Bruno Vilela Batista
Felipe Souza Fernandes
Rodrigo Nascimento Pereira

Pluviômetro Sem Partes Móveis: Aplicação de modelos CNN para Detecção Sonora de Chuvas em Dispositivos de Borda

Brasil

Dezembro de 2024

Bruno Vilela Batista
Felipe Souza Fernandes
Rodrigo Nascimento Pereira

Pluviômetro Sem Partes Móveis: Aplicação de modelos CNN para Detecção Sonora de Chuvas em Dispositivos de Borda

Monografia referente ao Trabalho Final de Graduação do curso de Engenharia da Computação da Universidade Federal de Itajubá

Universidade Federal de Itajubá – UNIFEI
Instituto de Engenharia de Sistemas e Tecnologia da Informação
Engenharia da Computação

Orientador: José Alberto Ferreira Filho
Coorientador: Marcelo José Rovai

Brasil
Dezembro de 2024

Este trabalho é dedicado aos que fazem da educação um ato de amor e transformação. Em especial, aos nossos professores, que nos mostraram o caminho, e às nossas famílias, que nos deram forças para percorrê-lo.

Agradecimentos

Expressamos nossa profunda gratidão aos nossos orientadores, José Alberto Ferreira Filho e Marcelo José Rovai, pelo valioso acompanhamento, orientação e dedicação ao longo de todo o desenvolvimento deste trabalho. Agradecemos também aos nossos pais e familiares, por seu apoio incondicional, e aos amigos, por sua parceria e incentivo, que foram fundamentais para a conclusão deste projeto.

Por vezes sentimos que aquilo que
fazemos é apenas uma gota no oceano. Mas,
o oceano seria menor se lhe faltasse uma gota.
(Madre Teresa de Calcutá)

Resumo

Este trabalho apresenta o desenvolvimento de um modelo de inteligência artificial capaz de realizar a medição de precipitação pluvial por meio de detecção sonora, que elimina a necessidade de partes móveis, comuns em pluviômetros tradicionais. A abordagem utiliza redes neurais convolucionais (CNNs) e dados do dataset Aurora, o qual possui mais de 3000 horas de áudio e informações meteorológicas. Foi implementado um sistema de classificação sonora para identificar a intensidade da chuva (sem chuva, chuva fraca, moderada e forte), por meio da plataforma Edge Impulse para integração em dispositivos de borda, como o Arduino Nano 33 BLE Sense. Os resultados demonstram a viabilidade da solução, destacando sua acurácia e potencial como alternativa econômica e de baixa manutenção para monitoramento climático. Este estudo contribui para avanços na aplicação de aprendizado de máquina em soluções ambientais sustentáveis.

Palavras-chave: inteligência artificial. CNN. chuva. áudio. Arduino. Edge Impulse

Abstract

This work presents the development of an artificial intelligence model designed to measure rainfall using sound detection, eliminating the need for moving parts typically found in traditional rain gauges. The approach employs convolutional neural networks (CNNs) and utilizes data from the Aurora dataset, which consists of over 3,000 hours of audio and meteorological information. A sound classification system was implemented to identify rain intensity categories: no rain, light rain, moderate rain, and heavy rain. This was achieved using the Edge Impulse platform to integrate edge devices like the Arduino Nano 33 BLE Sense. The results demonstrate the feasibility of the solution, highlighting its accuracy and potential as an economical, low-maintenance alternative for climate monitoring. This study contributes to advances in the application of machine learning for sustainable environmental solutions.

Keywords: artificial intelligence. CNN. rain. audio. Arduino. Edge Impulse.

Lista de ilustrações

Figura 1 – Pluviômetro automático basculante 22
Figura 2 – Fluxograma da extração de *features* com diferentes técnicas 25
Figura 3 – Esquemático de um neurônio 28
Figura 4 – Arduino Nano 33 BLE Sense 35
Figura 5 – Diagrama do dispositivo de captura do *dataset* Aurora 38
Figura 6 – Arquivos *.dat* presentes no *dataset* Aurora 38
Figura 7 – Estrutura dos dados presentes no *dataset* Aurora 39
Figura 8 – Gráfico da distribuição de chuva acumulada segundo o *dataset* Aurora . 39
Figura 9 – Gráfico da distribuição de chuva acumulada segundo as informações do World Weather Online fornecidas pela estação meteorológica da cidade de Neuchatel-Suíça 40
Figura 10 – Parte da tabela com as semanas disponíveis no Aurora *Dataset* 43
Figura 11 – Visualização da amplitude de sinais exemplos de cada classe do modelo 44
Figura 12 – Comparação entre os três modelos de DSP 45
Figura 13 – Janela de Hamming 46
Figura 14 – Espectrograma para janela de 6 segundos 47
Figura 15 – MFE das diferentes classes presentes no modelo 47
Figura 16 – Arquitetura do modelo 50
Figura 17 – Modelos obtidos pela ferramenta *EON Tuner* disponível na plataforma *Edge Impulse* 51
Figura 18 – Comparação dos modelos otimizados e não otimizados para *deploy* para biblioteca arduino por meio da ferramenta *EON Compiler*, disponível no *Edge Impulse* 52
Figura 19 – Janela de áudio exemplo para inferência na placa 53
Figura 20 – Inferência da Classe *HighRain* 53
Figura 21 – Inferência da Classe *MediumRain* 53
Figura 22 – Inferência da Classe *LowRain* 54
Figura 23 – Inferência da Classe *NoRain* 54
Figura 24 – Código para criação do CSV e Rotulação - parte 1 65
Figura 25 – Código para criação do CSV e Rotulação - parte 2 66
Figura 26 – Código para criação dos gráficos de chuva acumulada 67
Figura 27 – Código para seleção dos trechos de áudio - Parte 1 69
Figura 28 – Código para seleção dos trechos de áudio - Parte 2 70
Figura 29 – Código para demonstração do MFE - Parte 1 71
Figura 30 – Código para demonstração do MFE - Parte 2 72

Lista de tabelas

Tabela 1 – Classes e limites de precipitação utilizados pelo vencedor do *Next-Gen tinyML Smart Weather Station* 41
Tabela 2 – Classes e limites de precipitação utilizados no estudo italiano 41
Tabela 3 – Classes e limites de precipitação utilizados neste estudo 42

Lista de abreviaturas e siglas

TB	Tipping Bucket
USB	Universal Serial Bus
IDE	Integrated Development Environment
MFE	Mel-Frequency Energy
MFCC	Mel-Frequency Cepstral Coefficients
FFT	Fast Fourier Transform
DCT	Discrete Cosine Transform
IoT	Internet of Things
CNN	Convolutional Neural Network
NumPy	Numerical Python
CSV	Comma separated values
ML	Machine Learning
TinyML	Tiny Machine Learning
ReLU	Rectified Linear Unit
RNN	Recurrent Neural Network
LED	Light Emitting Diode

Sumário

1 INTRODUÇÃO . . . 19
1.1 Motivação . . . 19
1.2 Metodologia . . . 20
1.3 Objetivo . . . 20

2 REVISÃO BIBLIOGRÁFICA . . . 21
2.1 Meteorologia e pluviosidade . . . 21
2.2 Equipamentos para medição de precipitação . . . 21
2.2.1 Pluviômetro convencional . . . 21
2.2.2 Pluviômetro automático basculante . . . 22
2.3 *Python* e bibliotecas de processamento de dados . . . 22
2.3.1 *Numerical Python (NumPy)* . . . 23
2.3.2 *Pandas* . . . 23
2.3.3 *Matplotlib* . . . 23
2.4 Sinais de Áudio e Som . . . 23
2.4.1 Conversão analógico-digital . . . 24
2.5 Técnicas de extração de *features* para áudio . . . 24
2.5.1 Espectograma . . . 25
2.5.2 *Mel-frequency energy* (MFE) . . . 26
2.5.3 *Mel-frequency Cepstral Coefficients* (MFCC) . . . 26
2.6 *Machine Learning* (ML) . . . 27
2.6.1 *Tiny Machine Learning (TinyML)* . . . 27
2.6.2 Tarefas de aprendizado de máquina . . . 27
2.6.2.1 Classificação . . . 27
2.6.2.2 Regressão . . . 27
2.7 Redes neurais . . . 28
2.7.1 *Long Short-Term Memory* (LSTM) . . . 29
2.7.2 Redes neurais convolucionais . . . 29
2.7.3 Camadas Ocultas . . . 29
2.7.3.1 Camadas densas . . . 30
2.7.3.2 Camadas convolucionais . . . 30
2.7.3.3 *Pooling* . . . 30
2.7.3.4 *Dropout* . . . 31
2.7.3.5 *Flatten* . . . 31
2.7.3.6 *Reshape* . . . 31

2.7.4 Funções de Ativação 31
2.7.4.1 *Rectified Linear Unit (ReLU)* 31
2.7.4.2 *Softmax* 32
2.8 Métricas para validação do modelo 32
2.8.1 Matriz de confusão 32
2.8.2 Acurácia 33
2.8.3 F_1 *score* 33
2.9 Edge Impulse 33
2.10 Arduino 34
2.11 Estudos Anteriores 35

3 METODOLOGIA 37
3.1 Aurora *Dataset* 37
3.2 Visualização de dados 39
3.3 Construção da base de dados 40
3.3.1 Limites de precipitação 40
3.3.2 *Labels* 41
3.3.3 Finalização do *Dataset* 42
3.4 *Feature engineering* 44
3.5 Arquitetura 48
3.5.1 Estudo do modelo 48
3.5.2 Classificação 48

4 EXPERIMENTOS E RESULTADOS 51
4.1 Melhores modelos 51
4.2 Inferência 52

5 CONCLUSÃO 55
5.1 Trabalhos Futuros 55

REFERÊNCIAS 57

APÊNDICES 63

APÊNDICE A – CÓDIGO UTILIZADO PARA CRIAÇÃO DO CSV E DA ROTULAÇÃO 65

APÊNDICE B – CÓDIGO UTILIZADO PARA CRIAÇÃO DOS GRÁFICOS DE CHUVA ACUMULADA 67

APÊNDICE C – CÓDIGO UTILIZADO PARA SELEÇÃO DOS TRECHOS DE ÁUDIO 69

APÊNDICE D – CÓDIGO UTILIZADO PARA DEMONSTRAÇÃO DO MFE 71

1 Introdução

.

A medição da chuva é uma prática utilizada no campo da meteorologia para coletar informações sobre os recursos hídricos. Essa atividade contribui para a previsão do tempo, estudos climáticos e a gestão de bacias hidrográficas. O monitoramento da precipitação pluvial fornece dados que podem ser utilizados em áreas como agricultura, planejamento urbano, controle de enchentes e secas, além da formulação de políticas relacionadas ao meio ambiente e ao desenvolvimento sustentável.

A importância do monitoramento de precipitação se estende à capacidade de prever e mitigar desastres naturais. Enchentes e deslizamentos de terra, que frequentemente resultam em perdas humanas e materiais significativas, podem ser melhor previstos e gerenciados com base em dados pluviométricos confiáveis. Da mesma forma, a compreensão das variações sazonais e anuais da precipitação é fundamental para o planejamento agrícola, permitindo uma gestão mais eficiente dos recursos de irrigação e contribuindo para a segurança alimentar (KOBIYAMA et al., 2006, p. 89–90).

Este monitoramento é feito por pluviômetros, dispositivos que permitem medir a quantidade de precipitação em uma determinada área durante um período específico. Os mais básicos são compostos por um recipiente que coleta a água da chuva, permitindo calcular o volume acumulado. O valor obtido é normalmente expresso em milímetros, que corresponde a quantidade de litros de chuva em um metro quadrado.

1.1 Motivação

Os pluviômetros automáticos tradicionais, como o basculante, possuem diversas partes móveis, componentes que se deslocam para realizar a medição da precipitação, tornando-os suscetíveis a falhas devido à exposição às condições ambientais e à falta de manutenção (MEDEIROS; GOLDENFUM; CASTRO, 2003). O projeto é motivado pelo estudo de uma alternativa aos pluviômetros clássicos, buscando analisar a viabilidade de modelos de inteligência artificial, que realizem medições pluviométricas de forma eficiente e ao mesmo tempo superar os principais desafios enfrentados, como a presença de partes móveis e a necessidade de manutenção frequente.

1.2 Metodologia

Para desenvolvimento do trabalho foram utilizados dados de um repositório online de livre acesso (CSEM, 2024) que possui mais de 3000 horas de áudio de chuvas, além de dados meteorológicos, capturados ao longo de semanas de 2023 e 2024, na cidade Neuchâtel na Suíça. Para a criação de modelos, redes neurais convolucionais (CNNs) foram escolhidas. Com o intuito de treinar o modelo de classificação para intensidade de chuvas, foi escolhido a plataforma de desenvolvimento para aprendizado de máquina para dispositivos de borda, Edge Impulse.

A fim de integrar os modelos de inteligência artificial criados a um dispositivo de borda, a placa de desenvolvimento escolhida foi um Arduino NANO 33 BLE Sense, em função da afinidade dos autores com os softwares de implementação para a placa e sua disponibilidade.

Por fim, a integração entre software e hardware será realizada utilizando o próprio Edge Impulse, em conjunto com a plataforma de desenvolvimento do Arduino. Esses dois sistemas serão responsáveis por todas as etapas, desde a criação até a implementação e execução do modelo de inteligência artificial diretamente no dispositivo.

1.3 Objetivo

O trabalho tem como objetivo principal avaliar a viabilidade da utilização de inteligência artificial para medição das chuvas, visando permitir a implementação futura de um pluviômetro de baixo custo e sem partes móveis. Para isso, o foco é desenvolver um modelo de *machine learning* capaz de identificar a intensidade da chuva através do som, utilizando o *dataset* mencionado.

A partir do desenvolvimento desse modelo, espera-se fornecer uma base sólida para o aprimoramento de sistemas de medição de precipitação, possibilitando uma solução acessível e eficaz para monitoramento climático. O trabalho também busca contribuir para o avanço da aplicação de inteligência artificial em soluções ambientais e de baixo custo.

2 Revisão Bibliográfica

Este capítulo apresenta a fundamentação teórica necessária para o entendimento do trabalho. As seções seguintes foram separadas em tópicos individuais, partindo de conceitos mais amplos de meteorologia até tecnologias e modelos utilizados no desenvolvimento.

2.1 Meteorologia e pluviosidade

Meteorologia é a ciência que estuda atmosfera e os fenômenos relacionados ao clima e tempo, como chuva, ventos e temperatura (INMET, c2019) e as interações desses processos físicos e químicos com os sistemas litosfera, hidrosfera, criosfera e biosfera (YNOUE et al., 2017).

Nesse contexto, a pluviosidade, um dos fenômenos analisados pela meteorologia, refere-se ao volume de chuva que precipita em um determinado espaço em um certo período de tempo, em que a unidade de medida mais comum de ser utilizada é milímetros (mm), que corresponde a $1L/m^2$ de superfície (MENEZES, 2019). O monitoramento da pluviosidade, por exemplo, ajuda a entender a variação dos regimes de chuva, que pode causar impactos negativos em zonas agrícolas, cidades e na vida da população (PIAZZA et al., 2016).

2.2 Equipamentos para medição de precipitação

Pluviômetro é o nome dado ao instrumento que realiza a medição da precipitação de chuva em uma determinada localidade (MILANESI; ALVES; GALVANI, 2017). Esse equipamento é essencial em diversas áreas, especialmente na agricultura e meteorologia (JÚNIOR et al., 2018).

2.2.1 Pluviômetro convencional

O pluviômetro convencional é constituído apenas por uma abertura de captação e um recipiente para armazenamento da água captada. A aferição pode ser realizada de duas formas principais, o exterior do reservatório pode ser graduado em milímetros de precipitação, possibilitando uma leitura direta, ou a água coletada é medida posteriormente, bastando haver conhecimento sobre a área de captação.

Após a ocorrência de uma chuva ou término de um período de análise predeterminado, o equipamento deve ser lido e interpretado manualmente. Por fim, o recipiente deve ser esvaziado a fim de liberar o dispositivo para novas medições.

2.2.2 Pluviômetro automático basculante

Pluviômetros do tipo automático de báscula, também chamados de pluviômetros de cuba basculante ou pluviômetros de caçamba basculante, conhecidos em inglês como *Tipping Bucket* (TB), (DORNELLES; OLIVEIRA; REVEILLEAU,), utilizam-se de um coletor para direcionar água a duas básculas conectadas a um eixo, que, por sua vez, tem um ímã acoplado. Essas básculas são estruturas de tamanho conhecido que se enchem de água de forma alternada.

Quando uma se enche, o peso da água faz com que ela se incline, expulsando seu conteúdo. Simultaneamente, o outro recipiente assume a posição de coleta, começando a captar a água da chuva. Cada um desses movimentos de alternância, por meio do ímã conectado ao eixo e de um circuito embarcado presente na estrutura, gera um pulso elétrico. Os pulsos gerados são lidos e armazenados em um outro dispositivo que compõe o pluviômetro, chamado *datalogger*. Posteriormente, pulsos são convertidos em volume de chuva de acordo com o volume da báscula (BLAINSKI; GARBOSSA; ANTUNES, 2012).

Figura 1 – Pluviômetro automático basculante

Fonte: TB3... (c2024)

2.3 *Python* e bibliotecas de processamento de dados

O *Python* é uma linguagem de programação de alto nível, interpretada, de uso geral e popular devido a sua simplicidade e legibilidade. Ela foi criada por Guido Van Rossum no início da década de 90 e possui recursos como tipagem dinâmica e forte, multiparadigmas e código aberto (PYTHON, 2024).

Nos últimos anos, o *Python* ganhou relevância nas áreas de ciência de dados e aprendizado de máquina, tornando-se uma das linguagens mais populares devido ao suporte às bibliotecas, como *Pandas* e *NumPy* (MCKINNEY, 2018).

2.3.1 *Numerical Python (NumPy)*

A *NumPy* é uma biblioteca essencial para computação científica. Ela fornece *arrays* multidimensionais que são rápidos e eficientes, operações vetorizadas e matemáticas, álgebra linear, transformadas de Fourier, geração de números aleatórios e manipulação de dados em geral. Com essas funcionalidades, a *NumPy* se torna uma ferramenta indispensável, proporcionando tanto alta performance quanto flexibilidade na manipulação e processamento de grandes volumes de dados numéricos (NUMPY, c2022).

2.3.2 *Pandas*

O *Pandas* oferece estruturas de dados avançadas e funções poderosas, com o objetivo de tornar o trabalho com dados estruturados ou tabulares mais rápido, fácil e intuitivo. As principais estruturas são o *DataFrame*, que é uma tabela orientada a colunas e com rótulos para linhas e colunas, e a *Series*, um objeto unidimensional com rótulo, semelhante a um *array* (MCKINNEY, 2018). Além disso, a biblioteca possibilita a leitura e escrita de dados em diversos formatos, como *Comma Separeted Values* (CSV) e planilhas Excel, e oferece funcionalidades para lidar com dados faltantes, agregação e análise estatística.

2.3.3 *Matplotlib*

O *Matplotlib* é uma biblioteca que permite a criação de gráficos estáticos, animados e interativos a partir de dados, oferecendo ampla personalização de estilos e *layouts*. Extremamente flexível, ela suporta uma grande variedade de tipos de gráficos (MATPLOTLIB, c2024). É comumente usada em conjunto com o *Pandas*, para visualizar dados armazenados em *DataFrames*.

2.4 Sinais de Áudio e Som

O som é uma onda mecânica, ou seja, se propaga em meios vibrantes, a exemplo de partículas do ar (COSTA et al., 2023). Ele pode ser representado por um sinal analógico, que é um sinal contínuo que expressa uma medida física e pode assumir qualquer valor em um dado intervalo, além de poder variar de maneira sutil ao longo do tempo (ADAM, 1988). Matematicamente, um sinal é representado por uma função de variável independente t, em que na maioria dos casos, representa o tempo. Caso esta função seja contínua, ou seja, $\lim_{x \to a} f(x) = f(a)$, este sinal é analógico (ELIAS, 2020). Quando este sinal é transmitido, gravado ou reproduzido por meios digitais, é chamado de áudio (CAMBRIDGE, c2024).

Para a captura desses sinais, é utilizado um microfone, que é um dispositivo que converte ondas sonoras em sinais elétricos. Um microfone de pressão dinâmica, por exemplo,

possui um diafragma que se move em respostas a essas ondas, capta vibrações, gerando uma corrente elétrica proporcional à pressão sonora (EARGLE, 2004).

2.4.1 Conversão analógico-digital

A conversão de um sinal analógico para digital é chamada de conversão A/D. Esse processo é dividido em duas etapas de discretização, amostragem (tempo) e quantização (amplitude), que limitam a quantidade de informações de um sinal analógico para que ele possa ser analisado e interpretado de maneira correta, sem perdas significativas de informação (ELIAS, 2020).

Amostragem é a captura de informações de um sinal ao longo do tempo, dada uma frequência de amostragem ω_a. Essa frequência, caso respeite o teorema de Nyquist (teorema que demonstra a frequência de amostragem mínima para a recuperação de um sinal), garante que as características da forma serão mantidas após a transformação do contínuo para o digital (ELIAS, 2020).

A quantização, por sua vez, limita a quantidade de informações na amplitude, por meio da definição de faixas de valores finitas. Se a amplitude do sinal estiver em um certo intervalo, o valor é ajustado para o valor médio desse intervalo. É importante frisar que um sinal não será reconstruído com fidelidade total, há sempre um erro que não pode ser corrigido, que é chamado de erro de quantização (ELIAS, 2020).

2.5 Técnicas de extração de *features* para áudio

Um sinal bruto contém, em muitos casos, dados que escondem as características de interesse para uma análise (KRISHNA et al., 2021). Nesse sentido é necessário aplicar sobre esse sinal, técnicas de extração das chamadas *features* (Figura 2). Features são atributos ou propriedades específicas extraídas do sinal que representam informações relevantes de forma simplificada e compreensível, preservando os aspectos essenciais para a análise.. Para o âmbito da inteligência artificial e aprendizado de máquina, isso acelera o entendimento e generalização do modelo (DEO et al., 2020).

Antes da aplicação das técnicas de extração, é necessário realizar uma etapa de pré-processamento. Nessa etapa são realizados 4 passos, sendo eles: pré-ênfase, *frame blocking*, janelamento e transformação de Fourier (ABDICHE; HARRAR, 2021).

O processo é iniciado pela amplificação das altas frequências, portadoras de menos energia, do sinal (ABDICHE; HARRAR, 2021). Essa amplificação é realizada seguindo a Fórmula 2.1 (TURNER et al., 2011), onde a é igual a 0.95 ou 0.98.

$$X(t) = z(t) - az(t-1) \tag{2.1}$$

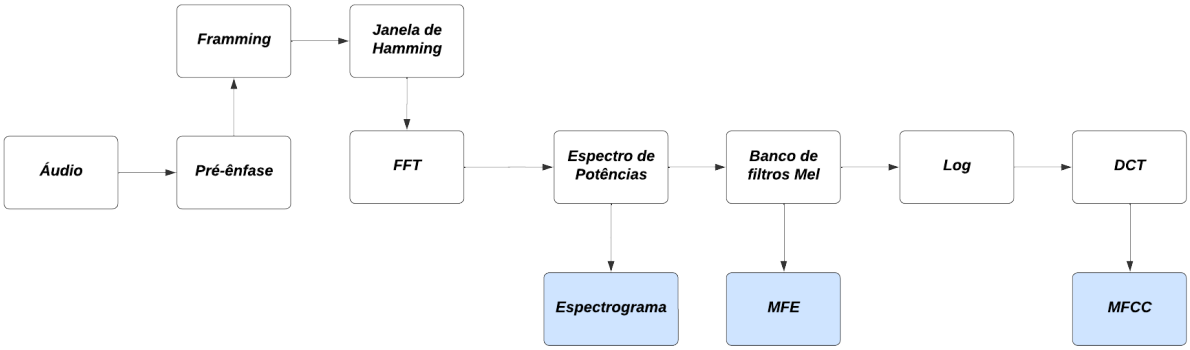


Figura 2 – Fluxograma da extração de *features* com diferentes técnicas

Fonte: Autoria própria

Após isso, o sinal é subdividido em "quadros" geralmente de duração entre 20 e 40 milissegundos (PRACTICAL..., c2012), o que garante estabilidade às amostras e permite uma análise detalhada de cada quadro (ABDICHE; HARRAR, 2021).

O sinal, agora dividido em quadros, tem cada subdivisão multiplicada por uma janela de Hammimg, caracterizada pela Equação 2.2 (GOEL; GUPTA, 2014), onde N é o número de amostras. Esse procedimento configura continuidade ao sinal e reduz perdas espectrais (ABDICHE; HARRAR, 2021; LIN et al., 2024).

$$W(n) = 0,54 - 0,46\cos(\frac{2\pi n}{N-1}), \quad 0 \leq n \leq N-1 \tag{2.2}$$

Neste momento é aplicada a transformada de Fourier, comumente sua variação rápida *Fast Fourier Transform* (FFT) seguindo a Equação 2.3 onde $n = 0, 1, 2, ..., N-1$. Isso realiza a transposição do sinal no domínio do tempo para o domínio da frequência, ou seja, para informação espectral (ABDICHE; HARRAR, 2021).

$$Z_2(n) = \sum_{k=0}^{n-1} Z_1(k)e^{-i\omega n/N} \tag{2.3}$$

2.5.1 Espectograma

Partindo da informação obtida após a aplicação do pré processamento, é necessário obter o espectro de potências que representa o espectrograma. Isso é realizado como mostra a Equação 2.4, onde N é o numero de pontos da FFT e $X(f)$ é a FFT do sinal (PROCESSING-BLOCKS/SPECTROGRAM..., 2023).

$$P(f) = (1/N) \cdot |X(f)|^2 \tag{2.4}$$

Essa técnica permite visualizar como as frequências e suas intensidades variam com o tempo. Também é válido citar que essa estratégia não foca em destacar pontos específicos do sinal, apenas apresenta a distribuição de potência do sinal de forma direta.

2.5.2 *Mel-frequency energy* (MFE)

As *features* MFE buscam representar o som como se ele fosse percebido pela audição humana. Nesse sentido, o MFE apresenta a distribuição de energia nas diferentes faixas de frequência do sinal, trazidas para a escala Mel, uma escala não linear que destaca as frequências mais baixas. Para obtenção dessas *features*, é necessário aplicar sobre o espectro de potências uma série de filtros triangulares, baseados na escala Mel, que compõe um banco de filtros. (LIN et al., 2024).

A escala Mel é caracterizada pela Equação 2.5 (O'SHAUGHNESSY, 1987), nela está disposto a conversão de frequência para escala Mel.

$$m = 2595 \log_{10}(1 + f/700) \tag{2.5}$$

A construção do banco de filtros segue a Equação 2.6 (ABDUL; AL-TALABANI, 2022), onde $f(m)$ é a frequência central do filtro e $\sum_m^{M-1} H_m(k) = 1$:

$$H_m(k) = \begin{cases} 0 & \text{se } k < f(m-1) \\ \dfrac{k - f(m-1)}{f(m) - f(m-1)} & \text{se } f(m-1) \leq k \leq f(m) \\ 1 & \text{se } k = f(m) \\ \dfrac{f(m+1) - k}{f(m+1) - f(m)} & \text{se } f(m) < k \leq f(m+1) \\ 0 & \text{se } k > f(m+1) \end{cases} \tag{2.6}$$

2.5.3 *Mel-frequency Cepstral Coefficients* (MFCC)

As *features* MFCC também apresentam a distribuição de energia do sinal. Entretanto, este método executa uma compactação extra que suaviza o espectro e permite a captura de informações mais finas do áudio, contidas no envelope espectral do sinal (JENSEN et al., 2006), fazendo com que uma das principais indicações de uso da técnica de MFCC seja tratar de áudios que envolvem voz humana. Para isso, é necessário aplicar a operação de *discrete cosine transform* (DCT), mostrada na Equação 2.7, onde X_n é o sinal discreto e N é o comprimento do sinal, sobre o logaritmo natural dos dados após aplicação dos filtros Mel (ABDICHE DEHIA; HARRAR, 2019).

$$X(k) = \sum_{n=0}^{N-1} x_n \cdot \cos\left(\frac{2\pi jnk}{N}\right), \quad k = 1, 2, 3, \ldots, N-1 \tag{2.7}$$

2.6 *Machine Learning* (ML)

Machine learning, ou aprendizado de máquina, é uma forma de estatística aplicada com ênfase no uso de computadores para estimar funções complexas, sem focar na prova de intervalos de confiança em torno dessas funções (BENGIO; GOODFELLOW; COURVILLE, 2017). Algoritmos de ML, são algoritmos que aprendem com dados. Segundo Mitchell (1997), em tradução livre, "Um programa de computador é dito aprender com uma experiência E em relação a uma classe de tarefas T e uma medida de desempenho P, se o seu desempenho nas tarefas em T, conforme medido por P, melhora com a experiência E".

2.6.1 *Tiny Machine Learning (TinyML)*

TinyML refere-se a aplicação de modelos de ML em dispositivos computacionais com recursos limitados. O objetivo é permitir que esses modelos sejam executados de maneira eficiente em dispositivos com memória reduzida, baixo poder de processamento e bateria limitada. Além disso, todo o processamento é realizado localmente, sem necessidade de enviar dados para servidores na nuvem. Esse conceito, conhecido como computação em borda, proporciona menor latência, maior privacidade e segurança (RAY, 2022).

2.6.2 Tarefas de aprendizado de máquina

Tarefas de aprendizado de máquina são descritas em termos de como o sistema deve processar um *example*. *Examples* são coleções de *features* quantitativamente medidas que o sistema deve processar. Uma notação para um exemplo é um vetor $x \in \mathbb{R}^n$ em que cada entrada x_i é uma nova *feature* (BENGIO; GOODFELLOW; COURVILLE, 2017). Diversas tarefas podem ser solucionadas com aprendizado de máquina. Neste trabalho, o foco será em duas dessas tarefas.

2.6.2.1 Classificação

Neste tipo de tarefa, o computador é solicitado a especificar a qual classe uma entrada k pertence. Para realizar essa tarefa, o algoritmo de aprendizado cria uma função $f : \mathbb{R}^n \to \{1, ..., k\}$. Dada uma igualdade $y = f(x)$, o modelo atribui uma entrada descrita pelo vetor x a uma categoria identificada pelo valor numérico y (BENGIO; GOODFELLOW; COURVILLE, 2017). Um exemplo de classificação é o reconhecimento, em que a entrada é uma imagem (que pode ser representada por um vetor com o valor de intensidade dos *pixels*) e a saída é um valor numérico, que identifica o objeto da imagem.

2.6.2.2 Regressão

Na regressão, o computador deve prever um valor numérico a uma dada entrada. Para realizar essa tarefa, o algoritmo de aprendizado cria uma função $f : \mathbb{R}^n \to \mathbb{R}$

(BENGIO; GOODFELLOW; COURVILLE, 2017). Há semelhanças com o processo de classificação, porém, a saída tem um formato diferente. Dada uma entrada descrita pelo vetor x, o algoritmo retorna um valor exato. Um exemplo de regressão é a previsão de dados financeiros, que por meio de um *dataset* de dados anteriores, prevê-se qual será o próximo valor.

2.7 Redes neurais

Nas seções anteriores, foram explorados os conceitos fundamentais de aprendizado de máquina, como extração de *features* e tarefas de um algoritmo. Nesta seção, será aprofundado o conceito de redes neurais, uma classe de modelos capaz de capturar padrões mais complexos e abstratos de dados.

Uma rede neural é um modelo computacional utilizado em inteligência artificial que simula o funcionamento de um cérebro (BENGIO; GOODFELLOW; COURVILLE, 2017). Ela é composta por neurônios artificiais, que atuam como unidades de processamento e são organizados em camadas (conjuntos de neurônios de mesma profundidade). Essas redes são capazes de processar informações e encontrar padrões de dados (NIELSEN, 2015).

De maneira mais detalhada, um neurônio recebe sinais de entrada, que são valores numéricos, de outros neurônios ou os próprios dados de entrada da rede. Cada uma dessas entradas tem um peso, em que o neurônio faz uma soma ponderada e adiciona um termo de ajuste de cálculo chamado *bias*. O resultado dessa soma passa como entrada de uma função de ativação, que retorna um valor numérico, o qual será utilizado por outros neurônios ou como a própria saída da rede neural, caso seja da camada de saída (BENGIO; GOODFELLOW; COURVILLE, 2017).

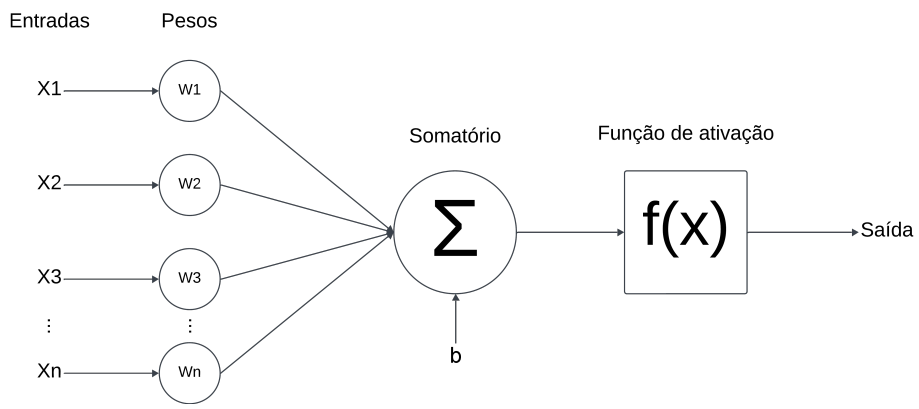


Figura 3 – Esquemático de um neurônio

Fonte: Autoria própria

As camadas podem ser separadas em camadas de entrada, ocultas ou saída. As camadas de entrada e saída, de maneira geral, são bem diretas. Em uma rede de exemplo, suponha que deseja-se determinar se uma imagem de uma escrita é um "9"ou não. Uma

maneira de criar a rede seria colocar a intensidade de cada pixel da imagem em um neurônio. Se a imagem for 64 por 64 em escala de cinzas, teremos portanto $64 \times 64 = 4096$ neurônios de entrada, em que a intensidade será um valor entre 0 e 1. A camada de saída terá apenas um neurônio, com valores de saída menores que 0.5 indicando "a imagem de entrada não é um 9", e valores maiores que 0.5 como "a imagem de entrada é um 9"(BENGIO; GOODFELLOW; COURVILLE, 2017). As camadas ocultas, por sua vez, não possuem um processo exato para o *design*, mas são importantes para o aumento de complexidade de uma rede neural.

2.7.1 *Long Short-Term Memory* (LSTM)

Uma rede do tipo *Long Short-Term Memory* é um tipo específico de rede neural recorrente. Redes neurais recorrentes (RNNs) são tipos especiais de redes neurais projetadas para problemas de sequência. Dada uma rede *multi-layer perceptron feedforward* padrão, uma RNN pode ser vista como a adição de loops à arquitetura. Por exemplo, em uma certa camada, cada neurônio pode passar seu sinal lateralmente ou também encaminhá-lo para a próxima camada. A saída da rede pode ser retroalimentado como uma entrada da rede com o próximo vetor de entrada, e assim por diante. (BROWNLEE, 2017).

A diferença de um modelo LSTM para uma RNN é evitar problemas comuns de RNNs, a exemplo de gradientes explosivos, que são gradientes que crescem exponencialmente durante a retroalimentação, o que causa a divergência do modelo. Esta capacidade é atingida por meio de uma estrutura de neurônio que utiliza memórias de curto e longo prazo, e que ao se conectar a outros neurônios, cria uma rede especializada em problemas de predição de sequência (BROWNLEE, 2017).

2.7.2 Redes neurais convolucionais

Redes neurais convolucionais (CNNs) são um tipo de rede neural especializado no processamento de dados em grade. Exemplos incluem séries temporais, que podem ser entendidos como uma grade 1D de intervalos de tempo regulares, e imagens, que podem ser interpretadas como uma grade 2D formada de *pixels*. O nome dessa rede indica o uso da operação matemática de convolução, um tipo específico de operação linear. CNNs são, portanto, redes neurais que utilizam convolução ao invés de multiplicação matricial em pelo menos uma de suas camadas (BENGIO; GOODFELLOW; COURVILLE, 2017).

2.7.3 Camadas Ocultas

As Camadas Ocultas em redes neurais são camadas que se encontram entre a camada de entrada e a camada de saída. São nessas camadas que ocorre o processo de aprendizado, na qual os dados da entrada são processados, características são destacadas

e padrões são aprendidos. O número de camadas ocultas pode variar: redes rasas possuem poucas camadas, enquanto redes profundas têm muitas camadas, tornando-as, portanto, mais complexas. Existem diferentes tipos de camadas ocultas, cada uma com uma função específica, como, por exemplo, camadas densas e camadas convolucionais (NIELSEN, 2015).

2.7.3.1 Camadas densas

Camadas Densas, também conhecidas como camadas totalmente conectadas, são comuns em redes neurais. Nessa camada, todos os seus neurônios estão conectados a todos os neurônios da camada anterior. Dessa forma, trata-se de uma camada versátil, podendo ser utilizada para tarefas de aprendizado, como classificação e regressão, devido a sua capacidade de aprender representações complexas dos dados (AGGARWAL et al., 2018).

2.7.3.2 Camadas convolucionais

São as camadas que caracterizam as CNNs, amplamente utilizadas no processamento de imagens. Nessas camadas, filtros, também denominados *kernels*, são aplicados à imagem com o objetivo de identificar a presença de características específicas, como bordas e texturas. Os filtros consistem em matrizes de pequeno tamanho que se deslocam pela imagem, sendo regidos por dois parâmetros fundamentais: o tamanho, que especifica as dimensões do *kernel*, e o *stride*, que determina o número de *pixels* que o filtro avança a cada operação de convolução. O resultado da aplicação dos filtros são mapas de ativação, que contém as características encontradas (BUNRIT et al., 2019).

2.7.3.3 *Pooling*

As camadas de *pooling*, comumente utilizadas após as camadas de convolução, têm a função de simplificar os resultados dessas camadas. Elas operam sobre cada mapa de ativação gerado pela convolução, condensando-o em um novo mapa com menor dimensionalidade, enquanto preservam as informações mais relevantes para a tarefa de aprendizado (NIELSEN, 2015). Existem dois tipos principais de *pooling*: *max pooling* e *average pooling*. O *max pooling*, sendo o mais frequentemente empregado, seleciona o valor máximo de cada região da janela de *pooling*, enquanto o *average pooling* calcula a média dos valores dessa região. Assim como na camada de convolução, a região de *pooling* é caracterizada por um tamanho definido, que especifica as dimensões da janela de *pooling*, e um *stride*, que determina o número de *pixels* que o filtro avança a cada operação de *pooling* (SAHA, 2018).

2.7.3.4 *Dropout*

As camadas de *dropout* funcionam de maneira diferente das camadas tradicionais, sendo uma técnica de regularização que tem como principal objetivo evitar o *overfitting*, problema no qual o modelo falha em generalizar novos dados em decorrência de um treinamento excessivo sobre os dados de treinamento. Durante o treinamento, neurônios das camadas ocultas são aleatoriamente desativados, ou seja, temporariamente "desligados", o que força a rede a aprender representações mais robustas e a não depender de neurônios específicos. Esse processo é repetido em várias iterações e, embora a rede não treine várias redes independentes, pode-se pensar que o *dropout* simula o comportamento de várias redes menores durante o treinamento, ajudando a rede a generalizar melhor e reduzir o *overfitting* (NIELSEN, 2015).

2.7.3.5 *Flatten*

A camada *flatten* também é distinta das camadas tradicionais e se assemelha mais a uma operação de reorganização de dados. Normalmente encontrada após camadas de *pooling*, a *flatten* é responsável por transformar dados multidimensionais em um vetor unidimensional. Isso é essencial, pois é comum que essa camada seja seguida por camadas densas, as quais exigem um vetor como entrada (CHOLLET, 2021).

2.7.3.6 *Reshape*

A camada *reshape* é semelhante à camada *flatten*, mas, enquanto a *flatten* transforma os dados em um vetor unidimensional, a *reshape* permite reorganizar os dados em qualquer formato desejado, mantendo o número total de elementos inalterado. Dessa forma, a *reshape* oferece mais flexibilidade (CHOLLET, 2021).

2.7.4 Funções de Ativação

Funções de ativação são funções matemáticas aplicadas a saída de cada neurônio em uma rede neural. Elas são responsáveis por decidir o valor de saída do neurônio, determinando se ele deve ser ativado ou não. Existem diferentes tipos de funções de ativação, sendo algumas das principais: *ReLU* e *softmax* (a função sigmoide, embora importante, não será tratada aqui) (AGGARWAL et al., 2018).

2.7.4.1 *Rectified Linear Unit (ReLU)*

A *ReLU* é uma função de ativação amplamente utilizada em CNNs. Seu principal objetivo é acelerar o treinamento da rede, introduzindo não linearidade enquanto mantém a simplicidade computacional. A função ReLU (2.8) transforma entradas negativas em zero, enquanto as entradas positivas são mantidas inalteradas. Isso ajuda a preservar as

características extraídas pelas camadas convolucionais e facilita o aprendizado, pois evita problemas como o desvanecimento do gradiente (BUNRIT et al., 2019).

$$f(x) = \begin{cases} 0 & \text{se } x < 0 \\ x & \text{se } x \geq 0 \end{cases} \tag{2.8}$$

2.7.4.2 *Softmax*

A função *softmax* é comumente utilizada na última camada de redes neurais voltadas para problemas de classificação multiclasse. Ela transforma, através da Fórmula 2.9, as saídas brutas em probabilidades, que somam um. Isso possibilita a interpretação dessas saídas como a probabilidade de um dado pertencer a cada classe, facilitando a comparação entre elas, uma vez que as probabilidades são valores entre 0 e 1, e a classe com o maior valor é a considerada mais provável (NIELSEN, 2015).

Na equação, z_j representa a saída de cada neurônio j na camada de saída, e^z, é o valor exponencial de z_j, e k refere-se à classe para a qual está sendo calculando a probabilidade (BUNRIT et al., 2019).

$$softmax(z_j) = \frac{e^{z_j}}{\sum_{k=1}^{k} e^{z_k}}, \quad j = 1, 2, ...k \tag{2.9}$$

2.8 Métricas para validação do modelo

Métricas de validação referem-se a métodos utilizados para avaliar o desempenho de um modelo durante o treinamento e validação. Elas contribuem na escolha do melhor modelo e permitem identificar problemas como o *overfitting*. Alguns exemplos comuns de métricas são a matriz de confusão e a acurácia (BRUCE; BRUCE; GEDECK, 2020).

2.8.1 Matriz de confusão

A matriz de confusão é uma tabela que permite visualizar o desempenho de um modelo de classificação, possibilitando observar a quantidade de previsões corretas e incorretas para cada classe.

Em um problema com duas classes, a matriz será composta por quatro elementos: verdadeiros positivos, verdadeiros negativos, falsos positivos e falsos negativos. Verdadeiros positivos referem-se ao número de acertos ao classificar instâncias como positivas, enquanto os verdadeiros negativos indicam o número de acertos ao classificar instâncias como negativas. Já os falsos positivos e falsos negativos indicam, respectivamente, os erros cometidos ao classificar incorretamente instâncias negativas como positivas e os erros ao classificar instâncias positivas como negativas.

Para problemas de classificação multiclasse, a matriz é expandida, sendo que os acertos ficam localizados na diagonal principal e os erros fora diagonal (BRUCE; BRUCE; GEDECK, 2020).

2.8.2 Acurácia

A acurácia (2.10) é uma métrica que indica a proporção de previsões corretas do modelo em relação ao número de amostras. É uma maneira mais simples de medir a performance da classificação.

No entanto, a acurácia pode ser enganosa, principalmente quando há um desbalanceamento nos dados. Em cenários onde uma classe é muito mais frequente que outra, o modelo pode ter uma alta acurácia apenas por sempre prever a classe majoritária, sem classificar corretamente a minoritária (BRUCE; BRUCE; GEDECK, 2020).

$$acurácia = \frac{\sum VerdadeiroPositivo + \sum VerdadeiroNegativo}{TamanhoAmostra} \tag{2.10}$$

2.8.3 F_1 *score*

F_1 *score* (2.11c) é uma média harmônica entre duas outra medidas: precisão (proporção das amostras classificadas como positivas que são realmente positivas) e *recall* (proporção de amostras positivas classificadas corretamente como positivas). Por ser uma média harmônica, os valores baixos tem mais peso e, portanto, o modelo só terá um F_1 *score* alto se ambos a precisão e o *recall* forem altos. Dessa forma, essa medida é útil quando é necessário um equilíbrio entre a precisão e o *recall* e, principalmente, quando o *dataset* é desbalanceado, já que evita que o modelo favoreça a classe majoritária (GÉRON, 2022).

$$precisão = \frac{\sum VerdadeiroPositivo}{\sum VerdadeiroPositivo + \sum FalsoPositivo} \tag{2.11a}$$

$$recall = \frac{\sum VerdadeiroPositivo}{\sum VerdadeiroPositivo + \sum FalsoNegativo} \tag{2.11b}$$

$$F_1 = 2 \cdot \frac{precisão \cdot recall}{precisão + recall} \tag{2.11c}$$

2.9 Edge Impulse

Segundo relatório da MarketsandMakets, companhia de inteligência e análise de mercado, o setor de inteligência artificial de borda (*Edge AI*) em 2023 já era uma indústria de 1,1 bilhão de dólares e possui projeções de alcançar o valor de 4,1 bilhões até 2028.

Esse avanço é impulsionado por diversos fatores, dentre eles, os principais são a adoção crescente de dispositivos *internet of things* (IoT) e necessidade, também em ascensão, de processamento de dados em tempo real (MARKETSANMARKETS, 2023).

Nesse sentido, a Edge Impulse (EDGE..., c2024), segundo a própria definição da organização, em tradução livre, é "[...] a plataforma líder para coleta de dados, treinamento e implantação de modelos para dispositivos computacionais de borda."(WHAT..., 2024). Esse sistema, em constante evolução, se propõe como um simplificador para os processos envolvidos no desenvolvimento e aplicação de modelos de *machine learning* em dispositivos embarcados de pequeno porte e consumo de energia(HYMEL et al., 2022).

Além disso, vale ressaltar que o ambiente de desenvolvimento virtual não se limita somente a simplificação dos processos, mas também inclui ferramentas avançadas para aprimorá-los. Um desses instrumentos é focado no *auto machine learning*, que permite variar hiperparâmetros de forma automatizada, facilitando a descoberta dos melhores parâmetros para um determinado modelo. Outro recurso trás a possibilidade de realizar o mesmo para parâmetros de pré-processamento(WHAT..., 2024).

2.10 Arduino

A Arduino surgiu em em 2005 na cidade de Ivrea, localizada na Itália, com a proposta de ser uma plataforma acessível, tanto no sentido financeiro como também no técnico, para se trabalhar com microcontroladores (EVANS; NOBLE; HOCHENBAUM, 2013). Desde então, é estimado que até 2013 mais de um milhão de placas Arduino, oficiais e não oficiais, já haviam sido comercializadas, indicando a atratividade desse modelo de baixo custo, baixa curva de aprendizagem e alta agilidade (MCROBERTS, 2015).

Essa plataforma é dividida em duas bases principais, *hardware* e *software*. Em quesito de *hardware*, as placas de desenvolvimento Arduino são baseadas em microcontroladores de 8 a 32 bits, microcontroladores estes que podem ser programados de maneira simples com as linguagens C/C++. Além disso, o *hardware* é composto por entrada de alimentação externa, entrada tipo *Universal Serial Bus* (USB) para comunicação, programador interno, pinos analógicos, pinos digitais, pinos de energização e pinos de aterramento. Já em *software* a Arduino disponibiliza uma interface de desenvolvimento, chamada Arduino *Integrated Development Environment* (IDE), para produção dos códigos e *deploy* do código em seus dispositivos (ISMAILOV; JO'RAYEV et al., 2022).

Além do mais, é válido destacar uma placa em específico, a Arduino NANO 33 BLE Sense. Essa atenção deve-se às expressivas vantagens que apresenta quando se trata de um projeto envolvendo inteligência artificial e aprendizado de máquina. Alguns dos benefícios apresentados por esse modelo são o custo relativamente baixo, armazenamento embarcado, bibliotecas de suporte abundantes, microfone e acelerômetro inclusos, além

do suporte a *TensorFlow Lite* (WAQAR et al., 2021). A placa possui 256kB de memória RAM, sua cpu possui 1MB de memória flash, o microcontrolador é um nRF52840 e o microfone é um MP34DT06JTR (ARDUINO..., c2024).

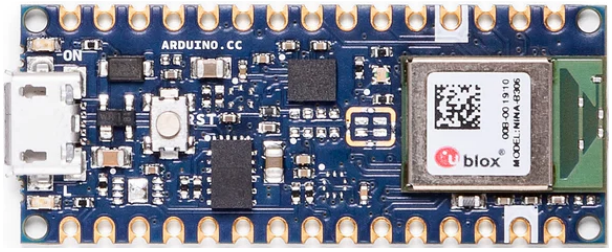


Figura 4 – Arduino Nano 33 BLE Sense

Fonte: Arduino... (c2024)

2.11 Estudos Anteriores

O estudo sobre pluviômetros baseados em *machine learning* é recente, porém já possui frentes de estudo. Algumas delas buscam classificar chuvas, outras se propõe até a estimar o volume de chuva despejado. Certos projetos, mais ambiciosos, almejam fornecer estações meteorológicas completas e de baixo custo.

O estudo de Alkhatib et al. (2023) conseguiu se apresentar como uma prova de conceito para estimativa de chuvas baseada em som. Contou com um *dataset* de autoria própria com mais de 2 anos de dados relacionados a chuva, incluindo o som produzido. Foram testadas duas técnica de processamento de sinais, Espectrograma e MFE, para extração de *features*, sendo MFE a mais eficiente. A utilização de estratégias de redução de ruído se apresentou como ponto essencial da pesquisa, uma vez que o ambiente urbano apresentou-se como dificultador do processo devido ao sons característicos que o compõe. O trabalho utilizou uma CNN, alimentada com trechos de 5 segundos de áudio, e focou sua atenção em chuvas fortes (acima de 60mm/h). Os resultados obtidos demonstram sua viabilidade, uma vez alcançados 98% de acurácia em seus teste de validação.

Outra abordagem apresentada por Wang et al. (2024) optou por estimar a intensidade de chuva ocorrida, baseando-se no som produzido pelo impacto da chuva no material do solo próximo aos microfones. O *dataset* utilizado foi de áudios provenientes de microfones de câmeras de segurança instaladas estrategicamente para coletar dados de diversos tipos de solo. Foram utilizados 3 tipos de pré-processamento: espectrograma (proveniente da aplicação da técnica Short-time Fourier transform (STFT)), MFE e MFCC. Também, 3 tipos de modelo foram avaliados, sendo eles: CNN, LSTM e modelo baseado em *transformers*. Para questão de comparação, 4 algoritmos clássicos também foram avaliados: árvore de decisão, floresta aleatória, *linear support vector machine* e *ridge regression model*. Os modelos foram alimentados com áudios de tamanho de janela

igual a 4 segundos. O melhor modelo obtido contou com *features* extraídas por MFCC e modelo baseado em *transformers*, apresentando o menor erro entre seus concorrentes ($RootMeanSquareError(RMSE) = 0,88mm * h^{-1}$).

Já Avanzato et al. (2019) se dispõe a realizar a classificação das chuvas, tendo como fonte de áudio o som produzido por pratos de diferentes materiais (aço, alumínio, cerâmica e plástico), localizados no solo próximo ao microfone. A classificação foi realizada para 4 categorias: fraca, moderada, forte e muito forte. Nesse estudo, apenas a combinação de MFCC com CNN foi utilizada, tendo como entrada áudios de 1 segundo dos diferentes *datasets* produzidos (1 para cada material). A acurácia obtida foi de 65% para o alumínio, 92% para o aço e 93% para plástico e cerâmica. Tais resultados indicam a validade do método proposto.

Também, Avanzato e Beritelli (2020) realizaram a classificação das chuvas, utilizando áudios provenientes de um *dataset* autoral coletado durante 5 dias em diferentes terrenos. A categorização contou com 7 categorias: sem chuva, fraca, moderada, forte, pancada e torrencial. Nesse trabalho, a rotulagem dos sons foi realizada de maneira automática utilizando um algoritmo desenvolvido para essa função. O modelo gerado foi baseado em CNN, alimentado por áudios de 3 segundos sem qualquer técnica de extração de *features* (*raw data*). Os testes resultaram em 75% de acurácia, subindo para 93% se forem desconsiderados as classificações erradas que ocorreram entre classes adjacentes.

Ademais, Batista e Fernandes (2024) classificaram a vazão de água de um chuveiro, buscando com isso propor a expansão do conceito para a classificação do volume de chuvas. As categorias analisadas foram: fraco, médio, forte e "ruído". O *dataset* foi gerado de maneira autoral, realizando-se a captação de 10 áudios de 10 segundos para cada classe. O projeto gerou 2 modelos de ML baseados em CNN. O primeiro contou com pré-processamento por espectrograma e áudios de 4 segundos. Já para o segundo modelo, as *features* foram extraídas por MFE e os áudios de entrada tinham 1 segundo de duração. Os testes resultaram em 100% de acurácia (modelo com MFE) para todas as classes. Isso mostra a validade do método e reforça a possibilidade de expansão do método para classificação de chuvas.

Além disso, é válido citar o trabalho desenvolvido por Zennaro et al. (2024). Apesar de não tratar de um modelo de pluviômetro baseado em *machine learning*, ele apresenta um cenário de aplicação relevante para os instrumentos de medição de precipitação de chuva. O projeto conseguiu ampliar a infraestrutura de meteorologia em um país predominante de ambientes montanhosos e frios, além de possuir limitada cobertura de sinal telefônico. Realizando o emprego de estações de meteorologia de baixo custo e da tecnologia *LoRaWAN*, a coleta de dados críticos para análise meteorológicas foi viabilizado em um ambiente desafiador.

3 Metodologia

Este capítulo tem como objetivo apresentar as metodologias e procedimentos utilizados na análise do *dataset*, Aurora, abrangendo tanto a visualização dos dados quanto a elaboração de um subconjunto reduzido, que será utilizado no treinamento de um modelo de *machine learning*.

O trabalho inicia-se com a exploração dos dados originais, na qual são utilizadas ferramentas de visualização para identificar as principais características do *dataset*. Logo após, é realizada uma comparação com outro *dataset* para verificar a fidelidade dos dados. Essa é uma fase importante, uma vez que permite conhecer os dados e selecionar as informações relevantes.

Em seguida, torna-se possível elaborar um subconjunto de dados mais eficiente, que contribui para o treinamento do modelo de *machine learning*, potencializando a capacidade do mesmo em prever resultados de forma mais precisa.

3.1 Aurora *Dataset*

Para o desenvolvimento de um modelo de inteligência artificial, capaz de estimar a pluviosidade, é necessário um *dataset* que possui dados de chuva ao longo do tempo. Para isso, foi estudada a possibilidade de criar um *dataset* local, mas pela conveniência de dados já captados, estruturados e disponibilizados publicamente, o *dataset* Aurora (CSEM, 2024) foi escolhido.

Esse *dataset* foi criado para servir como base na competição *Next-Gen tinyML Smart Weather Station*, que visa promover o desenvolvimento de estações meteorológicas inteligentes que sejam economicamente viáveis e energeticamente eficientes, por meio da tecnologia *TinyML*. As equipes tinham o objetivo de criar uma estação de baixo custo, consumo, precisa, fácil de instalar e sem partes mecânicas móveis para capturar dados de diversas condições climáticas, com foco em chuva e vento, baseado em *machine learning* para dispositivos de borda de ultrabaixo consumo. Para isso foi disponibilizado um *dataset*, criado pelo o vencedor do desafio do ano anterior, contendo gravações de microfone e dados de sensores ambientais através do dispositivo da Figura 5.

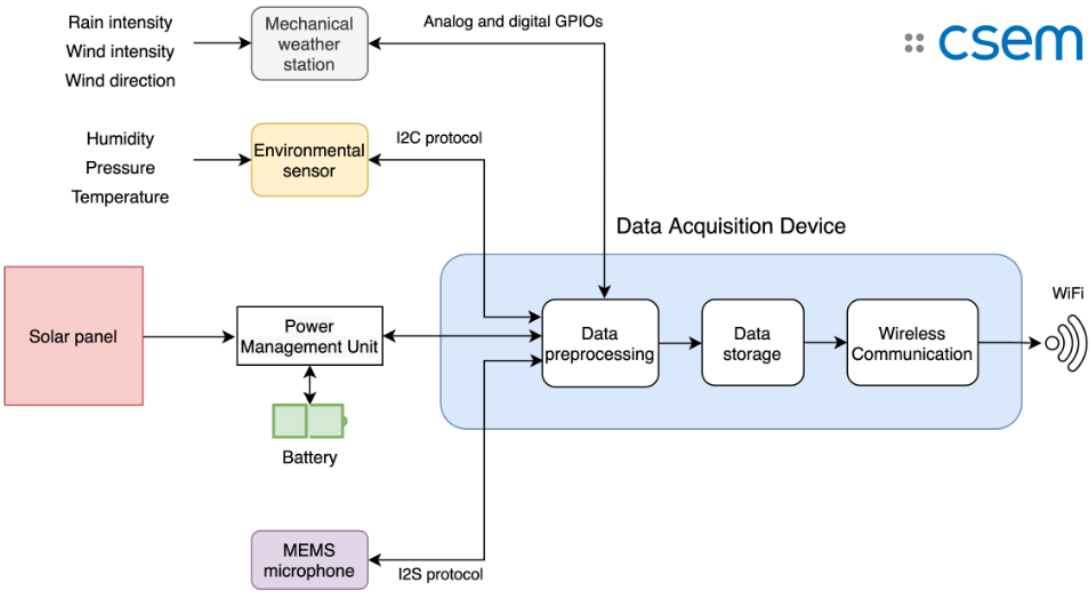


Figura 5 – Diagrama do dispositivo de captura do *dataset* Aurora

Fonte: CSEM (2024)

O *dataset* conta com informações de tempo (*timestamp*), temperatura, pressão, contagem de pulsos de vento, direção do vento, altitude e umidade, além de conter a quantidade de chuva, representada por interrupções elétricas geradas por um pluviômetro do tipo *tipping bucket*, e o som produzido por ela. Os dados estão organizados em arquivos *.dat* contendo aproximadamente 1 hora de informação (Figura 6).

raw_data_CSEMAurora1_2023-11-09_0	09/11/2023 00:59	DAT	157.921 KB
raw_data_CSEMAurora1_2023-11-09_1	09/11/2023 01:59	DAT	157.008 KB
raw_data_CSEMAurora1_2023-11-09_2	09/11/2023 02:59	DAT	155.855 KB
raw_data_CSEMAurora1_2023-11-09_3	09/11/2023 03:59	DAT	156.624 KB
raw_data_CSEMAurora1_2023-11-09_4	09/11/2023 04:59	DAT	157.873 KB
raw_data_CSEMAurora1_2023-11-09_5	09/11/2023 05:59	DAT	158.594 KB
raw_data_CSEMAurora1_2023-11-09_6	09/11/2023 06:59	DAT	160.275 KB
raw_data_CSEMAurora1_2023-11-09_7	09/11/2023 07:59	DAT	159.891 KB
raw_data_CSEMAurora1_2023-11-09_8	09/11/2023 08:59	DAT	158.738 KB
raw_data_CSEMAurora1_2023-11-09_9	09/11/2023 09:59	DAT	155.279 KB
raw_data_CSEMAurora1_2023-11-09_10	09/11/2023 10:59	DAT	156.095 KB
raw_data_CSEMAurora1_2023-11-09_11	09/11/2023 11:59	DAT	160.852 KB
raw_data_CSEMAurora1_2023-11-09_12	09/11/2023 12:59	DAT	160.083 KB

Figura 6 – Arquivos *.dat* presentes no *dataset* Aurora

Fonte: CSEM (2024)

Esses arquivos *.dat* são agrupados semanalmente e essas pastas cedidas no site do desafio, englobando um período de aproximadamente 29 semanas, compreendido entre a trigésima semana de 2023 e a sétima semana de 2024, na cidade de Neuchatel-Suiça. Dentro

de cada arquivo *.dat* os dados estão estruturados em pacotes de 49196 bytes, seguindo o arranjo da Figura 7.

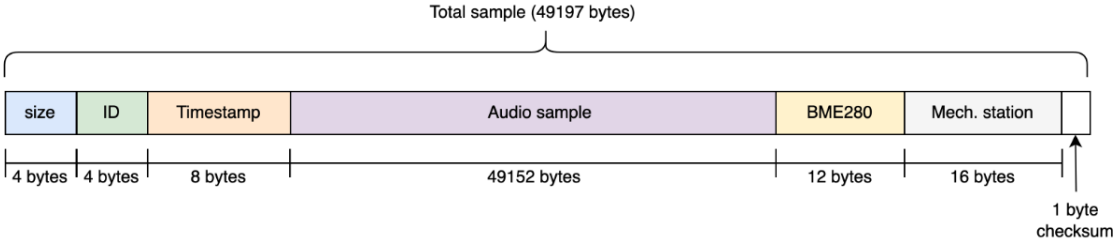


Figura 7 – Estrutura dos dados presentes no *dataset* Aurora

Fonte: CSEM (2024)

3.2 Visualização de dados

A partir do *dataset*, os dados foram transformados, inicialmente sem os áudios, através de um *script* de *python*, em um formato que permitisse a visualização. Foi criado um arquivo CSV para observar os valores em uma forma tabular.

Levando em consideração o objetivo do trabalho, era necessário criar uma nova coluna na tabela para a milimetragem. Através da contagem de interrupções de chuva, já presente no conjunto de dados, e levando em consideração que cada pulso corresponde a 0,2794 mm de chuva, foi possível calcular essa medida.

A fim de permitir uma maior compreensão dos dados, foi gerado um gráfico de precipitação por tempo (8), o que permitiu uma análise visual da quantidade de chuva ocorrida no período de interesse.

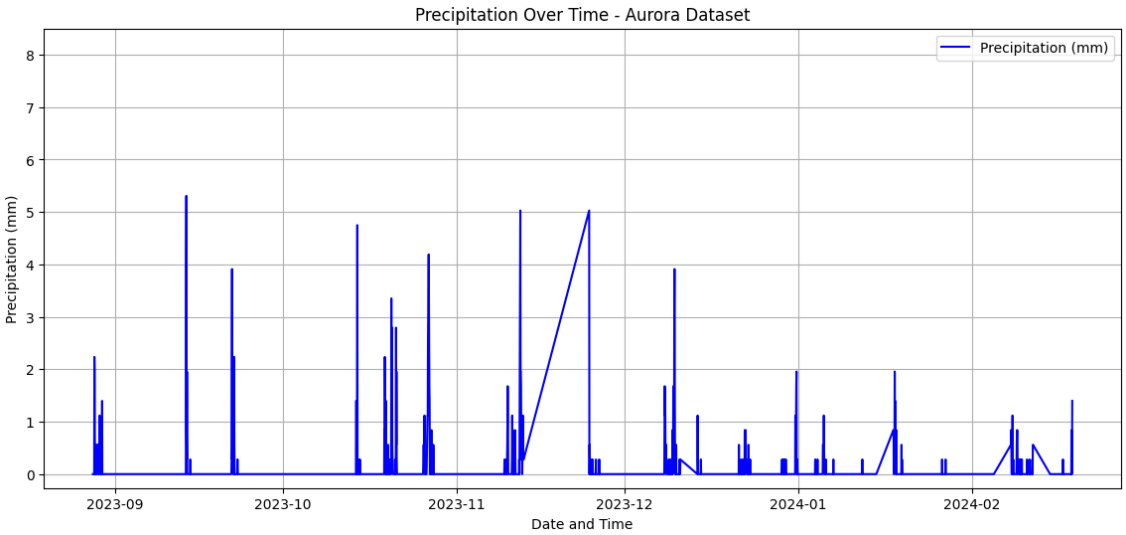


Figura 8 – Gráfico da distribuição de chuva acumulada segundo o *dataset* Aurora

Fonte: Autoria própria

Além disso, para garantir maior confiabilidade dos dados, foi produzida uma segunda visualização gráfica (9) para outro *dataset*, referente a estação meteorológica da cidade (WORLD..., c2024), excluindo alguns períodos de dados para aproximar o período disponível nesse conjunto de dados ao presente no Aurora.

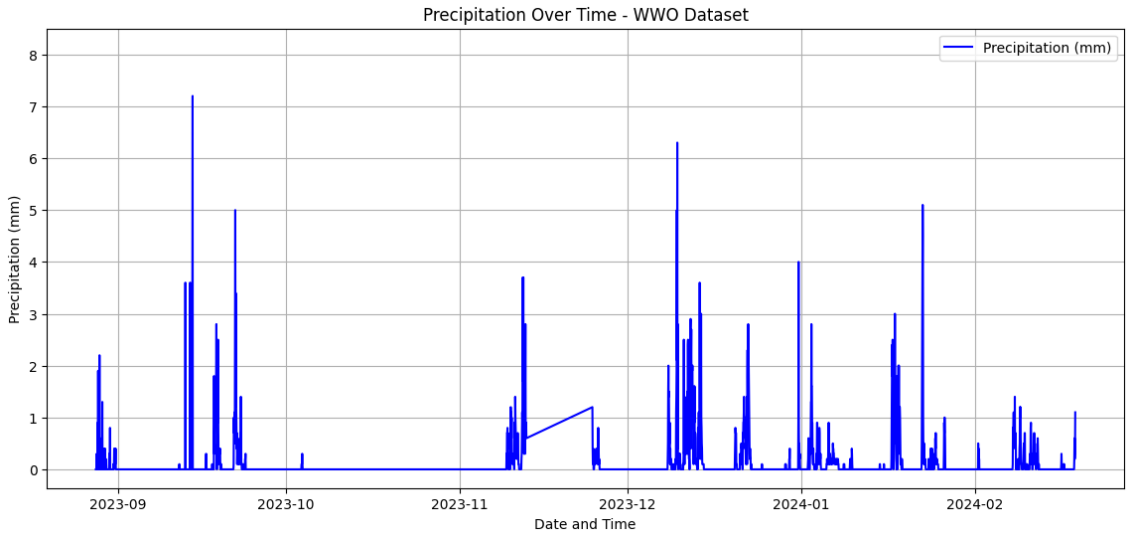


Figura 9 – Gráfico da distribuição de chuva acumulada segundo as informações do World Weather Online fornecidas pela estação meteorológica da cidade de Neuchatel-Suíça

Fonte: Autoria própria

Após uma análise comparativa entre os dois gráficos, é possível notar diferenças significativas entre os dados. Entretanto, considerando que o *dataset* Aurora possui diversos dias e períodos faltantes, é possível considerar que os dados são válidos e possíveis de serem utilizados no prosseguimento deste estudo.

3.3 Construção da base de dados

Com os dados já visualizados, iniciou-se a fase de obtenção de um subconjunto de dados, a partir do *dataset* original. Para isso, inicialmente foi necessário delimitar classes para o volume de chuva, baseando-se em artigos e trabalhos estudados previamente. Em seguida, foram realizados testes para definir o tempo de áudio, informação fundamental levando em consideração o tamanho do modelo e as limitações de processamento do Arduino. Isso, uma vez estabelecido, permitiu a rotulação dos áudios com as classes. O resultado foi um *dataset* sintetizado e pronto para ser utilizado no modelo.

3.3.1 Limites de precipitação

A fim de definir classes para o modelo, foram utilizadas algumas referências como ponto de partida. A primeira, trata-se do vencedor do desafio *Next-Gen tinyML Smart*

Weather Station, que apresentou duas soluções para o problema, uma com duas e outra com três classes de chuva e vento. Apesar de possuir uma acurácia com dados reais menor, a segunda solução se apresentou mais interessante, por incluir uma classe (chuva fraca) entre as classes chuva e não chuva (Tabela 1).

Classificação de chuva	**Precipitação (mm/h)**
Sem chuva	$< 0{,}5$
Chuva fraca	0,5 - 2
Chuva	> 2

Tabela 1 – Classes e limites de precipitação utilizados pelo vencedor do *Next-Gen tinyML Smart Weather Station*

Fonte: CSEM (2024)

No estudo de Avanzato e Beritelli (2020), cujo objetivo também foi a classificação da chuva, os autores utilizaram como referência a escala de classificação nacional da Itália, com algumas poucas modificações para obter um número homogêneo de exemplos para as diferentes categorias, e decidiram entre as classes da Tabela 2.

Classificação de chuva	**Precipitação (mm/h)**
Sem chuva	$< 0{,}5$
Chuva fraca	0,5 - 2
Chuva moderada	2 - 6
Chuva forte	6 - 10
Chuva muito forte	10 - 18
Pancada de chuva	18 - 30
Chuva torrencial	> 30

Tabela 2 – Classes e limites de precipitação utilizados no estudo italiano

Fonte: Avanzato e Beritelli (2020)

Visto isso, os limites utilizados neste trabalho (Tabela 3) foram definidos levando em consideração os estabelecidos nos projetos de CSEM (2024) e Avanzato e Beritelli (2020), alterando levemente algumas faixa de valores a fim de garantir uma quantidade suficiente de dados para cada classe, em especial "chuva forte", que apresentou poucas ocorrências ao utilizar um limite superior maior que 4mm/h para "chuva moderada".

3.3.2 *Labels*

Nesta seção, os áudios foram introduzidos na análise. Os dados meteorológicos, agora acompanhados de áudios correspondente a eles, passaram a ser trabalhados de

Classificação de chuva	**Precipitação (mm/h)**
Sem chuva	$< 0{,}5$
Chuva fraca	0,5 - 2,5
Chuva moderada	2,5 - 4
Chuva forte	> 4

Tabela 3 – Classes e limites de precipitação utilizados neste estudo

Fonte: Autoria própria

maneira conjunta. A partir dessa união e do foco na informação de intensidade de chuva, foi possível avançar para a fase de rotulagem.

A rotulagem desses áudios é parte essencial do processo de construção de um *dataset* para reconhecimento e classificação de sons. Essa etapa se demonstrou complexa, fazendo-se necessária a realização de diversos testes para definição de uma duração padrão para os áudios a serem rotulados. Inicialmente, os limites de intensidade de chuva estavam definidos para áudios com duração de 1 hora, mas foi necessário ajustá-los para outras durações. Para garantir a consistência, os limites foram adequados proporcionalmente de acordo com a nova duração. Por fim, foi utilizado um algoritmo condicional para realizar a rotulagem dos áudios automaticamente.

Em um primeiro momento, tentou-se evitar a rotulagem de um áudio completo de uma hora. Escolheu-se, portanto, durações pequenas: um e cinco minutos. Esses valores, inicialmente, foram julgados adequados para a tarefa de rotulagem. Entretanto, essa abordagem não foi válida, visto que o volume que cada pulso do pluviômetro representa é de 0,2794mm de chuva. A conversão para um minuto dos limites de cada intervalo de classe inviabiliza a rotulagem, uma vez que a ocorrência de um único pulso extrapola todos os limites, fazendo com que não exista a possibilidade de nenhum dado ser rotulado nas classes intermediárias. Algo semelhante ocorre com as transformações envolvidas para cinco minutos, onde um pulso extrapola os limites da classe de "chuva fraca".

Ao descartar a possibilidade de subdivisão da hora para rotulagem, foi observado que a utilização de uma hora completa (3600 segundos) também apresentava problemas, dado que os arquivos .dat (*raw dataset*) não continham exatamente 3600 amostras de um segundo. Logo, tendo em vista os problemas listados, optou-se pela alternativa de utilizar um áudio para cada arquivo .dat, gerando assim áudios de diferentes durações, mas próximos de uma hora. Para os limites, portanto, manteve-se os dispostos na tabela 3.

3.3.3 Finalização do *Dataset*

Definida a duração de áudio e as classes, foi iniciado o processo de análise e modelagem do *dataset*. Pela inviabilidade de utilizar todos os dados, seja pela memória

de disco necessária para descomprimir tudo, seja pelo aumento do tempo de treinamento, a utilização de partes do *dataset* com mais informações de chuva foi priorizada. Dessa forma, foram escolhidas as semanas com mais interrupções causadas pela chuva presentes no *dataset*, que são as semanas 43 e 45, que trazem em seu conteúdo dados de todas as classes (sem chuva, chuva fraca, chuva moderada e chuva forte).

Name	Year	Week	# Rain interrupts	# Wind interrupts	File size	Download link
CSEMAurora1	2023	39	0	468991	7.47 GB	Download here
CSEMAurora1	2023	40	0	384826	7.42 GB	Download here
CSEMAurora1	2023	41	37	703399	7.60 GB	Download here
CSEMAurora1	2023	42	91	606679	7.58 GB	Download here
CSEMAurora3	2023	41	5	904598	6.30 GB	Download here
CSEMAurora3	2023	42	47	736454	6.28 GB	Download here
CSEMAurora1	2023	43	117	840573	7.83 GB	Download here
CSEMAurora1	2023	45	128	911857	7.71 GB	Download here
CSEMAurora1	2023	47	26	378611	4.34 GB	Download here
CSEMAurora1	2023	49	94	629099	7.53 GB	Download here

Figura 10 – Parte da tabela com as semanas disponíveis no Aurora *Dataset*

Fonte: CSEM (2024)

Inicia-se nesse momento a preocupação com o modelo. Áudios de longa duração geram um grande volume de parâmetros, dificultando o processamento e exigindo muita memória e poder computacional. Para isso, os áudios de aproximadamente uma hora, agora rotulados e com uma distribuição balanceada entre cada classe, precisaram ser repartidos e selecionados para deixar o modelo mais leve, mas mantendo as características mais importantes. Para isso, optou-se por utilizar como nova duração 30 segundos, semelhante ao utilizado no estudo de Avanzato e Beritelli (2020).

Tendo em vista o objetivo, primeiramente foi testada uma abordagem manual. Os áudios de uma hora eram escutados e, em momentos de concentração de chuva, novos áudios de 30 segundos do *label* original eram gerados. Entretanto, essa estratégia se mostrou lenta, ineficiente e sujeita a erros humanos e, portanto, foi abortada.

Em outro teste, a hora inteira foi dividida em áudios de 30 segundos, sem qualquer tipo de seleção, todos mantendo o mesmo *label*. Porém, a chuva não é um fenômeno que mantém a mesma intensidade ao longo do tempo e, dessa maneira, os áudios gerados, apesar de na mesma classe, eram muito variados, de forma que alguns eram em momentos sem chuva e outros de chuva forte.

Visto os problemas apresentados, foi definida uma estratégia para definir os áudios que viriam a compor o grupo final de dados para ser utilizado no treinamento, validação e teste do modelo. Essa estratégia consiste em focar no momento em que os sinais de interrupção são gerados pelo pluviômetro presente na captação dos dados do *dataset* Aurora. Isso ocorre com base em uma variável, presente nos arquivos de dados e que indica

a ocorrência de pulsos de chuva. Com isso, é possível anotar as amostras onde esse evento ocorreu.

Em posse da informação temporal de quando os pulsos ocorreram, 15 segundos antes e depois desse instante foram agrupados e classificados de acordo com o *label* da hora em que a interrupção aconteceu. Por fim, para compor os dados da classe sem chuva, que não necessariamente conta com interrupções do pluviômetro, a estratégia foi adaptada e grupos de 30 segundos de áudio foram coletados aleatoriamente das horas classificadas nessa categoria. Na Figura 11, pode-se observar o formato final de um áudio de cada classe após realização dessas operações.

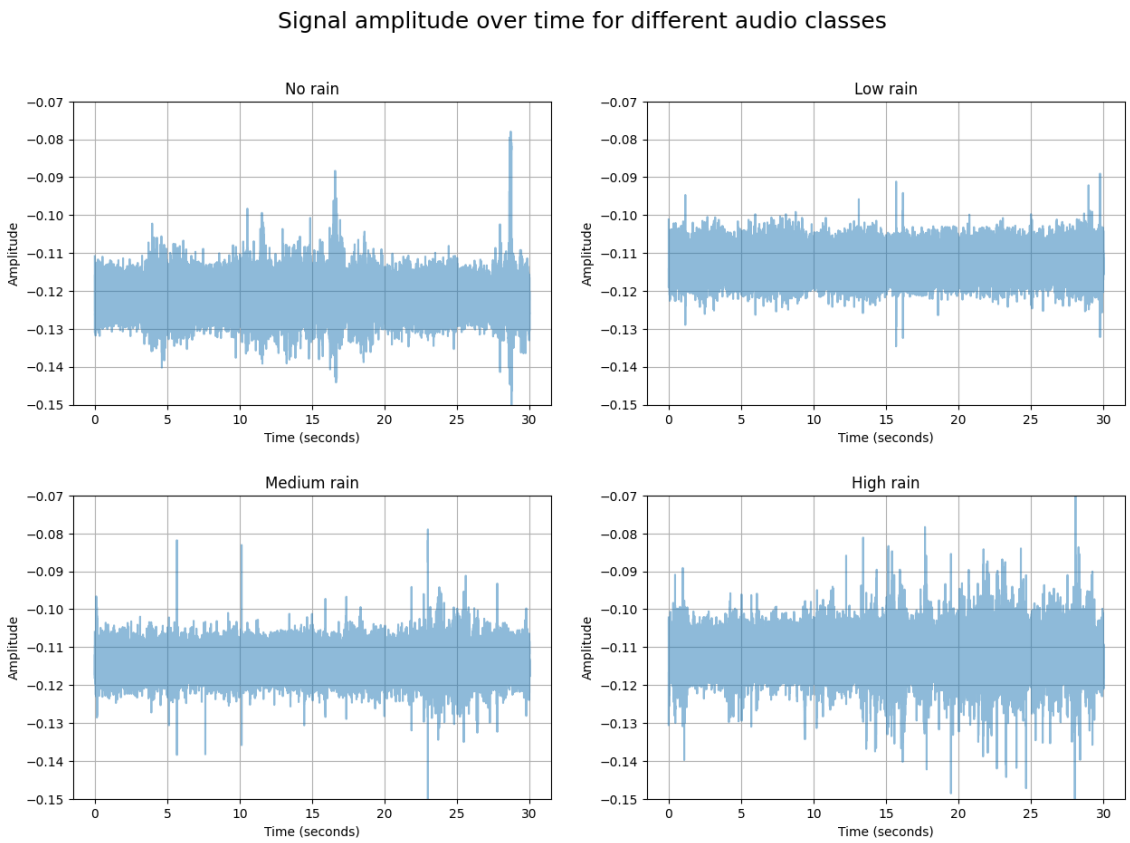


Figura 11 – Visualização da amplitude de sinais exemplos de cada classe do modelo

Fonte: Autoria própria

3.4 *Feature engineering*

Tendo o *dataset* final, tem início a fase de *feature engineering*, processo de transformação de dados brutos em informações que podem ser utilizadas no modelo de *machine learning*. A qualidade dos *features* gerados após a seleção e transformação dos dados, está diretamente relacionada ao desempenho do modelo, uma vez que é por meio dos valores de entrada que o mesmo aprende a realizar previsões e classificações. Para a realização

dessa tarefa, foi utilizada a plataforma *Edge Impulse*, que facilita o processo, por meio de uma interface gráfica e recomendações de parâmetros.

No *Edge Impulse*, o *dataset* é inserido, com uma parte destinada ao treinamento e outra ao teste. Por se tratar de arquivos de áudio (time series data), é necessário escolher, inicialmente, dois parâmetros: tamanho de janela (comprimento de segmento de áudio que será analisado) e incremento da janela (o quanto a janela se move). O *Edge Impulse* oferece sugestões baseadas em suas experiências, sendo esse o ponto de partida para os parâmetros escolhidos, alterando conforme necessário. Após testes, os melhores resultados foram obtidos com os seguintes valores: tamanho de janela de 6 segundos e incremento de 0,6 segundos.

Em seguida, foi necessária a escolha da técnica que seria utilizada para a extração de *features*. Foram realizados experimentos, através da ferramenta de testes automatizados para hiperparâmetros *EON Tuner*, disponibilizada pela Edge Impulse, com espectrogramas, MFCC e MFE. Os dois primeiros apresentaram resultados inferiores (10% e 54%) (Imagem 12) ao terceiro (76%) e, portanto, optou-se por utilizar o MFE. Para isso, novos parâmetros precisaram ser decididos: tamanho do quadro (número de amostras em cada segmento de áudio), passo do quadro (o quanto o quadro se move), número de filtros, tamanho da FFT, frequência baixa (limite inferior do intervalo de frequência), frequência alta (limite superior do intervalo de frequência) e piso de ruído (nível de ruído de fundo). Novamente, ajustes finos foram sendo realizados sobre os valores de acordo com a necessidade, a partir de valores recomendados pelo *Edge Impulse*. Ao final, os melhores resultados foram obtidos com os seguintes hiperparâmetros: tamanho do quadro de 0,08s, passo do quadro de 0,025s, número de filtros de 41, tamanho de FFT de 256, frequência baixa de 80Hz, frequência alta de 8kHz e piso de ruído de -64dB.

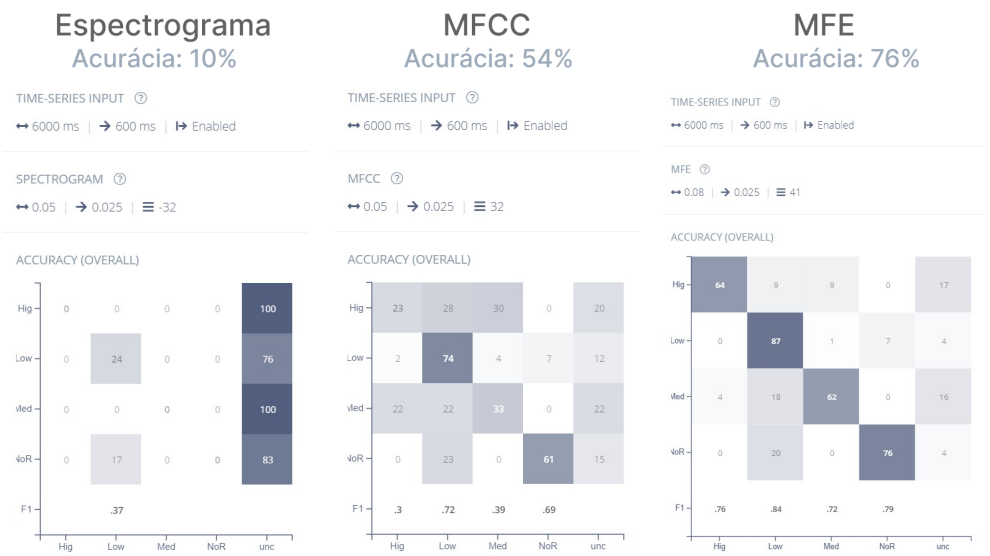


Figura 12 – Comparação entre os três modelos de DSP
Fonte: Autoria própria

A partir desses parâmetros, o áudio tem suas principais características extraídas após algumas etapas. Inicialmente o áudio é pré-amplificado nas frequências mais altas, dividido em *frames* e sobre cada um deles é aplicada uma janela de *Hamming*, como pode ser observado na Figura 13, que representa a aplicação dessa janela em 3 *frames* de um áudio do *dataset*. Sobre o resultado dessa sequência de operações é realizada também um transformada de Fourier e calculado o espectro de potência dessas amostras. A Figura 14 apresenta quatro espectros de potência, um para cada classe. Nessa imagem é possível observar a distribuição de potência, para cada frequência do sinal, e sua variação no tempo. Posteriormente, é construído o banco de filtros característico da técnica MFE. Esses filtros são aplicados sobre o sinal de cada *frame* produzindo as informações relevantes do sinal, que podem ser observadas na Figura 15. Essa imagem apresenta quatro visualizações da aplicação de Mel-Frequency Energy (MFE) em sinais de áudio do *dataset*, com o eixo x indicando o tempo (em segundos) e o eixo y as bandas de frequência Mel. As cores, por sua vez, mostram a energia em cada faixa de frequência e instante de tempo: tons vermelhos indicam maior intensidade, enquanto tons azuis apontam menor energia. Esse tipo de visualização ajuda a identificar características sonoras ao longo do tempo, como padrões específicos que diferenciam sons, neste caso, diferentes intensidades de chuva.

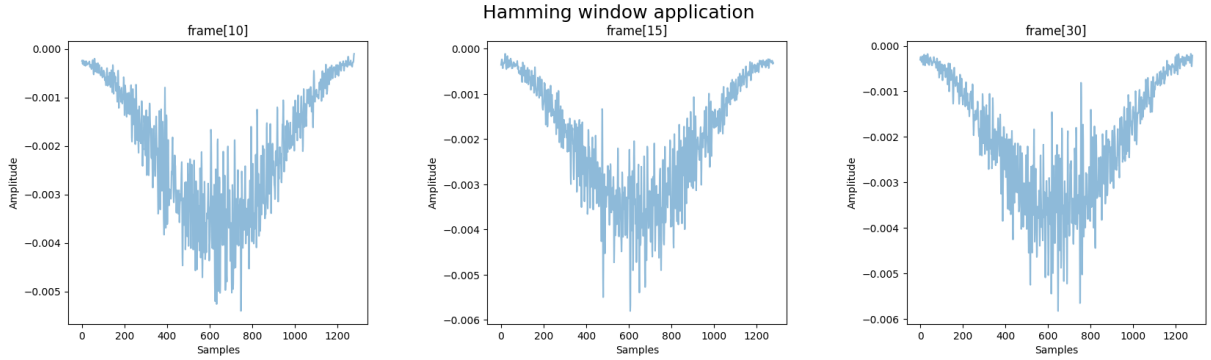


Figura 13 – Janela de Hamming

Fonte: Autoria própria

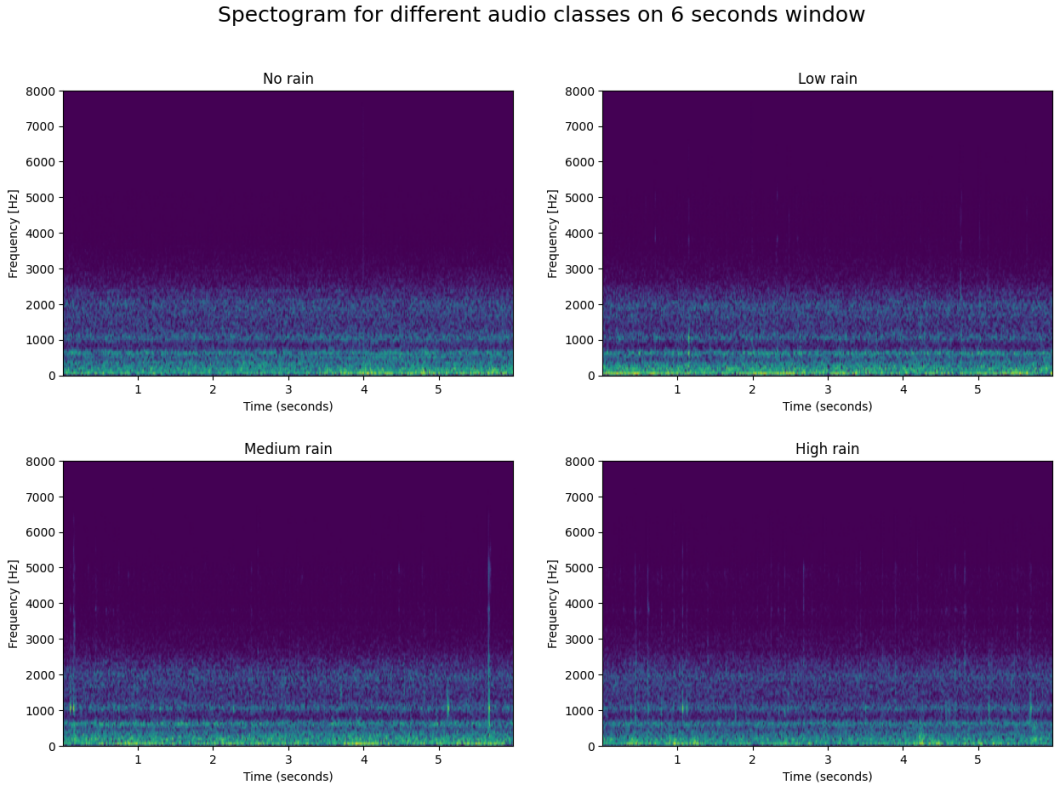


Figura 14 – Espectrograma para janela de 6 segundos

Fonte: Autoria própria

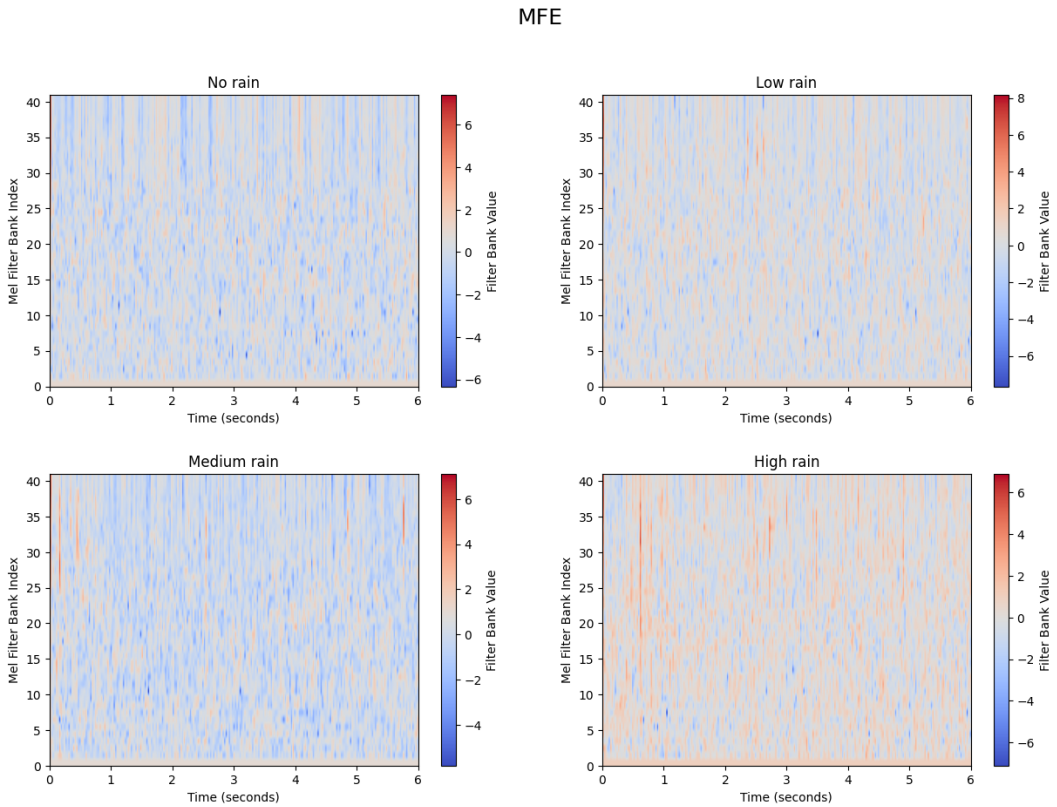


Figura 15 – MFE das diferentes classes presentes no modelo

Fonte: Autoria própria

3.5 Arquitetura

Arquitetura de *machine learning* refere-se à estrutura completa de uma rede neural, que consiste em camadas de neurônios e tipicamente possuem entradas, camadas ocultas e saídas (NIELSEN, 2015). Dada essa informação, foi necessário explorar diferentes arquiteturas de ML para identificar a mais adequada para a tarefa de classificação de áudios visando a estimativa do volume de chuva a partir do som.

3.5.1 Estudo do modelo

Ao início do trabalho, foram considerados dois tipos de arquiteturas principais para a classificação: LSTM e CNN. A primeira, eficiente para o trabalho com dados de séries temporais que apresentam dependência temporal, mostrou-se uma opção válida, levando em consideração o objetivo do estudo de encontrar uma maneira de estimar a milimetragem da chuva apenas utilizando o som. Isso ocorre pois, conforme discutido anteriormente, intensidade da chuva pode variar significativamente durante o intervalo de uma hora e, portanto, a LSTM seria capaz de capturar essas dependências temporais, identificando padrões nessas variações. Entretanto, essa estratégia foi descartada, uma vez que, no estado da arte e levando em conta estudos realizados anteriormente, como por exemplo em Batista e Fernandes (2024), modelos baseados em CNN são mais utilizados para tarefas de classificação de chuva, além de apresentarem bons resultados em termos de acurácia e serem mais facilmente implementados através do *Edge impulse*.

3.5.2 Classificação

O modelo utilizado para classificação conta com uma série de parâmetros como: número de camadas convolucionais, utilização de camadas de *dropout*, definição da quantidade de ciclos de treinamento, taxa de aprendizagem, entre outras especificações. Adotando uma abordagem empírica e utilizando novamente a ferramenta de testes automatizados, disponibilizada pela *Edge Impulse*, obteve-se os parâmetros que proporcionaram os melhores resultados. O modelo mais otimizado encontrado (Figura 16) conta com nove camadas: uma camada de *reshape*, duas camadas convolucionais, duas camadas de *MaxPooling* e duas camadas de *dropout*. Além disso, foi definido que o modelo contaria com uma taxa de aprendizagem de 0.001, 100 ciclos de treinamento, oito neurônios na primeira camada convolucional e 16 na segunda, a taxa de *dropout* seria de 0.25, utilizaria o otimizador *Adam* e não contaria com nenhum tipo de *data augmentation*. Definidos esses parâmetros, o modelo foi testado com 20% dos dados, separados para essa finalidade.

Em seguida, dá-se início à fase de implementação no dispositivo de borda escolhido, o Arduino NANO 33 BLE Sense. Para isso, a plataforma *Edge Impulse* oferece diferentes formas de *deploy*, além de otimizações para o modelo, reduzindo o seu tamanho e garantindo,

assim, que funcione em ambientes com limitações computacionais. Para o teste na placa, escolheu-se o *deploy* por meio de biblioteca Arduino e o otimizador *EON Compiler*. O modelo, agora compactado, pode ser carregado no dispositivo de borda através da ferramenta Arduino IDE. Para a realização da inferência, optou-se por utilizar a opção de um *static buffer*, na qual os dados de entrada são inseridos em sua forma bruta e o resultado da classificação pode ser visto através do monitor serial e do acendimento de *Light Emitting Diodes* (LEDs) da placa Arduino, que obedecem a seguinte regra.

- Branco → Sem chuva
- Verde → Chuva fraca
- Amarelo → Chuva moderada
- Vermelho → Chuva forte

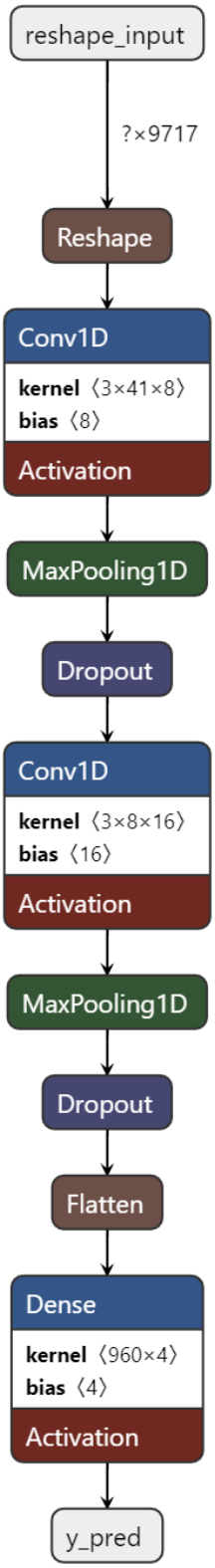


Figura 16 – Arquitetura do modelo

Fonte: Autoria própria

4 Experimentos e Resultados

Definida a metodologia de trabalho para desenvolver o modelo de detecção sonora de chuva, deu-se início aos testes. Nessa etapa, buscou-se validar a eficácia do modelo no *Edge Impulse* em classificar diferentes intensidades de chuva com áudios não utilizados no treinamento. Após isso foi realizada a inferência no dispositivo de borda com dados do próprio treinamento do modelo, a fim de validar seu funcionamento.

4.1 Melhores modelos

Os experimentos realizados promoveram diversas combinações para hiperparâmetros de pré-processamento e classificação. Esses experimentos, que incluíram treinamento, validação e testes de inferência, foram o principal argumento utilizado para definição dos parâmetros finais do modelo com maior acurácia, descrito na seção de metodologia deste trabalho. Esses ensaios podem ser visualizados na Figura 17, que apresenta a acurácia do modelo resultante (ao centro) e de dois modelos experimentais.

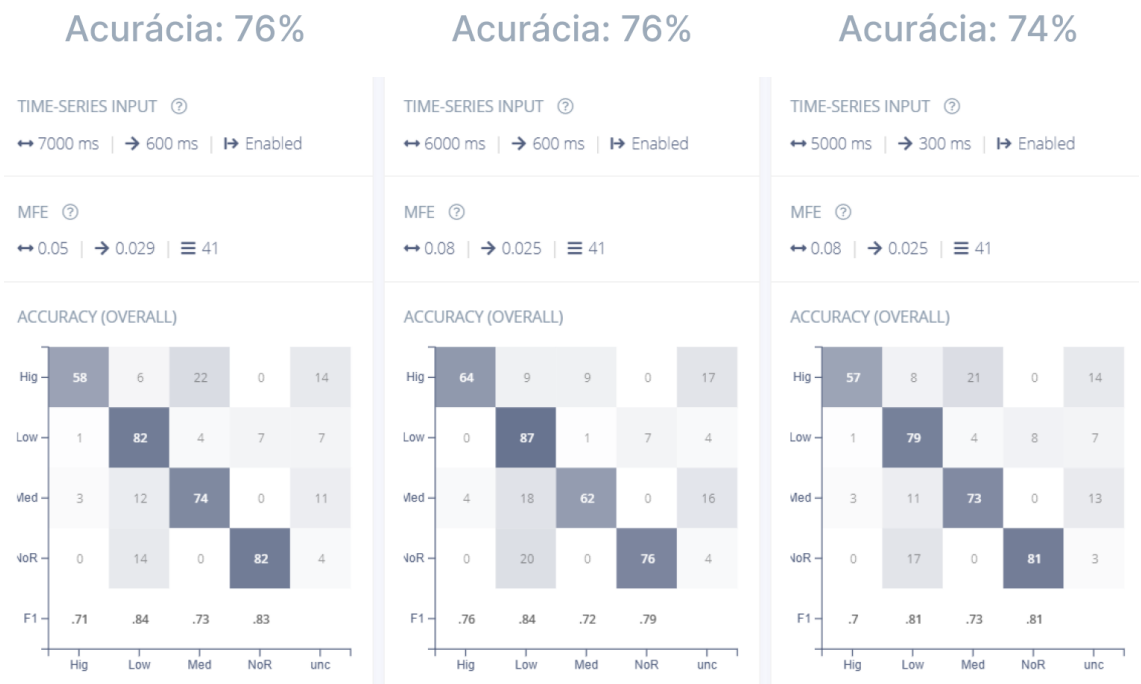


Figura 17 – Modelos obtidos pela ferramenta *EON Tuner* disponível na plataforma *Edge Impulse*

Fonte: Autoria própria

Ao observar a Imagem 17, é possível justificar a escolha do modelo apresentado ao centro, uma vez que ele apresentou uma maior acurácia, juntamente ao modelo à esquerda,

e utilizou um segundo a menos de áudio que seu concorrente com a mesma taxa de acerto, o que resulta em menos memória utilizada para armazenamento do áudio.

Após isso, o modelo escolhido foi testado novamente, incluindo mais formas de visualização dos resultados e, dessa vez, inserindo sua versão quantizada (int8) nos testes, a fim de analisar o impacto da quantização na eficácia geral do modelo.

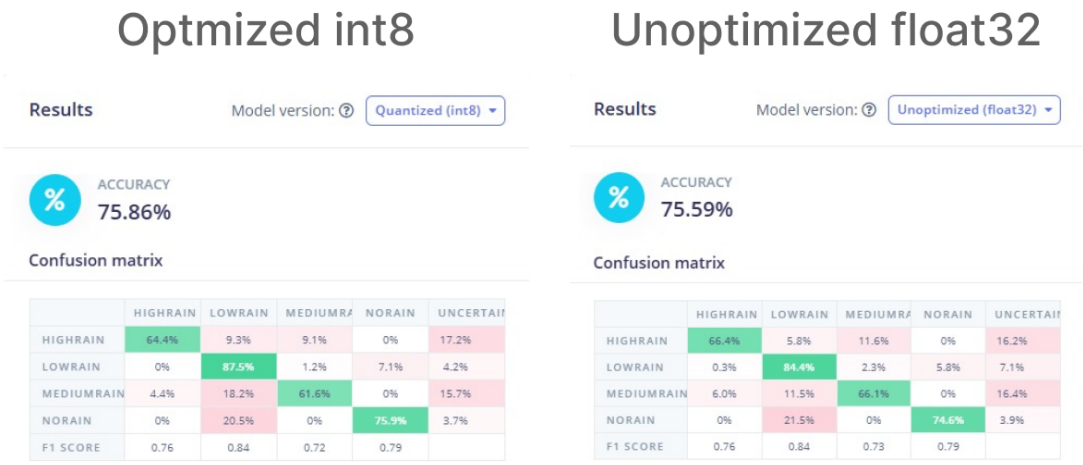


Figura 18 – Comparação dos modelos otimizados e não otimizados para *deploy* para biblioteca arduino por meio da ferramenta *EON Compiler*, disponível no *Edge Impulse*

Fonte: Autoria própria

Observando a Figura 18, nota-se que a quantização não causou impacto negativo no modelo e, pelo contrário, aumentou levemente a acurácia. Isso pode ser visto como um ponto positivo, já que o modelo manteve sua capacidade de generalização utilizando variáveis inteiras que resultam em menor ocupação de memória.

4.2 Inferência

Para validar o funcionamento do modelo em dispositivos de borda, foram realizadas inferências diretamente na placa Arduino, executando o procedimento para um áudio de cada classe. Os *raw features*, obtidos através da ferramenta *Edge Impulse* (19), foram carregados em um static buffer na placa Arduino. O modelo foi capaz de classificar corretamente com 99,6%, 87,1%, 97,6% e 99,6% de certeza as classes de chuva forte, chuva moderada, chuva fraca e sem chuva, respectivamente. Além disso, os LEDs foram acesos corretamente, indicando de forma visual a classificação de cada classe. Vale destacar que esta fase não se trata de uma análise estatística, mas sim de uma validação do funcionamento do modelo em um dispositivo de borda, com foco na execução prática.

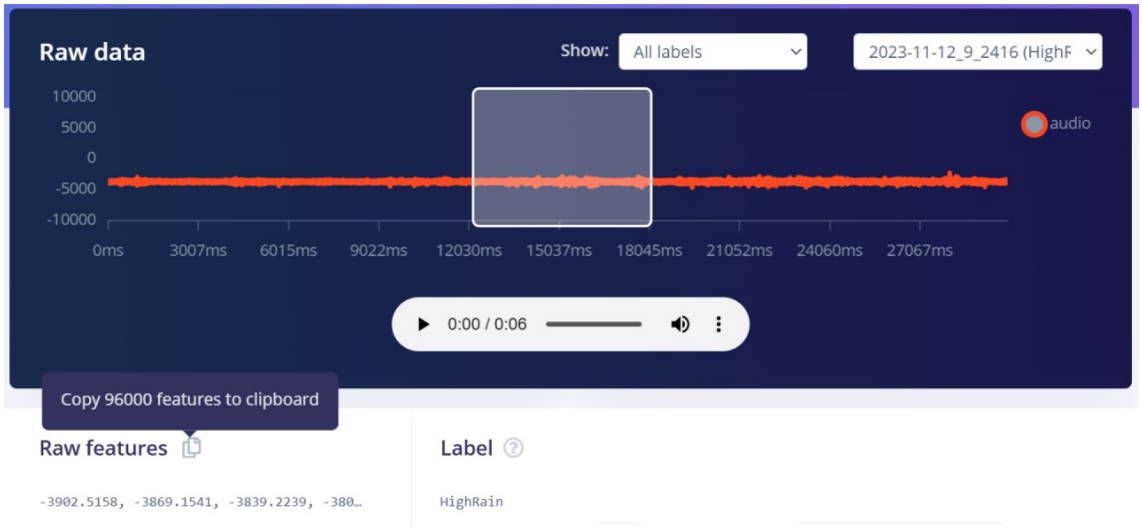
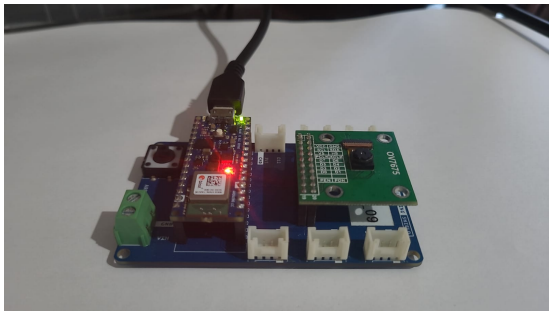


Figura 19 – Janela de áudio exemplo para inferência na placa

Fonte: Autoria própria

(a) LED



Fonte: Autoria própria

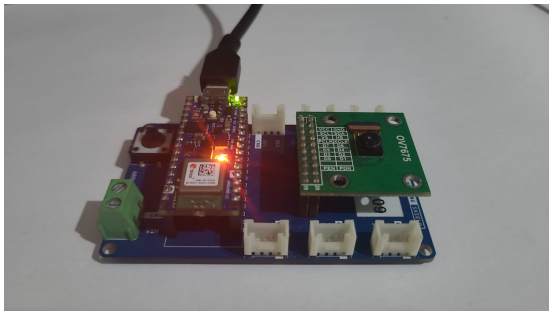
(b) Monitor serial

```
Edge Impulse standalone inferencing (Arduino)
run_classifier returned: 0
Timing: DSP 630 ms, inference 47 ms, anomaly 0 ms
Predictions:
  HighRain: 0.99609
  LowRain: 0.00000
  MediumRain: 0.00000
  NoRain: 0.00000
```

Fonte: Autoria própria

Figura 20 – Inferência da Classe *HighRain*

(a) LED



Fonte: Autoria própria

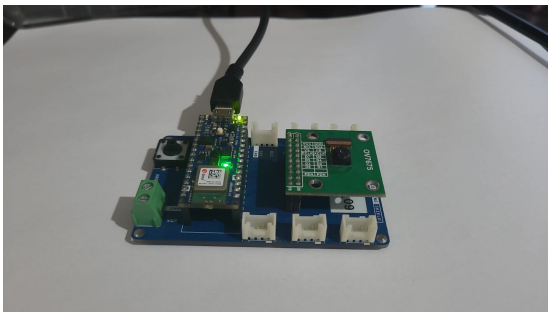
(b) Monitor serial

```
Edge Impulse standalone inferencing (Arduino)
run_classifier returned: 0
Timing: DSP 629 ms, inference 46 ms, anomaly 0 ms
Predictions:
  HighRain: 0.02344
  LowRain: 0.10547
  MediumRain: 0.87109
  NoRain: 0.00000
```

Fonte: Autoria própria

Figura 21 – Inferência da Classe *MediumRain*

(a) LED



Fonte: Autoria própria

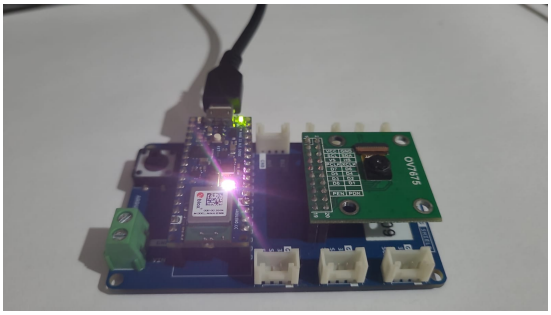
(b) Monitor serial

```
Edge Impulse standalone inferencing (Arduino)
run_classifier returned: 0
Timing: DSP 628 ms, inference 46 ms, anomaly 0 ms
Predictions:
  HighRain: 0.00781
  LowRain: 0.97656
  MediumRain: 0.01562
  NoRain: 0.00000
```

Fonte: Autoria própria

Figura 22 – Inferência da Classe *LowRain*

(a) LED



Fonte: Autoria própria

(b) Monitor serial

```
Edge Impulse standalone inferencing (Arduino)
run_classifier returned: 0
Timing: DSP 628 ms, inference 46 ms, anomaly 0 ms
Predictions:
  HighRain: 0.00000
  LowRain: 0.00000
  MediumRain: 0.00000
  NoRain: 0.99609
```

Fonte: Autoria própria

Figura 23 – Inferência da Classe *NoRain*

5 Conclusão

No decorrer deste trabalho, foi realizado um estudo sobre a viabilidade da utilização de modelos de *machine learning* baseados em CNN para detecção e identificação da intensidade de chuvas. Foi descrito todo o processo, desde a aquisição do *dataset* até a obtenção de um modelo de classificação capaz de identificar as diferentes classes de precipitação.

Ao analisar e trabalhar o *dataset* original, foi possível desenvolver um método para sintetizar e extrair informações, convergindo em um *dataset* mais especifico com características relevantes, que permite ao modelo aprender padrões e realizar inferências de maneira eficiente.

Portanto, visto que foi gerado um modelo capaz de classificar de maneira eficiente diferentes faixas de intensidade de chuva, esse estudo se apresenta como uma evidência de que é possível utilizar modelos de IA/ML baseados em CNN para determinar a intensidade de chuva em um certo momento, viabilizando pluviômetros sem partes móveis como uma alternativa aos pluviômetros automáticos tradicionais.

5.1 Trabalhos Futuros

Após o desenvolvimento deste trabalho, é possível identificar pontos a serem desenvolvidos em trabalho futuros, sendo alguns deles:

- Geração de um *dataset* na região de estudo para melhor representação da realidade local, concentrando a gravação em momentos de chuva, gerando, assim, mais amostras de cada classe.
- Implementação de modelo capaz de quantizar a intensidade de precipitação;
- Desenvolvimento de um hardware específico, com estrutura de proteção contra a chuva e maneiras de cancelamento de ruído, para padronização na aquisição do *dataset* e inferência do modelo.

Referências

ABDICHE, D.; HARRAR, K. Text-independent speaker identification using mel-frequency energy coefficients and convolutional neural networks. In: . [S.l.: s.n.], 2021. p. 204–209. Citado 2 vezes nas páginas 24 e 25.

ABDICHE DEHIA; HARRAR, K. A hybrid of deep neural network and extreme gradient boosting for automatic speaker identification. *International Journal of Computing and Digital Systems*, v. 11, p. 533–543, 2019. Citado na página 26.

ABDUL, Z. K.; AL-TALABANI, A. K. Mel frequency cepstral coefficient and its applications: A review. *IEEE Access*, v. 10, p. 122136–122158, 2022. Citado na página 26.

ADAM, J. Analog signal processing with microwave magnetics. *Proceedings of the IEEE*, v. 76, n. 2, p. 159–170, 1988. Citado na página 23.

AGGARWAL, C. C. et al. *Neural networks and deep learning*. [S.l.]: Springer, 2018. v. 10. Citado 2 vezes nas páginas 30 e 31.

ALKHATIB, M. I. I. et al. An urban acoustic rainfall estimation technique using a cnn inversion approach for potential smart city applications. *Smart Cities*, v. 6, n. 6, p. 3112–3137, 2023. ISSN 2624-6511. Disponível em: <https://www.mdpi.com/2624-6511/6/6/139>. Citado na página 35.

ARDUINO Hardware | Arduino. c2024. <https://www.arduino.cc/en/hardware>. (Accessed on 11/28/2024). Citado na página 35.

AVANZATO, R.; BERITELLI, F. An innovative acoustic rain gauge based on convolutional neural networks. *Information*, v. 11, n. 4, 2020. ISSN 2078-2489. Disponível em: <https://www.mdpi.com/2078-2489/11/4/183>. Citado 3 vezes nas páginas 36, 41 e 43.

AVANZATO, R. et al. A convolutional neural networks approach to audio classification for rainfall estimation. In: . [S.l.: s.n.], 2019. p. 285–289. Citado na página 36.

BATISTA, B.; FERNANDES, F. *UNIFEI-IESTI01-TinyML/00_Curse_Folder/2_Applications_Deploy/Final_Projects_2024/Monit_Acustico.pdf at main · Mjrovai/UNIFEI-IESTI01-TinyML*. 2024. <https://github.com/Mjrovai/UNIFEI-IESTI01-TinyML/blob/main/00_Curse_Folder/2_Applications_Deploy/Final_Projects_2024/Monit_Acustico.pdf>. (Accessed on 10/17/2024). Citado 2 vezes nas páginas 36 e 48.

BENGIO, Y.; GOODFELLOW, I.; COURVILLE, A. *Deep learning*. [S.l.]: MIT press Cambridge, MA, USA, 2017. v. 1. Citado 3 vezes nas páginas 27, 28 e 29.

BLAINSKI, É.; GARBOSSA, L. H. P.; ANTUNES, E. N. Estações hidrometeorológicas automáticas: recomendações técnicas para instalação. *Florianópolis: Epagri*, 2012. Citado na página 22.

BROWNLEE, J. *Long short-term memory networks with python: develop sequence prediction models with deep learning*. [S.l.]: Machine Learning Mastery, 2017. Citado na página 29.

BRUCE, P.; BRUCE, A.; GEDECK, P. *Practical statistics for data scientists: 50+ essential concepts using R and Python*. [S.l.]: O'Reilly Media, 2020. Citado 2 vezes nas páginas 32 e 33.

BUNRIT, S. et al. Text-independent speaker identification using deep learning model of convolution neural network. *International Journal of Machine Learning and Computing*, v. 9, n. 2, p. 143–148, 2019. Citado 2 vezes nas páginas 30 e 32.

CAMBRIDGE. *AUDIO | English meaning - Cambridge Dictionary*. c2024. <https://dictionary.cambridge.org/dictionary/english/audio#google_vignette>. (Accessed on 10/17/2024). Citado na página 23.

CHOLLET, F. *Deep learning with Python*. [S.l.]: Simon and Schuster, 2021. Citado na página 31.

COSTA, R. G. et al. Acústica para engenheiros. *Editora Científica*, 2023. Citado na página 23.

CSEM. *Aurora - Dataset*. 2024. <https://aurora.portal.csem.ch/dataset.html>. (Accessed on 09/13/2024). Citado 6 vezes nas páginas 20, 37, 38, 39, 41 e 43.

DEO, R. et al. Review of feature extraction techniques. In: *Proceedings of the 3rd International Conference on Advances in Science & Technology (ICAST)*. [S.l.: s.n.], 2020. Citado na página 24.

DORNELLES, A. M.; OLIVEIRA, V. Q. de; REVEILLEAU, S. C. Análise das precipitações registradas pelo pluviômetro digital (telemetria) na uhe ernestina-rs. Citado na página 22.

EARGLE, J. *The Microphone Book*. Focal Press, 2004. (Audio Engineering Society Presents Series). ISBN 9780240519616. Disponível em: <https://books.google.com.br/books?id=w8kXMVKOsY0C>. Citado na página 24.

EDGE Impulse - The Leading Edge AI Platform. c2024. <https://edgeimpulse.com/>. (Accessed on 10/11/2024). Citado na página 34.

ELIAS, F. G. de M. *Sinais e sistemas: uma introdução*. 1. ed. Curitiba: Intersaberes, 2020. E-book. Acesso em: 08 out. 2024. Disponível em: <https://plataforma.bvirtual.com.br>. Citado 2 vezes nas páginas 23 e 24.

EVANS, M.; NOBLE, J.; HOCHENBAUM, J. *Arduino em ação*. [S.l.]: Novatec Editora, 2013. Citado na página 34.

GÉRON, A. *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow*. [S.l.]: "O'Reilly Media, Inc.", 2022. Citado na página 33.

GOEL, A.; GUPTA, A. *Design of satellite payload filter emulator using hamming window*. 2014. Citado na página 25.

HYMEL, S. et al. Edge impulse: An mlops platform for tiny machine learning. *arXiv preprint arXiv:2212.03332*, 2022. Citado na página 34.

INMET. *Instituto Nacional de Meteorologia - INMET*. c2019. <https://portal.inmet.gov.br/sobre-meteorologia>. (Accessed on 10/08/2024). Citado na página 21.

ISMAILOV, A. S.; JO'RAYEV, Z. B. et al. Study of arduino microcontroller board. *Science and Education*, v. 3, n. 3, p. 172–179, 2022. Citado na página 34.

JENSEN, J. et al. Evaluation of mfcc estimation techniques for music similarity. In: *Proceedings of the 14th European Signal Processing Conference*. [S.l.: s.n.], 2006. European Signal Processing Conference ; Conference date: 04-09-2006 Through 08-09-2006. Citado na página 26.

JÚNIOR, O. dos S. et al. Análise das estimativas de precipitação: uma revisão. *Ciência e Natura*, p. 267–271, 2018. Citado na página 21.

KOBIYAMA, M. et al. *Prevenção de desastres naturais: conceitos básicos*. [S.l.]: Organic Trading Curitiba, 2006. Citado na página 19.

KRISHNA, G. et al. Generating eeg features from acoustic features. In: *2020 28th European Signal Processing Conference (EUSIPCO)*. [S.l.: s.n.], 2021. p. 1100–1104. Citado na página 24.

LIN, T.-H. et al. Real-time hollow defect detection in tiles using on-device tiny machine learning. *Measurement Science and Technology*, IOP Publishing, v. 35, n. 5, p. 056006, 2024. Citado 2 vezes nas páginas 25 e 26.

MARKETSANMARKETS. *Edge AI Software Market Size, Share and Global Market Forecast to 2028 | MarketsandMarkets*. 2023. <https://www.marketsandmarkets.com/Market-Reports/edge-ai-software-market-70030817.html>. (Accessed on 10/11/2024). Citado na página 34.

MATPLOTLIB. *Matplotlib — Visualization with Python*. c2024. <https://matplotlib.org/>. (Accessed on 10/09/2024). Citado na página 23.

MCKINNEY, W. *Python para análise de dados: Tratamento de dados com Pandas, NumPy e IPython*. [S.l.]: Novatec Editora, 2018. Citado 2 vezes nas páginas 22 e 23.

MCROBERTS, M. *Arduino Básico-2ª edição: Tudo sobre o popular microcontrolador Arduino*. [S.l.]: Novatec Editora, 2015. Citado na página 34.

MEDEIROS, J. D. F.; GOLDENFUM, J. A.; CASTRO, N. Problemas operacionais observados no monitoramento hidrológico de pequenas bacias rurais embutidas. *15º SIMPÓSIO BRASILEIRO DE RECURSOS HÍDRICOS*, 2003. Citado na página 19.

MENEZES, A. C. dos S. *Desenvolvimento e Calibração de Pluviômetros Automáticos de Baixo Custo*. 45 p. Dissertação (Dissertação (Mestrado)) — Universidade Federal do Vale do São Francisco - UNIVASF, Juazeiro, BA, 2019. Curso de Engenharia Agrícola. Citado na página 21.

MILANESI, M. A.; ALVES, R. R.; GALVANI, E. Comparativo entre instrumentos pluviométricos experimentais e automáticos. *Os Desafios da Geografia Física na Fronteira do Conhecimento*, v. 1, p. 2251–2261, 2017. Citado na página 21.

MITCHELL, T. *Machine Learning*. McGraw-Hill, 1997. (McGraw-Hill International Editions). ISBN 9780071154673. Disponível em: <https://books.google.com.br/books?id=EoYBngEACAAJ>. Citado na página 27.

NIELSEN, M. A. *Neural networks and deep learning*. [S.l.]: Determination press San Francisco, CA, USA, 2015. v. 25. Citado 5 vezes nas páginas 28, 30, 31, 32 e 48.

NUMPY. *NumPy Documentation*. c2022. <https://numpy.org/doc/>. (Accessed on 10/09/2024). Citado na página 23.

O'SHAUGHNESSY, D. *Speech Communication: Human and Machine*. Addison-Wesley Publishing Company, 1987. (Addison-Wesley series in electrical engineering). ISBN 9780201165203. Disponível em: <https://books.google.com.br/books?id=mHFQAAAAMAAJ>. Citado na página 26.

PIAZZA, G. A. et al. Análise espacial e temporal dos dados de precipitação das estações de entorno da bacia do ribeirão concórdia, lontras (sc) visando sua inserção no contexto de mudanças climáticas. *Revista Brasileira de Meteorologia*, SciELO Brasil, v. 31, n. 4 suppl 1, p. 580–592, 2016. Citado na página 21.

PRACTICAL Cryptography. c2012. <http://practicalcryptography.com/miscellaneous/machine-learning/guide-mel-frequency-cepstral-coefficients-mfccs/>. (Accessed on 10/15/2024). Citado na página 25.

PROCESSING-BLOCKS/SPECTROGRAM at master · edgeimpulse/processing-blocks · GitHub. 2023. <https://github.com/edgeimpulse/processing-blocks/tree/master/spectrogram>. (Accessed on 10/15/2024). Citado na página 25.

PYTHON. *Python 3.13.0 Documentation*. 2024. <https://docs.python.org/3/>. (Accessed on 10/08/2024). Citado na página 22.

RAY, P. P. A review on tinyml: State-of-the-art and prospects. *Journal of King Saud University-Computer and Information Sciences*, Elsevier, v. 34, n. 4, p. 1595–1623, 2022. Citado na página 27.

SAHA, S. A comprehensive guide to convolutional neural networks—the eli5 way. *Towards data science*, Medium publication, v. 15, p. 15, 2018. Citado na página 30.

TB3 Tipping Bucket Rain Gauge | KISTERS North America. c2024. <https://products.kisters.net/products/hardware/meteorology/tb3-tipping-bucket-rain-gauge>. (Accessed on 12/12/2024). Citado na página 22.

TURNER, C. et al. The wavelet and fourier transforms in feature extraction for text-dependent, filterbank-based speaker recognition. *Procedia Computer Science*, v. 6, p. 124–129, 2011. ISSN 1877-0509. Complex adaptive sysytems. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1877050911004893>. Citado na página 24.

WANG, M. et al. Estimating rainfall intensity based on surveillance audio and deep-learning. *Environmental Science and Ecotechnology*, v. 22, p. 100450, 2024. ISSN 2666-4984. Disponível em: <https://www.sciencedirect.com/science/article/pii/S2666498424000644>. Citado na página 35.

WAQAR, D. M. et al. Design of a speech anger recognition system on arduino nano 33 ble sense. In: IEEE. *2021 IEEE 7th International Conference on Smart Instrumentation, Measurement and Applications (ICSIMA)*. [S.l.], 2021. p. 64–69. Citado na página 35.

WHAT is Edge Impulse? | Edge Impulse Documentation. 2024. <https://docs.edgeimpulse.com/docs/concepts/edge-ai/what-is-edge-impulse>. (Accessed on 10/11/2024). Citado na página 34.

WORLD Weather API and Weather Forecast. c2024. <https://www.worldweatheronline.com/>. (Accessed on 11/18/2024). Citado na página 40.

YNOUE, R. Y. et al. *Meteorologia: noções básicas*. [S.l.]: Oficina de Textos, 2017. Citado na página 21.

ZENNARO, M. et al. From mountains to data: Low-cost weather stations in kyrgyzstan's challenging terrain. *Preprints*, Preprints, June 2024. Disponível em: <https://doi.org/10.20944/preprints202406.1898.v1>. Citado na página 36.

Apêndices

APÊNDICE A – Código utilizado para criação do CSV e da Rotulação

```python
#example multiple files
def read_data_multiple_files_hour(directory, save_in_directory,csv_filename):
    total_temperature = 0
    total_pressure = 0
    total_altitude = 0
    total_humidity = 0
    total_wind_count = 0
    total_rain_count = 0
    total_wind_direction = 0
    contagemAmostras = 0
    pulses = []
    #sound_data = []
    # Create directory if it doesn't exist
    os.makedirs(save_in_directory, exist_ok=True)

    # Setup for CSV file
    csv_headers = ['timestamp', 'temperature', 'pressure', 'altitude', 'humidity', 'wind_count', 'wind_km_h','rain_count', 'rain_mm', 'wind_direction', 'audio_file', 'label', 'amostras','pulses']

    # We will check if the CSV already exists to decide whether to write headers or not
    write_headers = not os.path.exists(csv_filename)

    with open(csv_filename, 'a', newline='') as csv_file:
        csv_writer = csv.DictWriter(csv_file, fieldnames=csv_headers)
        if write_headers:
            csv_writer.writeheader()

        # Get a list of all .dat files in the directory
        dat_files = glob.glob(os.path.join(directory, '*.dat'))

        # Sort the files by filename
        dat_files.sort(key=filename_to_datetime)

        for file_name in dat_files:
            try:
                print(f"Processing {file_name}...")
                # open the .dat file with "rb" (read binary) mode
                with open(file_name, "rb") as f:
                    data = f.read()
                    f.close()

                #o timestamp deve dever o nome do arquivo raw_data_CSEMAurora1_2023-08-28_0 alterado para o formato 2023-08-27 00_00_00
                timestamp_name = file_name.split('_')[3] + ' ' + file_name.split('_')[4][:-4] + '_' + '00' + '_' + '00'

                # read the binary data
                num_sample = int(len(data) / SAMPLE_SIZE) # should be an integer
                for j in range(num_sample):
                    index = j * SAMPLE_SIZE
                    packet_size = struct.unpack("<L", data[index: index + PACKET_SIZE])[0] # should be equal to SAMPLE_SIZE
                    index += PACKET_SIZE
                    packet_id = struct.unpack("<L", data[index: index + ID_SIZE])[0]
                    index += ID_SIZE
                    timestamp = struct.unpack("<Q", data[index: index + TIMESTAMP_SIZE])[0]
                    index += TIMESTAMP_SIZE
                    samples_raw =  data[index: index + AUDIO_SIZE]
                    index += AUDIO_SIZE
                    bme_values = list(struct.iter_unpack("<f", data[index: index + BME280_SIZE]))
                    index += BME280_SIZE
                    mechanical_values = list(struct.iter_unpack("<I", data[index: index + MECHANICAL_SIZE]))
                    index += MECHANICAL_SIZE

                    # post process and save the BME and mechanical data in a dict
                    BME_data = {"temperature":bme_values[0][0], "pressure":bme_values[1][0], "altitude":bme_values[2][0], "humidity":bme_values[3][0]}
                    mechanical_data = {"wind_count":mechanical_values[0][0],"rain_count":mechanical_values[1][0],"wind_direction":mechanical_values[2][0]}
```

Figura 24 – Código para criação do CSV e Rotulação - parte 1

Fonte: Autoria própria

```
        if mechanical_data['rain_count'] > 0:
            pulses.append(j)

        total_temperature += BME_data['temperature']
        total_pressure += BME_data['pressure']
        total_altitude += BME_data['altitude']
        total_humidity += BME_data['humidity']
        total_wind_count += mechanical_data['wind_count']
        total_rain_count += mechanical_data['rain_count']
        total_wind_direction += mechanical_data['wind_direction']
        contagemAmostras += 1

    med_temperature = total_temperature / contagemAmostras
    med_pressure = total_pressure / contagemAmostras
    med_altitude = total_altitude / contagemAmostras
    med_humidity = total_humidity / contagemAmostras
    med_wind_count = total_wind_count / contagemAmostras
    med_wind_direction = total_wind_direction / contagemAmostras
    wind_km_h = med_wind_count * 2.4
    rain_mm = total_rain_count * 0.2794
    wind_deg = station1.getWindDirection(med_wind_direction)
    label = ''
    if rain_mm < rain_limits[0]:
        label = 0
    elif rain_mm < rain_limits[1]:
        label = 1
    elif rain_mm < rain_limits[2]:
        label = 2
    else:
        label = 3

    row_data = {
        'timestamp': timestamp_name,
        'temperature': med_temperature,
        'pressure': med_pressure,
        'altitude': med_altitude,
        'humidity': med_humidity,
        'wind_count': med_wind_count,
        'wind_km_h': wind_km_h,
        'rain_count': total_rain_count,
        'rain_mm': rain_mm,
        'wind_direction': wind_deg,
        'audio_file': 'null',
        'label': label,
        'amostras': contagemAmostras,
        'pulses': pulses
    }
    csv_writer.writerow(row_data)

    total_temperature = 0
    total_pressure = 0
    total_altitude = 0
    total_humidity = 0
    total_wind_count = 0
    total_rain_count = 0
    total_wind_direction = 0
    contagemAmostras = 0
    pulses = []
    #sound_data = []
    time_ini = ''
    time_ini_name =''
except:

    print(f"Error processing {file_name}...")
```

Figura 25 – Código para criação do CSV e Rotulação - parte 2

Fonte: Autoria própria

APÊNDICE B – Código utilizado para criação dos gráficos de chuva acumulada

```
city = 'Neuchatel, Suiza'
df2 = pd.read_csv('weather_data_Neuchatel_hourly.csv')
df2.shape
df2['date_time'] = pd.to_datetime(df2['date_time'])
df2.set_index('date_time', inplace=True)

#eliminar todos os dados antes de 2023-08-28 00:00 e 2024-02-18 23:00
df2 = df2.loc['2023-08-28 00:00':'2024-02-18 23:00']

#eliminar dados entre 2023-10-09 00:00 e 2023-11-09 00:00
df2 = df2.drop(df2.loc['2023-10-09 00:00':'2023-11-09 00:00'].index)

#eliminar dados entre 2023-11-12 23:00 e 2023-11-24 16:00
df2 = df2.drop(df2.loc['2023-11-12 23:00':'2023-11-24 16:00'].index)

#eliminar dados entre 2023-11-26 23:00 e 2023-12-07 00:00
df2 = df2.drop(df2.loc['2023-11-26 23:00':'2023-12-07 00:00'].index)

df3 = pd.read_csv('./TestDataset/weather_data_with_audio_hour.csv')
#eliminar a segunda data da coluna timestamp para cada linha e converter para datetime
df3['timestamp'] = pd.to_datetime(df3['timestamp'],format='%Y-%m-%d %H_%M_%S')
df3.set_index('timestamp', inplace=True)
```

```
plt.figure(figsize=(14, 6))
plt.plot(df2['precipMM'], label='Precipitation (mm)', color='blue')
plt.title('Precipitation Over Time - WWO Dataset')
plt.xlabel('Date and Time')
plt.ylabel('Precipitation (mm)')
plt.ylim(top=8.5)
plt.grid(True)
plt.legend()
plt.show()

plt.figure(figsize=(14, 6))
plt.plot(df3['rain_mm'], label='Precipitation (mm)', color='blue')
plt.title('Precipitation Over Time - Aurora Dataset')
plt.xlabel('Date and Time')
plt.ylabel('Precipitation (mm)')
plt.ylim(top=8.5)
plt.grid(True)
plt.legend()
plt.show()
```

Figura 26 – Código para criação dos gráficos de chuva acumulada

Fonte: Autoria própria

APÊNDICE C – Código utilizado para seleção dos trechos de áudio

```python
#example multiple files
def save_audio_data_multiple_files_hour(directory, save_in_directory):

    countLowRain = 20
    countNoRain = 0
    countMediumRain = 99
    countHighRain = 99
    sound_data = []
    label_name = ''

    # Create directory if it doesn't exist
    os.makedirs(save_in_directory, exist_ok=True)

    #create directories for each label
    os.makedirs(f"{save_in_directory}/NoRain", exist_ok=True)
    os.makedirs(f"{save_in_directory}/LowRain", exist_ok=True)
    os.makedirs(f"{save_in_directory}/MediumRain", exist_ok=True)
    os.makedirs(f"{save_in_directory}/HighRain", exist_ok=True)

    # Get a list of all .dat files in the directory
    dat_files = glob.glob(os.path.join(directory, '*.dat'))

    # Sort the files by filename
    dat_files.sort(key=filename_to_datetime)

    for file_name in dat_files:
        #try:
            print(f"Processing {file_name}...")

            #o timestamp deve dever o nome do arquivo raw_data_CSEMAurora1_2023-08-28_0 alterado para o formato 2023-08-27 00_00_00
            #ele será comparado com o a coluna timestamp do csv, dependendo do label, o arquivo de audio será salvo em uma pasta específica
            timestamp_name = file_name.split('_')[3] + '_' + file_name.split('_')[4][:-4]
            print(timestamp_name)
            if timestamp_name in df['timestamp'].values:
                print(f"Processing {timestamp_name}...")
                label = df.loc[df['timestamp'] == timestamp_name, 'label'].values[0]
                print(f"Label: {label}")

            #if the four counts are zero, break the loop
            if countNoRain == 0 and countLowRain == 0 and countMediumRain == 0 and countHighRain == 0:
                break

            match label:
                case 0:
                    if countNoRain == 0:
                        continue
                    countNoRain -= 1
                    label_name = 'NoRain'
                case 1:
                    if countLowRain == 0:
                        continue
                    countLowRain -= 1
                    label_name = 'LowRain'
                case 2:
                    if countMediumRain == 0:
                        continue
                    countMediumRain -= 1
                    label_name = 'MediumRain'
                case 3:
                    if countHighRain == 0:
                        continue
                    countHighRain -= 1
                    label_name = 'HighRain'
```

Figura 27 – Código para seleção dos trechos de áudio - Parte 1

Fonte: Autoria própria

```python
# open the .dat file with "rb" (read binary) mode
with open(file_name, "rb") as f:
    data = f.read()
    f.close()

# read the binary data
for pulses in df.loc[df['timestamp'] == timestamp_name, 'pulses']:
    #convertendo a string para lista e tirando espacos vazios
    pulses = pulses[1:-1].split(',')

    for pulse in pulses:
        sound_data = []
        pulse = int(pulse)
        start_index = max(0, pulse - 15)
        end_index = min(len(data), pulse + 15)
        for j in range(start_index, end_index):
            index = j * SAMPLE_SIZE
            index += PACKET_SIZE
            # packet_id = struct.unpack("<L", data[index: index + ID_SIZE])[0]
            index += ID_SIZE
            # timestamp = struct.unpack("<Q", data[index: index + TIMESTAMP_SIZE])[0]
            index += TIMESTAMP_SIZE
            samples_raw =  data[index: index + AUDIO_SIZE]
            index += AUDIO_SIZE
            index += BME280_SIZE
            index += MECHANICAL_SIZE

            # post process the audio samples collected with I2S protocol
            samples = []
            for i in range(0, int(AUDIO_SIZE / AUDIO_SAMPLE_SIZE)): # get the number of audio samples, not the number of bytes
                value =     samples_raw[AUDIO_SAMPLE_SIZE*i      ] << 16
                value +=    samples_raw[AUDIO_SAMPLE_SIZE*i + 1  ] << 8
                value +=    samples_raw[AUDIO_SAMPLE_SIZE*i + 2  ]
                if(value>2**23 - 1):
                    value = -(2**24 - value)
                samples.append(value)
            sound_data += samples


        #save sound data in a .wav file
        sound_file = f"{save_in_directory}/{label_name}/{timestamp_name}_{pulse}.wav"
        obj = wave.open(sound_file, "w")
        obj.setnchannels(1)
        obj.setsampwidth(AUDIO_SAMPLE_SIZE)
        obj.setframerate(AUDIO_SAMPLING_FREQ)
        data_as_bytes = [struct.pack("<i", sample) for sample in sound_data] # store it as 4 bytes in LSB format (little endian)
        for data_bytes in data_as_bytes:
            obj.writeframes(data_bytes[0:3]) # remove the fourth unused byte one ().
        obj.close()
```

Figura 28 – Código para seleção dos trechos de áudio - Parte 2

Fonte: Autoria própria

APÊNDICE D – Código utilizado para demonstração do MFE

```python
def calc_plot_mfe_features(audio,
                           sample_rate,
                           alpha = 0.97,
                           NFFT=256,
                           low_freq_cut=80,
                           nfilt=41,
                           noise_floor_dB=-64,
                           frame_size=0.08,
                           frame_stride=0.025,
                           figsize=(10, 5),
                           title= 'MFE',
                           audio_duration=6  # Duration of the audio in seconds
                           ):

    # Pre-emphasis
    audio = np.append(audio[0], audio[1:] - alpha * audio[:-1])

    # Calculate frame length and frame step (convert from seconds to samples)
    frame_length, frame_step = frame_size * sample_rate, frame_stride * sample_rate
    frame_length = int(round(frame_length))
    frame_step = int(round(frame_step))

    # Calculate the total number of frames
    num_frames = int(np.ceil(float(np.abs(len(audio) - frame_length)) / frame_step))

    # Pad audio signal
    pad_audio_length = num_frames * frame_step + frame_length
    z = np.zeros((pad_audio_length - len(audio)))
    pad_audio = np.append(audio, z)

    # Initialize the frames
    indices = np.tile(np.arange(0, frame_length), (num_frames, 1)) + np.tile(np.arange(0, num_frames * frame_step, frame_step), (frame_length, 1)).T
    frames = pad_audio[indices.astype(np.int32, copy=False)]

    # Apply window function (Hamming)
    frames *= np.hamming(frame_length)

    # Perform FFT and calculate power spectrum
    mag_frames = np.absolute(fft.fft(frames, NFFT))
    pow_frames = ((1.0 / NFFT) * ((mag_frames) ** 2))

    # Take only up to (NFFT/2) + 1 elements after the FFT,
    # as those are the unique frequency components for real-valued signals.
    mag_frames = mag_frames[:, :NFFT//2 + 1]
    pow_frames = ((1.0 / NFFT) * ((mag_frames) ** 2))

    # Apply Mel filter banks
    low_freq_mel = 2595 * np.log10(1 + low_freq_cut / 700.0)  # Convert Hz to Mel
    high_freq_mel = (2595 * np.log10(1 + (sample_rate / 2) / 700))  # Convert Hz to Mel
    mel_points = np.linspace(low_freq_mel, high_freq_mel, nfilt + 2)  # Equally spaced in Mel scale
    hz_points = (700 * (10**(mel_points / 2595) - 1))  # Convert Mel to Hz
    bin = np.floor((NFFT + 1) * hz_points / sample_rate)

    fbank = np.zeros((nfilt, int(np.floor(NFFT / 2 + 1))))
    for m in range(1, nfilt + 1):
        f_m_minus = int(bin[m - 1])
        f_m = int(bin[m])
        f_m_plus = int(bin[m + 1])

        for k in range(f_m_minus, f_m):
            fbank[m - 1, k] = (k - bin[m - 1]) / (bin[m] - bin[m - 1])
        for k in range(f_m, f_m_plus):
            fbank[m - 1, k] = 1 - (k - bin[m]) / (bin[m + 1] - bin[m])
```

Figura 29 – Código para demonstração do MFE - Parte 1

Fonte: Autoria própria

```
filter_banks = np.dot(pow_frames, fbank.T)
filter_banks = np.where(filter_banks == 0, np.finfo(float).eps, filter_banks)
filter_banks = 20 * np.log10(filter_banks)  # dB

# Apply the noise floor
filter_banks = np.where(filter_banks > noise_floor_dB, noise_floor_dB, filter_banks)

# Normalize the Mel Filterbank Energies
mean = np.mean(filter_banks, axis=0)
std_dev = np.std(filter_banks, axis=0)
std_dev[std_dev == 0] = 1e-10
filter_banks -= mean
filter_banks /= std_dev

# Calculate time array for x-axis
time = np.linspace(0, audio_duration, num_frames)

return filter_banks
```

Figura 30 – Código para demonstração do MFE - Parte 2

Fonte: Autoria própria