Programa de Pós-Graduação em Instrumentação, Controle e Automação de Processos de Mineração (PROFICAM)

Escola de Minas, Universidade Federal de Ouro Preto (UFOP)

Associação Instituto Tecnológico Vale (ITV)

Dissertação

DESENVOLVIMENTO DE UMA SOLUÇÃO EMBARCADA PARA IDENTIFICAÇÃO DE FALHAS EM SISTEMAS *UPS* (*UNINTERRUPTIBLE POWER SUPPLY*) POR MEIO DE APRENDIZADO DE MÁQUINA

Patrick Rafael Portes Andrade

Ouro Preto

Minas Gerais, Brasil

2023

Patrick Rafael Portes Andrade

DESENVOLVIMENTO DE UMA SOLUÇÃO EMBARCADA PARA IDENTIFICAÇÃO DE FALHAS EM SISTEMAS *UPS* (*UNINTERRUPTIBLE POWER SUPPLY*) POR MEIO DE APRENDIZADO DE MÁQUINA

Dissertação apresentada ao Programa de Pós-Graduação em Instrumentação, Controle e Automação de Processos de Mineração da Universidade Federal de Ouro Preto e do Instituto Tecnológico Vale, como parte dos requisitos para obtenção do título de Mestre em Engenharia de Controle e Automação.

Orientador: Gustavo Pessin, D.Sc.

Ouro Preto
2023

SISBIN - SISTEMA DE BIBLIOTECAS E INFORMAÇÃO

A553d Andrade, Patrick Rafael Portes.
Desenvolvimento de uma solução embarcada para identificação de falhas em sistemas UPS (uninterruptible power supply) por meio de aprendizado de máquina. [manuscrito] / Patrick Rafael Portes Andrade. - 2023.
77 f.: il.: color., tab.. + Código.

Orientador: Prof. Dr. Gustavo Pessin.
Dissertação (Mestrado Profissional). Universidade Federal de Ouro Preto. Programa de Mestrado Profissional em Instrumentação, Controle e Automação de Processos de Mineração. Programa de Pós-Graduação em Instrumentação, Controle e Automação de Processos de Mineração.
Área de Concentração: Engenharia de Controle e Automação de Processos Minerais.

1. Sistemas embarcados (Computadores). 2. Inteligência artificial. 3. Aprendizado do computador. 4. Arduino (Controlador programável) - Nano 33 BLE Sense. 5. Python (Linguagem de programação de computador). 6. Uninterruptible Power Supply (UPS). I. Pessin, Gustavo. II. Universidade Federal de Ouro Preto. III. Título. CDU 681.5:622.2

Bibliotecário(a) Responsável: Maristela Sanches Lima Mesquita - CRB-1716

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE OURO PRETO
REITORIA
ESCOLA DE MINAS
PROGR. POS GRAD. PROF. INST. CONT. E AUT.
PROCESSOS DE MIN.

FOLHA DE APROVAÇÃO

Patrick Rafael Portes Andrade

Desenvolvimento de uma solução embarcada para identificação de falhas em sistemas UPS (uninterruptible power supply) por meio de aprendizado de máquina

Dissertação apresentada ao Programa de Pós-Graduação em Instrumentação, Controle e Automação de Processos de Mineração (PROFICAM), Convênio Universidade Federal de Ouro Preto/Associação Instituto Tecnológico Vale - UFOP/ITV, como requisito parcial para obtenção do título de Mestre em Engenharia de Controle e Automação na área de concentração em Instrumentação, Controle e Automação de Processos de Mineração

Aprovada em 09 de outubro de 2023

Membros da banca

Doutor - Gustavo Pessin - Orientador - Instituto Tecnológico Vale
Doutor - Bruno Nazário Coelho - Universidade Federal de Ouro Preto
Doutor - Geraldo Pereira Rocha Filho - Universidade Estadual do Sudoeste da Bahia

Gustavo Pessin, orientador do trabalho, aprovou a versão final e autorizou seu depósito no Repositório Institucional da UFOP em 27/11/2023

Documento assinado eletronicamente por **Bruno Nazário Coelho**, **COORDENADOR(A) DE CURSO DE PÓS-GRADUAÇÃO EM INST. CONTROLE AUTOMAÇÃO DE PROCESSOS DE MINERAÇÃO**, em 28/11/2023, às 10:20, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site http://sei.ufop.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0 , informando o código verificador **0631977** e o código CRC **35FF57E5**.

Referência: Caso responda este documento, indicar expressamente o Processo nº 23109.016218/2023-78 SEI nº 0631977

R. Diogo de Vasconcelos, 122, - Bairro Pilar Ouro Preto/MG, CEP 35402-163
Telefone: (31)3552-7352 - www.ufop.br

Agradecimentos

Gostaria de agradecer a Deus pela dádiva da vida, pela saúde, pela capacitação, inteligência, e, principalmente, pelas pessoas colocadas em minha vida. Sim, eu tenho muito a agradecer por ter sido filho do senhor Herme e da senhora Márcia, in memoriam, pelo caráter, cuidado e zelo com o qual fui criado. Obrigado pai. No momento mais difícil da minha vida, o senhor se fez fortaleza e me amparou com braços fortes.

Agradeço a minha esposa, Thaís, por estar sempre ao meu lado, me apoiando e me dando força nas vezes que pensei em desistir. Agradeço também pelo cuidado com nosso filho, Pedro, nos momentos que eu estive ausente.

Agradeço à UFOP e ao ITV, incluindo todos professores e funcionários, pela oportunidade de fazer parte de uma instituição referência no Brasil e pela qualidade no ensino. Agradeço, especialmente, ao meu orientador, Prof. Dr. Gustavo Pessin, pela orientação, transmissão de conhecimento, leveza e, sobretudo, pela paciência.

Agradeço a minha família que é parte dessa realização e, especialmente, ao meu irmão gêmeo, Plínio, pelo apoio e pela presença sempre constante na minha vida. Deixo registrado aqui o orgulho que sito por você e por suas realizações.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior, Brasil (CAPES), Código de Financiamento 001; do Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq); da Fundação de Amparo à Pesquisa do Estado de Minas Gerais (FAPEMIG); e da Vale SA.

Resumo

Resumo da Dissertação apresentada ao Programa de Pós-Graduação em Instrumentação, Controle e Automação de Processos de Mineração como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

DESENVOLVIMENTO DE UMA SOLUÇÃO EMBARCADA PARA IDENTIFICAÇÃO DE FALHAS EM SISTEMAS *UPS* (*UNINTERRUPTIBLE POWER SUPPLY*) POR MEIO DE APRENDIZADO DE MÁQUINA

Patrick Rafael Portes Andrade

Setembro, 2023

Orientadores: Gustavo Pessin – ITV

Sistemas que utilizam algoritmos de *Machine Learning* (*ML*) para classificação e predição de informações são cada vez mais comuns na indústria. Relatórios que unem inteligência analítica e *big data* são capazes de prover *insights* preciosos sobre comportamentos de clientes, tendências de mercado e oportunidades de negócio, contudo, o uso de IA embarcado no chão de fábrica ainda é reduzido. Com avanço do poder de processamento de microcontroladores e utilização de técnicas de otimização de algoritmos de *ML*, surgiram algumas bibliotecas dedicadas para embarcar modelos de *ML* em placas microcontroladas de baixo custo. O sistema *UPS* é de extrema importância para o Sistema Elétrico de Potência (SEP), uma vez que é o responsável por garantir monitoramento e comando no caso da falta de tensão primária. O retificador trifásico é a parte mais sensível do sistema *UPS* e é o mais susceptível a falhas. Os retificadores atuais possuem um sistema de alarmes para indicar falhas, todavia esses alarmes, na maioria das vezes, vêm de forma tardia, quando o equipamento parou de funcionar. Esse trabalho propõe o desenvolvimento de uma solução embarcada utilizando a placa Arduino Nano 33 BLE Sense e algoritmos de *ML* para identificação de falhas em sistemas *UPS* através do processamento do som emitido por esses equipamentos. Foram obtidos resultados com acurácia de 99,74% para identificação de retificadores com defeito.

Palavras-chave: Sistemas Embarcados, Inteligência Artificial, Aprendizado de Máquinas, Inspeção, Arduino Nano 33 BLE Sense, Python, Diagnóstico, Retificadores, Subestação, Uninterruptible Power Supply (*UPS*).

Macrotema: Usina; **Linha de Pesquisa:** Robótica aplicada à Mineração; **Tema:** Inspeção automática de ativos; **Área Relacionada da Vale:** Manutenção e Inspeção.

Abstract

Abstract of Research Project presented to the Graduate Program on Instrumentation, Control and Automation of Mining Process as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

DEVELOPMENT OF AN EMBEDDED SOLUTION FOR FAULT IDENTIFICATION IN UPS (UNINTERRUPTIBLE POWER SUPPLY) SYSTEMS BY MEANS OF MACHINE LEARNING

Patrick Rafael Portes Andrade

Setembro, 2023

Advisors: Gustavo Pessin – ITV

Systems that use Machine Learning (ML) algorithms to classify and predict information are increasingly common in the industry. Reports that combine analytical intelligence and big data can provide precious insights into customer behaviors, market trends and business opportunities, however, the use of embedded AI on the factory floor is still low. With advances in the processing power of microcontrollers and the use of ML algorithm optimization techniques, such as quantization, some dedicated libraries have emerged to embed ML models on low-cost microcontroller boards. The UPS system is extremely important for the Electrical Power System (EPS), as it is responsible for ensuring monitoring and command in the event of a lack of primary voltage. The three-phase rectifier is the most sensitive part of the UPS system and is the most susceptible to failure. Current rectifiers have an alarm system to indicate failures, however these alarms, in most cases, come late, when the equipment has stopped working. This work proposes the development of an embedded solution using the Arduino Nano 33 BLE Sense board and ML algorithms to identify faults in UPS systems by processing the sound emitted by this equipment. Results were obtained with an accuracy of 99.74% for identifying defective rectifiers.

Keywords: Embedded Systems, Artificial Intelligence, Machine Learning, Diagnosis, Arduino Nano 33 BLE Sense, Python, Rectifiers, Substation, Uninterruptible Power Supply (UPS).

Macro theme: Plant; **Research Line:** Robotics applied to Mining; **Theme:** Automatic asset inspection; **Related Area of Vale:** Maintenance and Inspection.

Lista de Figuras

Lista de Tabelas

Lista de Siglas e Abreviaturas

CA: Corrente Alternada

CC: Corrente Contínua

COC: Centro de Operação e Controle

***ERP*:** *Enterprise Resource Planning*

DFN: *Deep Feedforward Neural Network*

***DT*:** *Decision Tree*

IA: Inteligência Artificial

***IoT*:** Internet of Things

ITV: Instituto Tecnológico Vale.

***KNN*:** K-Nearest Neighbors

***ML*:** Machine Learning

***MFCC*:** Mel-Frequency Cepstral Coefficient

***MEMS*:** *Micro-Electro-Mechanical Systems*

PC: *Personal Computer*

PCAN: *Per-Channel Amplitude Normalization*

PWM: *Pulse Width Modulation*

***RF*:** Random Forest

RNC: Rede Neural Convolucional

SNR: *Signal-to-Noise Ratio*

***STFFT*:** Short-Time Fast Fourier Transform

SEP: Sistema Elétrico de Potência

***UPS*:** Uninterruptible Power Supply

Sumário

1. Introdução

1.1. Motivação

O componente mais crítico de um sistema de proteção, controle e monitoramento é o sistema auxiliar de energia de controle em corrente contínua. A falha no fornecimento de energia em corrente contínua pode tornar os dispositivos de detecção de falhas incapazes de identificar falhas, impedir que disjuntores atuem em caso de falhas e tornar as indicações locais e remotas inoperantes, entre outros problemas (THOMPSON et al., 2007).

Em muitos casos, o sistema em corrente contínua não é redundante, o que torna a confiabilidade uma consideração extremamente importante no projeto geral. O sistema auxiliar de energia de controle em corrente contínua inclui a bateria, carregador de bateria, também conhecido como *UPS* (do inglês: *Uninterruptible Power Supply*), sistema de distribuição, dispositivos de comutação e proteção, além de qualquer equipamento de monitoramento.

É graças ao sistema *UPS* que o Centro de Operação e Controle (COC) permanece com o monitoramento das instalações e pode operar os equipamentos de campo remotamente, e de forma ágil, para o restabelecimento de energia elétrica em caso de faltas, uma vez que a fonte primária de energia em Corrente Alternada (CA) não é uma opção durante faltas no SEP. Portanto, o sistema UPS é de extrema importância pois sem ele seria impossível abrir disjuntores de alta tensão para interrupção de curtos-circuitos.

Outra função do sistema *UPS* consiste em isolar o sistema de corrente contínua e filtrar as oscilações da rede de corrente alternada para que elas não prejudiquem os equipamentos de proteção e comunicação (EMADI et al., 2005).

O sistema *UPS* é composto basicamente por um retificador trifásico e um banco de baterias estacionárias que, geralmente, operam com tensão nominal de 125 VCC (tensão de corrente contínua). O retificador trifásico é a parte mais sensível do sistema *UPS* sendo o mais susceptível a falhas, já que opera de forma ininterrupta fornecendo alimentação em corrente contínua para os equipamentos da instalação, bem como mantém o banco de baterias carregado. Outro fato a ser considerado é que o banco de baterias possui vida útil determinada pelo material das baterias e são trocados a cada cinco anos, enquanto os retificadores foram feitos para funcionarem por décadas.

Atualmente o diagnóstico desse equipamento é fornecido através de alarmes do próprio equipamento, os quais chegam, muitas das vezes, quando o equipamento já não está mais

retificando, ou, no pior caso, não sobe alarme para o COC, o qual só toma conhecimento do problema quando perde totalmente o monitoramento e a possibilidade de comando.

O diagnóstico de falhas em retificadores é mais eficiente quando realizado por um técnico experiente e treinado ao visitar a subestação de energia elétrica. O técnico treinado é capaz de reconhecer um retificador com falha ou na iminência de falhar por meio do som característico emitido pelo equipamento. Retificadores com problemas costumam emitir ruídos de alta frequência, os quais não estão presentes em retificadores em bom estado de funcionamento.

Em (RAHNAMA, 2019) foi mostrado que é possível detectar faltas em retificadores de um gerador síncrono sem escovas através de sinais sonoros emitidos pelo retificador. A experiência em campo mostra que retificadores emitem sons que vão se alterando de acordo com a vida útil do equipamento. De forma geral, os sons emitidos vão se tornando mais agudos ao ouvido humano.

Um técnico experiente ao visitar uma subestação é capaz de distinguir se um retificador está com algum defeito, ou na iminência de dar problema, apenas pelo ruído emitido. Um sistema externo ao retificador capaz de classificá-lo em tempo real, de forma preditiva, entre bom, mediano e ruim com boa precisão se apresenta como uma solução mais confiável e segura em relação aos meios tradicionais de diagnóstico.

Sabe-se que algoritmos de *ML* possuem a capacidade de aprender com exemplos e fazer interpolações e extrapolações do que aprenderam (BRAGA, 2000). O presente trabalho parte desse princípio para buscar uma solução para o problema de diagnósticos, e possível predição, de problemas em retificadores trifásicos através de algoritmos de *ML* treinados, por meio de aprendizagem supervisionada, com áudios gravados de retificadores em três categorias: bom, mediano e ruim. São utilizadas bibliotecas em Python para treinamento e análise de performance de dois modelos: um modelo de RNC e outro de *RF*.

Por fim, é embarcado algoritmos de *ML* em um placa Arduino Nano 33 BLE Sense gerando uma solução portátil para a detecção de defeitos em retificadores trifásicos.

1.2 Objetivos

O objetivo geral do trabalho é propor, desenvolver e avaliar uma solução embarcada para classificação de falhas em equipamentos de retificação trifásico por meio da análise do som emitido por esses equipamentos.

1.2.1 Objetivos específicos

- Coletar dados de sons de retificadores trifásicos instalados em campo e formar uma base de dados.
- Investigar diferentes modelos e algoritmos de *ML* e definir qual o mais eficiente para o problema em questão.
- Embarcar modelos de IA em hardware e avaliar taxas de acerto e consumo de recursos.
- Desenvolver um sistema inteligente para identificação de falhas em equipamentos de retificação trifásico por meio de som

1.3 Organização do trabalho

Os próximos capítulos deste trabalho estão organizados da seguinte forma: No capítulo 2 - Fundamentação científica, é feito uma revisão sobre as ferramentas e conceitos teóricos necessários para o desenvolvimento desse trabalhado. No capítulo 3 – Trabalhos relacionados, são apresentados trabalhos utilizados como referência desta dissertação e a contribuição específica deste trabalho. No capítulo 4 – Materiais e métodos, são abordados os equipamentos, programas e métodos utilizados neste trabalho. No capítulo 5 - Resultado e discussão, os resultados são analisados e comparados. No capítulo 6 - Conclusão e trabalhos futuros, é feita uma conclusão geral das atividades e resultados deste trabalho. No capítulo 7 – Trabalhos futuros, é proposto opções de trabalhos que possam aperfeiçoar ou complementar os resultados alcançados no presente trabalho.

2. Fundamentação científica

Neste capítulo será apresentada uma revisão teórica e pesquisas similares aos interesses desta dissertação. Pesquisas relacionadas ao uso de Aprendizado de Máquinas em sistemas embarcados serão apresentadas primeiro, em sequência, serão apresentadas pesquisas de técnicas em aprendizado de máquina para classificação e identificação de sons ambientes.

2.1. Sistema *UPS*

Sistemas *UPS* são utilizados para prover uma fonte de alimentação confiável para equipamentos de missão crítica em caso de perda da alimentação principal ocasionada por panes ou defeitos. Toda subestação de energia elétrica possui um sistema *UPS* composto por um banco de baterias e um retificador trifásico, o qual alimenta as cargas em CC e mantém o banco de baterias sempre carregado e pronto para uso em caso de falha na tensão alternada.

A Figura 1 mostra um Retificador Trifásico típico utilizado em subestações de energia elétrica

Figura 1 – Retificador Trifásico

Fonte: (AUTOR, 2023).

A Figura 2 mostra um banco de baterias utilizado em conjunto com o retificador para o fornecimento de energia em caso da falta de tensão CA no sistema principal.

Figura 2 – Banco de Baterias

Fonte: (AUTOR, 2023).

2.1.1. Retificador trifásico

O retificador é composto basicamente de um transformador de entrada, com ponte retificadora tiristorizada, filtro de saída, circuitos de controle e de supervisão microprocessados e dispositivos de proteção de entrada e saída, conforme indicado no diagrama unifilar simplificado da Figura 3.

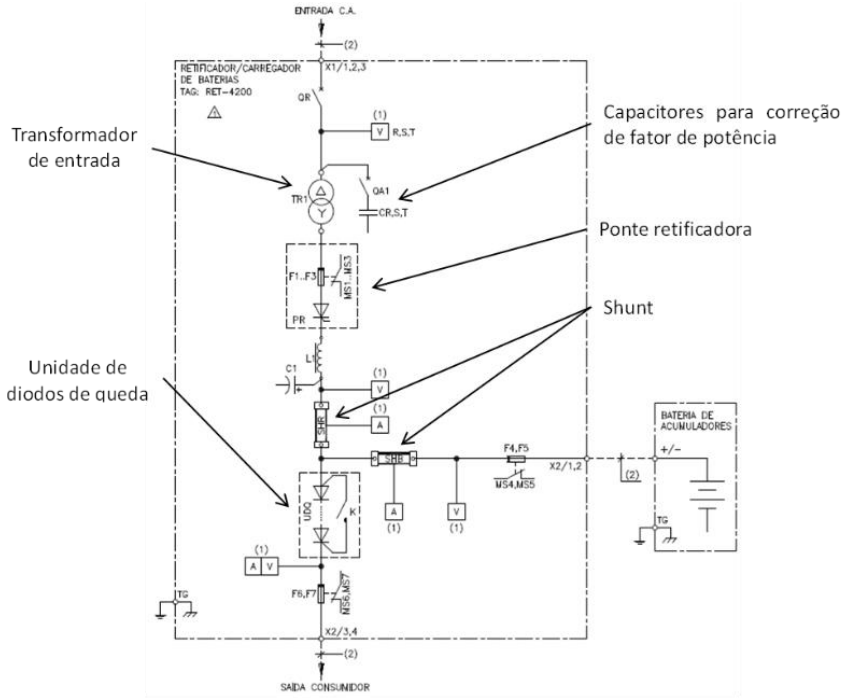

Figura 3 – Diagrama unifilar simplificado de um carregador de baterias trifásico

Fonte: (ADELCO, 2012).

Dado a essencialidade do retificador trifásico, torna-se necessário um sistema de monitoramento do seu estado. Atualmente esse monitoramento é feito por placas de circuito impresso presentes nos próprios retificadores. Acontece que esses equipamentos possuem vida útil de décadas (PLATTS et al., 1992) e nem sempre os circuitos de monitoramento estão confiáveis após tanto tempo e costumam falhar. Além disso, os sistemas de monitoramento atuais não conseguem diagnosticar, de forma preventiva, o problema e servem apenas para alarmar o problema depois de ocorrido.

2.2. Processamento de áudio

Processamento de áudio é uma das subáreas de uma área maior conhecida como Processamento Digital de Sinais. Em um sistema digital, um som é normalmente codificado como um vetor de amostras (*samples*), onde cada amostra corresponde à amplitude do sinal sonoro em um instante de tempo (OPPENHEIM, 2012). A Figura 4 mostra um trecho de 50 milissegundos de áudio amostrado a 16 kHz.

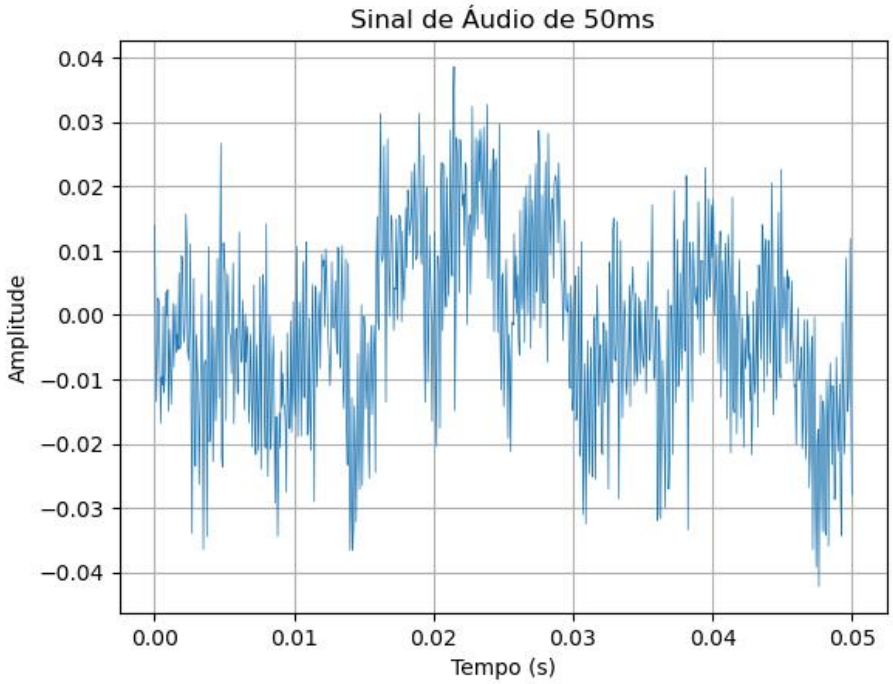

Figura 4 – 50 milissegundos de áudio com taxa de amostragem de 16 kHz

Fonte: (AUTOR, 2023).

A Figura 5 mostra um trecho de 5 milissegundos do mesmo áudio. Nessa figura é possível perceber que o sinal de áudio digital não é contínuo no tempo e possui espaçamentos temporais idênticos entre as amostras. A distância, no tempo, entre cada amostra é chamada de período de amostragem (OPPENHEIM, 2012).

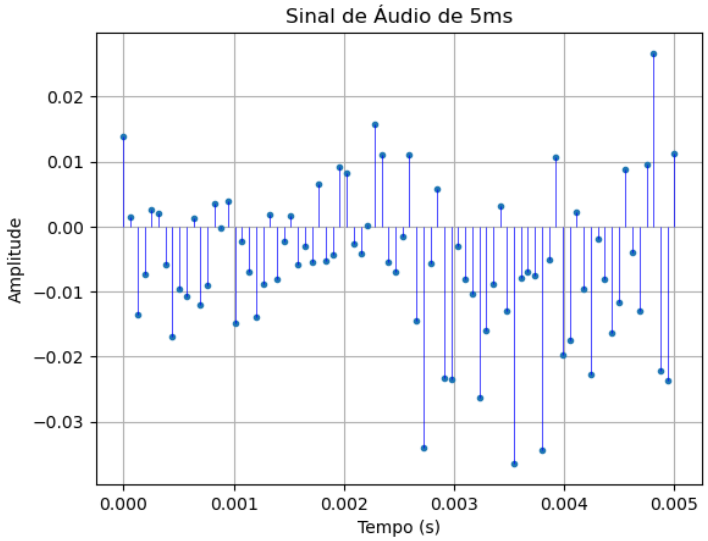

Figura 5 – 5 milissegundos de áudio com taxa de amostragem de 16 kHz

Fonte: (AUTOR, 2023).

É possível realizar o treinamento de um modelo de *ML* diretamente com as amostras de áudio puro, ou com o resultado de operações matemáticas (média simples, média quadrática, moda, etc.) sobre essas amostras, contudo, trabalhos recentes de (ERICEIRA, 2020) e (CRUZ, 2020), bem como a natureza ruidosa dos sons dos retificadores, demonstram que com o pré-processamento dos áudios para extração de características (*features*) no domínio da frequência é possível reduzir a quantidade de informações para alimentação do modelo, além de conseguir melhores resultados.

Do ponto de vista do sistema embarcado, a redução do volume de dados para alimentar um modelo de *ML* é uma característica muito relevante, uma vez que o poder de processamento, bem como os recursos (processador e memória) são limitados. Além do fato de que menos operações realizadas pelo processador significa economia de energia, o que é crucial quando se pensa em soluções embarcadas.

2.2.1. Extração de características do áudio

Existem alguns métodos conhecidos na literatura para extração de características de sons. De forma geral, esses métodos consistem no pré-processamento dos arquivos de áudio passando-os por algoritmos já conhecidos que são capazes de diminuir a dimensionalidade das amostras originais para que os algoritmos classificadores consigam lidar melhor com o conjunto de dados [RIJO, 2018]. Os principais métodos disponíveis na literatura são:

MOTIFS: Os MOTIFS são padrões frequentes que podem ser encontrados em sons e possuem uma importância especial, já que são as menores unidades estruturais que possuem uma identidade temática no contexto do som. Para extração desses "atributos" são utilizados algoritmos (ex. MrMotif), os quais discretizam os sinais e procuram padrões nas sequências (RIJO, 2018).

MFCC (Mel Frequency Cepstral Coefficients): Introduzidos por Davis e Mermelstein na década de 1980 *MFCCs* são recursos amplamente utilizados na área de processamento e classificação de áudios e desde então são considerados o estado da arte no que diz respeito a extração de características e reconhecimento de padrões em sons. Contudo, trata-se de uma técnica com elevado custo computacional, quando comparada a análises espectrais, e mais direcionadas para identificação de padrões de fala e/ou ritmos musicais (RIJO, 2018).

Análise espectral: A análise espectral nada mais é do que a soma da análise estatística das séries temporais mais os métodos de análise de Fourier (AGUIRRE, 1995). Para o contexto de

processamento de sons, a análise espectral consiste no pré-processamento do arquivo de áudio, passando-o por um algoritmo de *FFT* (*Fast Fourier Transform*), extraindo-lhe as componentes de frequência do sinal.

Para esse trabalho será utilizada a técnica de pré-processamento do áudio saindo do domínio do tempo para o domínio da frequência, tendo em vista que em trabalhos anteriores (ERICEIRA, 2020), (CRUZ, 2020) e (SILVA, 2021) essa técnica se mostrou satisfatória para detecção de falhas e/ou anomalias em equipamentos de campo através da gravação de sons/ruídos emitidos por esses equipamentos.

2.3. Aprendizado de máquina

Aprendizado de Máquinas é uma área da Inteligência Artificial cujo objetivo é o desenvolvimento de técnicas computacionais sobre o aprendizado bem como a construção de sistemas capazes de adquirir conhecimento de forma automática. Um sistema de aprendizado é um programa de computador que toma decisões baseado em experiências acumuladas através da solução bem-sucedida de problemas anteriores. Os diversos sistemas de aprendizado de máquina possuem características particulares e comuns que possibilitam sua classificação quanto à linguagem de descrição, modo, paradigma e forma de aprendizado utilizado (MONARD et al., 2003).

O processo de aprendizado da máquina está diretamente ligado a percepção de um padrão nos dados que alimentam o sistema. Se não houver um padrão, não há nada para o sistema inteligente aprender. Ou seja, o computador só é capaz de perceber uma regra e ser bem-sucedido na tomada de decisão a cerca de um assunto sobre o qual ele foi previamente treinado, se existir algum fator em comum nos dados apresentados no treinamento e, posteriormente, nos testes dos algoritmos (SILVA et al., 2010). É interessante notar que o computador é capaz de identificar padrões onde seres humanos são incapazes de enxergá-los.

2.3.1. Tipos de aprendizagem

Existem, basicamente, três métodos de aprendizado utilizados nos diversos algoritmos ou modelos de *ML*, são eles:

Aprendizado supervisionado: Este método de aprendizagem consiste em apresentar ao computador pares de entrada e saída. Ou seja, os dados de treinamento são rotulados de forma que é apresentado ao computador uma entrada e a sua respectiva saída real. Dessa forma, os dados passam a "ensinar" ao computador qual a resposta correta para cada amostra de entrada. Esses dados devem compor uma tabela entrada/saída que representa o processo. Assim, os dados serão utilizados de forma a continuamente ajustar os pesos da rede por meio do algoritmo de aprendizagem, sendo interrompido o processo apenas quando as saídas entregues pela rede forem condizentes com os valores da tabela entrada/saída. Essa conferência é normalmente realizada pelo próprio algoritmo. Será considerada treinada a rede que tiver valores de erro aceitáveis para a generalização de soluções (GÉRON, 2019). Este é o método mais utilizado pela maioria dos sistemas inteligentes em atividade hoje.

Aprendizado não supervisionado: Nesse método os dados são apresentados sem suas respectivas saídas. Apesar de parecer inútil, os algoritmos de aprendizado de máquinas são capazes de extrair informações e identificar padrões, os quais são suficientes para realizar a classificação em subconjuntos (do inglês: *clusters*). (GUIDO, 2017). Uma aplicação prática para o sistema capaz de realizar agrupamentos automáticos seria a segmentação de mercado, de modo a dividir os clientes pertencentes ao banco de dados de uma determinada empresa em grupos, fazendo com que ela possa direcionar seus esforços em medidas específicas que satisfaçam esses grupos.

Aprendizado com reforço: O aprendizado com reforço pode ser entendido como um meio termo entre o supervisionado e o não supervisionado. Nele, os dados que você trabalha são as entradas, algumas saídas e uma probabilidade dessas saídas serem satisfatórias ou não. Além do número limitado de saídas, o algoritmo não tem uma noção do quão 'perto' está da classificação desejada. Portanto, o sistema opera por tentativa e erro até conseguir compreender como determinada atividade funciona (GÉRON, 2019). Este é um algoritmo particularmente útil em jogos mais recentemente em *Chats* do tipo GPT.

Nesse trabalho, será utilizado o método de aprendizagem supervisionada, uma vez que os sons capturados dos retificadores já foram divididos em classes de retificadores em estado bom, mediano e com defeito.

2.3.2. Redes Neurais Artificiais (RNA)

As Redes Neurais Artificiais são estruturas formadas por neurônios artificiais, cuja inspiração veio do neurônio biológico. O neurônio artificial é um modelo matemático que simula o funcionamento de um neurônio biológico (SILVA et al., 2016). A Figura 6 mostra uma comparação entre uma rede de neurônios biológicos e uma rede de neurônios artificias.

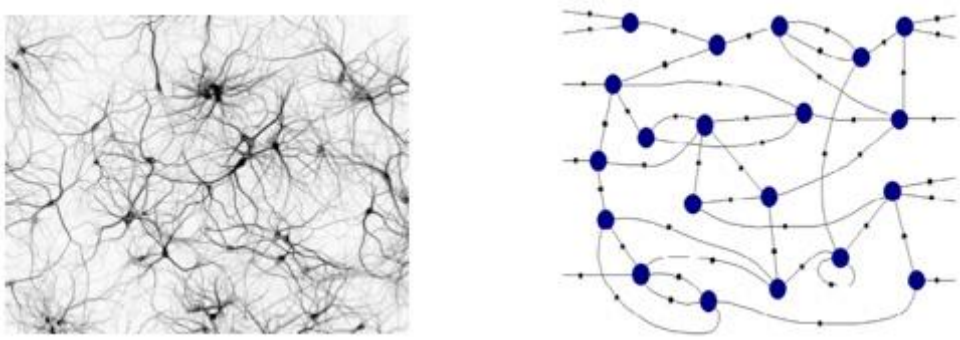

Figura 6 - Rede neural biológica (esquerda) e rede neural artificial (direta)

Fonte: (VILLA, 2018).

A entrada de um neurônio biológico é transmitida para outro neurônio após o atingimento de um nível crítico (*threshold*). O sinal neural é conduzido, estabelecendo comunicação com outros neurônios, por meio da sinapse. Sob o ponto de vista de uma rede neural composta por vários neurônios, esses sinais gerados proporcionam a ativação de outros neurônios, promovendo ondas de sinais. Dessa forma, têm-se diversas ondas reverberando dentro do tecido nervoso e, a partir delas, são gerados os pensamentos, as sensações e os comandos motores.

Com base nesse conceito, define-se o neurônio artificial como um modelo matemático que está representado na Figura 7.

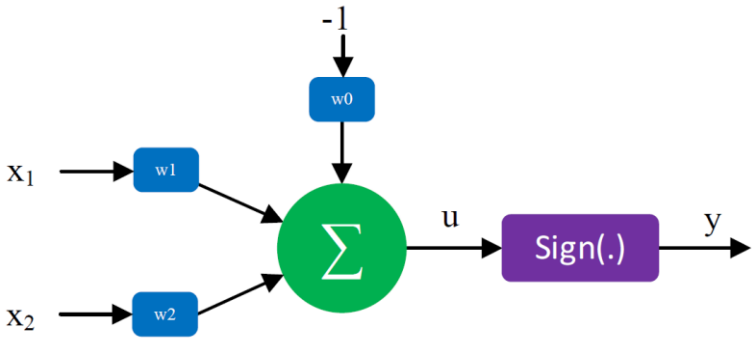

Figura 7 – Representação de um neurônio artificial

Fonte: (VILLA, 2018).

Em que o sinal de saída y é dado pelas equações:

$$y = f(u)$$

$$u = \sum_{i=1}^{m} x_i w_i - w_0$$

Sendo m a quantidade de entradas do neurônio.

Desse modo, o neurônio artificial recebe os sinais x_i de entrada e realiza a composição deles, por meio da soma ponderada de cada sinal, resultando em u. Para uma função de ativação $f(u)$, como apresentada na Figura 9, o neurônio artificial gera uma saída a positiva se $\sum_{i=1}^{m} x_i w_i > w_0$.

Em suma, o sistema exposto recebe os sinais da entrada e, de acordo com o valor desses sinais e dos valores dos pesos, pode-se interpretar as saídas $y = \{+1, -1\}$ como tendo um significado binário (sim/não). Este evento em muito se assemelha à condição de um neurônio biológico para transmitir sinais a outros neurônios. Por esse motivo, no contexto de uma rede neural artificial, o modelo matemático exposto pode ser considerado uma unidade básica da rede, denominado Perceptron. Dessa forma, uma rede neural artificial é uma rede composta por vários perceptrons.

2.3.3. Deep Learning (DL)

O aprendizado profundo é um ramo específico da aprendizagem de máquina que se concentra na aquisição de representações de dados cada vez mais significativas por meio de camadas sucessivas. O termo "profundo" em aprendizado profundo refere-se a essas camadas, e o número de camadas em um modelo é chamado de sua profundidade. O aprendizado profundo frequentemente envolve numerosas camadas de representações aprendidas automaticamente a partir dos dados de treinamento, enquanto outras abordagens de aprendizado de máquina tendem a se concentrar em apenas uma ou duas camadas de representações de dados, tornando-as "aprendizado superficial" (FRANÇOIS, 2021).

No aprendizado profundo, essas representações em camadas são adquiridas usando redes neurais, que são estruturadas como camadas literais empilhadas umas sobre as outras. Embora o termo "rede neural" seja inspirado na neurobiologia, os modelos de aprendizado profundo não são tentativas de replicar os mecanismos de aprendizado do cérebro. Não há evidências de que o cérebro use os mesmos métodos que os modelos modernos de aprendizado

profundo. É essencial evitar a associação do aprendizado profundo com a neurociência, pois é principalmente uma estrutura matemática para aprender a partir de dados (FRANÇOIS, 2021).

A Figura 8 mostra uma rede neural profunda utilizada para aprender sobre dígitos. Nela é mostrado o processamento de uma imagem do dígito 4 que passa por quatro camadas até ser classificado como dígito 4 na última camada.

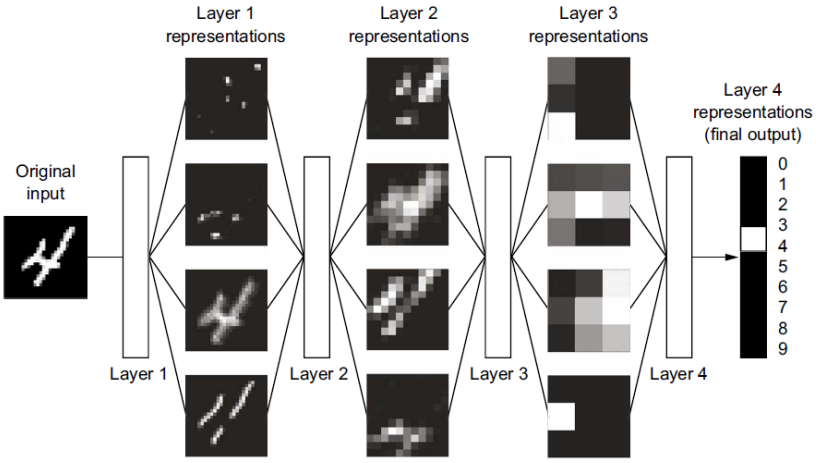

Figura 8 – Representação de um modelo de aprendizado tipo *Deep Learning*

Fonte: (FRANÇOIS, 2021).

Redes neurais profundas, também conhecidas como CNN, são ferramentas poderosas para modelos

2.3.3.1. *Convolutional Neural Network* (CNN) e imagens

Imagens são armazenadas no computador como um conjunto de matrizes com valores de cada pixel. A Figura 9 mostra como uma imagem de 4 x 4 pixels colorida é armazenada pelo computador. Basicamente, cada cor (ou canal) é armazenada em uma matriz da mesma dimensão da imagem, gerando uma matriz tridimensional, ou como é conhecido na computação gráfica, um tensor de dimensão (4, 4, 3).

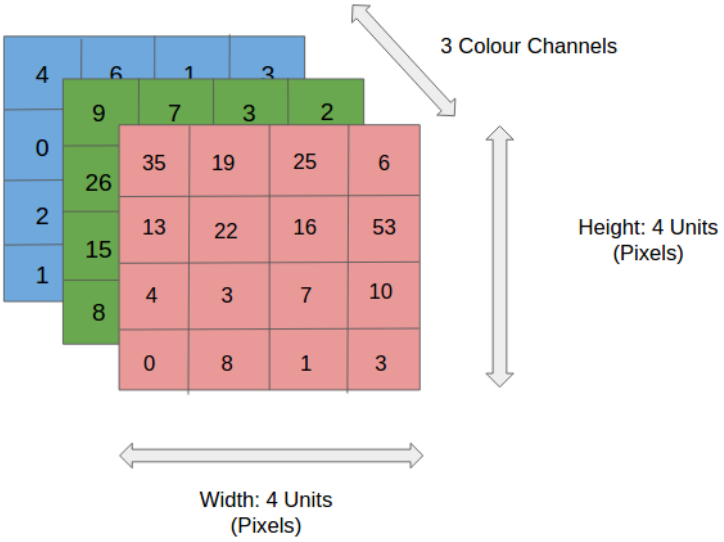

Figura 9 – Representação de uma imagem RGB

Fonte: (SAHA, 2018).

As CNNs são um tipo específico de rede neural profunda que se mostraram altamente eficazes no processamento de dados de grade, como imagens. Elas são projetadas para extrair automaticamente características relevantes de uma imagem, identificar padrões e reconhecer objetos. As CNNs têm camadas de convolução que aplicam filtros a partes da imagem e camadas de pooling que reduzem a dimensionalidade dos dados, preservando as características mais importantes (LI, 2021).

As CNNs são amplamente usadas em tarefas de visão computacional, como classificação de imagens, detecção de objetos, segmentação de imagens e reconhecimento facial. Elas aprenderam a representar e identificar características complexas em imagens, permitindo que os sistemas façam previsões precisas com base nessas características.

Uma das principais vantagens das CNNs e do aprendizado profundo em geral é a capacidade de aprender representações hierárquicas dos dados. Isso significa que as redes podem extrair automaticamente características simples nas camadas iniciais e, em camadas mais profundas, combinar essas características para formar representações mais complexas e abstratas (ALBAWI, 2017).

As principais operações em uma CNN é a convolução e o agrupamento (*pooling*). Abaixo será mostrado o princípio de cada uma dessas operações.

A convolução consiste em aplicar um filtro com pesos que vai “deslizando” por toda a imagem multiplicando cada pixel da imagem pelo respectivo peso do filtro. Esta operação pode ser usada para extrair características da imagem, aumentar ou diminuir o tamanho da imagem. O filtro, que também é conhecido por *kernel*, é formado por pesos inicializados aleatoriamente,

e vão sendo atualizados a cada nova entrada durante o processo de *backpropagation*. A pequena região da entrada onde o filtro é aplicado é chamada de *receptive field* (SAHA, 2018).

A figura 10 mostra a aplicação de um filtro em uma imagem monocromática (com um único canal) com o objetivo de realçar as linhas verticais e remover informações desnecessárias da imagem.

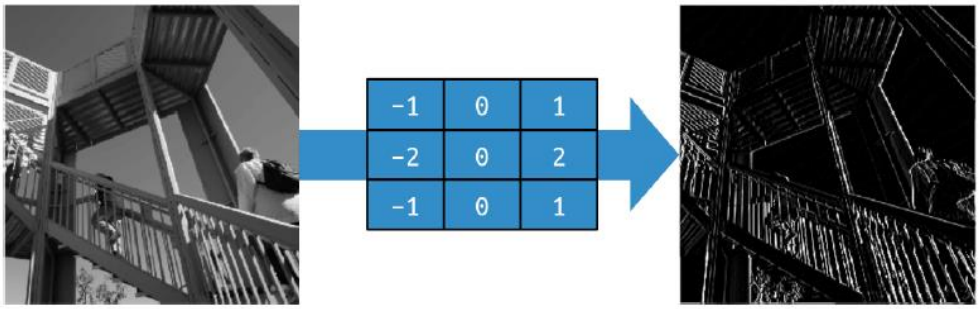

Figura 10 – Filtro para obter linhas verticais

Fonte: (MORONEY, 2020).

A operação de *pooling*, por sua vez, serve para diminuir o tamanho da imagem e consiste em "deslizar" uma janela de tamanho definido, por exemplo 2 x 2, aplicando operações estatísticas, como máximo ou média, em um conjunto de pixels sob essa janela.

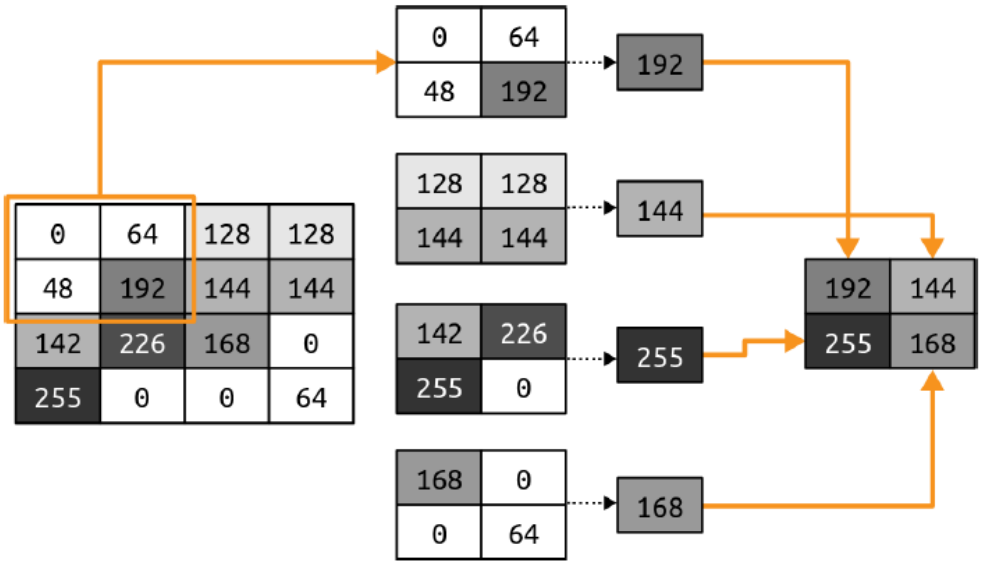

Figura 11 – Demonstração da operação *max pooling*

Fonte: (MORONEY, 2020).

A Figura 11 mostra a aplicação de uma operação *max pooling* em uma matriz 4 x 4 (ou imagem monocromática de tamanho 4 x 4 pixels com profundidade de imagem de 8 bits – valores possíveis para cada pixel variam entre 0 e 255) reduzindo a dimensionalidade de uma

imagem 4 x 4 para 2 x 2, ou seja, 25% menos pixels. Essa operação é extremamente útil para reduzir a quantidade de dados processados em uma CNN.

2.4. Audio Augmentation

Audio Augmentation significa estender os dados existentes realizando transformações, mas com preservação de rótulos para aumentar a variedade de dados de áudio disponíveis para treinar modelos de aprendizado de máquina, melhorando assim o desempenho e a capacidade de generalização do modelo. Essas transformações não modificam o conteúdo semântico dos dados, mas introduzem variações anteriormente invisíveis nos dados.

Exemplos simples de aumento de dados são adição de ruído de fundo para dados de áudio e rotação para dados de imagem. Além de usar o aumento de dados para criar dados ruidosos a partir de dados limpos existentes, também pode ser usado para criar mais dados quando não há dados suficientes disponíveis. Além disso, dados adicionais diminuem a chance de overfitting e, portanto, melhoram o desempenho. Diferentes técnicas de aumento para dados de áudio são discutidas na próxima seção (VIRTANEN, 2018).

Várias técnicas de aumento de dados de áudio são apresentadas na literatura. Essas técnicas modificam, por exemplo, a relação sinal-ruído (SNR), os tempos de reverberação e a intensidade dos sons. Algumas técnicas de aumento de dados, como mudança de tom e alongamento de tempo, são implementadas via software utilizando apenas os áudios existentes. Outras podem exigir dados externos, como gravações de ruído de fundo ou respostas de impulso, embora usá-los exija apenas operações simples de adição e convolução (EKLUND, 2019).

Algumas das principais técnicas de aumento de áudio são:

Deslocamento no tempo: Essa técnica envolve atrasar ou adiantar o áudio em relação ao tempo original, introduzindo variações temporais. Isso pode ajudar a tornar o modelo mais robusto a pequenas mudanças de tempo nas amostras de áudio.

Mudança na velocidade: Alterar a taxa de reprodução do áudio, aumentando ou diminuindo a velocidade, é outra técnica comum. Isso pode simular diferentes taxas de gravação e sotaques, por exemplo.

Mudança de tom: Alterar o tom do áudio pode simular variações nas características vocais, como idade ou gênero. Isso pode ser feito ajustando a frequência do áudio.

Adição de ruído: A introdução de ruído branco ou outros tipos de ruído ao áudio pode ajudar a tornar o modelo mais resistente a ambientes ruidosos.

Mascaramento de frequência e tempo: Aplicar máscaras de frequência e tempo ao áudio pode remover partes específicas do sinal de áudio, o que pode ser útil para treinar modelos a focar em informações relevantes.

Alteração de amplitude: Ajustar a amplitude do sinal de áudio pode simular variações na intensidade da fala ou do som ambiente.

Reversão temporal: Inverter a ordem das amostras de áudio pode criar uma representação completamente diferente do sinal, o que pode ser útil para treinar modelos a reconhecer padrões em áudio de maneira mais robusta.

Concatenação e mistura: Combinação de várias amostras de áudio pode criar sequências mais longas e variadas para treinamento.

Essas técnicas podem ser usadas individualmente ou em combinação para aumentar a diversidade dos dados de áudio de treinamento, melhorando assim a capacidade do modelo de se adaptar a uma ampla gama de condições e variações. Os *scripts* utilizados disponibilizados pela biblioteca TensorFlow para extração de *features* de áudios possuem parâmetros para execução de algumas dessas técnicas de *Audio Augmentations5* .

2.5. TensorFlow Lite Micro

A biblioteca TensorFlow Lite Micro é uma versão reduzida para microcontroladores da versão TensorFlow completa para computadores pessoais. Assim, o treinamento da rede deve ser feito utilizando a infraestrutura de um computador pessoal e depois o modelo é convertido para a versão compatível com a biblioteca para microcontroladores escrita na linguagem C++. Não é possível treinar modelos de *ML* em microcontroladores utilizando a biblioteca TensorFlow Lite Micro (TENSORFLOW, 2023).

A Figura 12 mostra as etapas percorridas, começando pela definição do modelo, treinamento utilizando a biblioteca TensorFlow em Python, até a implantação do modelo treinado utilizando a biblioteca TensorFlow Lite Micro para microcontroladores.

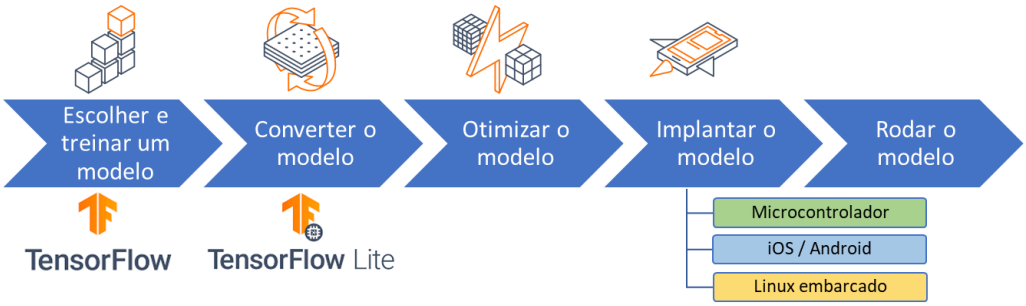

Figura 12 – Passo a passo para criação de um modelo TensorFlow Lite Micro

Fonte: (AUTOR, adaptado de TENSORFLOW, 2023).

A biblioteca TensorFlow Lite Micro original (disponível no site do projeto) consiste em um conjunto de arquivos .h (*header* file) e .cc (*C++ source file*), contudo vários fabricantes de placas, como a Arduíno, portaram a biblioteca para suas plataformas de desenvolvimento fazendo ajustes no código original para que fosse possível a utilização de recursos específicos em suas placas, como é o caso da placa Arduino Nano 33 BLE Sense que possui periféricos voltados para sensoriamento, como microfone, giroscópio e sensor de proximidade (ARDUINO, 2023). Assim, a biblioteca Arduino_TensorFlowLite é a versão para Arduino da biblioteca TensorFlow Lite Micro.

Em Warden et al. (2019), os autores da biblioteca detalham toda a estrutura do TensorFlow Lite Micro descrevendo cada subdiretório e a função de cada arquivo de código utilizado na biblioteca. A tabela abaixo mostra os diretórios e arquivos mínimos necessários para usar biblioteca TensorFlow Lite Micro em microcontroladores.

Nome	Tipo	Função
micro_mutable_op_resolver.h	Arquivo	Fornece as operações usadas pelo interpretador para executar o modelo
micro_error_reporter.h	Arquivo	Gera informações de depuração
micro_interpreter.h	Arquivo	Contém código para carregar e executar modelos
schema_generated.h	Arquivo	Contém o esquema para o formato de arquivo de modelo *TensorFlow Lite FlatBuffer* utilizado no model.cc
version.h	Arquivo	Fornece informações de versão para o esquema do TensorFlow Lite.
model.h e model.cc	Arquivo	Modelo fornecido como um vetor C++ gerado, convertido e exportado com a biblioteca TensorFlow para computadores (em Python)
micro	Diretório	Diretório raiz que contém a maioria dos arquivos mostrados acima
kernel	Diretório	Contém implementação de operações e o código associado
examples	Diretório	Contém código de exemplos

Tabela 1 – Principais arquivos e diretórios da biblioteca TensorFlow Lite Micro

Fonte: (AUTOR, 2023).

Além desses arquivos, para cada plataforma ou exemplos existem arquivos específicos para o fornecimento de recursos da placa, os quais serão detalhados no capítulo de desenvolvimento.

2.6. Random Forest

Random Forest (*RF*) é simplesmente uma coleção de árvores de decisão que são geradas usando um subconjunto aleatório de dados. O nome "*Random Forest*" vem da combinação da aleatoriedade que é usada para escolher o subconjunto de dados com um monte de árvores de decisão, portanto, uma floresta (HARTSHORN, 2016).

Para entender uma *RF*, é preciso entender uma Árvore de Decisão. Árvores de decisão são um método popular em aprendizado de máquina e análise de dados para tomada de decisões baseada em regras. Elas são estruturas de árvore que representam um conjunto de decisões e suas possíveis consequências. O processo de criação de uma árvore de decisão envolve dividir o conjunto de dados em subconjuntos menores com base em características ou atributos, a fim de fazer previsões ou tomar decisões (HASTIE, 2009).

Cada nó da árvore representa uma decisão ou teste em um atributo, e as arestas (ramos) que saem desse nó representam as possíveis saídas dessa decisão. As folhas da árvore representam os resultados ou as previsões.

Árvores de decisão podem ser usadas em uma variedade de tarefas, incluindo classificação, regressão e tarefas de tomada de decisões complexas. Elas são conhecidas por sua fácil compreensão e facilidade de visualização, pois permitem entender o processo de tomada de decisão de forma transparente.

A Figura 13 mostra um exemplo de uma árvore de decisão para ajudar a decidir sobre o tipo de veículo a ser adquirido.

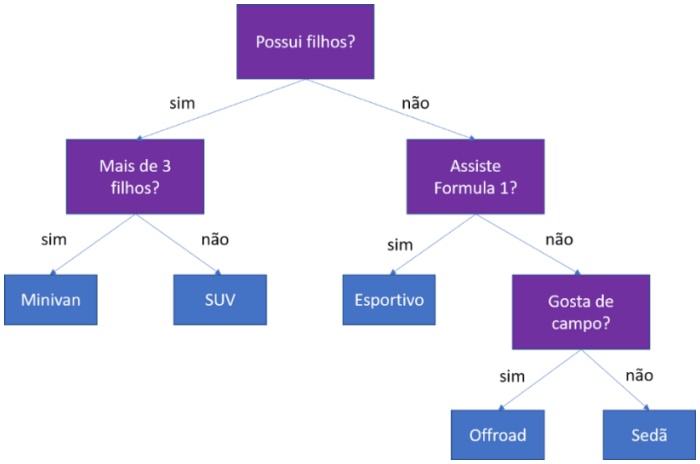

Figura 13 – Exemplo de árvore de decisão para escolha do tipo de veículo

Fonte: http://bit.ly/3RJHAVV. Acesso em: 25 de junho de 2023.

Uma *RF* é composta por várias árvores de decisão, cada uma delas gerada de forma ligeiramente diferente. Para gerar essas árvores de decisão, você precisa de um conjunto de dados inicial. Esse conjunto de dados inicial precisa ter recursos e resultados. Os resultados são a resposta final que você obtém quando está tentando categorizar algo. No exemplo acima, o resultado é o tipo de veículo a ser adquirido. Você precisa de um conjunto de dados onde conheça os resultados para começar, para que possa usá-los para gerar as árvores de decisão.

2.7. Sistemas embarcados

Sistemas embarcados são sistemas de computação dedicados a tarefas específicas e incorporados em dispositivos eletrônicos, como microcontroladores, sistemas de controle automotivo, dispositivos médicos, eletrodomésticos inteligentes, entre outros. Eles são projetados para realizar funções específicas de maneira eficiente e confiável (HEATH et al., 2003).

Dentre as características principais dos sistemas embarcados, pode-se citar:

Propósito Específico: Os sistemas embarcados são projetados para executar tarefas específicas ou funções dedicadas. Eles não são computadores gerais, como PCs ou laptops, que executam uma variedade de aplicativos.

Restrições de Recursos: Geralmente, os sistemas embarcados têm recursos limitados em termos de poder de processamento, memória e armazenamento. Essas restrições são impostas para otimizar o desempenho e a eficiência em seu ambiente específico.

Tempo Real: Muitos sistemas embarcados operam em tempo real, o que significa que eles devem responder a eventos e entradas em um tempo determinado. Isso é crítico para aplicações como sistemas de controle de automóveis e dispositivos médicos.

Hardware e Software Integrados: Os sistemas embarcados envolvem tanto hardware quanto softwares personalizados. O hardware é projetado para atender às necessidades da aplicação, e o software é desenvolvido para controlar o hardware e realizar as tarefas necessárias.

Ciclo de Vida Prolongado: Muitos sistemas embarcados têm um ciclo de vida prolongado, o que significa que eles precisam funcionar de maneira confiável por um longo período. Isso é comum em sistemas industriais e equipamentos incorporados.

Comunicação: Muitos sistemas embarcados precisam se comunicar com outros dispositivos ou sistemas. Isso pode envolver interfaces de comunicação, como UART, SPI, I2C, Ethernet ou sem fio.

Segurança e Confiabilidade: A segurança e a confiabilidade são fundamentais em sistemas embarcados, especialmente em aplicações críticas, como sistemas médicos, aeroespaciais e de automóveis.

Está disponível no mercado uma infinidade de placas com microcontroladores e conjuntos de entradas e saídas e interfaces de comunicação, as quais podem ser utilizadas para embarcar sistemas computacionais. Para esse trabalho optou-se por utilizar uma placa Arduino Nano 33 BLE Sense.

2.7.1. Arduino Nano 33 BLE Sense

O Arduino Nano 33 BLE Sense é uma placa de desenvolvimento baseada em microcontrolador da família Arduino. Ela é projetada para permitir a criação de projetos de *IoT* (*Internet of Things*) e dispositivos vestíveis (*wearables*) com recursos avançados de sensores e conectividade sem fio. (ARDUINO, 2023).

A Figura 14 mostra uma foto da placa Arduino Nano 33 BLE Sense Rev2 com *headers* (pinos para conexão das entradas e saídas).

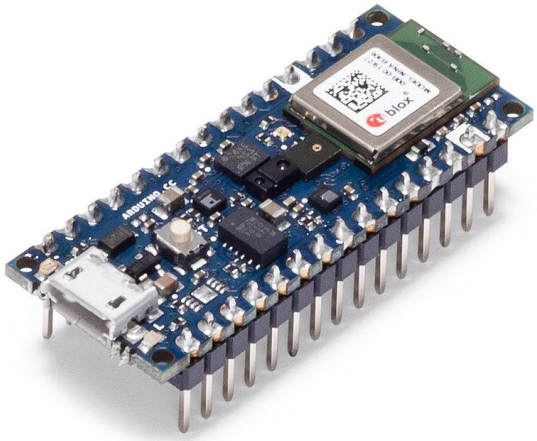

Figura 14 – Arduino Nano 33 BLE Sense Rev2 com *headers*
Fonte: (ARDUINO, 2023).

Algumas das características e funcionalidades do Arduino Nano 33 BLE Sense são:

Microcontrolador ARM: Ele é equipado com um microcontrolador Nordic Semiconductor nRF52840, que é baseado na arquitetura ARM Cortex-M4. Isso fornece poder de processamento e capacidade de memória suficientes para executar aplicativos de *IoT*.

Conectividade Bluetooth: O BLE no nome refere-se ao *Bluetooth Low Energy*, que permite a comunicação sem fio com outros dispositivos compatíveis, como smartphones, tablets e outros dispositivos *IoT*.

Sensores Embutidos: O Nano 33 BLE Sense possui uma variedade de sensores embutidos, incluindo um acelerômetro de 9 eixos, giroscópio, magnetômetro, sensor de umidade relativa, temperatura ambiente, pressão atmosférica e um sensor de luz ambiente.

Microfone MEMS: Ele também inclui um microfone MEMS (*Micro-Electro-Mechanical Systems*) para captura de áudio.

Entradas/Saídas (I/O): A placa possui uma série de pinos de entrada/saída digital e analógica, permitindo a conexão de componentes externos e sensores adicionais.
Alimentação: Pode ser alimentado por meio de uma porta USB ou bateria.
Suporte de Software: O Arduino Nano 33 BLE Sense é compatível com a plataforma de desenvolvimento Arduino, o que facilita a programação e o desenvolvimento de projetos. Ele também é suportado pelo Arduino IoT Cloud.

Com todas essas características, o Arduino Nano 33 BLE Sense tornou-se uma escolha popular para projetos de *IoT*, *wearables* e projetos de sensores ambientais, onde a conectividade sem fio e sensores avançados são necessários. Essa placa foi escolhida por ser a mais barata da família Arduino a possuir sensores acoplados, como microfone, bem como por ser, oficialmente, compatível com a biblioteca TensorFlow Lite Micro do Google (TensorFlow, 2023).

3. Trabalhos relacionados

Neste capítulo é apresentada de forma sucinta uma revisão sobre os trabalhos relacionados ao emprego de *ML* na indústria para diagnóstico de máquinas e equipamentos e mostrado a contribuição do presente trabalho.

3.1. Uso de *Machine Learning* na indústria

Em (RONGJIE et al., 2013) é apresentado um método inovador de diagnóstico de falhas em retificadores trifásicos, com base na Análise de Componentes Principais (*PCA*) e *Support Vector Machine* (*SVM*). O processo começa com a análise de sinais de falha por meio do *PCA* para extrair características relacionadas a diferentes tipos de falha. Em seguida, um classificador de reconhecimento de padrões baseado em SVM é utilizado para identificar os tipos de falha.

Foi utilizado o software Simulink 6.4 do Matlab R2006a para simulação de 13 tipos de falhas em um retificador trifásico controlável de ponte completa. Até mesmo a operação normal foi considerada como uma forma de falha para demonstrar a capacidade do método em distinguir entre correntes normais e anormais em retificadores de eletrônica de potência. Todas as análises foram feitas utilizando o formato de onda da tensão na saída do retificador trifásico (na carga).

Para validar o método proposto, foram usados conjuntos de amostras de três classes diferentes de falhas, incluindo variações no ângulo de disparo dos tiristores e adição de ruído gaussiano branco com diferentes relações sinal/ruído (*SNR*). Os resultados mostraram que o método proposto foi capaz de identificar com precisão todos os tipos de falhas e a localização dos elementos com falhas em todas as amostras de teste, incluindo cenários com dois tiristores com falha.

O artigo se concentrou-se em duas tecnologias-chave: extração de características de falha e diagnóstico de falhas, utilizando *PCA* e *SVM*. Os resultados da simulação demonstram um desempenho superior em termos de robustez ao ruído e complexidade de cálculo em comparação com métodos convencionais.

Em (GLOWACZ, 2018) o autor descreveu uma técnica de diagnóstico de falhas baseada em sinais acústicos para motores de indução trifásicos. Foram analisados quatro estados do motor: saudável, com uma barra de rotor quebrada, com duas barras de rotor quebradas e com anel defeituoso na gaiola de esquilo. Duas técnicas de extração de características foram

avaliadas. Foram usados três métodos de reconhecimento: *KNN*, rede neural de *backpropagation* e um classificador modificado baseado em codificação de palavras.

Os resultados da detecção de sinais acústicos foram muito bons para dados reais, com uma taxa de reconhecimento variando de 88,19% a 100%. Além disso, as técnicas de diagnóstico de falhas acústicas apresentadas não são caras, com um custo acessível, incluindo um laptop e um microfone de capacidade por cerca de 350 dólares. As vantagens dessa técnica incluem sua não invasividade, baixo custo e medição instantânea dos sinais acústicos. No entanto, o estudo se concentrou em quatro estados do motor, sugerindo que as técnicas propostas também podem ser aplicadas a mais estados no futuro. Além disso, há a possibilidade de combinar várias técnicas de diagnóstico de falhas, como imagem térmica, análise de vibração e análise de corrente elétrica, para melhorar ainda mais o diagnóstico de falhas em motores elétricos rotativos.

Em (KOU et al., 2020) é proposto o diagnóstico de falhas em retificadores trifásicos que baseados em *PWM* (*Pulse Width Modulation*), utilizando uma abordagem baseada em uma rede neural *feedforward* profunda (*DFN*). O trabalho focou na geração de características sintéticas transitórias para a detecção de falhas. Essas características transitórias são informações relevantes que surgem durante o processo de operação do retificador e podem indicar problemas, as quais foram usadas para treinar o classificador da *DFN*. A precisão do diagnóstico de falhas alcançou precisão de 97,85% com essas características sintéticas, representando um aumento de mais de 1% em relação às características transitórias originais.

Além disso, o método foi aplicado a experimentos de diagnóstico online, com resultados precisos na localização de falhas em IGBTs. O uso de múltiplos resultados contribui para a precisão e confiabilidade dos diagnósticos. Comparado a outros métodos, a abordagem das características sintéticas transitórias aumentou significativamente a precisão do diagnóstico de falhas e demonstrou possuir aplicabilidade generalizada em sistemas de eletrônica de potência.

Em resumo, o artigo propôs um método de diagnóstico de falhas baseado em redes neurais profundas com características transitórias sintéticas para retificadores de *PWM* de três fases. Esse método se destaca por seu caráter data-driven e sua capacidade de aprimorar a precisão do diagnóstico, sendo uma solução eficaz para identificar falhas em IGBTs em sistemas de eletrônica de potência.

Em Ericeira (2020) é apresentado métodos de detecção de defeitos em rolos de transportadores de correia por meio de sensoriamento ultrassônico dos sinais por eles emitidos. O autor utilizou um sensor comercial capaz de captar sons nas frequências de 20 kHz até 100

kHz. O conjunto de dados de áudios de rolos bons e defeituosos foi utilizado para testes de desempenho em dois algoritmos de aprendizagem: *Random Forest* e *Multilayer Perceptron*. Dentre os experimentos realizados pelo autor, o de maior acurácia alcançou média de 83,68% quanto a classificação correta dos rolos, tendo obtido 90% de acerto no melhor resultado dentre todos os testes.

Da Cruz (2020) também propôs um algoritmo baseado em aprendizado de máquinas para detecção de falhas de rolos de transportadores de correia. O trabalho de Da Cruz (2020) se diferenciou do trabalho de Ericeira (2019) principalmente pelo método utilizado para gravação dos ruídos emitidos pelos rolos dos transportadores de correia. Em Da Cruz (2020) o autor utilizou um microfone do tipo direcional convencional com taxa de amostragem de 96 kHz. Dentre os três algoritmos de *ML* utilizados, o que obteve melhor acurácia foi o *KNN* (K-*nearest neighbors*) com precisão de 99,1%.

Santos (2020) utilizou uma *CNN* para identificação de sujeira nas estruturas de transportadores de correia a partir do processamento de imagens RGB obtidas por uma câmera profissional FLIR AX8. Na avaliação do modelo com dados novos do campo a acurácia foi de 89,20% no melhor cenário.

Em (GRANDHI et al., 2021) foi proposto um modelo que envolveu a coleta de sinais acústicos de diferentes motores elétricos por meio de um telefone celular portátil, o sinal foi filtrado para remover o ruído ambiente. Esses sinais filtrados foram divididos em amostras menores para extrair recursos computando parâmetros de domínio de tempo e frequência, como média, mediana, assimetria, curtos e frequência de pico.

Usando esse conjunto de dados gerado, os sinais foram classificados aplicando vários algoritmos de aprendizado de máquina, como Árvore de Decisão, *KNN* e *SVM*. A precisão de 90,55% para classificação dos motores defeituosos dos não defeituosos foi obtida com o *KNN*, enquanto o modelo com *SVM* e *Decision Tree* (*DT*) mostrou uma precisão de 85,66% e 87,76% respectivamente. Esta abordagem facilita a detecção de motores defeituosos, fazendo uso de um simples aplicativo de celular.

3.2. Contribuições desse trabalho

O presente trabalho se diferencia dos demais trabalhos apresentados ao propor uma solução de classificação e diagnóstico de falhas em retificadores trifásicos através de sinais sonoros emitidos por esses equipamentos, ao invés de análises convencionais baseadas em sinais de tensão e corrente presentes na saída dos retificadores. O diagnóstico através de sinais acústicos possui a vantagem de não exigir contato com o equipamento ou manipulação de qualquer grandeza elétrica, além de obter o resultado rapidamente.

Outro ponto de contribuição, foi a criação de uma base de dados sonora de retificadores trifásicos instalados em campo com suas respectivas classificações.

Ao embarcar uma solução de *ML* em uma placa Arduino é gerado também um produto portátil e bastante eficaz, do ponto de vista de precisão e consumo, capaz de identificar o estado de retificadores trifásicos com base nos sons emitidos por estes equipamentos.

4. Materiais e métodos

Neste capítulo são apresentados os materiais e os procedimentos metodológicos que foram empregados para alcançar o objetivo de fazer um algoritmo de aprendizado capaz de diferenciar se um retificador está defeituoso ou não através do som emitido por ele.

4.1. Exemplo *micro_speech* do TensorFlow Lite Micro

Dentre os vários exemplos disponibilizados pela biblioteca TensorFlow Lite Micro está o exemplo *micro_speech*, que consiste em treinar uma CNN para reconhecimento de palavras simples. Esse exemplo, originalmente, utiliza um conjunto de dados chamado *Speech Commands Dataset*, que consiste em mais de 105.000 arquivos de áudio WAVE de pessoas dizendo trinta palavras diferentes, como '*up'*, '*down'*, '*left,* '*right' etc.* Esses dados foram coletados pelo Google e divulgados sob uma licença CC BY. (TensorFlow, 2023).

Por se tratar de um exemplo com o uso de reconhecimento de áudio, o presente trabalho tomou o exemplo *micro_speech* como ponto de partida para treinar o mesmo modelo com novos áudios das classes dos retificadores no lugar dos áudios originais de palavras. O exemplo é disponibilizado como um *Notebook Jupyter* com vários arquivos Python de suporte que são importados e usados ao longo do processo.

Como todo modelo de ML para microcontrolador, o exemplo se divide em duas partes: A primeira realizada no PC (Personal Computer) e a segunda realizada no microcontrolador. As duas partes serão descritas brevemente nas seções seguintes. Já no Capítulo 5 será mostrado com mais detalhes quando for tratado os ajustes necessários para tornar o modelo compatível com o propósito desse trabalho.

4.1.1. Tarefas executadas no PC

É utilizado o arquivo do *Notebook Jupyter* chamado *micro_speech.ipynb*, o qual realiza o treinamento da CNN – o que envolve pré-processamento do áudio, definição do modelo *Keras/TensorFlow* e treinamento da CNN em si, congelamento do modelo e conversão do modelo para o formato TensorFlow Lite Micro.

Na Figura 15 é mostrado os componentes utilizados na etapa realizada no PC. Logo abaixo segue uma descrição de cada um dos componentes baseada nos comentários dos

desenvolvedores presente nos códigos-fonte de cada arquivo do exemplo ou de bibliotecas utilizadas no exemplo.

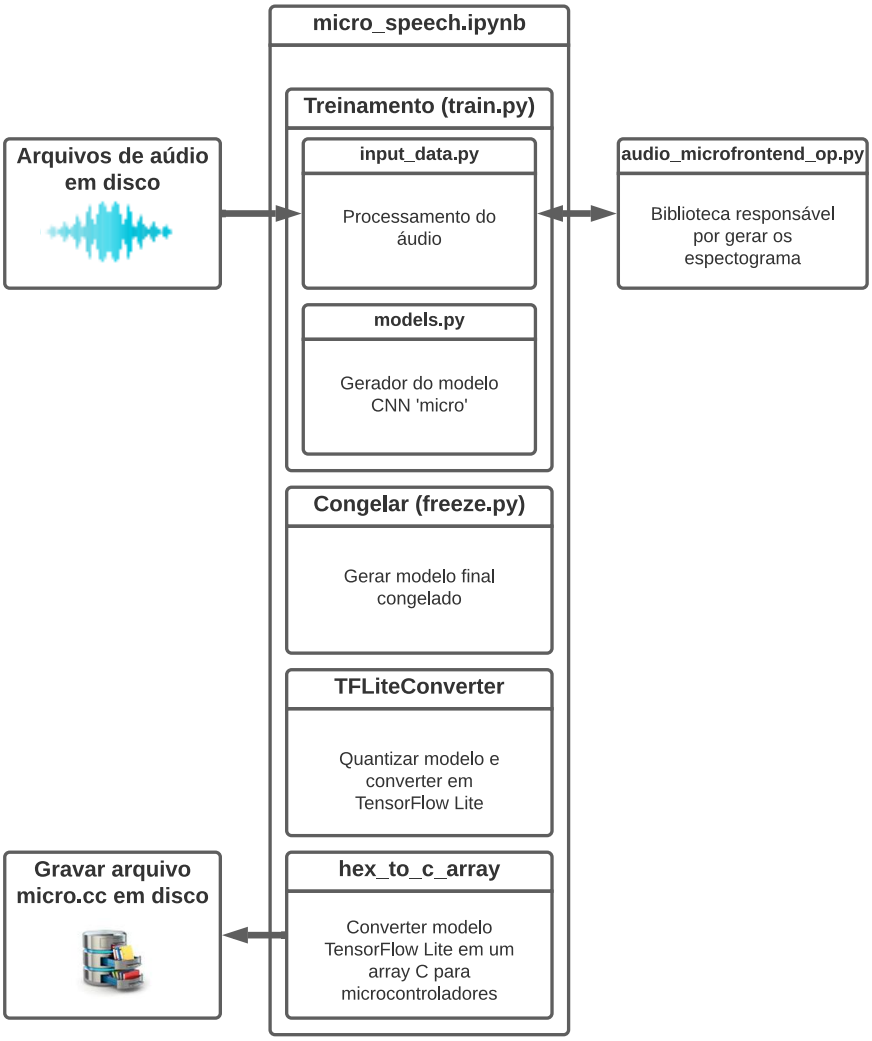

Figura 15 – Componentes do exemplo micro_speech (PC)

Fonte: (AUTOR, 2023).

***micro_speech.ipynb*:** É o arquivo de Notebook Jupyter que contém todo o código do exemplo. Isso inclui a parametrização, leitura dos arquivos em disco, pré-processamento, treinamento e conversão, como será detalhado nos demais elementos.

***train.py*:** É o arquivo *script* do *notebook*. Responsável por fazer a interface dos dados de entrada (arquivos de áudio) com o *script input_data.py* e gerar o modelo CNN a ser treinado com a ajuda do *script models.py*. O script *train.py* é responsável por treinar o modelo CNN com os dados (*features*) extraídos dos áudios, bem como repassar uma série de parâmetros recebidos para o script extrator de *features*. A saída do processo de treinamento são arquivo de pontos de checagem (*checkpoint files*) gerados a cada 1000 *loops* de treinamento. Todos esses números

são parametrizáveis no *notebook Jupyter* e são passados como argumento na chamada do *script train.py*.

***input_data.py*:** Esse *script* faz a interface recebe uma série de parâmetros repassados pelo *script train.py* e, de acordo com alguns parâmetros escolhe qual será o formato dos features extraído dos áudios. Para o exemplo micro_speech, o *script input_data.py* faz uso da biblioteca *audio_microfrontend_op.py*, que será descrita abaixo.

***audio_microfrontend_op.py*:** Essa biblioteca recebe vários parâmetros e o fluxo de dados do arquivo *train.py*, através do arquivo *input_data.py*, e realiza uma série de operações sobre os dados do áudio no domínio do tempo até entregar a matriz de dimensão 49 x 40 representando as *features* no domínio da frequência. Os parâmetros recebidos nesse *script* alteram significativamente a precisão do modelo embarcado no microcontrolador, por isso, será dado uma ênfase a esse script e seus parâmetros no Capítulo 5.

models.py: Esse script recebe do script train.py a informação de qual modelo utilizar para o treinamento da CNN e monta o modelo utilizando a API do TensorFlow. Os modelos são fixos e para esse trabalho foi utilizado o mesmo modelo CNN utilizado para o exemplo *micro_speech*. Nas seções abaixo será dado mais detalhe do modelo utilizado. A saída do processo de

freeze.py: Converte um ponto de verificação (*checkpoint file*) treinado em um modelo congelado (estático) para inferência em dispositivos móveis. Depois de treinar um modelo usando o *script train.py*, você pode usar esta ferramenta para convertê-lo em um arquivo *GraphDef* binário que pode ser carregado no TensorFlow Lite para Android, iOS ou Raspberry Pi. A saída desse script é um arquivo *tiny_conv.pb*. Em resumo, congelar um modelo no TensorFlow é uma etapa crucial na transição do modelo treinado para um ambiente de produção. Ele melhora a eficiência, segurança e compatibilidade do modelo, tornando-o pronto para uso em aplicativos do mundo real.

TFLiteConverter: Este componente faz parte da biblioteca do TensorFlow para PC e é responsável por converter o modelo estático para um formato de dados chamado *FlatBuffers*, que é uma serialização de dados hierárquicos. Isso permite armazenar o modelo como um único vetor plano, de fácil implementação em qualquer linguagem de programação. O formato *FlatBuffers* trabalha usando esquemas (*schemes*), os quais definem a estrutura de dados a ser serializada. Esse estágio gera dois arquivos: *float_model.tflite*, que é o modelo com entradas e pesos em *float*, e *model.tflite*, que é o modelo com as entradas e pesos quantizados como int8 (-128 a 127).

hex_to_c_array: Função definida em Python para converter o arquivo do *TensorFLow Lite* gerado (*model.tflite*) para um arquivo tipo C++ chamado model.cc com a definição de um vetor do tipo char no formato hexadecimal para compilação junto com o código do TensorFlow Lite Micro.

Acima foi dado uma visão geral dos componentes envolvidos no treinamento do exemplo *speech_command*, o qual foi utilizado como base para a solução proposta nesse trabalho. Abaixo será dada uma visão geral de como o modelo é executado no microcontrolador (código em C++). No capítulo seguinte será visto as modificações necessárias nos códigos em Python e em C++ para tornar o modelo funcional para o problema de identificação dos retificadores nas três classes propostas.

Em Warden et al. (2019), os autores da biblioteca explicam com detalhes a implementação da biblioteca e mostram o funcionamento dos vários exemplos disponibilizados com a biblioteca TensorFlow Lite.

4.1.2. Tarefas executadas no microcontrolador

A Figura 16 dá uma visão geral de como o exemplo funciona no microcontrolador. Logo abaixo segue uma descrição de cada um dos componentes.

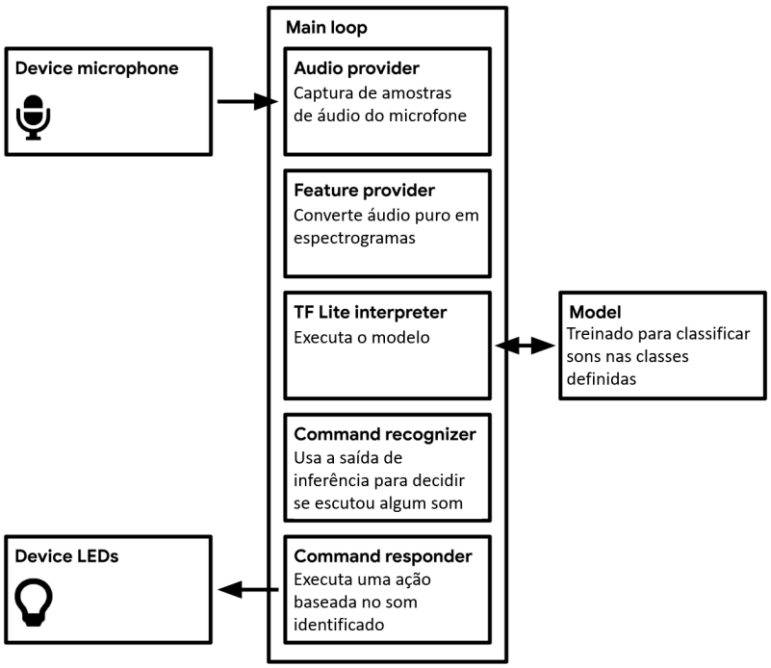

Figura 16 – Componentes do exemplo micro_speech (microcontrolador)

Fonte: (AUTOR, adaptado de (WARDEN, 2019)).

Main loop: No microcontrolador a aplicação roda em um loop contínuo. Todos os processos subsequentes estão contidos nele e são executados continuamente, tão rápido quanto o microcontrolador pode executá-los, ou seja, várias vezes por segundo.

Audio provider: O provedor de áudio captura dados de áudio brutos do microfone. Na biblioteca TensorFlowLite Micro para o Arduino Nano 33 BLE Sense este componente captura aúdio de um sensor de áudio (microfone) do tipo PCM embutido na placa.

Feature provider: O provedor de recursos converte dados de áudio brutos no formato de espectrograma exigido por nosso modelo. Ele faz isso continuamente como parte do loop principal, fornecendo ao intérprete uma sequência de janelas sobrepostas de um segundo. Para prover os espectrogramas é utilizada a biblioteca *microfrontend* escrita em C++, a mesma utilizada na escrita em Python durante o treinamento. Isso garante compatibilidade features utilizados no treinamento com os features extraídos dos áudios durante o processo de inferência. No Capítulo 5 será mostrado os parâmetros alterados no código em C++ para tornar a solução funcional com os áudios dos retificadores.

TF Lite interpreter: O intérprete executa o modelo TensorFlow Lite, transformando o espectrograma de entrada em um conjunto de probabilidades.

Model: O modelo é incluído como uma matriz de dados e executado pelo interpretador. A matriz está localizada em *model.cc* gerado durante ao fim do processo de treinamento no PC.

Command recognizer: Como a inferência é executada diversas vezes por segundo, a classe *RecognizeCommands* agrega os resultados e determina se, em média, uma palavra conhecida foi ouvida. Isso assegura que um resultado só será exibido após três constatações de que a identificação está correta.

Command responder: O *Command responder* usa os recursos de saída do dispositivo para informar o usuário. Para o Arduino Nano 33 BLE Sense a responta consiste em enviar para a saída serial qual áudio/categoria foi identificada e colorir o LED RGB de acordo com a categoria identificada.

4.1.3. Arquitetura do modelo TensorFlow Lite Micro

Ao pensar em uma solução de IA embarcada, deve-se tomar o cuidado com as operações de pré-processamento de dados, já que essas operações também precisam ser realizadas nos dados aquisitados pelo sistema embarcado antes de alimentar o modelo de *ML*. Outro ponto a ser observado é o tamanho final do modelo, já que há limitação de recursos de processamento, memória RAM e memória para armazenamento do modelo (WARDEN, 2019).

Baseado nessas limitações, foi escolhido o mesmo modelo CNN do exemplo *micro_speech* da própria biblioteca TensorFlow Lite Micro, o qual é adequado em tamanho e eficiência para o objetivo deste trabalho. Trata-se do modelo mostrado de forma gráfica na Figura 17.

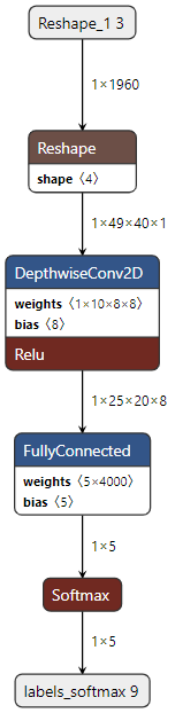

Figura 17 – Modelo CNN representado graficamente pelo software Netron

Fonte: (AUTOR, 2023).

O modelo em questão recebe como entrada a imagem do espectrograma redimensionada como um vetor de dados com dimensão de 1 x 1960, cuja dimensão original era 49 x 40. Esse formato de entrada serve para tornar o modelo compatível com o vetor de *features* do tipo int8 utilizado no microcontrolador para alimentar o modelo. Esse vetor é então redimensionado para o tamanho 1 x 49 x 40 x 1, recompondo a sua forma de imagem bidimensional embrulhada em duas outras dimensões: a primeira dimensão corresponde ao lote e a última aos canais (ou cores) da imagem. Esse formato com 4 dimensões é um formato padrão de entrada para a camada de operação de convolução.

Após ser redimensionado o dado passa por uma camada convolucional, na qual é aplicado 8 filtros de dimensão 10 x 8. A dessa camada produz uma imagem menor com 8 canais (um canal para cada filtro) e dimensão de 25 x 20.

Essa imagem de 8 canais alimenta uma camada totalmente conectada, a qual compara a entrada com pesos, gerando uma saída para cada classe. A camada *softmax* aprimora as pontuações, fornecendo a previsão do modelo. O processo inclui ativações ReLU e vieses. O resultado são quatro números, representando as categorias, com a maior pontuação indicando a previsão do modelo e sua confiança.

O modelo da Figura 14 se assemelha muito ao modelo apresentado na página do projeto Tensor Flow Lite Micro, diferindo apenas na primeira camada, já que para esse projeto houve um processo de quantização do modelo, processo em que a entrada do sistema e os pesos dos neurônios do modelo original, que estavam em formato de ponto flutuante – floatt16, foram quantizados em variáveis do tipo inteiro de 8 bits – int8. O processo de quantização, quando bem realizado, não implica em redução da acurácia do modelo, podendo reduzir seu tamanho em até 4x e acelerar em até 3x o seu processamento (TENSORFLOW, 2023).

4.2. Gravação de sons em campo

Idealmente, a gravação dos áudios de treinamento deveria ocorrer no mesmo dispositivo no qual o modelo foi embarcado, contudo, o hardware do Arduino Nano 33 BLE Sense empregado nesse projeto não possui capacidade de armazenamento devido as limitações de apenas 256 kB de memória SRAM e 1 MB de memória de flash, além de não possuir qualquer tipo de sistema operacional com sistema de arquivo para manipulação de leitura e escrita de dados. De fato, as bibliotecas de *ML* para microcontroladores disponíveis no momento não contemplam o processo de aquisição ou treinamento da rede pela plataforma embarcada (TENSORFLOW, 2023).

As gravações foram feitas por meio de um *smartphone* Google® Pixel 4 XL com uso de aplicativo específico para gravação de áudio configurado para gravar em formato *wave* puro (.wav) com uma taxa de amostragem de 32 kHz e 16 bits de resolução. Foram realizadas gravações de 5 minutos de 10 retificadores em bom estado, totalizando 50 minutos de áudio, 3 gravações de 5 minutos de retificadores ruidosos, porém que não apresentavam defeitos, totalizando 15 minutos de áudio e 2 gravações de 5 minutos de retificadores com defeitos, totalizando 10 minutos de áudio. Essas gravações foram agrupadas, respectivamente, em três grupos de retificadores: bons, medianos e ruins.

Na Figura 18 é possível ver o procedimento utilizado para gravar os áudios dos retificadores em campo.

Figura 18 – Procedimento de gravação dos sons com smartphone

Fonte: (AUTOR, 2023).

O menor número de áudios de retificadores com defeitos está relacionado ao fato da baixa disponibilidade equipamentos defeituosos em campo, já que são equipamentos que são substituídos imediatamente após falharem pela sua criticidade para o SEP. As gravações foram feitas nos retificadores das subestações da Cemig Distribuição da região do Leste de Minas Gerais sendo necessário 6 meses para conseguir amostra de 2 retificadores com defeito.

Para efeito de equilíbrio da base de dados, procurou-se manter, aproximadamente, o mesmo número de fragmentos com o tamanho de um segundo para todas as classes, fazendo um trabalho de mistura para as classes que possuem maior número de fragmentos.

4.3. Construção da base de dados para os modelos

Os arquivos de gravação foram pré-processados para equalização do volume divididos em trechos de um segundo utilizando a biblioteca Librosa para adequação ao formato de dados da biblioteca utilizada no treinamento do modelo. Todos os áudios passaram pelo processo de *resampling* passando de 32 kHz para 16 kHz. Isso se fez necessário para adequar o áudio de treinamento ao áudio disponível na placa Arduino, já que a placa possui taxa máxima de amostragem de 16 kHz. A Figura 12 mostra o espectro de frequência para áudios, tomados como exemplo, das três classes.

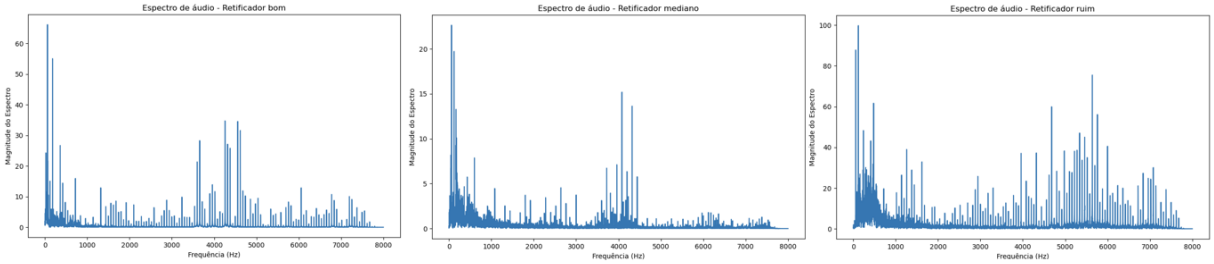

Figura 19 – Espectro de frequência de áudios das 3 classes

Fonte: (AUTOR, 2023).

Pelo Teorema da Amostragem de *Nyquist-Shannon* (OPPENHEIM, 2012), sinais amostrados na frequência de 16 kHz conseguem guardar informações de frequências até 8 kHz. Na Figura 19, é possível perceber que a maior diferença entre as três classes no domínio da frequência está em frequências superiores a 4 kHz, o que torna o nosso hardware disponível válido para a execução do modelo.

Foram realizadas duas abordagens distintas: Na primeira, utilizou-se a biblioteca TensorFlow para PC para extração de *features* dos áudios e treinamento do modelo do TensorFlow Lite Micro e posteriormente a biblioteca TensorFlow Lite Micro em C++ para embarcar o interpretador do modelo *CNN* treinado; Na segunda, utilizou-se a biblioteca Scikit-Learn para treinamento de um modelo de *RF* com os mesmos *features* extraídos com os *scripts* do TensorFlow e a biblioteca MicroMLGen para converter o modelo RF treinado em código C++ e permitir a portabilidade para o Arduino. Para as duas abordagens foram utilizados fragmentos de um segundo de áudio. As duas abordagens e seus resultados serão apresentados no próximo capítulo.

4.3.1. Definição das classes dos áudios

Inicialmente, foram definidas três classes para os áudios dos retificadores: “Bom”, “Mediano” e “Ruim”, que refletem os estados desses equipamentos. Contudo, o *script* de treinamento do exemplo *micro_speech* da biblioteca *TensorFlow Lite Micro* acrescenta mais duas classes às classes definidas pelo usuário. Uma classe oculta ao processo de personalização, chamada “*Silence*” e outra classe chamada de “*Unknown*”, a qual nesse trabalho foi renomeada como “Ambiente”.

Para definição das classes de áudio é necessário configurar a variável “*WANTED_WORDS*” com os nomes das classes separados por vírgula. Além disso, o *script* de

treinamento rotula os áudios em suas respectivas classes com base no nome da pasta em que eles estão armazenados. Por conta disso, foi necessário criar uma pasta para cada classe desejada.

O script de treinamento toma como referência para a classe "*Unknown*" os demais áudios armazenados em pastas com nomes diverso àqueles das classes definidas na variável "*WANTED_WORDS*". Sabendo disso, foi adicionada uma pasta com áudios de som ambiente, como ventiladores, barulho de passos e fala que foram gravados em ambientes diversos utilizando o mesmo *smartphone* usado nas gravações dos sons dos retificadores.

A definição da quarta classe de dados chamada "Ambiente" se mostrou interessante para que o algoritmo tornasse mais preciso e passasse a diferenciar os sons ambientes dos sons dos retificadores bons, que geralmente são mais silenciosos e, por isso, os áudios se assemelham aos dos sons ambientes.

4.3.2. Transformação dos áudios em imagens (espectrograma)

A ideia básica por detrás desse modelo é transformar os áudios em uma matriz, a qual pode ser visualizada e processada como uma imagem. A partir daí, pode-se utilizar modelos de *ML* voltados para o processamento de imagens para identificação de características “visuais” no espectrograma de cada áudio. Abaixo pode ser visto uma explicação resumida do processo de geração dos espectrogramas.

Os áudios passam pelo algoritmo de STFFT com uma janela de 30 milissegundos e saltos de 20ms para a construção do espectrograma. Cada janela de 30 milissegundos produz 240 informações de frequência, e cada conjunto de seis informações é tomada a média. Isso produz 40 informações de frequência (*frequency bins/buckets*) por janela (*slice*). A janela vai se movendo com salto de 20 milissegundos fazendo com que esse processo se repita por 49 vezes, conforme mostrado na Figura 16. O resultado desse processo é uma matriz de tamanho 49 x 40, também chamada de espectrograma do áudio, com o total de 1960 informações. Essa é a quantidade de neurônios da primeira camada do modelo.

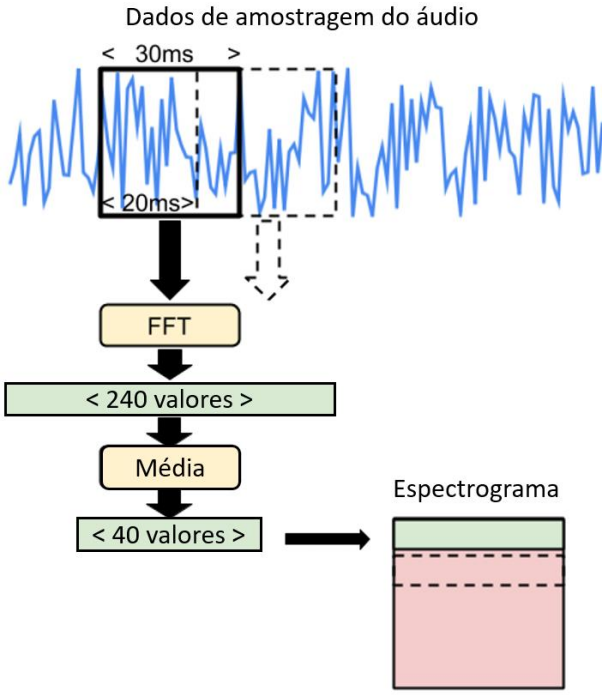

Figura 20 – Exemplo gráfico de aplicação da FFT em janelas de 30ms (STFFT)

Fonte: (AUTOR, adaptado de (TENSORFLOW, 2023)).

A matriz gerada por esse processo pode ser plotada como um espectrograma, conforme mostrado na Figura 20. Assim, todas as técnicas utilizadas em modelos de visão computacional podem ser utilizadas a partir desse ponto. É o que ocorre na camada do meio da rede. Já a

última camada possui uma função de ativação e apenas cinco neurônios, que é exatamente o número total de classes definidas no modelo.

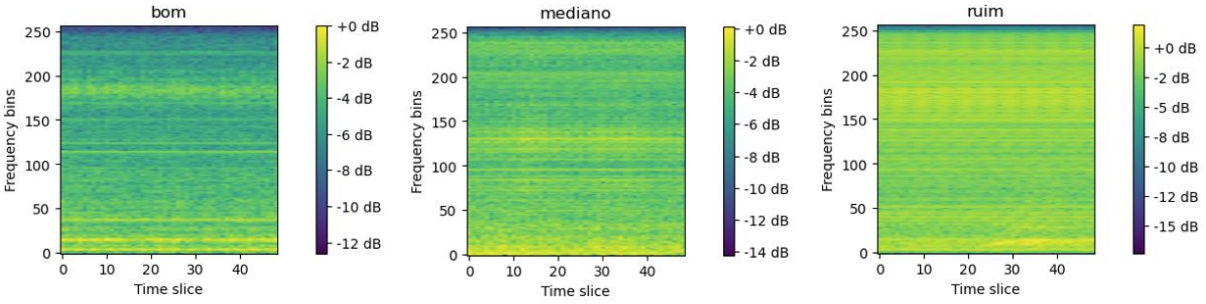

Figura 21 – Exemplos de espectrogramas das três classes utilizadas

Fonte: (AUTOR, 2023).

Através da Figura 17, é possível perceber que no domínio da frequência as características dos áudios de cada classe se tornam mais eminentes. Motivo pelo qual modelos que utilizam como entradas *features* de áudios no domínio da frequência tendem a ser mais precisos (TENSORFLOW, 2023).

5. Resultados e Avaliações

Neste capítulo serão apresentados os resultados alcançados e a avaliação dos modelos embarcados no Arduino Nano 33 BLE Sense. Todos os códigos e scripts utilizados neste trabalho estão disponíveis como um repositório público na plataforma GitHub no link: https://github.com/patrickpandrade/micro_classifier.

5.1. Definição de parâmetros

O *notebook Jupyter* utilizado como base para este trabalho contém vários parâmetros para o processo de treinamento, quantização e conversão do modelo. Esses parâmetros são passados para os scripts *train.py* e *freeze.py*, os quais são responsáveis por criarem interface com vários outros scripts e objetos da biblioteca TensorFlow utilizada neste trabalho.

O *notebook Jupyter* de base, chamado *micro_speech.ipynb*, foi renomeado para *micro_classifier.ipynb* para melhor descrever a finalidade do modelo e ser possível alterar os parâmetros sem que cause confusão com o *notebook* base.

Na Tabela 2 está uma lista dos parâmetros que exercem influência sobre a qualidade do treinamento, bem como a função de cada parâmetro.

PARÂMETRO	VALOR	*SCRIPT* DE EXECUÇÃO	FINALIDADE
BACKGROUND_FREQUENCY	0.8	*input_data.py*	Proporção dos áudios que serão usados para mixagem de sons de background para o processo de *Audio Augmentation*
BACKGROUND_VOLUME_RANGE	0.1	*input_data.py*	Intensidade (volume) dos áudios de background que serão mixados na taxa de 80% com os áudios dos retificadores
CLIP_DURATION_MS	1000	*input_data.py*	Duração de cada arquivo de áudio em milissegundos
DATASET_DIR	'dataset-16kHz-16b-1s'	*input_data.py*	Diretório de armazenamento dos arquivos de áudio em suas respectivas classes
FEATURE_BIN_COUNT	40	*input_data.py*	Número de *features* (*frequency buckets*) por cada janela de 30 milissegundos de áudio
LEARNING_RATE	"0.001,0.0001"	*train.py*	Taxa de aprendizagem de cada uma das etapas de treinamento separadas por vírgula
MODEL_ARCHITECTURE	*'tiny_conv'*	*models.py*	Tipo de arquitetura do modelo CNN. Para microcontrolador a arquitetura mais indicada é a "*tiny_conv*" (TensorFlow, 2023)

PARÂMETRO	VALOR	*SCRIPT* DE EXECUÇÃO	FINALIDADE
PREPROCESS	'micro'	*input_data.py*	Tipo de pré-processamento dos áudios antes de alimentar o modelo CNN
QUANT_INPUT_MAX	26.0	*input_data.py*	Valor máximo possível para os dados do espectrograma
QUANT_INPUT_MIN	0.0	*input_data.py*	Valor mínimo possível para os dados do espectrograma
SAMPLE_RATE	16000	*input_data.py*	Taxa de amostragem dos áudios em Hz
SILENT_PERCENTAGE	*equal_percentage_of_training_samples*	*input_data.py*	Percentual de áudios que serão gerados para a classe "*Silence*"
TESTING_PERCENTAGE	10	*input_data.py*	Percentual dos áudios que serão usados para teste
TIME_SHIFT_MS	100.0	*input_data.py*	Tempo em milissegundos permitido deslocar aleatoriamente os clipes para o processo de *Audio Augmentation*
TRAINING_STEPS	"12000,3000"	*train.py*	Número de etapas de treinamento totalizando 15000 etapas
UNKNOWN_PERCENTAGE	*equal_percentage_of_training_samples*	*input_data.py*	Percentual de áudios que serão gerados para a classe "Unknown" = "Ambiente"
VALIDATION_PERCENTAGE	10	*input_data.py*	Percentual dos áudios que serão usados para validação
WANTED_WORDS	"bom, mediano, ruim"	*models.py*	Classes de áudios disponibilizadas em pastas com o mesmo nome dentro do diretório definido em *DATASET_DIR*
WINDOW_SIZE_MS	30.0	*models.py e input_data.py*	Tamanho da janela da FFT em milissegundos
WINDOW_STRIDE	20.0	*models.py e input_data.py*	Tamanho do deslocamento/passo (*step*) da janela da FFT em milissegundos

Tabela 2 – Parâmetros do modelo CNN utilizado

Fonte: (AUTOR, 2023).

Abaixo são discutidos mais a fundo os principais arquivos utilizados no processo de treinamento e como cada parâmetro influência o modelo, bem como os trechos de códigos que foram alterados para adequar o modelo *micro_speech*, que serve para reconhecimento de fala, para o modelo *micro_classifier*, adequado ao reconhecimento de sons, sobretudo de alta frequência.

5.1.1. Arquivo *model.py*

Este *script* é responsável por montar o modelo CNN a ser treinado com os dados/*features* disponibilizados pelo script *input_data.py*. O parâmetro

MODEL_ARCHITECTURE define qual dos modelos pré-concebidos será utilizado para o treinamento da CNN.

O modelo voltado para embarcar em microcontroladores é o "*tiny_conv*" (mostrado na Figura 17), o qual originalmente foi projetado para ser usado como o primeiro estágio de um pipeline, rodando em um hardware de baixo consumo de energia que pode estar sempre ligado e, em seguida, ativar chips de maior potência quando uma possível expressão for encontrada, para que uma análise mais precisa possa ser feito (WARDEN, 2019).

Nenhuma modificação foi feita diretamente no código fonte do arquivo *model.py*.

5.1.2. Arquivo *train.py*

Este arquivo recebe a totalidade dos parâmetros definidos no *notebook micro_classifier*.ipynb relacionados à entrada de dados, pre-processamento de áudios, quantidade de etapas de treino e ao modelo a ser utilizado. O *script* é responsável por fazer interface com o *script models.py*, responsável por definição do modelo, e o *script input_data.py*, responsável por pré-processar os arquivos de áudio e entregar as *features* para treinamento do modelo.

Para esse trabalho experimentou-se diminuir o número total de etapas de treinamento para 5 mil, sendo as primeiras 4 mil etapas com taxa de aprendizagem em 0.001 e as últimas mil etapas com taxa de aprendizagem de 0.0001, porém foi observado que a acurácia do modelo aumentava continuamente proporcionalmente ao número total de etapas de treinamento. Por isso, foi definido o total de 15 mil etapas de treinamento, o que exige um tempo total de treinamento de aproximadamente 2,5 horas em um computador com processador AMD Ryzen 7 5700U. Para definição desse valor foi levado em consideração a acurácia aceitável x tempo de treinamento.

Não foi feita modificação diretamente no código fonte do *script train.py*.

5.1.3. Arquivo *input_data.py*

Este *script* é responsável por converter os áudios em espectrogramas, e dividir os dados em *datasets* de treinamento, teste e validação de acordo com os parâmetros passados. Ele também é responsável por realizar o processo de *audio augmentation* com as técnicas de

deslocamento no tempo (*time shift*) e adição de ruído, além de criar as classes "*unknown*" e "*silence*" mesma proporção dos áudios disponíveis para as classes "bom", "mediano" e "ruim". Isso é definido pelos parâmetros *UNKNOWN_PERCENTAGE* e *SILENT_PERCENTAGE*, respectivamente.

A adição de ruído branco consiste em mixar um segundo de áudio, definido de forma randômica dentre os áudios disponíveis na pasta *_background_noise_*, nos áudios das classes para treinamento na proporção definida pelo parâmetro *BACKGROUND_FREQUENCY* e com volume definido pelo parâmetro *BACKGROUND_VOLUME_RANGE*. Essa pasta faz parte do *dataset speech_commands* disponibilizado pela Google (TensorFlow, 2023). Já o deslocamento no tempo é aplicado em todos os áudios deslocando um valor aleatório compreendido entre -TIME_SHIFT_MS e TIME_SHIFT_MS nos áudios de treinamento.

A classe "*unknown*" é criada com os áudios armazenados em quaisquer pastas dentro do diretório definido pelo parâmetro *DATASET_DIR* cujo seus nomes não estejam listados como classes no parâmetro *WANTED_WORDS*, excetuando-se a pasta *_background_noise_* que tem a finalidade explicada logo acima. Já a classe "*silence*" é definida como um uma sequência de 16 mil valores com valor 0, o que para uma taxa de amostragem de 16 kHz equivale a um áudio vazio (mudo) de um segundo.

O modelo *micro_classifier* foi treinado alterando vários dos parâmetros relacionados à definição das classes extras e *audio augmentation* e até mesmo desabilitando esse recurso durante o treinamento não obtendo diferenças consideráveis durante a fase de treinamento e testes no computador. Contudo, ao embarcar o modelo no microcontrolador foi percebido que com os valores registrados na Tabela 2 se obteve melhores resultados. Além disso, pelo fato do *dataset* com os áudios dos retificadores ser pequeno, as técnicas acima se demonstraram convenientes para aumentar a robustez do modelo e garantir uma maior generalização.

O script *input_data.py* também define como o espectrograma é processado para produzir as *features*. Os parâmetros *WINDOW_SIZE_MS* e *WINDOW_STRIDE* definem o tamanho do espectrograma gerado pela a FFT.

De acordo com matemática envolvida no cálculo da FFT, com uma janela de 30 milissegundos (480 amostras em 16 kHz) e passos de 20 milissegundos (320 amostras em 16 kHz) em um áudio de 1 segundo de com taxa de amostragem de 16 kHz (16 mil amostras) são produzidos: $Próxima Potência De Dois\left(\frac{480}{2}\right) + 1 \Rightarrow 256 + 1 = 257$ *bins/bukets* de frequência por janela. Para um áudio com duração de 1 segundo, a janela se desloca por:

$\frac{1600 - 480}{320} = 49$ vezes. Isso produz um espectrograma como uma forma de imagem de resolução $257\ x\ 49$ *pixels* ou $257\ x\ 49 = 12593$ *features* para serem alimentados na *CNN*.

Essa quantidade de informação é muito grande para ser processada em um modelo de *CNN* pensado para microcontroladores. Além disso, não é necessário esse nível de detalhe para classificação, por isso, pode-se reduzir a quantidade de informação sem necessariamente perder precisão no algoritmo. Um método simples seria calcular a média agrupando valores adjacentes, mas uma abordagem mais sofisticada é aplicar o algoritmo *MFCC* para reduzir a representação (WARDEN, 2019).

Os parâmetros *PREPROCESS* = "*micro*" instrui o script *input_data.py* a utilizar uma biblioteca chamada *audio_microfrontend* para gerar o espectrograma e aplicar algumas transformações de modo a obter, para cada janela de 30 milissegundos, a quantidade de *frequency bins/buckets* definida no parâmetro *FEATURE_BIN_COUNT*. Devido a complexidade dessa biblioteca e a necessidade de alteração do código fonte para melhorar a performance do modelo, ela será vista com mais detalhes abaixo.

5.1.4. Arquivo *audio_microfrontend_op.py*

A biblioteca *audio_microfronted* é utilizada tanto na fase de treinamento, quanto na fase de inferência no microcontrolador para extrair as *features* dos áudios. No primeiro momento a biblioteca é utilizada nos áudios gravados dos retificadores e depois nos áudios capturados pelo microfone da placa Arduino Nano 33 BLE Sense para alimentação do modelo *CNN* embarcado. Para que isso seja possível, a biblioteca possui sua versão em Python fornecida pelo arquivo *audio_microfrontend_op.py* e sua versão em C++ fornecida pelo arquivo *frontend.c*.

Esta biblioteca de geração de recursos (também chamada de *frontend* ou *microfronted*) recebe como entrada o áudio bruto e produz bancos de filtros (um vetor de valores). Espera-se que a entrada de áudio bruto tenha recursos PCM de 16 bits, com uma taxa de amostragem configurável. Mais especificamente o sinal de áudio passa por um filtro de pré-ênfase (opcional). Em seguida, é dividido em quadros (potencialmente sobrepostos) e uma função de janela é aplicada a cada quadro. Depois, uma Transformada de Fourier é aplicada em cada quadro (ou mais especificamente uma STFFT) para calcular o espectro de potência e posteriormente calcular os bancos de filtros (WARDEN, 2019)

Recursos extras desta biblioteca *frontend* incluem um módulo de redução de ruído, bem como um módulo de controle de ganho.

Cancelamento de ruído: Remove o ruído estacionário de cada canal do sinal usando um filtro passa-baixo.

Controle de ganho: Uma nova compactação dinâmica baseada em controle de ganho automático para substituir a compactação estática amplamente usada, como *log* ou *root* (WANG et al., 2017).

A Figura 22 mostra de forma esquematizada os processos realizados pela biblioteca *microfrontend* que não abrange as etapas de Cancelamento de ruídos e controle de ganho, mas serve para dar uma ideia dos processos envolvidos desde a entrada dos dados de áudio puro na parte superior até a saída dos *features* na parte inferior.

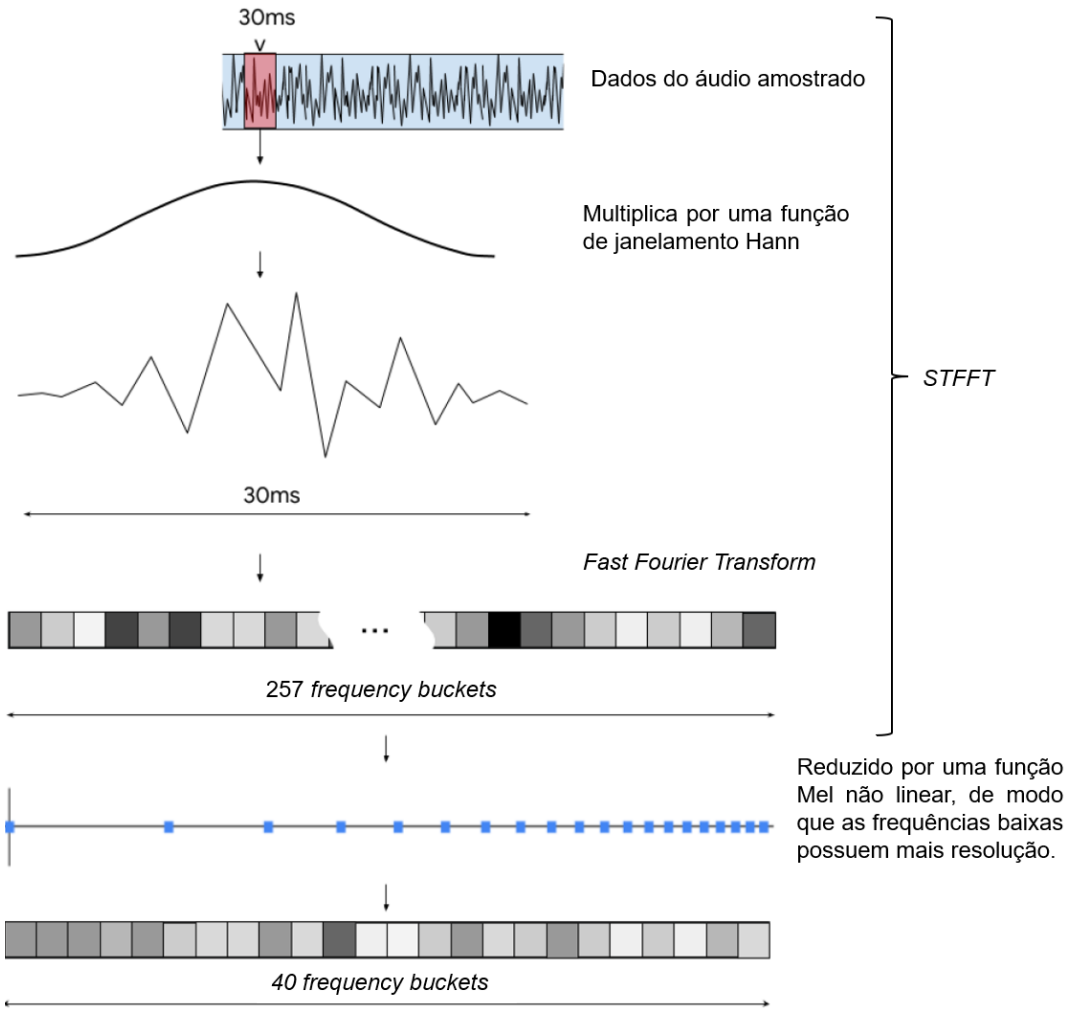

Figura 22 – Esquema do processo de geração de *features* (*frontend*)

Fonte: (AUTOR, modificado a partir de (WARDEN, 2019)).

O modelo *micro_speech*, que serviu de base para este trabalho, é voltado para o reconhecimento de fala, por isso os parâmetros configurados para a biblioteca *frontend* foi pensado na otimização das *features* para potencializar sons da fala humana e diminuir sinais considerados espúrios (ruídos e sinais diversos). Contudo, para o modelo *micro_classifier* o objeto é a identificação para classificação de sons de retificadores que possuem características bem distintas dos sons da fala humana, por isso, foi necessário alterar vários parâmetros do processo de geração de *features* que não foram disponibilizados no notebook Jupyter e tiveram que ser alterados diretamente no código da biblioteca em Python e em C++.

A função de processamento dos áudios da biblioteca *frontend* recebe um conjunto com mais de 20 parâmetros, o que torna inviável discorrer sobre cada um aqui. A Tabela 3 mostra um resumo dos parâmetros da biblioteca *frontend*, que foram alterados tanto no código-fonte em Python (*audio_microfrontend_op.py*), quanto no código-fonte em C++ (*micro_features_micro_features_generator.cpp*), com os valores padrão para reconhecimento de fala e os valores ajustados para reconhecimento de sons dos retificadores.

PARÂMETRO	PADRÃO	AJUSTADO	DEFINIÇÃO
upper_band_limit	7500	7999	Float, a frequência mais alta incluída nos bancos de filtros.
lower_band_limit	125	0.0	Float, a frequência mais baixa incluída nos bancos de filtros.
min_signal_remaining	0.05	0.80	Float, fração do sinal a ser preservada na suavização.
enable_pcan	True	False	Bool, habilite o controle de ganho automático PCAN.

Tabela 3 – Parâmetros alterados no código-fonte da biblioteca *frontend*

Fonte: (AUTOR, 2023).

Ao olhar para a Figura 19, é possível perceber que os sons dos retificadores, independente da classe, possuem componentes de frequências inferiores a 125 Hz e superiores a 7500 Hz. A alteração dos parâmetros *upper_band_limit* e *lower_band_limit* fez com que aumentasse o range de frequências (ou banda) processada dos áudios. Incluindo, assim, frequências mais baixas e chegando no limite das frequências presentes nos áudios, que é de 8 kHz (para áudios amostrados com 16 kHz). Isso melhorou significativamente a qualidade dos *features* disponibilizadas pela biblioteca *frontend*.

Já os parâmetros min_signal_remaining e enable_pcan habilitavam um filtro antirruído que funcionava mantendo uma média contínua do valor em cada intervalo de frequência e, em seguida, subtraindo essa média do valor atual (WARDEN, 2019). Algo, bom para identificação de fala, mas extremamente ruim quando se está trabalhando com áudio de equipamentos que

contém o ruído conta para a classificação. Por isso foi aumentado a preservação do ruído de fundo e desabilitado o filtro PCAN, que tinha a função de dar ênfase nas frequências de vozes humanas.

Com a alteração desses parâmetros foi possível obter acurácia superiores a 95%, o que ainda não havia acontecido treinando o modelo *CNN* alterando apenas os parâmetros disponíveis no *notebook micro_classifier.ipynb.*

A desativação do ganho automático *PCAN* (*Per-Channel Amplitude Normalization*) fez com que o range das *features* disponibilizadas pela biblioteca *frontend* para o treinamento da *CNN* mudasse de 0 a 665.6 para 0 a 954, por uma questão de implementação do código pelo autor. Essa saída, por sua vez é ajustada para o range 0 a 26 para compatibilização com outros geradores de recursos (*features*) que fazem uso da estrutura do TensorFlow.

Devido ao novo range, foi necessário alterar os arquivos *input_data.py* e *micro_features_micro_features_generator.cpp*, conforme mostrado abaixo:

input_data.py

```
        #************* Início da modificação ***************
        micro_frontend = frontend_op.audio_microfrontend(
            int16_input,
            sample_rate=sample_rate,
            window_size=window_size_ms,
            window_step=window_step_ms,
            num_channels=model_settings['fingerprint_width'],
        #************* Linhas adicionadas ***************
            upper_band_limit=7999.0,
            lower_band_limit=0.0,
            min_signal_remaining=0.80,
            enable_pcan=False,
        #************ Fim Linhas adicionadas *************
            out_scale=1,
            out_type=tf.float32)
        #self.output_ = tf.multiply(micro_frontend, (10.0 / 256.0))
        self.output_ = tf.multiply(micro_frontend, (6.9769 / 256.0))
        #************* Fim da modificação ******************
```

micro_features_micro_features_generator.cpp

```
      // ************* Início da modificação ***************
      /*
      constexpr int32_t value_scale = 256;
      constexpr int32_t value_div = static_cast<int32_t>((25.6f * 26.0f) +
0.5f);
    int32_t value =
        ((frontend_output.values[i] * value_scale) + (value_div / 2)) /
        value_div;
    */

      constexpr int32_t value_scale = 255; //MODIFICADO DE 256 PARA 255
      constexpr int32_t value_div = static_cast<int32_t>(954.0f);
    int32_t value = ((frontend_output.values[i] * value_scale) /
value_div);
```

```
// ************* Fim da modificação*******************
```

5.1.5. Arquivo *freeze.py*

O *script freeze.py* é utilizado para congelar o modelo CNN após o treinamento e definição dos pesos dos de cada parâmetro treinado. Congelar um modelo no TensorFlow é um processo comum em treinamento de modelos de aprendizado de máquina, especialmente em tarefas de inferência e implantação.

Um modelo de aprendizado profundo pode ser composto por milhões de parâmetros, o que pode tornar a implantação demorada e ineficiente, principalmente em dispositivos com recursos limitados, como dispositivos móveis ou sistemas embarcados. Ao congelar o modelo, os parâmetros treinados são convertidos em constantes, reduzindo significativamente o tamanho do modelo e, consequentemente, os requisitos de memória e recursos de computação. Isso faz com que o modelo ganhe eficiência na implantação.

Para realizar o processo de congelamento é necessário conhecer o formato da entrada do modelo *CNN* e para fazer isso o *script freeze.py* faz uso da biblioteca *frontend* por intermédio do *script input_data.py*. Por isso, foi necessário alterar o código-fonte na parte de normalização dos valores entregues pela biblioteca *frontend*. Segue trecho do código alterado:

```
#************* Início da modificação ***************
#fingerprint_input = tf.multiply(micro_frontend, (10.0 / 256.0))
fingerprint_input = tf.multiply(micro_frontend, (6.9769 / 256.0))
#************* Fim da modificação ******************
```

5.2. Treinamento do modelo

Após a definição dos vários parâmetros como as características dos áudios utilizados (duração e taxa de amostragem), características da janela e deslocamento utilizados no algoritmo na geração de *features*, o treinamento foi realizado pela chamada de do script em Python *train.py*. O treinamento do modelo foi realizado em 15 mil passos, o que levou aproximadamente 2,5 horas.

O processo de treinamento foi realizado várias vezes alterando-se os vários parâmetros descritos nas seções anteriores até se conseguir obter o melhor resultado de 98,88% de acurácia.

A Figura 23 mostra o gráfico gerado durante o processo de treinamento. É possível observar que a cada 1000 etapas de treinamento foi feito uma etapa de validação para aferir a

acurácia do modelo com dados diferentes dos usados no processo de treinamento. Os dados de validação correspondem a 10% dos áudios de cada classe que foram separados do *dataset* completo.

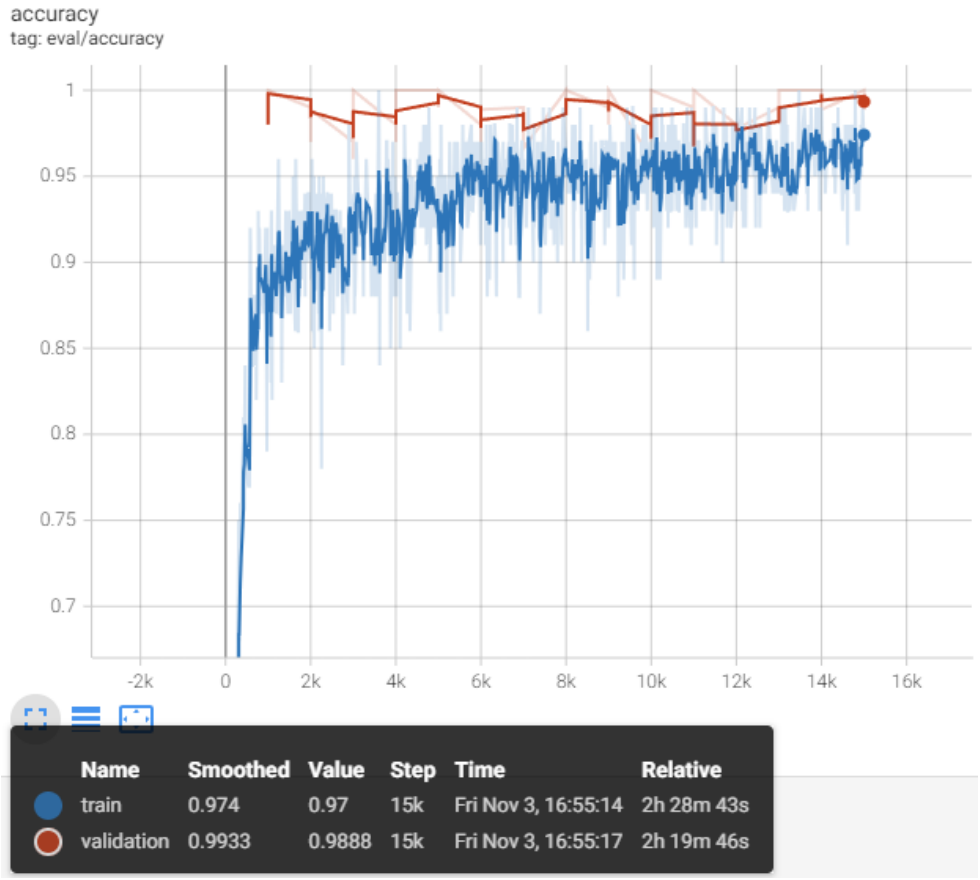

Figura 23 – Gráfico de acurácia do modelo por passo de treinamento

Fonte: (AUTOR, 2023).

O quadro escuro presente na Figura 23 mostra um resumo com a acurácia final do modelo antes de ser convertido para o formato TensorFlow, antes do modelo ser convertido para o TensorFlow Lite.

A Figura 24 mostra a matriz confusão gerada no final do processo de treinamento.

CLASSE		Valores Preditos				
		Silence	Ambiente	Bom	Mediano	Ruim
Valores Reais	Silence	54	3	0	0	0
	Ambiente	0	57	0	0	0
	Bom	0	0	108	0	0
	Mediano	0	0	0	91	0
	Ruim	0	0	0	0	84

Figura 24 – Matriz Confusão com dados de validação no TensorFlow

Fonte: (AUTOR, 2023).

5.3. Conversão do modelo e implantação

Com a utilização de scripts em Python foi feita a conversão do modelo para a biblioteca TensorFlow Lite Micro. Esse processo gerou dois modelos para a biblioteca TensorFlow Lite Micro: O primeiro é o modelo com os dados de entrada e pesos da rede neural em formato numérico do tipo *float* de 32 bits. Esse modelo possui 84460 bytes. O segundo é um modelo que passa por um processo de quantização, que é a adequação dos valores de entrada e pesos da rede para o tipo numérico int8_t, que são inteiros de 8 bits, compreendidos entre -128 a 127. Esse modelo possui 22960 bytes.

Após a conversão do modelo é feito uma etapa de teste com 10% dos áudios do *dataset* completo inéditos, ou seja, que não foram utilizados nem para treinamento, nem para validação. A Figura 24 mostra a matriz confusão gerada no teste de 10% dos áudios (397 amostras).

CLASSE		Valores Preditos				
		Silence	Ambiente	Bom	Mediano	Ruim
Valores Reais	Silence	54	1	0	0	0
	Ambiente	0	57	0	0	0
	Bom	0	0	108	0	0
	Mediano	0	0	0	91	0
	Ruim	0	0	0	0	84

Figura 25 – Matriz Confusão com dados de testes no TensorFlow Lite

Fonte: (AUTOR, 2023).

É possível notar que após a conversão e utilizando os dados reservados para testes, o modelo apresentou uma acurácia de 99,74%.

A partir do modelo em formato do TensorFlow Lite, é feito um processo de representação do modelo como um grande vetor em linguagem C++. Isso se torna necessário, pelo fato da maioria dos sistemas embarcados não possuírem sistema operacional com sistema de arquivos para armazenamento do arquivo do modelo quantizado "*model.fllite"* exportado pelo script de conversão.

Essa representação consiste, basicamente, em criar um arquivo .cpp com a definição de um vetor do tipo int8_t com tamanho de 22960 bytes, cujo conteúdo são os bytes do arquivo original. Esse arquivo de nome *micro_features_model.cpp* gerado foi importado no projeto do Arduino junto com a biblioteca TensorFlow Lite Micro. Após compilação e envio para o Arduino a predição se tornou funcional.

Foram feitos pequenos ajustes no código fonte disponibilizado pela biblioteca TensorFlow Lite Micro a fim de personalizar as classes e as cores do LED pelas quais cada classe é identificada, os quais serão tratados na próxima seção.

5.4. Arquivos alterados na biblioteca TensorFlow Lite Micro

Além dos ajustes nos arquivos C++ da biblioteca TensorFlow Lite Micro descritos nas seções acima, foram necessárias alterações nos arquivos que definem as classes para classificação e o arquivo que sinaliza qual classe está sendo identificada através da cor do LED RGB da placa Arduino Nano 33 BLE Sense. Segue abaixa as alterações que foram feitas:

micro_features_micro_model_settings.cpp

```
const char* kCategoryLabels[kCategoryCount] = {
    "Silence",
    "Ambiente",
    "Bom",
    "Mediano",
    "Ruim",
};
```

arduino_command_responder.cpp

```
    if (found_command[0] == 'B') {
      digitalWrite(LEDG, LOW);  // Green for 'Bom'
    } else if (found_command[0] == 'M') {
      digitalWrite(LEDB, LOW);  // Blue for 'Mediano'
    } else if (found_command[0] == 'R') {
      digitalWrite(LEDR, LOW);  // Red for 'Ruim'
    } else if (found_command[0] == 'A') {
      digitalWrite(LEDR, 0);  // Yellow for 'Ambiente'
```

```
      digitalWrite(LEDG, 0);
      digitalWrite(LEDB, 255);
    } else {
      digitalWrite(LEDR, 200);      // Gree for 'Silence'
      digitalWrite(LEDG, 200);
      digitalWrite(LEDB, 200);
      // silence
    }
```

5.5. *Scikit-Learn* e *Random Forest*

Para o treinamento do modelo *Random Forest* foi utilizada a biblioteca *Scikit-Learn* em um *notebook Jupyter* também disponibilizado no repositório GitHub desse trablaho. Esse *notebook* foi derivado do *notebook* de treinamento do exemplo *speech_command* da biblioteca *TensorFlow*. Isso quer dizer que o modelo RF foi treinado com os mesmos *features* extraídos dos dados de áudios pelo *script input_data.py*

Seguindo o mesmo padrão, a entrada para o modelo *RF* consistiu em um vetor com 1960 valores para cada fragmento de um segundo de áudio representando os valores das *features* no domínio da frequência. Foram definidas quatro classes para a saída baseada nas seguintes classes de áudio: “Ambiente”, “Bom”, “Mediano” e “Ruim”.

Foram testados vários modelos de *RF* variando o número de árvores até que se encontrou o número mínimo de árvores no qual o modelo apresentou acurácia de 100% com os dados de validação. A validação foi realizada com dados separados que não foram utilizados no treinamento do modelo. A escolha do menor número de árvores cuja acurácia atingiu 100% tem a ver com a limitação de recursos do sistema embarcado.

Do ponto de vista de desempenho e sendo conservador, poderia se optar por um número maior de árvores a fim de garantir, em teoria, maior confiabilidade do modelo, contudo, foi observado que o custo computacional de cada árvore adicional no modelo é elevado demais para o sistema embarcado. Isso pode ser percebido após a conversão do modelo para código C++. Cada árvore adicionava, em média, 500 condicionais no código gerado.

5.5.1. Representação e avaliação do modelo

Foram utilizadas algumas ferramentas disponíveis na biblioteca *Scikit-Learn* e dtreeviz (*Decision Tree Visualization*) para representação e avaliação do modelo *RF*. A Figura 26 mostra a visualização esquemática de a primeira das 16 árvores utilizadas no modelo. É possível notar que o modelo ficou bastante robusto pelos vários níveis de decisão e ramificações mostrado na Figura 26.

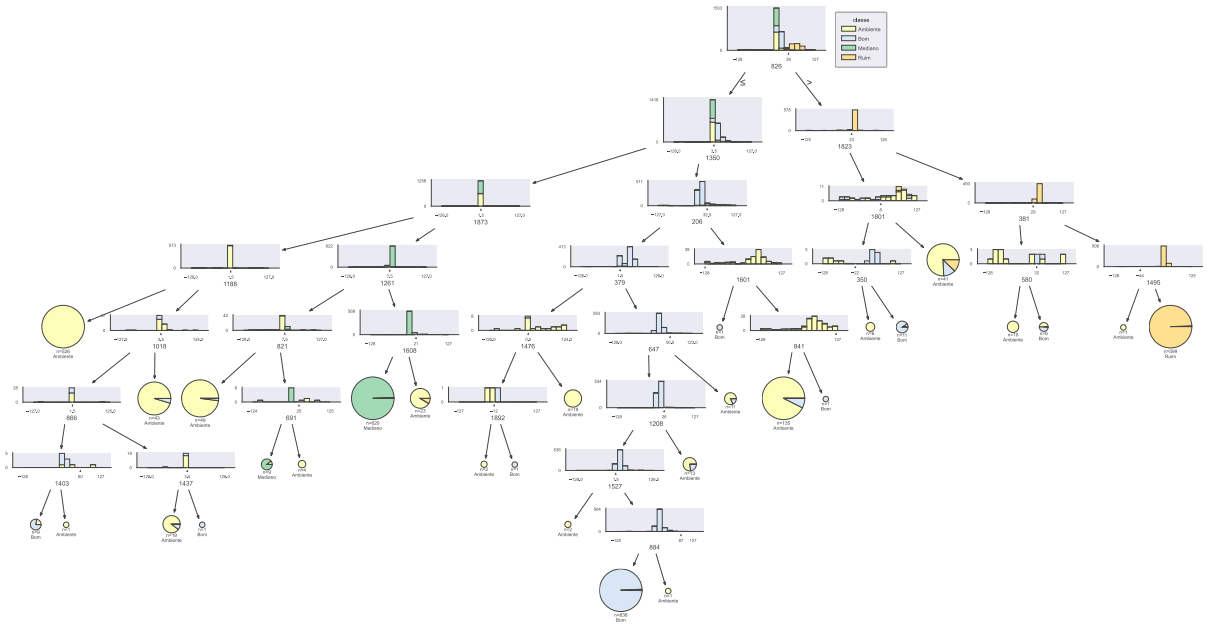

Figura 26 – Representação visual de uma das árvores da Floresta Aleatória (*RF*)

Fonte: (AUTOR, 2023).

Na Figura 27 é mostrado quais são os *features* mais importantes, dentre os 1960 valores, para a decisão de qual classe cada áudio pertence.

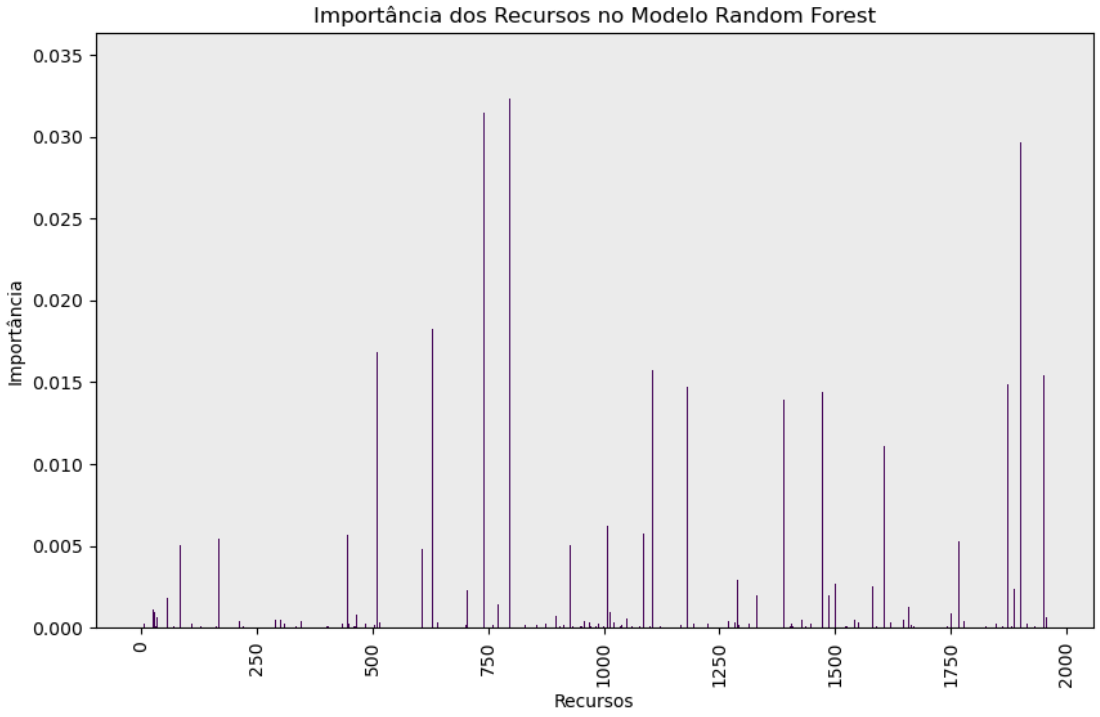

Figura 27 – Importância de cada *feature* na Floresta Aleatória (*RF*)

Fonte: (AUTOR, 2023).

Para avaliação do modelo foram plotados a Matriz Confusão e o relatório com as principais métricas mostradas nas Figuras 28 e 29, respectivamente.

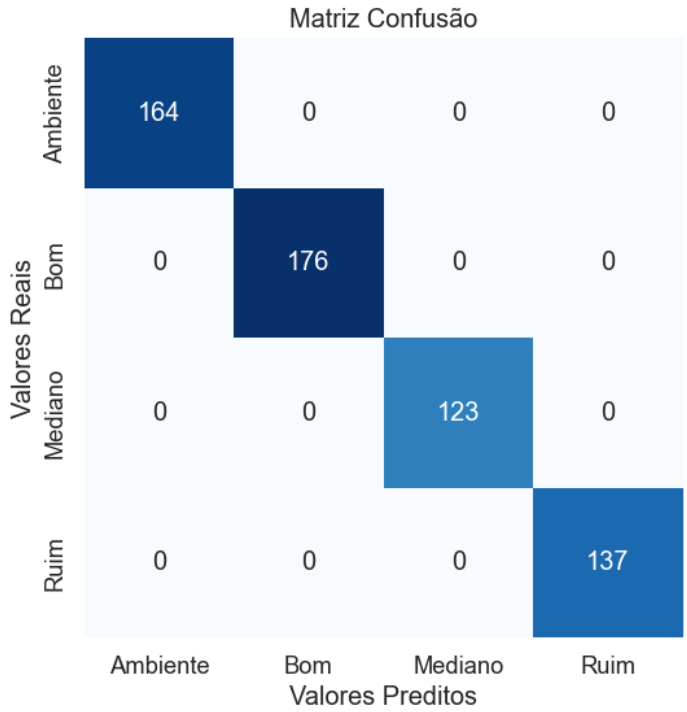

Figura 28 – Matriz Confusão do modelo RF

Fonte: (AUTOR, 2023).

	Precision	Recall	F1-Score	Support
0	1.00	1.00	1.00	164
1	1.00	1.00	1.00	176
2	1.00	1.00	1.00	145
3	1.00	1.00	1.00	115
Accuracy			1.00	600,00
Macro Avg	1.00	1.00	1.00	600
Weighted Avg	1.00	1.00	1.00	600
Weighted Avg	1.00	1.00	1.00	600

Figura 29 – Relatório com principais métricas do modelo *RF*

Fonte: (AUTOR, 2023).

5.5.2. Conversão e implantação do modelo *RF*

Para conversão do modelo *RF* em código C++ foi utilizado a biblioteca *MicroMLGen* gerando um arquivo *random_forest_classifier.h* (*header file*) com uma série de condicionais mostrando os vários critérios para as decisões tomadas pelo algoritmo *RF*. O arquivo .h gerado ficou com o tamanho de 130432 bytes, o qual foi importado no projeto Arduino. Esse arquivo gerado contém uma classe que possui um método para receber o vetor de *features* e retornar a classe correspondente.

Diferentemente da biblioteca *TensorFlow* para Python, que possui uma biblioteca específica (*TensorFlow Lite*) para levar o modelo para o Arduino, a biblioteca *MicroMLGen* apenas converte o modelo para um formato de linguagem C++ compatível com o Arduino. Por isso, para optou-se por treinar o modelo *RF* com os mesmos dados utilizados no modelo TensorFlow, uma vez que foi possível utilizar o mesmo provedor de features da biblioteca TensorFlow Lite Micro para alimentar o modelo em *RF*.

Para implantação do modelo foi necessário modificar o arquivo de projeto do Arduino (*rectifier_classifier.ino*) em dois pontos.

Primeiro, incluir o arquivo *random_forest_classifier.h* como *header* do arquivo *rectifier_classifier.ino*:

```
#include "RandomForestClassifier.h"
```

Segundo, incluir a parte de inferência do modelo RF logo após ter processados novos *features*:

```
  // Aqui entra a personalização para o objeto RandomForest
  if(is_new_command){
    MicroPrintf("RADOMFOREST: Heard %s",
classifier.predictLabel(model_input_buffer));
  }
  // Fim do personalização
```

Após essas alterações, compilação e envio para o Arduino, o modelo já se tornou funcionou.

5.6. Avaliação dos modelos no sistema embarcado

Devido a indisponibilidade de equipamentos em campo para testes, a avaliação dos algoritmos foi feita pela reprodução de um conjunto de áudios separados para esse fim em uma caixa de som e a verificação da identificação da classe na placa Arduino.

A placa foi programada de modo que ela reporte na forma de escrita para a interface serial qual classe foi identificada. Essa escrita é recebida em tempo real pela ferramenta chamada "*Serial Monitor*" da IDE do Arduino. O LED da placa também foi programado de forma que as classes "Ambiente", "Bom", "Mediano" e "Ruim" são identificadas pelas cores amarelo, verde, azul e vermelho, respectivamente.

A Figura 30 mostra o momento em que a placa identifica um áudio da classe "Mediano", o qual está sendo reproduzido pelo notebook.

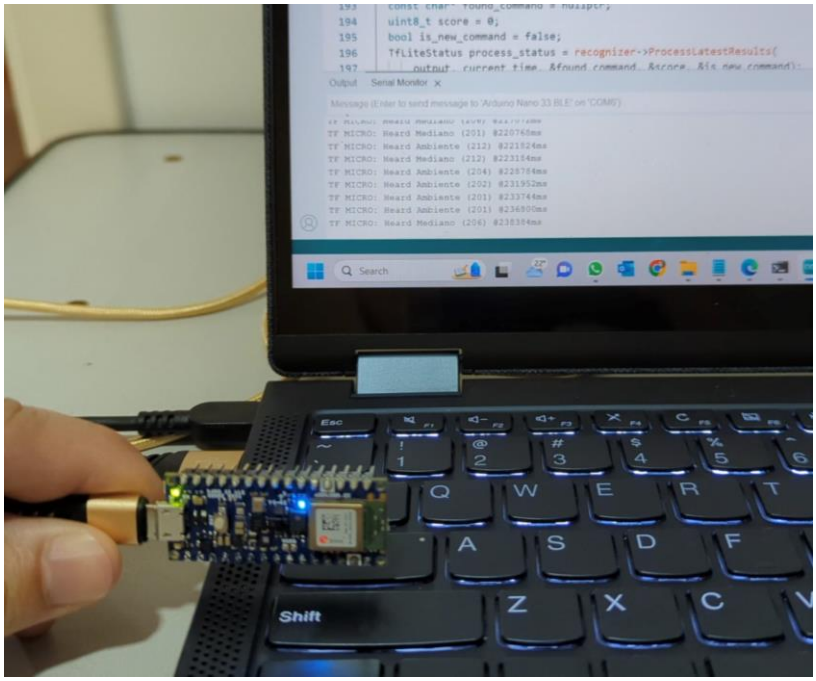

Figura 30 – Teste da solução embarcada

Fonte: (AUTOR, 2023).

A avaliação do modelo embarcado foi realizada de forma empírica e alcançou um acurácia de aproximadamente 75% no modelo *CNN* do *TensorFlow Lite Micro* e 45% no modelo Random Forest.

6. Considerações finais e conclusão

No contexto da Indústria 4.0, modelos de *ML* vem ganhando espaço, seja em análise de grande volume de dados (*big data*) de máquinas conectadas com os sistemas *ERP* (*Enterprise Resource Planning*), modelo conhecido como Data-Driven (WEN, et al, 2018), ou de forma embarcada em instrumentos microprocessados para análise e diagnósticos de máquinas e equipamentos em campo.

Nesse mesmo sentido, Warden et al. (2019) diz que o uso de Inteligência Artificial de forma embarcada em microcontroladores é um campo novo e promissor.

A chegada recente de extensões para microcontroladores de bibliotecas de *ML* já conhecidas pelo público, associada ao fato do baixíssimo custo de microcontroladores mais poderosos, tem atraído a atenção de desenvolvedoras e entusiastas para novas soluções de *ML* embarcada para a Indústria (ZHANG et al., 2023).

Neste trabalho foi apresentada uma solução embarcada de baixo custo para classificação em tempo real de retificadores trifásicos entre bom, mediano e ruim utilizando bibliotecas *open source* para treinamento e implementação na placa Arduino Nano 33 BLE Sense Rev2.

Durante a fase de validação no computador foram obtidos excelentes resultados, sendo que o melhor modelo de *ML* para o problema foi o *Random Forest*, o qual obteve uma acurácia de 100, enquanto o modelo *CNN* pela biblioteca *TensorFlow Lite* alcançou uma acurácia de 99,74%. Contudo, tal rendimento não se manteve ao embarcar as soluções na placa Arduino.

Em testes práticos com reprodução de áudios de retificadores nas três classes o modelo do *TensorFlow Lite Micro* foi capaz de obter taxa de acerto de aproximadamente 75%, enquanto o modelo de *RF* não conseguiu resultado superior a 50% de acerto.

Deve-se considerar a limitação do hardware utilizado no trabalho, já que a *Arduino Nano 33 BLE Sense Rev2* é a placa com sensores *build-in* mais barata da família Arduino. Outro fator preponderante é a quantidade limitada de amostras de retificadores utilizadas para o treinamento, apesar do esforço empreendido em aumentar os dados com técnicas de *audio augmentation*, para uma solução mais genérica entende-se que mais dados são necessários.

Apesar do baixo rendimento em campo, quando comparado aos altos valores de acurácia durante a fase de validação e teste no *PC*, o presente trabalho se apresenta como um caminho para a introdução de *ML* de forma embarcada em empresas e indústrias, sobretudo nas de energia, em que o uso de *ML* é muito tímido (AHMAD et al., 2021).

7. Trabalhos futuros

Como sugestões de trabalhos futuros pode-se explorar se o processo de quantização teve influência na queda da precisão do modelo embarcado e qual é o ganho em performance na utilização dessa técnica, além de avaliar o consumo do modelo em *float* frente ao modelo quantizado em *int8*.

É sugerido também a utilização do mesmo *framework* para treinamento de modelos com utilização de base de dados acústicas de outros equipamentos, bem como utilização de outros sinais como vibração e temperaturas.

Por último, pode-se fazer um estudo comparativo entre os diversos *frameworks* e algoritmos de *ML* disponíveis para sistemas embarcados.

Referências Bibliográficas

ALBAWI, Saad; MOHAMMED, Tareq Abed; AL-ZAWI, Saad. **Understanding of a convolutional neural network**. In: 2017 international conference on engineering and technology (ICET). Ieee, 2017. p. 1-6.

ARAUJO, R. **Aprendizado com Árvores de Decisão**. Disponível em: https://ricardomatsumura.medium.com/aprendizado-com-%C3%A1rvores-de-decis%C3%A3o-73d874664d1. Acesso em Outubro 2022.

AUDACITY SOFTWARE. **Audacity (2023)**. Disponível em: https://www.audacityteam.org/. Acesso em: 1 de junho de 2023.

BARBOSA, L. C. M. Áudio Digital: **Uma Abordagem ao Áudio pela Perspectiva de Ensino**. Tese de Doutorado, Instituto Politécnico do Porto (Portugal), 2012.

BARCHIESI, D.; GIANNOULIS, D.; STOWELL, D.; PLUMBLEY, M. D. **Acoustic scene classification**: Classifying environments from the sounds they produce. IEEE Signal Processing Magazine, v. 32, n. 3, p. 16-34, 2015.

BRAGA, Antônio de Pádua; LUDERMIR, Teresa Bernarda; CARVALHO, André Carlos Ponce de Leon Ferreira. **Redes neurais artificiais**: teoria e aplicações. 2000.

BREIMAN, L. **Random forests. Machine learning**, v. 45, n. 1, pp. 5–32, 2001.

CARVALHO JÚNIOR, J. R. D. **Processamento digital de imagens para a identificação automática de falhas em rolos dos transportadores de correias**. Dissertação de Mestrado, UFOP. Disponível em: https://www.repositorio.ufop.br/handle/123456789/10070. Acesso em Outubro 2022.

CHASE, Otavio; ALMEIDA, F. **Sistemas embarcados. Mídia Eletrônica**. Disponível em: <www.sbajovem.org/chase>. Acesso em Outubro 2022.

CHEN, Nianyi. **Support vector machine in chemistry**. World Scientific, 2004.

CHEN, S. H.; JAKEMAN, A. J.; NORTON, J. P. **Artificial intelligence techniques: an introduction to their use for modelling environmental systems**. Mathematics and computers in simulation, v. 78, n. 2-3, pp.

CRUZ, E. B. D. **Detector de falhas de rolos de transportadores de correia baseado em aprendizado de máquina**. Dissertação de Mestrado, UFOP. Disponível em: https://repositorio.ufop.br/handle/123456789/13038. Acesso em Outubro 2022.

VILLA, D. K. D. **Sistemas Inteligentes**. Minas Gerais, 2018. (Apostila).

DE OLIVEIRA, S. **Internet das Coisas com ESP8266, Arduino e Raspberry Pi**. Novatec Editora Ltda., 2017.

EKLUND, Ville-Veikko. **Data augmentation techniques for robust audio analysis**. 2019. Dissertação de Mestrado.

EMADI, A.; NASIRI, A.; BEKIAROV, B. S. **Uninterruptible Power Supplies and Active Filters**. 1. ed. New York: CRC Press, 2015. 285 p.

ERICEIRA, Daniel R. et al. **Early failure detection of belt conveyor idlers by means of ultrasonic sensing**. In: 2020 International Joint Conference on Neural Networks (IJCNN). IEEE, 2020. p. 1-8.

ERICEIRA, D. R. **Detecção automática de defeitos em rolos de transportadores de correia utilizando sensoriamento Ultrassônico**. Tese de Mestrado, Universidade Federal de Ouro Preto, Ouro Preto, 2019. Disponível em: https://www.repositorio.ufop.br/handle/123456789/12554. Acesso em Outubro 2022.

FRANÇOIS CHOLLET. **Deep Learning with Python, Second Edition**. [s.l.] Shelter Island, Ny Manning Publications, 2021.

GÉRON, A. **Hands-on Machine Learning with Scikit-Learn**, Keras, and TensorFlow. O'Reilly Media, Inc., 2022.

GLOWACZ, **Adam. Acoustic based fault diagnosis of three-phase induction motor. Applied Acoustics**, v. 137, p. 82-89, 2018. ISSN 0003-682X. Disponível em: https://doi.org/10.1016/j.apacoust.2018.03.010. Acesso em Outubro 2022.

GOLDBERGER, J. et al. **Neighbourhood components analysis**. Em: Advances in neural information processing systems, pp. 513–520, 2005. Acesso em Outubro 2022.

GRANDHI, R. T.; KRISHNA PRAKASH, N. **Machine-Learning Based Fault Diagnosis of Electrical Motors Using Acoustic Signals**. In: I. Jeena Jacob, S. Kolandapalayam Shanmugam, S. Piramuthu, P. Falkowski-Gilski (Eds.) Data Intelligence and Cognitive Informatics. Algorithms for Intelligent Systems. Springer, Singapore, 2021. Disponível em: https://doi.org/10.1007/978-981-15-8530-2_52. Acesso em Outubro 2022.

HASTIE, T.; TIBSHIRANI, R.; FRIEDMAN, J. **The Elements of Statistical Learning: Data Mining, Inference, and Prediction**. Springer, 2009.

HEARST, Marti A. et al. **Support vector machines. IEEE Intelligent Systems and their applications**, v. 13, n. 4, p. 18-28, 1998.

HEATH, Steve. **Embedded Systems Design**. Ano de publicação: 2021.

JAGATHEESAPERUMAL, S. K.; RAHOUTI, M.; AHMAD, K.; AL-FUQAHA, A.; GUIZANI, M. **The Duo of Artificial Intelligence and Big Data for Industry 4.0**: Applications, Techniques, Challenges, and Future Research Directions. IEEE Internet of Things Journal, v. 9, n. 15, p. 12861-12885, 2021.

LI, Zewen et al. **A survey of convolutional neural networks: analysis, applications, and prospects**. IEEE transactions on neural networks and learning systems, 2021.

MORONEY, L. **AI and Machine Learning for Coders**: A Programmer's Guide to Artificial Intelligence. O'Reilly Media, Inc., 2020.

KOU, Lei et al. **Fault diagnosis for three-phase PWM rectifier based on deep feedforward network with transient synthetic features**. ISA transactions, v. 101, p. 399-407, 2020.

Librosa. 2023, March. Disponível em: https://librosa.org/doc/latest/index.html. Acesso em: 29 de maio de 2023.

LORENA, Ana Carolina; DE CARVALHO, André CPLF. **Uma introdução às support vector machines**. Revista de Informática Teórica e Aplicada, v. 14, n. 2, p. 43-67, 2007.

MAMMONE, Alessia; TURCHI, Marco; CRISTIANINI, Nello. **Support vector machines. Wiley Interdisciplinary Reviews**: Computational Statistics, v. 1, n. 3, p. 283-289, 2009.

MAZIERO. **Processamento de Áudio**. Disponível em: http://wiki.inf.ufpr.br/maziero/doku.php?id=prog2:processamento_de_audio. Acesso em 10 de março de 2023.

MicroMLGen library for Python | Eloquent Arduino. 2022, December. Disponível em: https://eloquentarduino.com/libraries/micromlgen/. Acesso em: 25 de setembro de 2023.

MONARD, M. C.; BARANAUSKAS, J. A**. Conceitos sobre aprendizado de máquina. Sistemas inteligentes - Fundamentos e aplicações**, v. 1, n. 1, pp. 32, 2003.

MONARD, M. C.; BARANAUSKAS, J. A. **Conceitos sobre Aprendizado de Máquina. Sistemas Inteligentes - Fundamentos e Aplicações**. Barueri - SP: Editora Manole Ltda, 2003, v. , p. 89-114.

MÜLLER, A. C.; GUIDO, S. **Introduction to machine learning with Python:** a guide for data scientists. Beijing: O'reilly, 2017.

Nano 33 BLE Sense Rev2. 2022, November. Arduino. Disponível em: https://docs.arduino.cc/hardware/nano-33-ble-sense-rev2. Acesso em: 29 de maio de 2023.

Netron. 2023, May. Disponível em: https://github.com/lutzroeder/netron. Acesso em: 29 de maio de 2023.

NOBLE, William S. **What is a support vector machine?**. Nature biotechnology, v. 24, n. 12, p. 1565-1567, 2006.

OPPENHEIM, A. V.; SCHAFER, R. W. **Processamento em Tempo Discreto de Sinais**. Tradução Daniel Vieira. 3ª ed. São Paulo: Pearson Education do Brasil, 2012.

PETER, M. **Embedded Systems Design**. Kluwer Academic Publishers, 2003.

PLATTS, J.; ST. AUBYN, J. **Uninterruptible Power Supplies**. 1. ed. London: Peregrinus on behalf of the Institution of Electrical Engineers, 1992. 130 p.

RAHNAMA, Mehdi; VAHEDI, Abolfazl. **Application of acoustic signals for rectifier fault detection in brushless synchronous generator. Archives of acoustics**, v. 44, n. 2, p. 267-276, 2019.

RETTBERG, A.; ZANELLA, M.; DOMER, R.; GERSTLAUER, A.; RAMMIG, F. (Eds.). **Embedded System Design: Topics, Techniques and Trends**: IFIP TC10 Working Conference: International Embedded Systems Symposium (IESS), May 30-June 1, 2007, Irvine (CA), USA, vol. 231. Springer, 2010.

RODRIGUES, V. **Métricas de Avaliação: acurácia, precisão, recall... quais as diferenças?** Disponível em: https://medium.com/@vitorborbarodrigues/m%C3%A9tricas-de-avalia%C3%A7%C3%A3o-acur%C3%A1cia-precis%C3%A3o-recall-quais-as-diferen%C3%A7as-c8f05e0a513c. Acesso em outubro 2022.

RONGJIE, Wang et al. **A fault diagnosis method for three-phase rectifiers**. International Journal of Electrical Power & Energy Systems, v. 52, p. 266-269, 2013.

SAHA, Sumit. **A comprehensive guide to convolutional neural networks—the ELI5 way**. Towards data science, v. 15, p. 15, 2018.

SANTOS, A. A. **Sistema automático para a inspeção visual de transportadores de correia por meio de redes neurais convolucionais**. Dissertação de Mestrado, UFOP. Disponível em: https://www.repositorio.ufop.br/handle/123456789/13097.

SILVA, A. C. P. **Monitoramento da qualidade de SINTER FEED através de dados espectrais associados a aprendizado de máquina–estudo de caso: Mina de Carajás Serra Sul (S11D)**. Tese de Mestrado, Universidade Federal de Ouro Preto, Ouro Preto, 2021. Disponível em: https://www.repositorio.ufop.br/handle/123456789/14322. Acesso em Outubro 2022.

SILVA, I. N.; SPATTI, D. H.; FLAUZINO, R. A. **Redes Neurais Artificiais para Engenharia e Ciências Aplicadas**. ArtLiber Editora, 2010.

SUGUNA, N.; THANUSHKODI, K. **An improved k-nearest neighbor classification using genetic algorithm**. International Journal of Computer Science Issues, v. 7, n. 2, pp. 18–21, 2010.

TensorFlow Lite for Microcontrollers. 2023, March. TensorFlow. Disponível em: https://www.tensorflow.org/lite/microcontrollers. Acesso em: 29 de maio de 2023.

TensorFlow. 2023, March. TensorFlow. Disponível em: https://www.tensorflow.org/.

THOMPSON, Michael J.; WILSON, David. **Auxiliary DC control power system design for substations**. In: 2007 60th Annual Conference for Protective Relay Engineers. IEEE, 2007. p. 522-533.

Processamento de áudio. VELARDO, V. Disponível em: https://valeriovelardo.com/. Acesso em: 7 de junho de 2023.

VALVANO, Jonathan W. Embedded Systems: **Introduction to ARM Cortex-M Microcontrollers**. Ano de publicação: 2018.

VIRTANEN, Tuomas; PLUMBLEY, Mark D.; ELLIS, Dan (Ed.). **Computational analysis of sound scenes and events**. Berlin, Germany: Springer International Publishing, 2018.

WANG, Yuxuan et al. **Trainable frontend for robust and far-field keyword spotting**. In: 2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE, 2017. p. 5670-5674.

WARDEN, P.; SITUNAYAKE, D. **TinyML: Machine Learning with TensorFlow Lite on Arduino and Ultra-Low-Power Microcontrollers**. O'Reilly Media, 2019.

ZHANG, Z.; LI, J. **A Review of Artificial Intelligence in Embedded Systems**. Micromachines, v. 14, n. 5, p. 897, 2023.

ZHU, B.; XU, K.; WANG, D.; ZHANG, L.; LI, B.; PENG, Y. **Environmental sound classification based on multi-temporal resolution convolutional neural network combining with multi-level features**. In: Advances in Multimedia Information Processing–PCM 2018: 19th Pacific-Rim Conference on Multimedia, Hefei, China, September 21-22, 2018, Proceedings, Part II 19. Springer International Publishing, p. 528-537, 2018.

Apêndice A: Trabalhos Gerados

A seguinte produção foi gerada ao longo desta dissertação:

- Andrade, P. R. P.; Pessin, G. Identificação de Falhas em Sistemas UPS (Uninterruptible Power Supply) por meio de Deep Learning em Sistema Embarcado. In: XVI Simpósio Brasileiro de Automação Inteligente (SBAI 2023) - Automining, Manaus, 2023.