if len(zcr) > 1 else 0
169     zcr_range = np.max(zcr) - np.min(zcr) if len(zcr) > 1 else 0
170
171     # Correla  o entre canais
172     corr_features = []
173     for i in range(min(6, x.shape[1]-1)):
174         corr = np.corrcoef(x[:, i], x[:, i+1])[0,1]
175         corr_features.append(0 if np.isnan(corr) else corr)
176     while len(corr_features) < 6:
177         corr_features.append(0)
178
179     # Concatenar TODAS as features
180     return np.concatenate([
181         # Dom nio do tempo (base)
182         rms, curt, assim, crest, energia, media_abs,
183         zcr, forma, pulse_idx, clearance,
184         q90_q10_ratio, q75_q25_ratio,
185         dx_rms, dx_mean_abs,
186
187         # Entropia multi-bins
188         entropia_multi,
189
190         # Percentis expandidos
191         features_percentis,
192
193         # Dom nio da frequ ncia
194         features_freq,
195
```

```
196         # Features globais
197         [zcr_std, zcr_range] + corr_features
198     ])
199
200 def preparar_dados_completos(root_dir):
201     """Prepara dados com todas as features avan adas"""
202     arquivos, rotulos, contagem_classes = coletar_arquivos_e_rotulos(
            root_dir)
203
204     print(f"\n[FASE 2] Extraindo features avan adas...")
205     X = []
206     for p in tqdm(arquivos, desc="Processando arquivos"):
207         features = extrair_features_avancadas_tempo_frequencia(p)
208         X.append(features)
209
210     X, y = np.array(X), np.array(rotulos)
211     print(f"[FASE 2] Dimens o final dos dados: {X.shape}")
212
213     # Salvar informa  es das features
214     feature_info = {
215         'num_amostras': X.shape[0],
216         'num_features': X.shape[1],
217         'num_classes': len(np.unique(y)),
218         'distribuicao_classes': contagem_classes,
219         'timestamp': datetime.now().isoformat()
220     }
221
222     with open(f"{RESULTS_DIR}/feature_info.json", 'w', encoding='utf
            -8') as f:
223         json.dump(feature_info, f, indent=2, ensure_ascii=False)
224
225     # Padroniza  o
226     scaler = StandardScaler()
227     X_scaled = scaler.fit_transform(X)
228
229     # Codifica  o de r tulos
230     le = LabelEncoder()
231     y_enc = le.fit_transform(y)
232     y_cat = keras.utils.to_categorical(y_enc)
233
234     # Split estratificado
235     X_train, X_test, y_train_cat, y_test_cat, y_train_enc, y_test_enc
```

```
        = train_test_split(
236         X_scaled, y_cat, y_enc, test_size=0.2, random_state=SEED,
                stratify=y_enc
237     )
238
239     print(f"[FASE␣2]␣Split␣final␣->␣Treino:␣{X_train.shape},␣Teste:␣{
            X_test.shape}")
240
241     return X_train, X_test, y_train_cat, y_test_cat, y_train_enc,
            y_test_enc, le, scaler
242
243 def criar_modelo_95_percent(input_dim, num_classes):
244     """Modelo␣otimizado␣para␣95%␣de␣acur cia"""
245     model = keras.Sequential([
246         keras.layers.Dense(1024, activation='relu', input_shape=(
                input_dim,),
247                            kernel_regularizer=keras.regularizers.l2(1e
                                -4)),
248         keras.layers.BatchNormalization(),
249         keras.layers.Dropout(0.4),
250
251         keras.layers.Dense(512, activation='relu',
252                            kernel_regularizer=keras.regularizers.l2(1e
                                -4)),
253         keras.layers.BatchNormalization(),
254         keras.layers.Dropout(0.3),
255
256         keras.layers.Dense(256, activation='relu',
257                            kernel_regularizer=keras.regularizers.l2(1e
                                -4)),
258         keras.layers.BatchNormalization(),
259         keras.layers.Dropout(0.2),
260
261         keras.layers.Dense(128, activation='relu'),
262         keras.layers.BatchNormalization(),
263         keras.layers.Dropout(0.1),
264
265         keras.layers.Dense(64, activation='relu'),
266         keras.layers.BatchNormalization(),
267
268         keras.layers.Dense(num_classes, activation='softmax')
269     ])
```

```
270
271     return model
272
273 def get_callbacks_avancados():
274     """Callbacks avan ados para treino"""
275     return [
276         keras.callbacks.EarlyStopping(
277             monitor='val_loss',
278             patience=PATIENCE,
279             restore_best_weights=True,
280             verbose=1
281         ),
282         keras.callbacks.ReduceLROnPlateau(
283             monitor='val_loss',
284             factor=0.5,
285             patience=20,
286             min_lr=1e-7,
287             verbose=1
288         ),
289         keras.callbacks.ModelCheckpoint(
290             f"{RESULTS_DIR}/melhor_modelo_95.h5",
291             monitor='val_accuracy',
292             save_best_only=True,
293             mode='max',
294             verbose=1
295         ),
296         keras.callbacks.CSVLogger(
297             f"{RESULTS_DIR}/historico_treinamento.csv",
298             separator=',',
299             append=False
300         )
301     ]
302
303 def plotar_matriz_confusao_simples(cm, classes, acuracia, save_path):
304     """Plot simples da matriz de confus o sem seaborn"""
305     plt.figure(figsize=(16, 14))
306
307     # Plot b sico da matriz
308     plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
309     plt.title(f'Matriz de Confus o\nAcur cia: {acuracia:.4f}',
        fontsize=16, fontweight='bold')
310     plt.colorbar()
```

```
311
312     # Adicionar valores
313     thresh = cm.max() / 2.
314     for i in range(cm.shape[0]):
315         for j in range(cm.shape[1]):
316             plt.text(j, i, format(cm[i, j], 'd'),
317                     ha="center", va="center",
318                     color="white" if cm[i, j] > thresh else "black",
319                     fontsize=8)
320
321     tick_marks = np.arange(len(classes))
322     plt.xticks(tick_marks, classes, rotation=45, ha='right', fontsize
        =8)
323     plt.yticks(tick_marks, classes, fontsize=8)
324     plt.tight_layout()
325     plt.ylabel('R tulo Verdadeiro')
326     plt.xlabel('R tulo Predito')
327     plt.savefig(save_path, dpi=300, bbox_inches='tight')
328     plt.close()
329
330 def plotar_curvas_aprendizado(history, save_path):
331     """Plot das curvas de aprendizado"""
332     plt.figure(figsize=(12, 4))
333
334     plt.subplot(1, 2, 1)
335     plt.plot(history.history['loss'], label='Train Loss', linewidth
        =2)
336     plt.plot(history.history['val_loss'], label='Val Loss', linewidth
        =2)
337     plt.title('Curvas de Loss')
338     plt.xlabel(' poca ')
339     plt.ylabel('Loss')
340     plt.legend()
341     plt.grid(True, alpha=0.3)
342
343     plt.subplot(1, 2, 2)
344     plt.plot(history.history['accuracy'], label='Train Acc',
        linewidth=2)
345     plt.plot(history.history['val_accuracy'], label='Val Acc',
        linewidth=2)
346     plt.title('Curvas de Acur cia')
347     plt.xlabel(' poca ')
```

```
348     plt.ylabel('Acur cia')
349     plt.legend()
350     plt.grid(True, alpha=0.3)
351 
352     plt.tight_layout()
353     plt.savefig(save_path, dpi=300, bbox_inches='tight')
354     plt.close()
355 
356 def plotar_metricas_por_classe(report, classes, save_path):
357     """Plot␣das␣m tricas␣por␣classe"""
358     f1_scores = [report[cls]['f1-score'] for cls in classes if cls in
            report]
359     precision_scores = [report[cls]['precision'] for cls in classes
           if cls in report]
360     recall_scores = [report[cls]['recall'] for cls in classes if cls
           in report]
361 
362     x = np.arange(len(classes))
363     width = 0.25
364 
365     plt.figure(figsize=(20, 8))
366     plt.bar(x - width, precision_scores, width, label='Precision',
           alpha=0.8)
367     plt.bar(x, recall_scores, width, label='Recall', alpha=0.8)
368     plt.bar(x + width, f1_scores, width, label='F1-Score', alpha=0.8)
369 
370     plt.xlabel('Classes')
371     plt.ylabel('Scores')
372     plt.title('M tricas␣por␣Classe')
373     plt.xticks(x, classes, rotation=45, ha='right', fontsize=8)
374     plt.legend()
375     plt.grid(True, alpha=0.3)
376     plt.tight_layout()
377     plt.savefig(save_path, dpi=300, bbox_inches='tight')
378     plt.close()
379 
380 def executar_experimento_95_percent():
381     """Executa␣experimento␣completo␣para␣95%"""
382 
383     print("="*80)
384     print("EXPERIMENTO␣TCC␣-␣BUSCA␣95%␣DE␣ACUR CIA")
385     print("="*80)
```

```
386
387     # Preparar dados
388     X_train, X_test, y_train_cat, y_test_cat, y_train_enc, y_test_enc
            , le, scaler = preparar_dados_completos(ROOT_DIR)
389
390     # Criar e compilar modelo
391     model = criar_modelo_95_percent(X_train.shape[1], y_train_cat.
            shape[1])
392
393     model.compile(
394         optimizer=keras.optimizers.Adam(learning_rate=1e-4),
395         loss='categorical_crossentropy',
396         metrics=['accuracy']
397     )
398
399     print("\n[FASE␣3]␣Arquitetura␣do␣Modelo:")
400     model.summary()
401
402     # Class weights para dados desbalanceados
403     classes = np.unique(y_train_enc)
404     class_weights = compute_class_weight('balanced', classes=classes,
            y=y_train_enc)
405     class_weight_dict = {i: w for i, w in zip(classes, class_weights)
            }
406
407     # Treinar modelo
408     print(f"\n[FASE␣4]␣Iniciando␣treinamento...")
409     history = model.fit(
410         X_train, y_train_cat,
411         validation_split=0.2,
412         epochs=EPOCHS,
413         batch_size=64,
414         callbacks=get_callbacks_avancados(),
415         class_weight=class_weight_dict,
416         verbose=1
417     )
418
419     # Avaliar modelo
420     print(f"\n[FASE␣5]␣Avaliando␣modelo...")
421     test_loss, test_acc = model.evaluate(X_test, y_test_cat, verbose
            =0)
422     y_pred = np.argmax(model.predict(X_test, verbose=0), axis=1)
```

```
423     macro_f1 = f1_score(y_test_enc, y_pred, average='macro')
424
425     # Gerar visualizações
426     print(f"\n[FASE 6] Gerando visualizações...")
427
428     # Matriz de confusão
429     cm = confusion_matrix(le.inverse_transform(y_test_enc),
430                           le.inverse_transform(y_pred),
431                           labels=le.classes_)
432     plotar_matriz_confusao_simples(cm, le.classes_, test_acc, f"{
            RESULTS_DIR}/matriz_confusao.png")
433
434     # Curvas de aprendizado
435     plotar_curvas_aprendizado(history, f"{RESULTS_DIR}/
            curvas_aprendizado.png")
436
437     # Relatório de classificação
438     report = classification_report(le.inverse_transform(y_test_enc),
439                                    le.inverse_transform(y_pred),
440                                    target_names=le.classes_,
441                                    output_dict=True)
442
443     # Métricas por classe
444     plotar_metricas_por_classe(report, le.classes_, f"{RESULTS_DIR}/
            metricas_por_classe.png")
445
446     # Salvar relatório completo
447     report_df = pd.DataFrame(report).transpose()
448     report_df.to_csv(f"{RESULTS_DIR}/relatorio_classificacao_completo
            .csv",
449                      encoding='utf-8', index=True)
450
451     # Análise detalhada das classes problemáticas
452     classes_problematicas = []
453     for classe in le.classes_:
454         if classe in report and report[classe]['f1-score'] < 0.8:
455             classes_problematicas.append({
456                 'Classe': classe,
457                 'F1-Score': report[classe]['f1-score'],
458                 'Precision': report[classe]['precision'],
459                 'Recall': report[classe]['recall'],
460                 'Support': report[classe]['support']
```

```
            })

    if classes_problematicas:
        problem_df = pd.DataFrame(classes_problematicas)
        problem_df = problem_df.sort_values('F1-Score')
        problem_df.to_csv(f"{RESULTS_DIR}/classes_problematicas.csv",
            index=False, encoding='utf-8')

    # Resultado final
    print(f"\n{'='*80}")
    print("RESULTADO FINAL")
    print(f"{'='*80}")
    print(f"Acur cia: {test_acc:.4f}")
    print(f"Macro-F1: {macro_f1:.4f}")
    print(f"Loss: {test_loss:.4f}")

    # Melhor  poca
    best_epoch = np.argmin(history.history['val_loss']) + 1
    print(f"Melhor  poca : {best_epoch}")

    # Classes problem ticas
    if classes_problematicas:
        print(f"\nClasses com F1-Score < 0.8: {len(
            classes_problematicas)}")
        print(problem_df.to_string(index=False))

    # Mensagem final
    print(f"\n{'='*80}")
    if test_acc >= 0.95:
        print("                  OBJETIVO 95% ATINGIDO! PARAB NS!
                              ")
    elif test_acc >= 0.90:
        print("   EXCELENTE RESULTADO! Pr ximo dos 95%!")
    elif test_acc >= 0.88:
        print("       TIMO PROGRESSO! Continue ajustando!")
    else:
        print("       BOM RESULTADO! Considere mais features ou dados
            .")

    print(f"\nTodos os resultados salvos em: {RESULTS_DIR}")
    print(f"{'='*80}")

```

```
499     # Salvar resumo executivo
500     resumo = {
501         'acuracia': float(test_acc),
502         'macro_f1': float(macro_f1),
503         'loss': float(test_loss),
504         'melhor_epoca': int(best_epoch),
505         'total_epocas': len(history.history['loss']),
506         'timestamp': datetime.now().isoformat(),
507         'classes_problematicas': [cls['Classe'] for cls in
                classes_problematicas] if classes_problematicas else []
508     }
509
510     with open(f"{RESULTS_DIR}/resumo_executivo.json", 'w', encoding='
            utf-8') as f:
511         json.dump(resumo, f, indent=2, ensure_ascii=False)
512
513 if __name__ == "__main__":
514     set_seeds(SEED)
515     executar_experimento_95_percent()
```