Universidade de Brasília
Faculdade de Tecnologia
Departamento de Engenharia Elétrica

Classificação da Condição de Pavimentos Utilizando Redes Neurais e Sistemas Embarcados

Giulianno Querino Mattei

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Engenharia Elétrica

Orientador
Prof. Dr. Edson Mintsu Hung

Brasília
2023

Universidade de Brasília
Faculdade de Tecnologia
Departamento de Engenharia Elétrica

Classificação da Condição de Pavimentos Utilizando Redes Neurais e Sistemas Embarcados

Giulianno Querino Mattei

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Engenharia Elétrica

Prof. Dr. Edson Mintsu Hung (Orientador)
Universidade de Brasília

Dr. Renam Castro da Silva
Dell Technologies

Prof. Dr. Tiago Alves Fonseca
Universidade de Brasília

Prof. Dr. Eduardo Peixoto Fernandes da Silva
Coordenador do Bacharelado em Engenharia Elétrica

Brasília, 10 de Dezembro de 2023

Dedicatória

Dedico este trabalho aos meus pais, que foram sempre meus maiores incentivadores.

Dedico também ao meu avô Arimatéa, cujo jornal, presenteado a mim no dia da minha aprovação no vestibular, guardo com carinho até hoje.

Agradecimentos

Agradeço aos meus pais, ao meu irmão, à minha namorada e aos meus amigos pelo apoio e paciência nos momentos difíceis.

Agradeço também aos meus professores pelos ensinamentos e contribuições para o meu desenvolvimento acadêmico e pessoal.

Resumo

O propósito deste trabalho consiste em desenvolver um classificador referente à condição do pavimento utilizando modelos de Inteligência Artificial e sistema embarcado. O classificador é binário e categoriza o estado do pavimento em duas classes: perfeito ou imperfeito. Esta classificação é feita com base em dados coletados por um sensor acelerômetro/giroscópio embutido no interior de um veículo, e com base em imagens capturadas por uma câmera direcionada para o pavimento instalada no veículo. Neste trabalho, são desenvolvidos dois modelos de Inteligência Artificial, sendo um voltado para a classificação dos dados coletados pelo sensor e outro para a classificação das imagens. Ao longo do trabalho, são detalhados a implementação do *hardware* embarcado e o desenvolvimento dos modelos de IA, assim como os resultados obtidos, os quais foram validados em saídas de campo.

Palavras-chave: Redes Neurais, Sistemas Embarcados, Processamento de Sinais

Abstract

The purpose of this work is to develop a classifier related to pavement condition using Artificial Intelligence models and embedded system. The classifier is binary, categorizing the pavement condition into two classes: perfect or imperfect. This classification is based on data collected by an accelerometer/gyroscope sensor embedded in a vehicle, and on images captured by a camera aimed at the pavement installed in the vehicle. In this study, two Artificial Intelligence models are developed, one for classification of data collected by the sensor and another for the classification of images. Throughout the study, the implementation of the embedded hardware and the development of AI models are detailed, as well as the obtained results, which were validated in fieldworks.

Keywords: Neural Networks, Embedded Systems, Signal Processing

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

CNT Confederação Nacional do Transporte.

CPU Central Processing Unit.

DRAM Dynamic Random Access Memory.

EEPROM Electrically Erasable Programmable Read-Only Memory.

EPROM Erasable Programmable Read-Only Memory.

FFT Fast Fourier Transform.

I^{2}C Inter IC Bus.

IA Inteligência Artificial.

IDE Integrated Development Environment.

PROM Programmable Read-Only Memory.

RAM Random Access Memory.

ROM Read-Only Memory.

RTC Real-time Clock.

SGD Stochastic Gradient Descent.

SRAM Static Random Access Memory.

TDF Transformada Discreta de Fourier.

UART Universal Asynchronous Receiver/Transmitter.

USB Universal Serial Bus.

1 Introdução

O transporte é um elemento vital na construção da sociedade e da economia, sendo um reflexo direto da qualidade de vida das populações urbanas e rurais. A mobilidade eficiente está intrinsecamente ligada ao acesso a serviços básicos, à geração de empregos, ao desenvolvimento econômico e à promoção de uma convivência harmoniosa nas cidades. No cenário nacional, a infraestrutura rodoviária, apesar de abranger uma extensa malha de estradas, muitas vezes se depara com problemas de manutenção, falta de investimento adequado e ineficiência operacional. Nesse contexto, a Pesquisa CNT de Rodovias surge como uma importante ferramenta de avaliação das condições da malha rodoviária nacional. [19]

Realizada desde 1995, a Pesquisa CNT de Rodovias é uma iniciativa da empresa CNT (Confederação Nacional do Transporte) que visa avaliar o estado das rodovias brasileiras, fornecendo dados sobre sua qualidade e condição. Durante a pesquisa são percorridos mais de 100 mil quilômetros, em que são avaliadas diferentes características da via, como a condição do pavimento, a existência de sinalização horizontal e vertical e a geometria da via. Dentro da análise do pavimento, que será o foco deste trabalho, a classificação é feita com base no tipo de defeito do pavimento (ou na ausência de defeitos) nas seguintes categorias: perfeito, desgastado, trincas em malha/remendos, afundamentos/ondulações/buracos ou destruído. Na análise de sinalização horizontal e vertical observa-se a existência de faixas centrais, faixas laterais e placas. Na análise de geometria são observadas propriedades espaciais da via, como largura de faixa e declividade [19]. O presente trabalho visa aprimorar a pesquisa desenvolvida pela CNT, por meio da utilização de sistemas embarcados e Inteligência Artificial, conforme será melhor explicado nas seções seguintes.

O objetivo deste trabalho é desenvolver um classificador de pavimento em duas classes: perfeito e imperfeito. Neste trabalho, não será feita uma classificação específica para cada tipo de defeito dentro da classe dos imperfeitos, porém recomenda-se considerar esta sugestão como uma perspectiva relevante para futuros estudos.

Para implementar este classificador, serão utilizados um sensor acelerômetro/giroscópio integrado a um microcontrolador e uma câmera, ambos embutidos no interior de um veículo próprio da CNT utilizado durante a pesquisa. Além disso, serão desenvolvidos dois

modelos de IA, sendo um para classificação das imagens capturadas pela câmera, e outro para classificação dos dados coletados pelo sensor.

1.1 Objetivos

O presente trabalho apresenta os objetivos gerais e específicos descritos a seguir.

1.1.1 Objetivo Geral

Desenvolver um classificador de pavimento em duas classes: perfeito e imperfeito.

1.1.2 Objetivos específicos

- Desenvolver protótipo de *hardware* embarcado para classificação de condições do pavimento utilizando microcontrolador e sensor acelerômetro/giroscópio
- Desenvolver modelo de IA para classificar a condição do pavimento com base em imagens
- Desenvolver modelo de IA para classificar a condição do pavimento com base em sinais coletados pelos sensores

1.2 Organização do trabalho

O trabalho possui seis capítulos, sendo este o primeiro. O Capítulo 2 trata de uma revisão teórica acerca de conceitos necessários para o desenvolvimento do projeto. O Capítulo 3 descreve como foi feita a implementação do *hardware* embarcado. O Capítulo 4 descreve a estrutura dos modelos de IA desenvolvidos e o processo de treinamento do modelo baseado em imagem. O Capítulo 5 apresenta os testes práticos feitos com o protótipo embarcado. Além disso, detalha o processo de treinamento do modelo de IA baseado nos dados provenientes do sensor, e os resultados obtidos com ambos modelos de IA. Por fim, no Capítulo 6 são expostas as conclusões finais e perspectivas de trabalhos futuros.

2 Referencial Teórico

2.1 Microcontroladores

Um microcontrolador é um circuito integrado que contém os componentes básicos de um computador, como CPU, memória e componentes de entrada e saída. Geralmente também contém periféricos, como conversores de sinais analógicos ou digitais, barramentos de comunicação serial e temporizadores.

Uma CPU é formada por uma unidade de controle, uma unidade de execução e registradores. O processador busca instruções na memória do programa e as decodifica na unidade de controle. Então, a unidade de controle gera comandos que serão lidos pela unidade de execução, onde são produzidas microoperações lógicas que executam a instrução passada. [20]

Existem duas categorias de memória presentes em microcontroladores: RAM e ROM. Essencialmente, a principal diferença entre elas é que a memória RAM é volátil e, portanto, perde o seu conteúdo caso não esteja conectada à energia. Já a memória ROM é não-volátil e, portanto, mantém seu conteúdo mesmo sem energia.

A memória ROM pode ser classificada entre os seguintes tipos: PROM, EPROM, EEPROM ou FLASH. A memória PROM é uma memória ROM programável, porém só uma vez. Não é possível reprogramá-la. A memória EPROM é uma memória programável e apagável. Porém, o apagamento era feito com luz ultravioleta. E, portanto, é necessário retirar fisicamente a memória do aparelho para reprogramá-la e depois colocar de volta. A memória EEPROM funciona da mesma forma que a EPROM, porém com a diferença que sua reprogramação é feita eletricamente. Por fim, a memória FLASH possui funcionamento análogo ao da memória EEPROM, porém sua programação é feita em blocos de bytes. [21]

A memória RAM pode ser dividida entre DRAM e SRAM. A memória DRAM é uma memória dinâmica que necessita de uma atualização constantemente para manter os dados ativos. Essa atualização é chamada de *refresh* e é feita automaticamente. A consequência disso é que essa memória é mais lenta que a SRAM. Porém, essa memória é implementada com menos transistores e ocupa menos espaço no chip, o que diminui seu

custo. A memória SRAM é uma memória estática mais rápida que a DRAM, por não ser necessário o *refresh*, porém possui um custo mais elevado. [21]

No tocante aos periféricos de um microcontrolador, os elementos de maior utilidade para este projeto são os barramentos de comunicação serial, que serão detalhados nas Seções 2.1.1 e 2.1.2.

2.1.1 I²C

O I²C é um protocolo de comunicação serial que estabelece uma comunicação síncrona e bidirecional utilizando apenas dois condutores, sendo um para envio dos dados e outro para o clock, conforme ilustrado na Figura 2.1. O I²C opera no modo mestre-escravo e o clock é controlado pelo mestre. A lógica do protocolo é que a linha só vai para nível alto quando todos os dispositivos conectados a ela estiverem em nível alto. Caso um dispositivo coloque a saída em 0, a linha vai para nível baixo. [2]

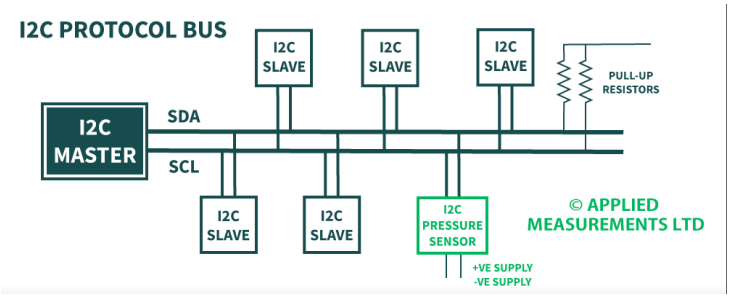

Figura 2.1: Barramento do protocolo I²C. Fonte: [1].

Nomenclatura

Mestre: é o dispositivo que determina quando inicia e quando termina a comunicação

Escravo: dispositivo endereçado pelo mestre

Transmissor: dispositivo que coloca dados no barramento

Receptor: dispositivo que lê os dados do barramento

Tanto mestre quanto escravo podem transmitir ou receber dados. Portanto, pode-se ter mestre transmissor, mestre receptor ou escravo transmissor e escravo receptor.

Transferência de dados

Na comunicação I²C, a linha SDA corresponde à linha de dados e a linha SCL corresponde à linha do clock. Os dados são enviados quando o clock SCL está em nível alto. Nesse momento, a linha de dados tem que permanecer estável e não pode ser alterada. Os dados a serem enviados podem ser alterados quando a linha de clock SCL estiver em nível baixo. Existem duas exceções para essa estrutura: as condições de *start* e *stop*. A condição de *start* ocorre quando a linha SCL está em nível alto e a linha SDA vai de 1 para 0 (borda de descida). A condição de *stop* ocorre quando a linha SCL está em nível alto e a linha SDA vai de 0 para 1 (borda de subida). As condições de *start* e *stop* só podem ser geradas pelo mestre. A Figura Figura 2.2 ilustra as condições de partida e parada. [2] [21]

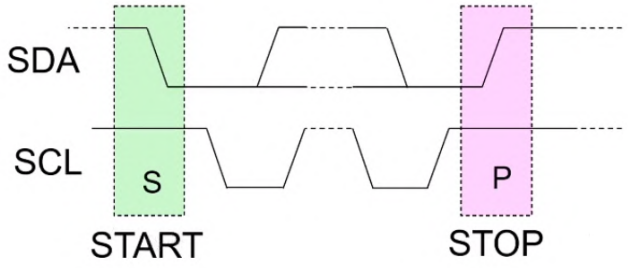

Figura 2.2: Condição de partida e parada da comunicação I²C. Fonte: [2].

Os dados são enviados em pacotes de 8 bits, sendo os 7 primeiros respectivos ao endereço do escravo e o último bit determina se aquele escravo irá ler ou escrever dados na transmissão subsequente. A cada byte transmitido, o escravo responde com um *acknowledge* (ACK), indicando que os dados foram recebidos. [2] [21]

2.1.2 UART

Diferentemente do protocolo I²C, a comunicação UART é assíncrona, podendo ser unidirecional ou bidirecional. Como a comunicação é assíncrona, os dados transmitidos devem conter um bit de partida e um bit de parada.

Na comunicação assíncrona, a linha fica em nível alto quando está inativa. Assim que se inicia uma transmissão, a linha vai para nível baixo e assim permanece até o fim da transmissão, que é sinalizado por um bit de parada que coloca a linha em nível alto novamente. [22]

Taxa de transmissão dos dados

Na comunicação UART, a velocidade de transmissão dos dados é definida pelo baud rate. O baud rate é a taxa de transmissão dos dados em bits/segundo. Ele é gerado a partir do clock do sistema e deve ser definido tanto no transmissor quanto no receptor. Tanto o transmissor quanto o receptor devem ser configurados com a mesma taxa de baud para que a comunicação ocorra corretamente. O baud rate é gerado a partir de um escalonamento do clock do sistema e, portanto, não é possível gerar de forma exata determinadas taxas de baud rate. O cálculo para a geração do baud rate é ilustrado na Equação 2.1, em que B é o baud rate, f é a frequência do clock do sistema e s e C são fatores de escalonamento que assumem valores inteiros. Geralmente o fator s é fixo e o valor C pode ser alterado. A geração do baud rate é feita combinando esses dois fatores, de forma que a taxa de transmissão fique o mais próximo possível da desejada. [21]

$$B = \frac{f}{s \cdot C} \tag{2.1}$$

Por exemplo, para gerar um baud rate de 115200 bps, com um clock de frequência 8 MHz, estando fixo o fator s = 8, o valor para C com o qual o baud rate seja o mais próximo do desejado é de C = 9. Neste caso, o baud rate gerado é de 111,11 kbps.

2.2 Sinais

Essencialmente, um sinal é um conjunto de dados ou informações. Um sinal pode variar em função do tempo ou de outras variáveis, como o espaço. Mas, para o trabalho em questão, apenas nos interessam os sinais que variam ao longo do tempo. [3]

Com relação à representação no tempo, existem sinais contínuos e discretos no tempo. Essencialmente, a principal diferença entre um sinal em tempo contínuo e um sinal em tempo discreto é sua variável t, correspondente ao tempo. Se t assume valores contínuos (ou seja, pode assumir infinitos valores), então o sinal é contínuo. Caso o sinal só exista para determinados valores de t (neste caso, portanto, a variável t é discreta), então o sinal é discreto. [3]

Função Impulso Unitário

Existem algumas funções matemáticas que são muito utilizadas na representação de sinais, como a *função degrau unitário*, a *função impulso unitário* e a *função exponencial*. Aqui, será abordada apenas a função impulso unitário, que será utilizada em demonstrações nos capítulos posteriores.

A função impulso unitário $\delta(t)$ possui duas propriedades fundamentais, mostradas nas Equações 2.2 a 2.3.

$$\delta(t) = 0 \qquad t \neq 0 \tag{2.2}$$

$$\int_{-\infty}^{\infty} \delta(t)dt = 1 \tag{2.3}$$

A função impulso assume valores nulos para qualquer $t \neq 0$. Para $t = 0$, seu valor é indefinido. O impulso unitário pode ser entendido como um sinal com o formato de um retângulo de largura muito fina e altura muito grande em torno do instante $t = 0$. A área desse retângulo é unitária, com uma largura $\epsilon \to 0$ e, consequentemente, uma altura $1/\epsilon \to \infty$. A Figura 2.3 ilustra duas formas de representação da função impulso. [3]

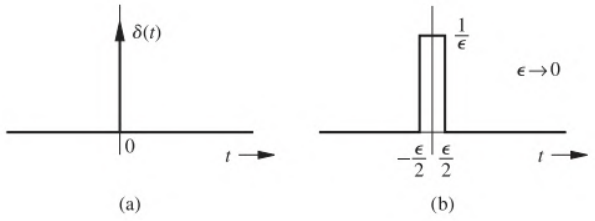

Figura 2.3: Representação da função impulso $\delta(t)$. Fonte: [3].

Sistemas

Para o escopo deste trabalho, um sistema é responsável por processar um sinal, extraindo informações sobre ele. Um sistema recebe um sinal de entrada e gera um sinal de saída. Ele pode ser constituído de componentes físicos, elétricos, mecânicos ou mesmo implementado computacionalmente. [3]

Com relação à representação no tempo, um sistema contínuo processa sinais contínuos, e um sistema discreto processa sinais discretos. Na prática, sistemas discretos são muito utilizados para processar sinais discretos resultantes da amostragem de sinais contínuos no tempo. Particularmente neste trabalho, o processamento dos sinais será em tempo discreto, implementado computacionalmente.

Na próxima seção, serão abordados os conceitos de representação de um sinal em frequência. Para este trabalho, a análise de sinais na frequência terá grande importância para definir parâmetros como a frequência de amostragem do sensor e frequências de corte em filtros.

2.2.1 Série de Fourier

Qualquer sinal periódico pode ser representado como uma soma de senoides com frequências diferentes. Essencialmente, um sinal periódico é um sinal que possui a propriedade explicitada pela Equação 2.4, para todo t. O menor valor de T_0 que satisfaz essa condição é denominado *período fundamental* de x(t). [3]

$$x(t) = x(t + T_0) \tag{2.4}$$

A seguir, será demonstrado que é possível representar um sinal periódico por um somatório de senoides. Para isso, será considerado um sinal x(t) constituído por senos e cossenos, conforme explicitado na Equação 2.5. [3]

$$x(t) = a_0 + \sum_{n=1}^{\infty} a_n cos(n\omega_0 t) + b_n sen(n\omega_0 t) \quad , \quad \omega_0 = \frac{2\pi}{T_0} \tag{2.5}$$

A frequência ω_0 é denominada *frequência fundamental*, com um *período fundamental* T_0 associado. Primeiramente, será demonstrado que esse sinal x(t) é, de fato, periódico. Para isso, a condição na Equação 2.4 tem que ser válida. [3]

$$\begin{aligned} x(t + T_0) &= a_0 + \sum_{n=1}^{\infty} a_n cos(n\omega_0(t + T_0)) + b_n sen(n\omega_0(t + T_0)) \\ &= a_0 + \sum_{n=1}^{\infty} a_n cos(n\omega_0 t + n\omega_0 T_0) + b_n sen(n\omega_0 t + n\omega_0 T_0) \end{aligned} \tag{2.6}$$

Pela Equação 2.5, tem-se que $\omega_0 T_0 = 2\pi$. Portanto, a Equação 2.6 pode ser reescrita da seguinte forma:

$$\begin{aligned} x(t + T_0) &= a_0 + \sum_{n=1}^{\infty} a_n cos(n\omega_0 t + 2\pi n) + b_n sen(n\omega_0 t + 2\pi n) \\ &= a_0 + \sum_{n=1}^{\infty} a_n cos(n\omega_0 t) + b_n sen(n\omega_0 t) \\ &= x(t) \end{aligned} \tag{2.7}$$

Portanto, conclui-se que a representação do sinal em função de componentes senoidais não altera a sua periodicidade, independentemente dos valores dos coeficientes a_0, a_n e b_n. A série exposta à direita da igualdade na Equação 2.5 é denominada *série trigonométrica de Fourier* do sinal x(t). A seguir, será demonstrado como obter as expressões equivalentes aos coeficientes a_0, a_n e b_n. [3]

Considerando a integral I definida na Equação 2.8, podemos reescrevê-la conforme mostrado na Equação 2.9, utilizando propriedades trigonométricas. [3]

$$I = \int_{T_0} cos(n\omega_0 t)cos(m\omega_0 t)dt \tag{2.8}$$

$$I = \frac{1}{2}\left[\int_{T_0} cos((n+m)\omega_0 t)dt + \int_{T_0} cos((n-m)\omega_0 t)dt\right] \tag{2.9}$$

Considerando $n,\ m$ inteiros e diferentes de zero, observa-se que a primeira integral da Equação 2.9 é nula para qualquer valor de $n,\ m$, pois é uma integral em um período T_0 e, portanto, suas áreas positivas e negativas se anulam. Já para a segunda integral, observa-se que só não será nula para $n = m$. Portanto, a Equação 2.8 pode ser reescrita da seguinte forma: [3]

$$\int_{T_0} cos(n\omega_0 t)cos(m\omega_0 t)dt = \begin{cases} 0 & n \neq m \\ \frac{T_0}{2} & m = n \neq 0 \end{cases} \tag{2.10}$$

Utilizando raciocínio análogo, pode-se demonstrar também o exposto nas Equações 2.11 a 2.12. [3]

$$\int_{T_0} sen(n\omega_0 t)sen(m\omega_0 t)dt = \begin{cases} 0 & n \neq m \\ \frac{T_0}{2} & m = n \neq 0 \end{cases} \tag{2.11}$$

$$\int_{T_0} sen(n\omega_0 t)cos(m\omega_0 t)dt = 0 \quad \forall\ n, m \tag{2.12}$$

Para determinar a_0, basta integrar ambos os lados da Equação 2.5, conforme mostrado nas Equações 2.13 a 2.15. [3]

$$\int_{T_0} x(t)dt = a_0 \int_{T_0} dt + \sum_{n=1}^{\infty}\left[a_n \int_{T_0} cos(n\omega_0 t)dt + b_n \int_{T_0} sen(n\omega_0 t)dt\right] \tag{2.13}$$

$$\int_{T_0} x(t)dt = a_0 \int_{T_0} dt = a_0 T_0 \tag{2.14}$$

$$a_0 = \frac{1}{T_0}\int_{T_0} x(t)dt \tag{2.15}$$

Para determinar a_n, multiplica-se ambos os lados da Equação 2.5 por $cos(m\omega_0 t)$ e integra-se os dois lados. Utilizando as propriedades expostas nas Equações 2.10 a 2.12, é possível obter a expressão final para a_n, conforme mostrado nas Equações 2.16 a 2.18. [3]

$$\begin{aligned} &\int_{T_0} x(t)cos(m\omega_0 t)dt = a_0 \int_{T_0} cos(m\omega_0 t)dt \\ &+ \sum_{n=1}^{\infty}\left[a_n \int_{T_0} cos(n\omega_0 t)cos(m\omega_0 t)dt + b_n \int_{T_0} sen(n\omega_0 t)cos(m\omega_0 t)dt\right] \end{aligned} \tag{2.16}$$

$$\int_{T_0} x(t)cos(m\omega_0 t)dt = \frac{a_m T_0}{2} \tag{2.17}$$

$$a_m = \frac{2}{T_0}\int_{T_0} x(t)cos(m\omega_0 t)dt \tag{2.18}$$

De forma análoga, é possível obter a expressão para b_n multiplicando ambos os lados da Equação 2.5 por $sen(n\omega_0 t)$ e os integrando, chegando à expressão explicitada na Equação 2.19 [3]

$$b_m = \frac{2}{T_0}\int_{T_0} x(t)sen(m\omega_0 t)dt \tag{2.19}$$

Com isso, concluímos que um sinal periódico, de fato, pode ser representado por um somatório de senoides, cuja expressão é mostrada na Equação 2.5. Essa expressão é denominada *série trigonométrica de Fourier*. Mas há também outra forma de representar essa série, em termos de exponenciais. Neste caso, a série recebe o nome de *série exponencial de Fourier*. [3]

A expressão associada à série exponencial de Fourier está explicitada na Equação 2.20. É possível chegar a essa expressão aplicando identidades trigonométricas na expressão associada à série trigonométrica de Fourier, uma vez que é possível representar senoides e cossenoides em função de exponenciais. Porém, aqui a demonstração será feita de forma independente da série trigonométrica de Fourier. [3]

$$x(t) = \sum_{n=-\infty}^{\infty} D_n e^{jn\omega_0 t} \tag{2.20}$$

Para obter a expressão associada aos coeficientes D_n, basta multiplicar ambos os lados da Equação 2.20 por $e^{-jm\omega_0 t}$ e integrar em um período T_0, conforme mostrado na Equação 2.21. [3]

$$\int_{T_0} x(t)e^{-jm\omega_0 t}dt = \sum_{n=-\infty}^{\infty} D_n \int_{T_0} e^{j(n-m)\omega_0 t}dt \tag{2.21}$$

A expressão dentro da integral depois da igualdade na Equação 2.21 pode ser reescrita como mostrado na Equação 2.22. Analisando essa segunda expressão, observa-se que ela é nula para qualquer $n \neq m$, uma vez que são integrais de senoides em um período, cujas áreas se anulam. Porém, para $n = m$, a expressão é equivalente à T_0. Então, pode-se reescrever a Equação 2.21 conforme mostrado na Equação 2.23. [3]

$$\int_{T_0} e^{j(n-m)\omega_0 t}dt = \int_{T_0} cos((n-m)\omega_0 t)dt + j\int_{T_0} sen((n-m)\omega_0 t)dt \tag{2.22}$$

$$D_m T_0 = \int_{T_0} x(t) e^{-jm\omega_0 t} dt \tag{2.23}$$

Com isso, a expressão final para os coeficientes D_m é mostrada na Equação 2.24. Essa é uma forma alternativa e mais compacta para representar sinais periódicos em função de suas componentes de frequência. [3]

$$D_m = \frac{1}{T_0} \int_{T_0} x(t) e^{-jm\omega_0 t} dt \tag{2.24}$$

2.2.2 Transformada de Fourier

Conforme apresentado no Capítulo 2.2.1, a série de Fourier é empregada para a representação de sinais periódicos. Entretanto, essa abordagem não é adequada para sinais não-periódicos. Nesse contexto, surge a transformada de Fourier, utilizada para calcular o espectro de sinais não-periódicos.

Assumir que um sinal é não-periódico é o mesmo que assumir que esse sinal possui um período T_0, tal que $T_0 \to \infty$. Portanto, considerando um sinal não-periódico x(t) e um sinal periódico $x_{T_0}(t)$ formado a partir da repetição de x(t) em intervalos de tempo T_0, ilustrados nas Figuras 2.4 a 2.5, temos a relação expressa pela Equação 2.25 [3]

$$\lim_{T_0 \to \infty} x_{T_0}(t) = x(t) \tag{2.25}$$

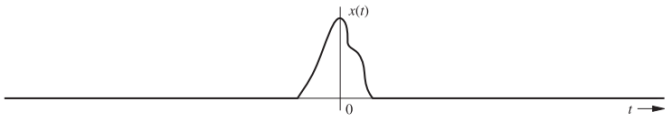

Figura 2.4: Sinal x(t) não-periódico. Fonte: [3].

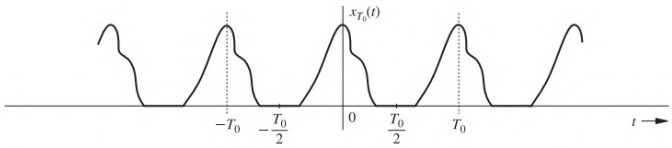

Figura 2.5: Sinal $x_{T_0}(t)$ periódico. Fonte: [3].

Como $x_{T_0}(t)$ é periódico, pode ser representado por sua série de Fourier, conforme mostrado nas Equações 2.26 a 2.27

$$x_{T_0}(t) = \sum_{n=-\infty}^{\infty} D_n e^{jn\omega_0 t} \tag{2.26}$$

$$D_n = \frac{1}{T_0}\int_{-T_0/2}^{T_0/2} x_{T_0}(t)e^{-jn\omega_0 t}dt \quad , \quad \omega_0 = \frac{2\pi}{T_0} \tag{2.27}$$

Pelas Figuras 2.4 a 2.5, observa-se que integrar o sinal $x_{T_0}(t)$ de $-T_0/2$ a $T_0/2$ é equivalente a integrar o sinal x(t) de $-\infty$ a ∞. Portanto, pode-se reescrever:

$$D_n = \frac{1}{T_0}\int_{-\infty}^{\infty} x(t)e^{-jn\omega_0 t}dt \tag{2.28}$$

Definindo uma função auxiliar X(ω), temos que:

$$X(\omega) = \int_{-\infty}^{\infty} x(t)e^{-j\omega t}dt \tag{2.29}$$

$$D_n = \frac{1}{T_0}X(n\omega_0) \tag{2.30}$$

E, substituindo de volta na Equação 2.26, temos:

$$x_{T_0}(t) = \sum_{n=-\infty}^{\infty} \frac{X(n\omega_0)}{T_0}e^{jn\omega_0 t} \tag{2.31}$$

A partir da Equação 2.30, concluímos que os coeficientes da série exponencial de Fourier referente ao sinal original podem ser expressos em função de amostras da representação espectral X(ω) uniformemente espaçadas. É fácil observar que, conforme aumentamos T_0, a frequência ω_0 diminui, até um limite em que ela assume valores infinitesimais. Para fins de representação, vamos denotar essas frequências infinitesimais por $\Delta\omega$, conforme explicitado na Equação 2.32. Com isso, podemos reescrever a Equação 2.31 da seguinte forma: [3]

$$\Delta\omega = \omega_0 = \frac{2\pi}{T_0} \tag{2.32}$$

$$x_{T_0}(t) = \sum_{n=-\infty}^{\infty} \left[\frac{X(n\Delta\omega)\Delta\omega}{2\pi}\right] e^{jn\Delta\omega t} \tag{2.33}$$

Retornando à Equação 2.25, temos:

$$\begin{aligned} x(t) &= \lim_{T_0\to\infty} x_{T_0}(t) \\ &= \lim_{\Delta\omega\to 0} \frac{1}{2\pi}\sum_{n=-\infty}^{\infty} X(n\Delta\omega)e^{jn\Delta\omega t}\Delta\omega \end{aligned} \tag{2.34}$$

Por fim, a partir da definição de soma de Riemann [23], pode-se demonstrar que:

$$x(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} X(\omega)e^{j\omega t}d\omega \tag{2.35}$$

A integral na Equação 2.35 é denominada *integral de Fourier*, e é a forma de representar sinais não periódicos no domínio da frequência. A função X(ω), explicitada na Equação 2.29, é denominada *transformada de Fourier* de x(t). [3]

Propriedades da transformada de Fourier

A transformada de Fourier apresenta diversas propriedades relevantes, as quais podem ser aprofundadas com maior detalhe em [3]. Aqui, será tratada apenas sobre a *propriedade de deslocamento em frequência*, que será abordada em seções seguintes.

A propriedade de deslocamento na frequência afirma que:

Se

$$x(t) \Leftrightarrow X(\omega)$$

então,

$$x(t)e^{j\omega_0 t} \Leftrightarrow X(\omega - \omega_0) \tag{2.36}$$

A demonstração dessa propriedade é explicitada na Equação 2.37. [3]

$$\mathcal{F}\left[x(t)e^{j\omega_0 t}\right] = \int_{-\infty}^{\infty} x(t)e^{j\omega_0 t}e^{-j\omega t}dt = \int_{-\infty}^{\infty} x(t)e^{-j(\omega-\omega_0)t}dt = X(\omega - \omega_0) \tag{2.37}$$

A partir desta propriedade, observa-se que:

$$x(t)cos(\omega_0 t) = \frac{1}{2}\left[x(t)e^{j\omega_0 t} + x(t)e^{-j\omega_0 t}\right] \tag{2.38}$$

$$x(t)cos(\omega_0 t) \Leftrightarrow \frac{1}{2}\left[X(\omega - \omega_0) + X(\omega + \omega_0)\right] \tag{2.39}$$

A propriedade explicitada na Equação 2.39 será utilizada para demonstrações posteriores.

2.2.3 Amostragem de sinais

É possível processar um sinal contínuo utilizando um sistema que opere em tempo discreto, a partir da amostragem desse sinal contínuo. Para isso, é fundamental que a taxa de amostragem seja suficientemente alta, assegurando assim a reconstrução do sinal original sem erro ou com um erro dentro de uma tolerância pré-definida. [3]

A seguir será demonstrado que um sinal real com espectro limitado em faixa a B Hz (X(ω) = 0 para $|\omega| > 2\pi B$) pode ser reconstruído exatamente para uma frequência de

amostragem $f_s > 2B$. Para ilustrar, será considerado um sinal fictício, cujo comportamento no tempo é ilustrado na Figura 2.6 e seu espectro é ilustrado na Figura 2.7. [3]

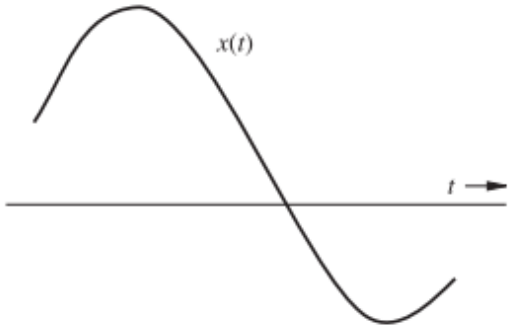

Figura 2.6: Sinal x(t) contínuo no tempo. Fonte: [3].

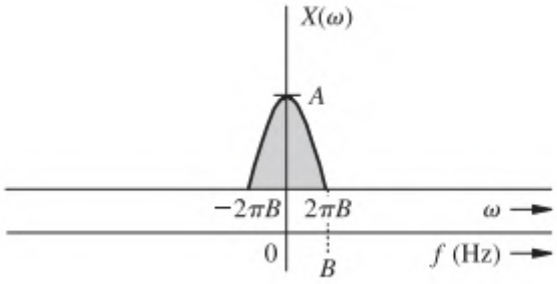

Figura 2.7: Espectro do sinal x(t). Fonte: [3].

Pela Figura 2.7, observa-se que o sinal em questão é limitado em frequência em B Hz. Para amostrar esse sinal, deve-se multiplicá-lo por um trem de impulsos igualmente espaçados no tempo por um período T_s, equivalente a $1/f_s$, em que f_s é a frequência de amostragem do sinal. A representação matemática do sinal amostrado é descrita na Equação 2.40, e o aspecto do sinal é ilustrado na Figura 2.8. Como o trem de impulso $\delta_T(t)$ é um sinal periódico, pode ser representado por sua série trigonométrica de Fourier, conforme a Equação 2.41. [3]

$$\overline{x}(t) = x(t)\delta_T(t) = \sum_n x(nT_s)\delta(t - nT_s) \tag{2.40}$$

$$\delta_T(t) = \frac{1}{T_s}\left(1 + 2cos(\omega_s t) + 2cos(2\omega_s t) + 2cos(3\omega_s t) + ...\right) \quad , \quad \omega_s = \frac{2\pi}{T_s} \tag{2.41}$$

Portanto,

$$\overline{x}(t) = x(t)\delta_T(t) = \frac{1}{T_s}\left(x(t) + 2x(t)cos(\omega_s t) + 2x(t)cos(2\omega_s t) + 2x(t)cos(3\omega_s t) + ...\right) \tag{2.42}$$

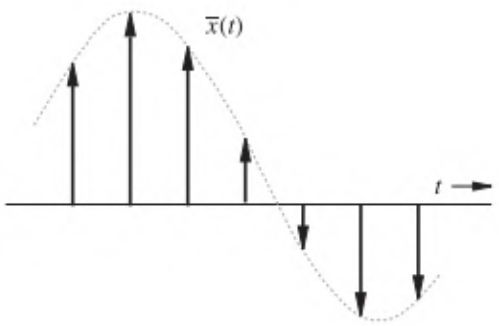

Figura 2.8: Sinal x(t) amostrado. Fonte: [3].

Calculando a transformada de Fourier de $\overline{x}(t)$, utilizando a propriedade descrita na Equação 2.39, obtém-se a expressão descrita na Equação 2.43.

$$\overline{X}(\omega) = \mathcal{F}\left\{\overline{x}(t)\right\} = \frac{1}{T_s}\sum_{n=-\infty}^{\infty} X(\omega - n\omega_s) \tag{2.43}$$

Portanto, observa-se que a transformada de Fourier do sinal amostrado é equivalente à transformada de Fourier do sinal original repetida periodicamente, com período $\omega_s = 2\pi/T_s$, como ilustrado na Figura 2.9. Conclui-se que é possível reconstruir o sinal original x(t) a partir de sua transformada X(ω), desde que não haja repetições sobrepostas de X(ω), e essa condição é garantida se:

$$f_s > 2B \tag{2.44}$$

A frequência descrita na Equação 2.44 corresponde à taxa de Nyquist, que é a menor taxa de amostragem necessária para preservar a informação do sinal original. [3]

2.2.4 Transformada Discreta de Fourier (TDF)

Para realizar o cálculo numérico da transformada de Fourier de um sinal é necessário que ele seja amostrado, uma vez que um computador digital só consegue fazer cálculos com valores discretos. Conforme explicado no Capítulo 2.2.3, o espectro de um sinal amostrado no tempo é equivalente ao espectro do sinal original repetido periodicamente.

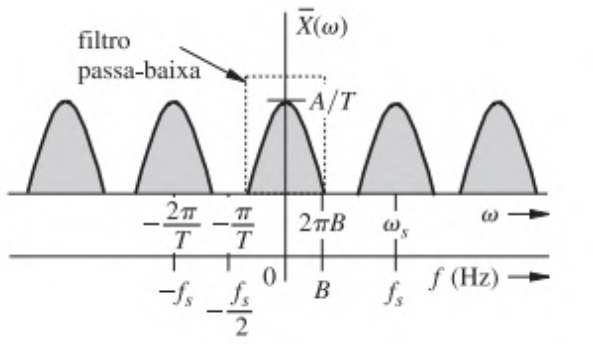

Figura 2.9: Espectro do sinal amostrado. Fonte: [3].

Para realizar o cálculo da TDF, deve-se relacionar as amostras do sinal temporal e de seu espectro. [3]

Para realizar a demonstração da expressão equivalente à transformada discreta de Fourier de um sinal, será utilizado raciocínio análogo ao da demonstração da transformada de Fourier contínua, no Capítulo 2.2.2.

Para esta demonstração, será considerado um sinal amostrado não-periódico x(t), cujo comportamento é ilustrado na Figura 2.10. Assim como foi feito na demonstração no Capítulo 2.2.2, será considerado que esse sinal é periódico, com um período $T_0 \to \infty$.

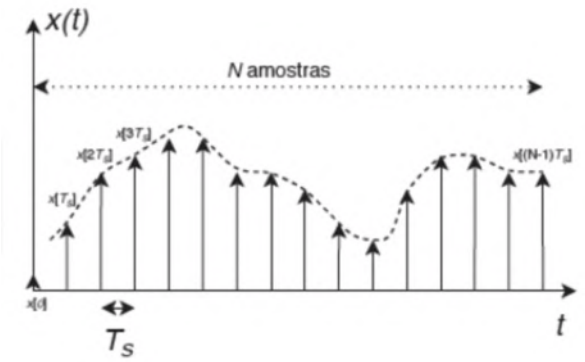

Figura 2.10: Sinal amostrado x(t). Fonte: [4].

Pela definição de série de Fourier, pode-se representar o k-ésimo coeficiente de Fourier deste sinal conforme mostrado na Equação 2.45.

$$c_k = \frac{1}{T_0} \int_{T_0} x(t) e^{-j\frac{2\pi}{T_0}kt} dt \tag{2.45}$$

Porém, como o sinal em questão é um sinal discreto, podemos reescrever a Equação 2.45 como um somatório para todas as N amostras, em que $T_0 = NT_s$, $t = nT_s$ e $dt = T_s$, conforme mostrado nas Equações 2.46 a 2.48. [4]

$$c_k = \frac{1}{NT_s} \sum_{n=0}^{N-1} x[n] e^{-j\frac{2\pi}{NT_s} knT_s} T_s \tag{2.46}$$

$$c_k = \frac{1}{N} \sum_{n=0}^{N-1} x[n] e^{-j\frac{2\pi}{N} kn} = \frac{X(k)}{N} \tag{2.47}$$

$$X(k) = \sum_{n=0}^{N-1} x[n] e^{-j\frac{2\pi}{N} kn} \tag{2.48}$$

A expressão mostrada na Equação 2.48 corresponde à *transformada discreta de Fourier* do sinal $x[n]$.

FFT (*Fast Fourier Transform*)

A transformada rápida de Fourier (FFT, do inglês *Fast Fourier Transform*) é uma forma alternativa de calcular a TDF. Essencialmente, a FFT e a TDF são iguais, porém a FFT é implementada de uma forma que o custo computacional para realizar seu cálculo é menor.

A TDF original tem um custo computacional de N^2 operações aritméticas, sendo N o número de amostras. Com a FFT, o custo é reduzido para $Nlog(N)$. [3]

Devido à transformada de Fourier ser linear, pode-se obter a transformada de um sinal a partir da soma de transformadas de subdivisões desse sinal. O funcionamento da FFT consiste em dividir o sinal em partes menores, e realizar as operações sobre essas partes. Esse processo é chamado de decimação. Essencialmente, o sinal original é dividido em dois sinais, um contendo as amostras pares e o outro contendo as ímpares. Então, cada parte é dividida pela metade de forma contínua, até que a menor parte do sinal possua apenas duas amostras. [3] [24]

Quando cumprem-se todos os estágios, pode-se demonstrar que o custo computacional para calcular a FFT é de $Nlog(N)$ operações. Neste trabalho, não será feita uma abordagem aprofundada sobre a teoria por trás do cálculo da FFT, mas tais conceitos podem ser vistos em [25].

2.3 IA

2.3.1 Redes Neurais Artificiais

Redes neurais artificiais são modelos matemáticos concebidos como uma tentativa de mapear os neurônios do cérebro humano. O *perceptron* é a unidade mais básica de uma rede neural artificial. A estrutura de um *perceptron* é ilustrada na Figura 2.11.

Desenvolvido em 1958 pelo cientista Frank Rosenblatt [26], o modelo *perceptron* é um modelo matemático cujo funcionamento consiste em receber entradas, multiplicar cada entrada por um peso correspondente e realizar o somatório de todos os termos. Então, o resultado desse somatório é passado por uma função de ativação, que retorna um valor que pode variar entre $[0, 1]$ ou $[-1, 1]$, correspondente à saída final daquele neurônio. Os cálculos realizados pelo modelo de *perceptron* são mostrados nas Equações 2.49 a 2.50, em que o subíndice k indica que a saída está relacionada a um neurônio apenas, w_{kj} corresponde ao peso associado a uma entrada, x_j corresponde à entrada e $g(u_k)$ é a função de ativação. Na Equação 2.49, o primeiro termo do somatório é um valor constante denominado *bias*. [8]

Uma função de ativação limita a amplitude do sinal de saída do neurônio. Funções de ativação comuns incluem a função degrau, função sigmoide e a função tangente hiperbólica. As Equações 2.51 a 2.53 mostram as expressões matemáticas correspondentes a essas três funções, respectivamente. As Figuras 2.12 a 2.14 ilustram o comportamento de cada uma dessas funções.

$$u_k = \sum_{j=0}^{m} w_{kj} \cdot x_j \tag{2.49}$$

$$y_k = g(u_k) \tag{2.50}$$

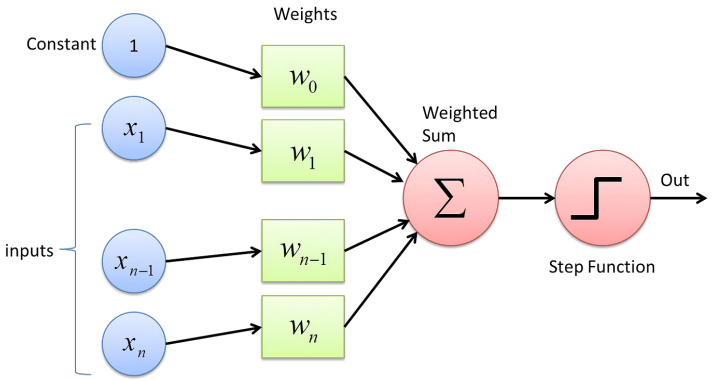

Figura 2.11: Estrutura geral de um *perceptron*. Fonte: [5].

$$f(x) = \begin{cases} 1 & se \quad x \geq 0 \\ 0 & se \quad x < 0 \end{cases} \tag{2.51}$$

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.52}$$

$$f(x) = tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.53}$$

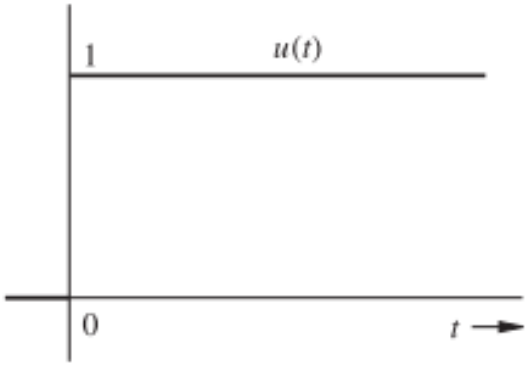

Figura 2.12: Função degrau. Fonte: [3].

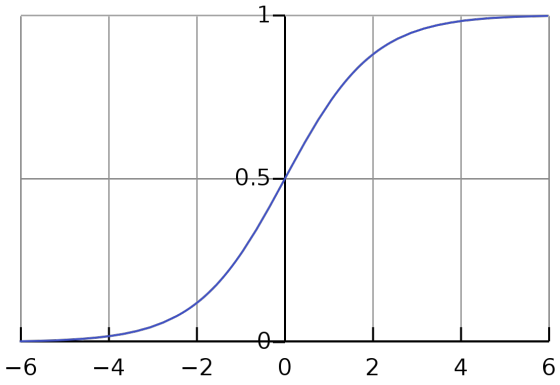

Figura 2.13: Função sigmoide. Fonte: [6].

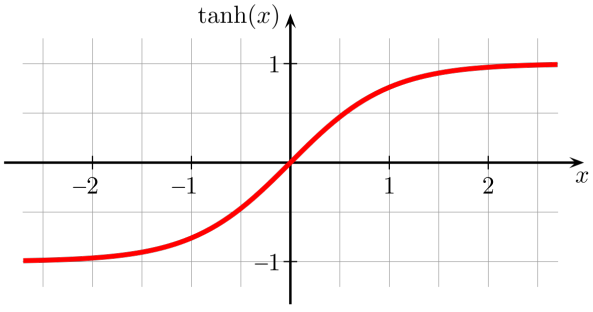

Figura 2.14: Função tangente hiperbólica. Fonte: [7].

A estrutura mostrada na Figura 2.11 corresponde a um *perceptron* de camada única. Porém, é possível interligar vários *perceptrons* em camadas. De forma geral, um *perceptron* multicamadas é composto por uma camada de entrada, camadas intermediárias (ou camadas ocultas) e uma camada de saída.

A quantidade de camadas intermediárias é arbitrária, sendo nessas camadas onde ocorre a maior parte do processamento. Adicionar muitas camadas ocultas pode fazer com que a rede aprenda mais detalhes sobre as entradas fornecidas. Porém, pode acabar ocorrendo o fenômeno denominado *overfitting*, que é quando a rede assimila de maneira excessiva as características do conjunto de dados com que foi treinada, tendo um grau de especialização muito alto para aqueles dados, mas com uma capacidade baixa de generalização. Por outro lado, adicionar camadas ocultas com poucos neurônios ou poucas camadas pode causar o fenômeno do *underfitting*, que é o oposto do *overfitting*. Nesse caso, a rede não consegue efetivamente aprender as características dos dados. Portanto, deve-se ter um balanço na estrutura da rede, de forma a evitar esses dois fenômenos. No artigo publicado por Saurabh Karsoliya [27], são discutidas estratégias para determinar a quantidade ideal de camadas ocultas e neurônios, sendo concluído que, para a maior parte dos casos, uma a duas camadas ocultas são suficientes.

Existem diferentes tipos de arquiteturas de redes, mas, de forma geral, pode-se agrupar as arquiteturas em duas classes principais: redes *feedforward* e redes recorrentes ou realimentadas [8]. Neste trabalho, o foco será direcionado para redes *feedforward*.

Em uma rede feedforward, o fluxo de informações é unidirecional. Os neurônios de uma camada se comunicam apenas com os neurônios da camada subsequente. Camadas com todos os neurônios conectados são denominadas camadas *fully connected* (totalmente conectadas). Um exemplo de estrutura de rede *feedforward* totalmente conectada é mostrado na Figura 2.15. A principal diferença de redes *feedforward* para redes recorrentes é que em redes recorrentes existe um *loop* de realimentação, em que as saídas são comparadas com as entradas por meio de um mecanismo de atraso. Já nas redes *feedforward*, a informação é passada adiante de camada em camada sem que haja uma comparação. [8]

Backpropagation

O treinamento de redes *feedforward* é realizado por um algoritmo denominado *backpropagation*. Esse algoritmo utiliza aprendizado supervisionado, que consiste em fornecer para a rede dados de entrada rotulados com as saídas desejadas. De forma sucinta, o algoritmo de *backpropagation* calcula o erro total da rede a partir das diferenças entre as saídas previstas pela rede e as saídas originais, visando reduzir esse erro a um valor mínimo. [8]

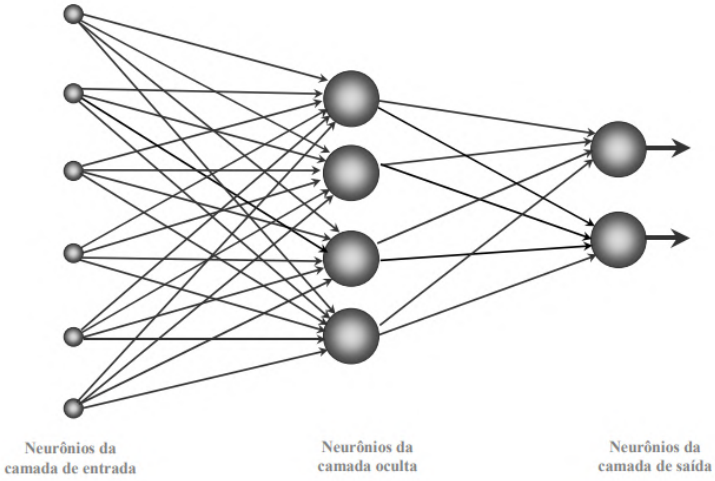

Figura 2.15: Rede *Feedforward* totalmente conectada. Fonte: [8].

Considerando um modelo *perceptron*, cuja equação de saída é dada pela Equação 2.54, em que os elementos θ_i são os parâmetros da rede, tem-se que a função de custo dessa rede é dada pela Equação 2.55. A função de custo, essencialmente, representa o erro total da rede. Deseja-se encontrar valores para θ_i que minimizem essa função. [9]

$$h_\theta(x) = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + ... + \theta_n x_n \tag{2.54}$$

$$J(\theta) = \frac{1}{2} \sum_{i=1}^{m} \left(h_\theta \left(x^{(i)} \right) - y^{(i)} \right)^2 \tag{2.55}$$

Na Equação 2.55, m corresponde à quantidade de entradas da rede e o sobrescrito i corresponde ao índice da entrada, e não a uma potência. Para fins de visualização A Figura 2.16 mostra um exemplo de função de custo para duas variáveis (θ_0 e θ_1).

A minimização da função $J(\theta)$ é feita utilizando um algoritmo denominado gradiente descendente. A ideia desse algoritmo é, por meio de cálculos de derivadas, encontrar o valor mínimo para uma função determinada, conforme será demonstrado adiante.

Esse algoritmo consiste em, primeiramente, inicializar os parâmetros θ_i com valores aleatórios. Então, esses parâmetros são atualizados conforme mostrado na Equação 2.56, em que a designação ":="significa que, a cada iteração, aquele valor é atribuído à variável θ_j. Além disso, o parâmetro α corresponde ao *learning rate*, que é a taxa de aprendizado do algoritmo, que cadencia o tamanho de cada passo. Por fim, vale ressaltar que foi feita uma atribuição $\theta = \vec{\theta} = [\theta_0 \quad \theta_1 \quad \theta_2 \quad ... \quad \theta_n]$. [9]

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta) \tag{2.56}$$

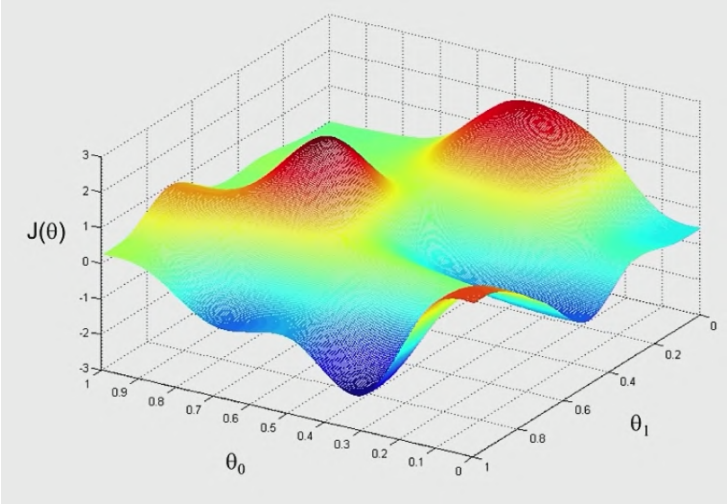

Figura 2.16: Função de custo para duas variáveis. Fonte: [9].

As Equações 2.57 a 2.59 demonstram como calcular a derivada parcial presente na Equação 2.56, utilizando a regra da cadeia. Para fins de simplificação, a demonstração foi feita para apenas uma entrada, sendo depois generalizada para m entradas. [9]

$$\frac{\partial}{\partial\theta_j}J(\theta) = \frac{\partial}{\partial\theta_j}\frac{1}{2}\left(h_\theta(x) - y\right)^2 \tag{2.57}$$

$$\frac{\partial}{\partial\theta_j}J(\theta) = 2\cdot\frac{1}{2}\left(h_\theta(x) - y\right)\cdot\frac{\partial}{\partial\theta_j}(h_\theta(x) - y) \tag{2.58}$$

$$\frac{\partial}{\partial\theta_j}J(\theta) = \left(h_\theta(x) - y\right)\cdot\frac{\partial}{\partial\theta_j}(\theta_0 x_0 + \theta_1 x_1 + ... + \theta_n x_n - y) \tag{2.59}$$

Na Equação 2.59, observa-se que a derivada será nula para todos os índices diferentes de j. E quando isso ocorrer, será igual a x_j. Portanto, pode-se reescrever a Equação 2.59 conforme mostrado na Equação 2.60.

$$\frac{\partial}{\partial\theta_j}J(\theta) = \left(h_\theta(x) - y\right)\cdot x_j \tag{2.60}$$

Por fim, uma vez que a derivada de uma soma é a soma das derivadas, pode-se generalizar a Equação 2.60 para m entradas, conforme mostrado na Equação 2.61. Com isso, a expressão final correspondente à atualização dos parâmetros θ_i a cada iteração é mostrada na Equação 2.62. [9]

$$\frac{\partial}{\partial\theta_j}J(\theta) = \sum_{i=1}^{m}\left(h_\theta(x)^{(i)} - y^{(i)}\right)\cdot x_j^{(i)} \tag{2.61}$$

$$\theta_j := \theta_j - \alpha \sum_{i=1}^{m} \left(h_\theta(x)^{(i)} - y^{(i)} \right) \cdot x_j^{(i)} \tag{2.62}$$

Esse algoritmo se repete até que o treinamento da rede alcance a convergência.

Esse método é uma das formas de calcular o algoritmo de *backpropagation*. O algoritmo em questão é baseado no cálculo do erro quadrático médio, que tem um desempenho satisfatório para problemas envolvendo funções de predição lineares. Mas para problemas de classificação binária, por exemplo, o erro quadrático médio já não é tão eficiente. Para problemas de classificação binária, usualmente é utilizada a função de ativação sigmoide, que é não linear. O cálculo do erro quadrático médio nesse tipo de função resulta em uma função com vários mínimos locais, o que pode atrapalhar a convergência do treinamento. Para esse tipo de problema, usa-se geralmente outra função de perda, denominada entropia cruzada. A função de perda de entropia cruzada L pode ser representada pela expressão mostrada na Equação 2.63, em que y é o valor da saída original e $\hat{y}$ é o valor da saída prevista pelo modelo. [28]

$$L = -\left[y log(\hat{y}) + (1 - y) log(1 - \hat{y}) \right] \tag{2.63}$$

E, nesse caso, a função de custo é dada pela Equação 2.64. [29]

$$J = \frac{1}{m} \sum_{i=1}^{m} - \left[y^{(i)} log(\hat{y}^{(i)}) + (1 - y^{(i)}) log(1 - \hat{y}^{(i)}) \right] \tag{2.64}$$

Neste trabalho, não será demonstrado como calcular os pesos para este caso, mas o procedimento é análogo ao procedimento anterior, sendo necessário calcular as derivadas parciais com relação aos pesos, conforme mostrado na Equação 2.56. Essa demonstração pode ser vista em [30].

2.3.2 Redes Neurais Convolucionais

Em uma rede neural artificial, as entradas fornecidas são unidimensionais. Isso ocasiona um desempenho menos eficiente na classificação de categorias distintas de entradas, tais como imagens de alta resolução. Nesse tipo de classificação, seria necessário converter a matriz de pixels em um vetor unidimensional, e então fornecer como entrada para a rede. E, a depender da quantidade de imagens, o custo computacional para realizar o treinamento da rede poderia aumentar substancialmente.

Nesse contexto, surgem as redes neurais convolucionais. O primeiro modelo de rede neural convolucional foi desenvolvido em 1988 pelo cientista da computação francês Yann LeCun. O modelo foi denominado *LeNet*, e tinha como principal objetivo realizar o reconhecimento de caracteres. As redes neurais convolucionais surgem com ênfase predo-

minante em tarefas de classificação de imagens, mas, ao longo do tempo, suas aplicações foram ampliadas para diversas áreas.

Essencialmente, uma imagem é um conjunto de pixels. E cada pixel pode assumir um valor entre 0 e 255, dependendo da sua tonalidade, seguindo o padrão RGB. Portanto, para fins de interpretação de uma máquina, uma imagem é uma matriz de números. Uma rede neural convolucional é composta por filtros que percorrem essa matriz de valores, multiplicando cada valor por um coeficiente do filtro. Os coeficientes desses filtros não precisam ser explicitados. A própria rede irá determinar os valores desses coeficientes durante o treinamento. Esses filtros irão exercer uma função de identificadores de *features* da classe em questão. *Features*, essencialmente, são características que compõem a classe. Por exemplo, em uma rede que identifica seres humanos, os *features* englobariam as características corporais do ser humano, como olhos, boca ou nariz. A interpolação desses filtros com a imagem original gera uma representação denominada *feature map*, que é basicamente um mapa de ativação correspondente àquela camada, que ilustra quais características da classe a rede aprendeu até então. [11]

Esse processo de multiplicação dos coeficientes do filtro pela matriz de entrada é denominado convolução. O filtro é uma janela deslizante de tamanho fixo, que percorre a matriz de entrada. Matematicamente, a operação de convolução em duas dimensões pode ser descrita pela Equação 2.65, em que $I_{i,j}$ correspondem aos valores da matriz de entrada, $K_{i,j}$ correspondem aos coeficientes do filtro, m é o número de linhas da matriz de entrada e n é o número de colunas da matriz de entrada. No caso da convolução em imagens, de forma geral, ela ocorre em três dimensões, sendo essa última dimensão composta pelos canais RGB da imagem. É possível que uma imagem tenha mais de 3 canais, como imagens de satélites, mas para a grande maioria das imagens são utilizados apenas os 3 canais RGB. [10]

$$S_{i,j} = \sum_{a=-\frac{m}{2}}^{\frac{m}{2}} \sum_{b=-\frac{n}{2}}^{\frac{n}{2}} I_{i-a,j-b} \cdot K_{\frac{m}{2}+a,\frac{n}{2}+b} \tag{2.65}$$

Além disso, há outros dois parâmetros que devem ser explicitados no treinamento de uma rede convolucional: *stride* e *padding*. *Stride* é o tamanho do passo com que o filtro percorre a entrada. *Padding* consiste em completar a entrada com zeros, para que o tamanho da saída seja o mesmo da entrada após a realização da convolução. As expressões para calcular a dimensão da saída de uma operação de convolução são mostradas nas Equações 2.66 a 2.67, em que W_2 é a largura da matriz de saída, W_1 é a largura da matriz de entrada, H_2 é a altura da matriz de saída, H_1 é a altura da matriz de entrada, F é o tamanho do filtro, P é a quantidade de zeros adicionados no *padding* e S é o tamanho do *stride*. A Figura 2.17 ilustra essas dimensões. A dimensão K mostrada na Figura 2.17

corresponde à quantidade de filtros. [10]

$$W_2 = \frac{W_1 - F + 2P}{S} + 1 \tag{2.66}$$

$$H_2 = \frac{H_1 - F + 2P}{S} + 1 \tag{2.67}$$

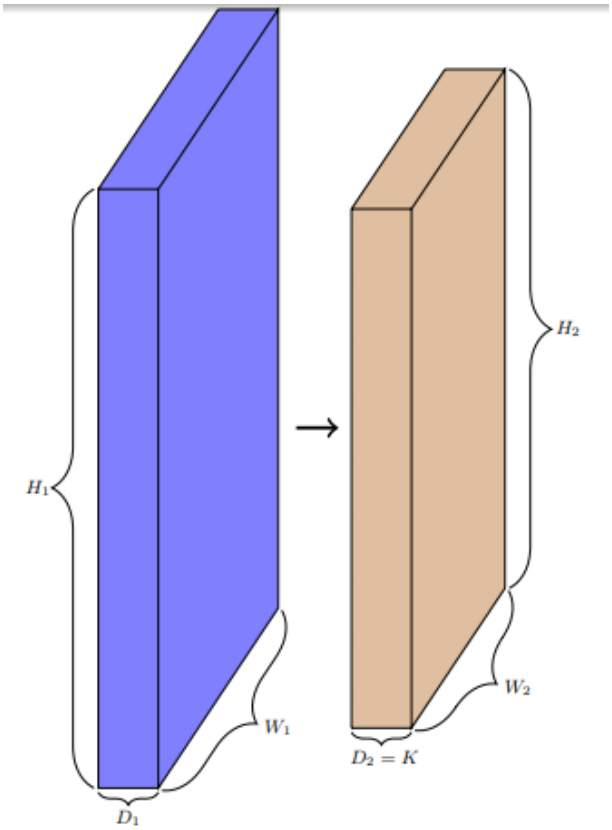

Figura 2.17: Dimensões em uma operação de convolução. Fonte: [10].

De forma geral, a estrutura de uma rede neural convolucional é composta por camadas convolucionais intercaladas por camadas de *pooling*, com uma rede totalmente conectada no final.

A camada de *pooling* tem como objetivo reduzir a dimensão dos *feature maps*, a partir de operações matemáticas em janelas fixas. Os tipos de *pooling* mais utilizados são o *Max Pooling* e o *Average Pooling*. O processo de *Max Pooling* consiste em percorrer a matriz de *feature maps*, selecionando o maior valor dentro de cada janela fixa pré-definida. E então gera uma saída, em que essas janelas serão substituídas pelo respectivo valor selecionado. A Figura 2.18 ilustra o processo de *Max Pooling* para uma janela de tamanho 2x2, com um *stride* $= (1, 1)$. O processo de *Average Pooling* é análogo, porém, neste caso, em vez de selecionar o maior valor presente na janela, é calculada a média entre todos os valores

da janela. A Figura 2.19 ilustra esse processo para uma janela de tamanho 2x2, com um *stride* $= (2, 2)$. O *pooling* reduz a dimensão dos dados e, consequentemente, o custo computacional. Com isso, reduz a probabilidade de que haja *overfitting*, uma vez que há menos parâmetros. [11]

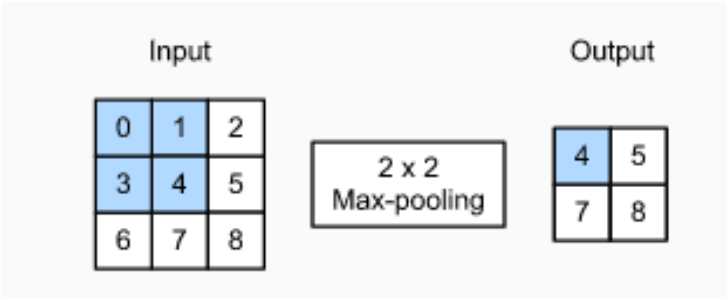

Figura 2.18: *Max Pooling* para uma janela 2x2 com *stride* $= (1, 1)$. Fonte: [11].

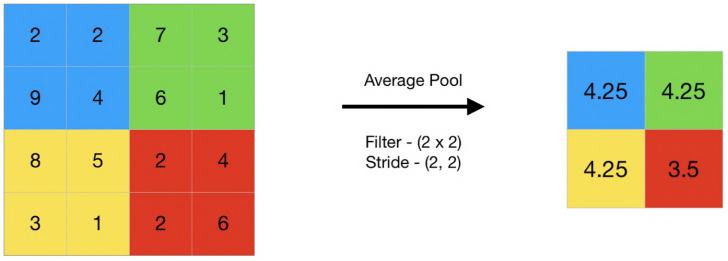

Figura 2.19: *Average Pooling* para uma janela 2x2 com *stride* $= (2, 2)$. Fonte: [12].

A rede totalmente conectada no final possui a estrutura de uma rede neural artificial. Antes de avançar para as camadas totalmente conectadas, é necessário submeter a saída das camadas convolucionais a um processo denominado *flattening*. O *flattening* consiste em, basicamente, converter todos os dados para um vetor unidimensional, uma vez que esse é o formato das entradas de redes neurais artificiais.

2.3.3 Deep Learning

O *Deep Learning* é uma subcategoria de *Machine Learning* voltada para a análise de redes neurais constituídas por um número maior de camadas. Não há um consenso no meio acadêmico com relação à quantidade de camadas necessárias para que uma rede seja caracterizada como uma rede de aprendizado profundo. Mas, como uma regra geral, considera-se que redes neurais compostas por duas ou mais camadas ocultas já podem ser inseridas dentro do contexto de *Deep Learning*. Neste trabalho, não será feita uma

abordagem aprofundada sobre a teoria por trás de *Deep Learning*, mas tais conceitos podem ser vistos em [31].

ResNet

Intuitivamente, pode-se pensar que adicionar mais camadas a uma rede irá melhorar seu desempenho, uma vez que a rede torna-se capaz de identificar características mais específicas de cada classe. Porém, estudos mostram que aumentar demais o número de camadas pode causar um aumento no erro de treinamento e teste. [13]

Nesse contexto, surgem as ResNets (abreviatura para redes residuais, em inglês). As redes residuais surgem com o objetivo de solucionar o problema do desaparecimento do gradiente. Esse problema ocorre quando o vetor gradiente assume valores muito pequenos. Logo, ao realizar a atualização dos pesos durante o *backpropagation*, observa-se uma variação mínima neles, o que prejudica a convergência do treinamento. Geralmente, esse problema do desaparecimento do gradiente é observado nas camadas iniciais da rede.

A arquitetura de uma ResNet é composta por uma série de blocos residuais, ao invés de uma série de camadas independentes. Dentro de cada bloco residual, a entrada é passada através de uma conexão de salto direta para a saída. Essa conexão pula uma ou mais camadas intermediárias. A saída do bloco residual é a soma do resíduo com a entrada original, o que permite que a rede se concentre em aprender apenas a parte nova dos parâmetros. A arquitetura ResNet é consideravelmente profunda, podendo atingir centenas de camadas, sem exigir um custo computacional muito elevado. Isso se deve ao fato de a ResNet desconsiderar algumas camadas durante o treinamento. A Figura 2.20 ilustra a estrutura de um bloco residual. [13]

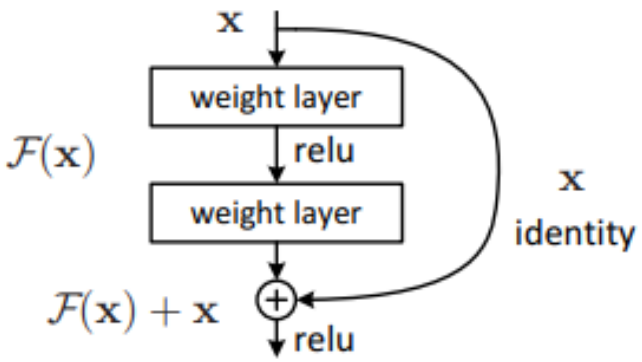

Figura 2.20: Estrutura de um bloco residual. Fonte: [13].

3 *Hardware*

Neste capítulo, será abordado como foi feita a implementação do *hardware*. O protótipo final foi constituído por um microcontrolador, um sensor acelerômetro/giroscópio, um *datalogger* para armazenamento dos dados, um módulo Bluetooth e uma bateria externa. A implementação desses componentes será detalhada nas seções seguintes.

3.1 Componentes

3.1.1 Microcontrolador

O microcontrolador utilizado foi o ESP32-S2 da *Adafruit Industries*. O microcontrolador em questão possui conectividade Wi-Fi de 2,4 GHz, processador LX7 de 32 bits com um *core*, 128 KB de memória ROM e 320 KB de memória SRAM. Além disso, conta com 43 pinos de entrada e saída programáveis, 2 pinos de I^2C, 2 pinos de UART e um display LCD. A Figura 3.1 ilustra um modelo desse microcontrolador. [32]

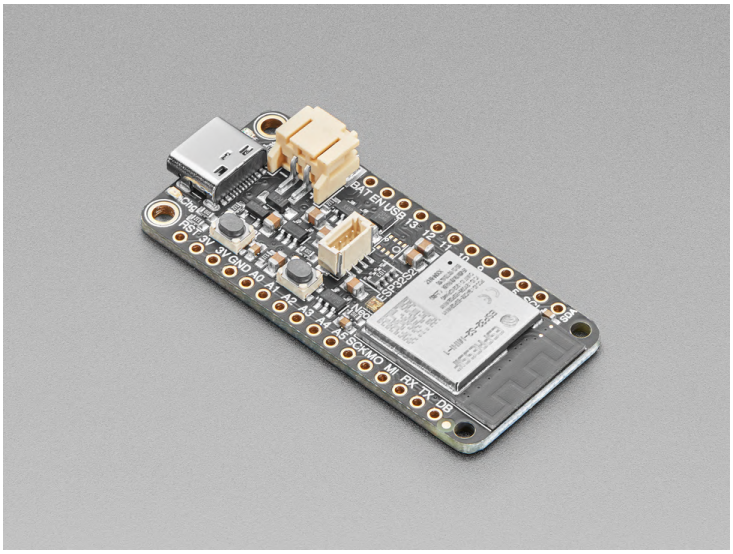

Figura 3.1: Microcontrolador ESP32-S2. Fonte: [14].

Além disso, particularmente para este modelo, há um conector STEMMA QT para comunicação I^2C. O sensor acelerômetro/giroscópio será conectado nessa interface.

A programação desse microcontrolador pode ser feita em linguagem C ou Python. A programação em Python é feita através do *software* CircuitPython [33], uma solução em Python voltada para a programação de sistemas embarcados. Neste trabalho, foi escolhida a programação em linguagem C por meio da IDE do Arduino. Essa decisão foi tomada com base na velocidade de execução dos códigos. A IDE do Arduino é mais rápida que o CircuitPython, pelo fato de o código ser compilado diretamente em linguagem de máquina, enquanto em CircuitPython o código passa por um interpretador antes.

3.1.2 Datalogger

Foi utilizado um *datalogger* também da Adafruit [15] para armazenamento dos dados. O *datalogger* possui uma interface para inserção de cartão SD, onde os dados serão armazenados. Além disso, o *datalogger* possui um módulo RTC (Real-time Clock) integrado. Esse módulo RTC será útil para determinar com precisão os momentos nos quais os dados foram capturados. Além disso, o datalogger possui um espaço para inserção de uma bateria circular de lítio, modelo CR1220. Essa bateria é necessária para o funcionamento correto do RTC. Por se tratar do mesmo fabricante, a estrutura desse *datalogger* é compatível com a do microcontrolador, sendo possível conectar um em cima do outro. A Figura 3.2 ilustra um modelo desse *datalogger*.

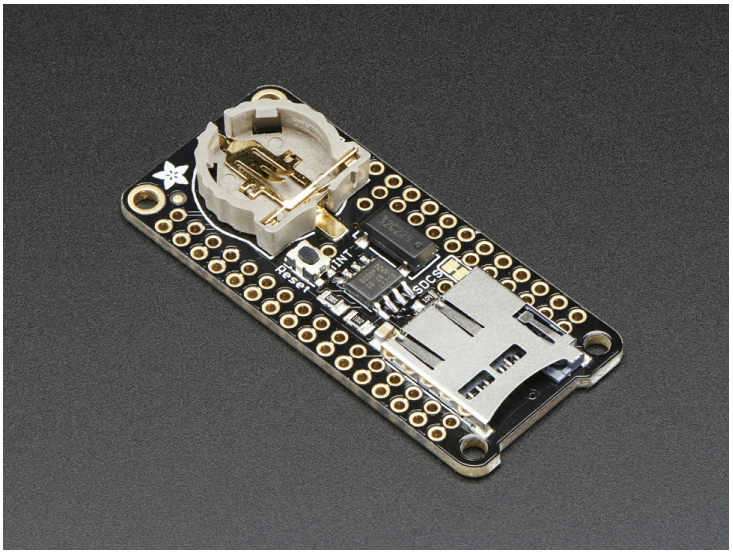

Figura 3.2: Datalogger. Fonte: [15].

3.1.3 Sensor acelerômetro/giroscópio

Foi utilizado um sensor com as funções de acelerômetro e giroscópio. O modelo utilizado foi o ISM330DHCX, da Adafruit [16]. Inicialmente, foram adquiridos dois modelos de acelerômetro: o ISM330DHCX e o LSM6DSOX [34]. Foram feitos testes com os dois acelerômetros e optou-se pelo ISM330DHCX, que possui maior precisão e qualidade industrial, e maior robustez a variações de temperatura.

Esse sensor calcula a aceleração e velocidade angular de rotação nos eixos x, y e z. Esses dados de aceleração e velocidade angular serão utilizados como entrada para alimentar a IA que irá classificar a condição do pavimento. O acelerômetro pode operar em uma faixa de $\pm 2/\pm 4/\pm 8/\pm 16$ g. O giroscópio pode operar em uma faixa de $\pm 125/\pm 250/\pm 500/\pm 1000/\pm 2000/\pm 4000$ *dps* (*degrees per second*). A taxa de amostragem do sensor pode ser definida para um intervalo entre 12,5 Hz e 6,67 KHz. Para este trabalho, foi definida uma faixa de 8g para o acelerômetro e 500 dps para o giroscópio. Ambos valores são padrões na configuração do sensor e, ao longo dos ensaios, se mostraram suficientes. A taxa de amostragem do sensor escolhida foi de 104 Hz. Para a escolha da taxa de amostragem, foi necessária uma análise mais específica da resposta em frequência dos sinais, que será mais detalhada no Capítulo 5. O sensor se comunica com o microcontrolador via protocolo I^2C. A Figura 3.3 ilustra esse modelo de sensor.

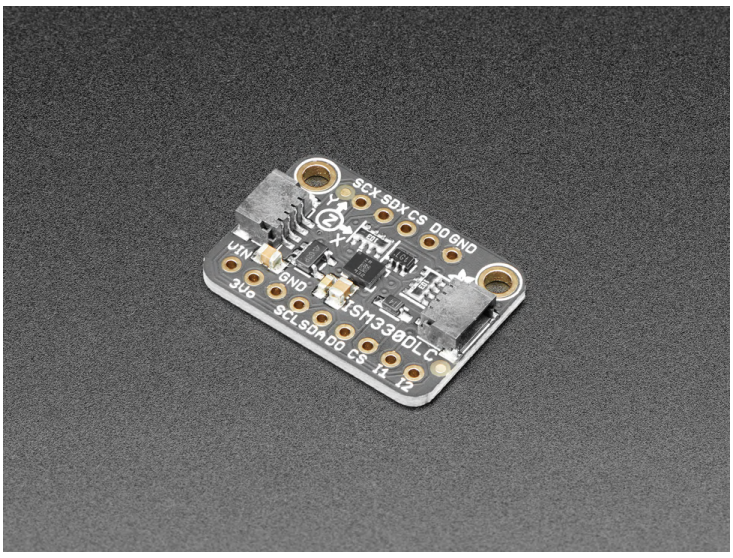

Figura 3.3: Sensor acelerômetro/giroscópio ISM330DHCX. Fonte: [16].

3.1.4 Módulo Bluetooth

Durante a Pesquisa CNT de Rodovias, faz-se uso de um tablet para coleta de dados. Nesse tablet, são registradas as avaliações acerca das condições do pavimento, sinalização e geometria da via dentro de um aplicativo desenvolvido pela própria CNT. Por meio deste aplicativo, é possível sincronizar as informações de posição do GPS com a avaliação correspondente ao trecho em questão.

Durante o desenvolvimento deste trabalho, observou-se que seria necessário estabelecer uma comunicação entre o tablet e o microcontrolador para extrair informações de horário do tablet para sincronia com o RTC do microcontrolador. E essa comunicação é feita via Bluetooth, por meio de um módulo externo. O modelo de microcontrolador utilizado nesse trabalho (ESP32-S2) não possui Bluetooth integrado, por isso a necessidade de um módulo externo.

O módulo Bluetooth utilizado é o JDY-31-SPP. O módulo em questão é compatível com o protocolo Bluetooth Classic (versão 3.0), opera em uma banda de frequências de 2,4 GHz, possui tensão de operação entre 1,8 e 3,6V e alcance de transmissão de dados de até 30 metros. A comunicação com o módulo é feita através do protocolo UART, e ele suporta taxas de *baud rate* entre 9600 e 128000 bps. Funciona apenas no modo escravo. Sua configuração é feita por meio de comandos AT, sendo possível configurar parâmetros como o *baud rate* e senha de pareamento. A Figura 3.4 ilustra o módulo Bluetooth utilizado. [35]

Figura 3.4: Módulo Bluetooth JDY-31 SPP. Fonte: [17].

3.1.5 Bateria externa

Foi utilizada uma bateria externa de lítio para realizar a alimentação do circuito. A bateria também foi adquirida na Adafruit [18]. Possui uma capacidade nominal de 500mAh, tensão nominal de 3,7V e uma tensão máxima de carregamento de 4,2V.

A ideia de utilizar essa bateria é ter uma fonte de alimentação de reserva para o circuito. Primariamente, a alimentação do circuito será feita via USB, por meio de conexão com a entrada USB do veículo. Contudo, por questões de segurança, a bateria externa seria adicionalmente conectada ao microcontrolador, a fim de mitigar possíveis falhas na alimentação pelo USB. A Figura 3.5 ilustra a bateria utilizada.

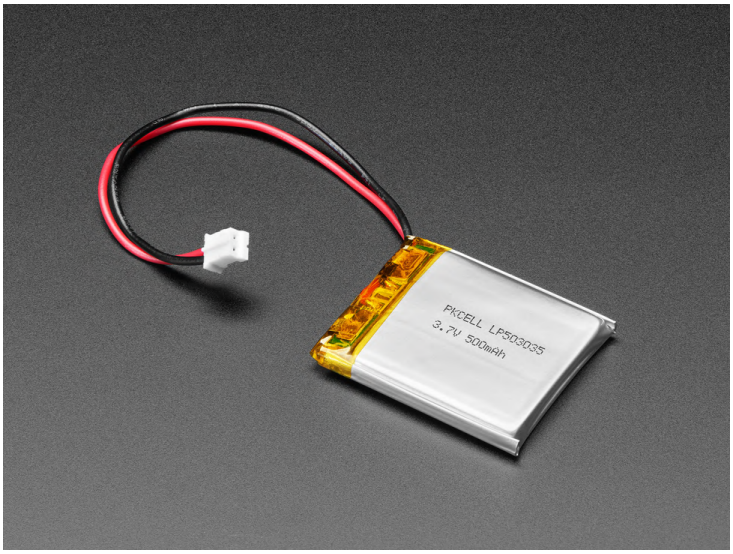

Figura 3.5: Bateria externa para o microcontrolador. Fonte: [18].

3.2 Implementação

3.2.1 Estrutura e conexões

Primeiramente, foi feita a soldagem de *headers* nos pinos do microcontrolador e do *datalogger*, para que fosse possível conectar um em cima do outro. Então, foi feita a conexão de todos componentes. O acelerômetro foi conectado na interface STEMMA QT do microcontrolador, por meio de um cabo com 4 conectores [36], sendo dois para a linha de dados e *clock* do I^2C e dois para alimentação. O *datalogger* foi conectado em cima do microcontrolador. A conexão dos pinos do módulo Bluetooth foi feita da seguinte forma:

- VCC (módulo) → 3V (ESP32)
- GND (módulo) → GND (ESP32)
- TXD (módulo) → RX (ESP32)
- RXD (módulo) → TX (ESP32)

- Os pinos STATE e EN do módulo não foram conectados a nenhum pino do ESP32

A bateria externa foi conectada no conector *LiPoly* do ESP32. Todo esse conjunto foi implementado em uma protoboard e, durante os testes, foi fixado no painel do carro com fita dupla face. A posição do protótipo dentro do carro será melhor abordada no Capítulo 5. A Figura 3.6 ilustra o *hardware* completo (sem a conexão da bateria externa). Na Figura 3.6, observa-se a presença de um suporte preto envolvendo a protoboard. Esse suporte foi desenvolvido pelo LAB (Laboratório Aberto de Brasília). Optou-se por desenvolver esse suporte considerando o método de fixação do conjunto no interior do carro. Como o conjunto é fixado com fita dupla face, foi considerado mais adequado colar a fita em outra superfície, e não diretamente na parte de baixo da protoboard. Foi feito um teste inicialmente passando a fita diretamente na parte de baixo da protoboard. Ao final desse teste, constatou-se que a protoboard havia sido bastante danificada, tendo em vista a forte aderência da fita. Além disso, o acelerômetro foi colado na protoboard com fita dupla face. Foi dedicada particular atenção à fixação do acelerômetro, observando-se a orientação de seus eixos. Em todos os testes, a orientação dos eixos foi mantida constante.

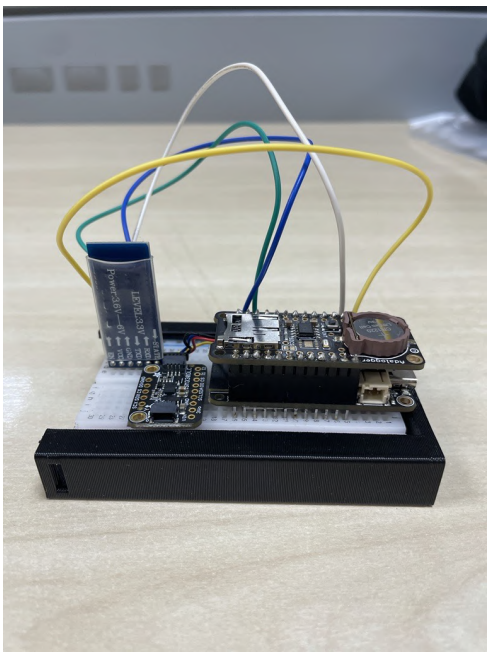

Figura 3.6: Implementação do *hardware*.

3.2.2 Lógica do *software*

Como já explicado anteriormente, o código foi todo escrito em linguagem C. Durante o desenvolvimento do código, foram utilizadas várias funções já prontas presentes em

bibliotecas da Adafruit, o que simplificou o processo.

Configurações Iniciais

Primeiramente, foram feitas as configurações iniciais do sistema. Nessa parte inicial, o *display* LCD presente no ESP32 foi utilizado para auxiliar no *debug* de cada etapa do código. A cada configuração, é escrito no *display* uma mensagem positiva ou de erro.

Primeiramente, são feitas as inicializações do RTC, acelerômetro/giroscópio e cartão SD. Então, são especificados os parâmetros do acelerômetro. São definidos uma faixa de 8g para o acelerômetro, 500 dps para o giroscópio e uma frequência de amostragem de 104 Hz para o sensor. Por fim, é feita a configuração da interface UART do ESP32. São definidos um *baud rate* de 115200 bps, 8 bits de dados, sem paridade e 1 bit de parada. Além disso, previamente, o *baud rate* do módulo Bluetooth já havia sido configurado para 115200 bps através de comandos AT. A escolha desse valor de *baud rate* será justificada na subseção seguinte.

Ao final da configuração inicial, é exibida a mensagem *Coletando...* no *display* LCD. As Figuras 3.7 a 3.11 ilustram as mensagens exibidas no *display* referentes à cada etapa em ordem.

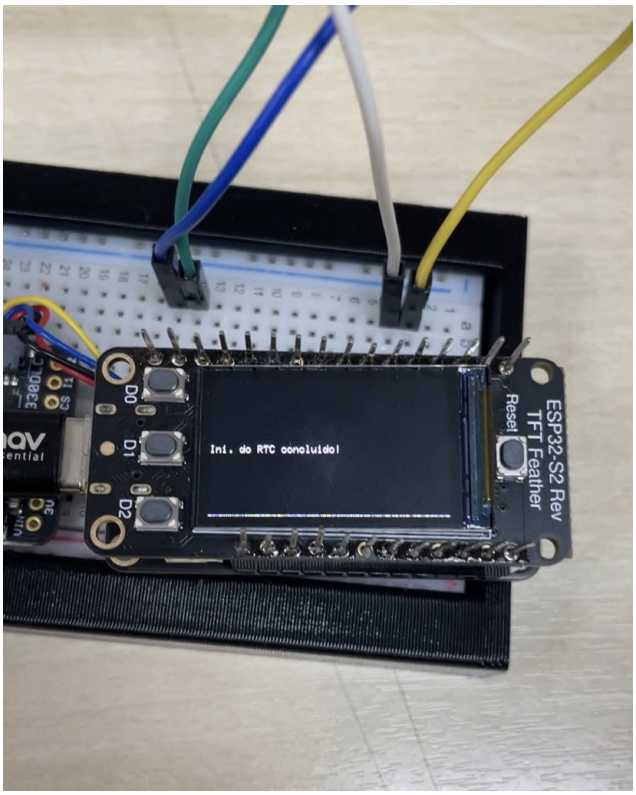

Figura 3.7: Inicialização do RTC.

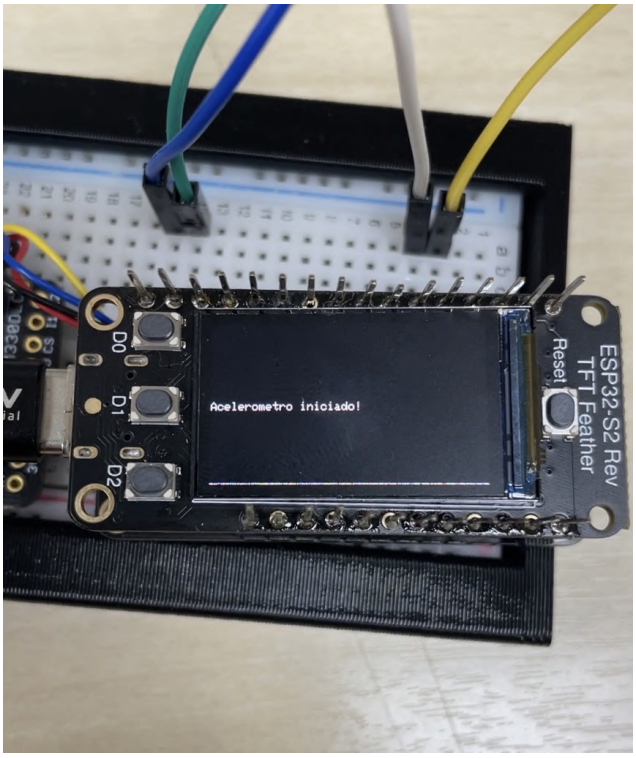

Figura 3.8: Inicialização do acelerômetro.

Figura 3.9: Inicialização do cartão SD.

Figura 3.10: Definição de parâmetros do acelerômetro.

Figura 3.11: Coletando dados.

Programa Principal

Ao entrar no *loop* pela primeira vez, o programa fica travado esperando a conexão Bluetooth do ESP32 com o tablet. Para realizar o pareamento pela primeira vez no tablet, é necessário digitar o PIN 1234 (padrão do módulo Bluetooth). Então, após a conexão ser estabelecida, o tablet envia uma mensagem *CONNECTED* para o ESP32. A etapa seguinte consiste em extrair o horário do tablet para sincronizar com o RTC.

Para estabelecer essa comunicação, foi desenvolvido um aplicativo de comunicação Bluetooth para o tablet em linguagem *Dart*, utilizando o *framework Flutter* [37]. A escolha pelo *Flutter* foi com o objetivo de integrar essa funcionalidade ao aplicativo desenvolvido pela CNT para classificação de rodovias, citado na Seção 3.1.4, o qual foi escrito em *Flutter*. Neste trabalho, não será abordado em detalhes o desenvolvimento deste aplicativo em *Flutter* para comunicação Bluetooth, tendo em vista que não é o foco do trabalho. De forma resumida, após feito o pareamento, o funcionamento deste aplicativo consiste em ficar esperando por um comando específico por Bluetooth. Ao receber este comando, o tablet responde com o seu horário atual.

Então, retornando ao programa principal, ao receber a mensagem *CONNECTED*, o ESP32 envia o comando *CMD_1* para o tablet. Este comando foi definido como o comando para envio do *timestamp* pelo tablet. Ao receber o *timestamp*, o programa ajusta o horário do RTC para que fique igual ao horário do tablet. Então, cria um arquivo de texto com nomenclatura contendo as informações de data e horário, no formato *yyyy-mm-dd_hh-mm-ss*. Nesse arquivo, são armazenados os dados captados pelo acelerômetro/giroscópio. São salvos os valores de aceleração nos eixos *x, y, z* e velocidade angular de rotação nos eixos *x, y, z* nessa ordem, separando-se cada valor por vírgula. A Figura 3.12 ilustra um exemplo de um arquivo. Esses arquivos são criados precisamente em intervalos de 1 minuto e armazenados no cartão SD.

Além de escrever os dados em arquivos de texto e armazenar no cartão SD, o programa também envia esses dados para o tablet por Bluetooth através da porta serial. Idealmente, esse envio para o tablet é por segurança. Caso, por algum motivo, o armazenamento no cartão SD falhe, ou a memória seja sobrecarregada, os dados não seriam perdidos. Porém, até o presente momento, esses dados são apenas enviados para o tablet, mas ainda não foi implementado um mecanismo para armazená-los dentro do tablet. A Figura 3.13 ilustra o funcionamento do aplicativo de comunicação Bluetooth no tablet após a realização do pareamento.

Portanto, o arquivo mostrado na Figura 3.12 contém todas as amostras coletadas pelo sensor em um intervalo de um minuto. A taxa de amostragem definida é de 104 Hz. Porém, na prática, observou-se que a taxa de amostragem efetiva era ligeiramente superior à estabelecida, alcançando aproximadamente 109 Hz. Durante a realização de testes,

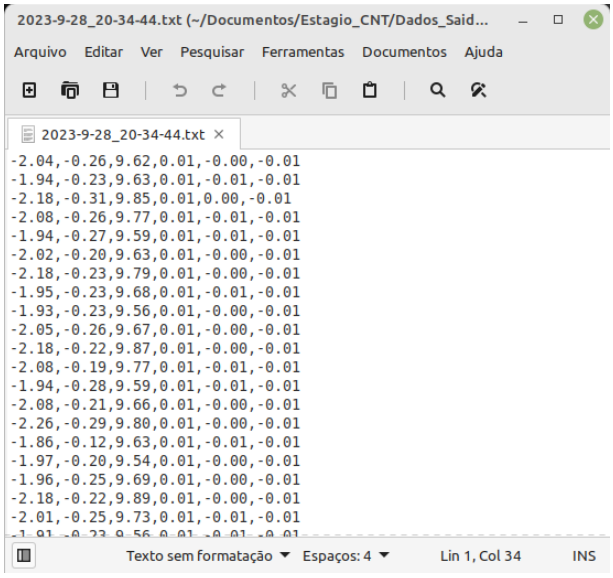

Figura 3.12: Exemplo de arquivo onde são armazenados os dados do sensor.

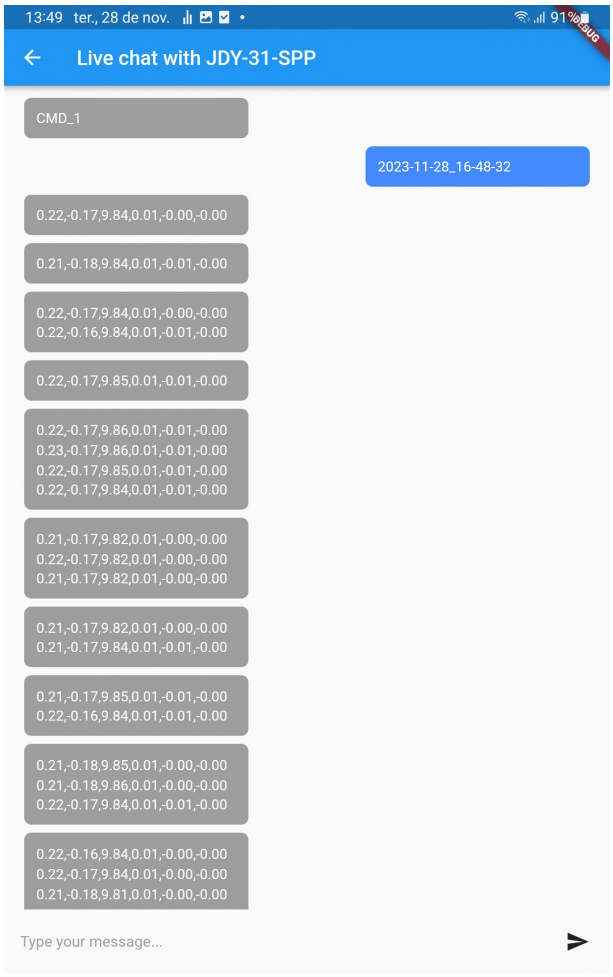

Figura 3.13: Funcionamento do aplicativo de comunicação Bluetooth.

constatou-se que, ao implementar a funcionalidade de envio de dados por Bluetooth, a taxa de amostragem do sensor diminuía. Verificou-se que isso acontecia devido à taxa de *baud rate* definida para a interface UART. Inicialmente, definiu-se um *baud rate* mais baixo (por volta de 9600 bps). Porém, observa-se que uma amostra pode ser constituída de até 43 caracteres (se todos valores forem negativos, contando com os caracteres de *carriage return* e *new line* no final). Portanto, no máximo, uma amostra seria constituída de 43 bytes. Como a taxa de amostragem observada é de 109 Hz, tem-se que a taxa de envio de dados é de 37496 bits/segundo ($43 \times 109 \times 8$). Portanto, uma taxa de *baud rate* a partir de 38400 bps seria suficiente. Mas, preventivamente, optou-se por um *baud rate* mais alto, de 115200 bps, para caso fosse necessário aumentar a taxa de amostragem do sensor posteriormente.

O programa permanece nesse loop infinito, coletando dados e os armazenando no cartão SD a cada minuto. A Figura 3.14 ilustra um fluxograma do funcionamento do código implementado na ESP32.

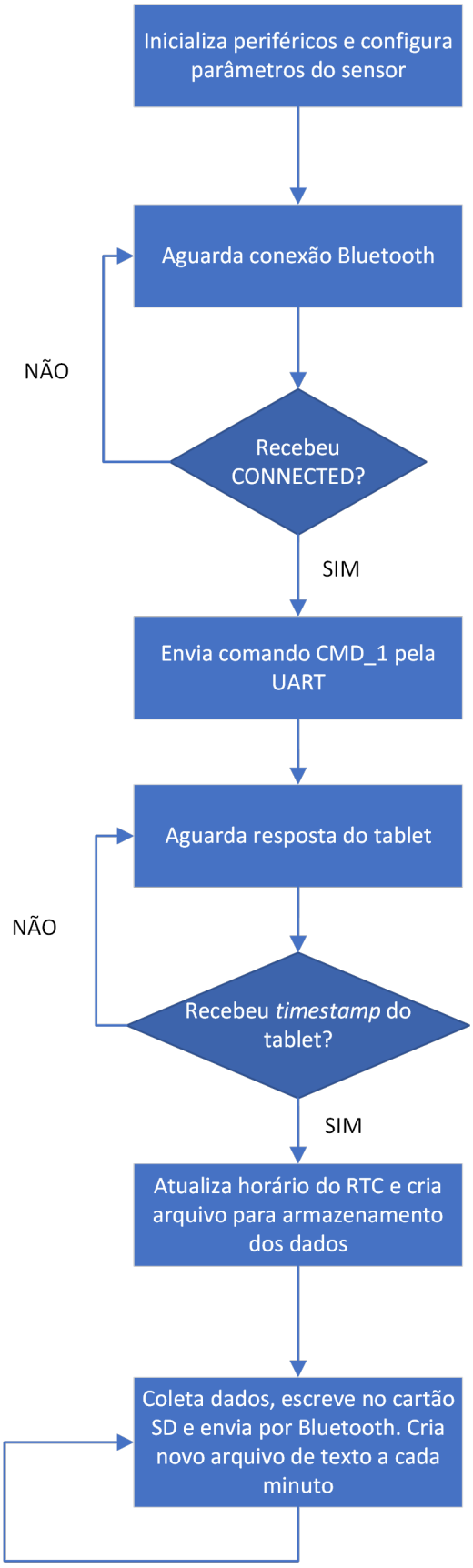

Figura 3.14: Fluxograma do código embarcado.

4 IA

Neste capítulo, será abordado como foi feita a implementação dos dois modelos de Inteligência Artificial para classificação do pavimento. Um dos modelos realiza a classificação com base em imagens, e o outro com base nos dados coletados pelo sensor.

4.1 IA de imagem

Conforme discutido no Capítulo 1, os carros utilizados durante a Pesquisa CNT de Rodovias são equipados com câmeras. A ideia deste modelo de IA é realizar a classificação da condição do pavimento com base em *frames* capturados por essa câmera.

O modelo implementado é constituído por uma rede *ResNet50* [38] pré-treinada no início e três camadas totalmente conectadas no final. Uma com 1000 neurônios, outra com 256 e a última com 2 neurônios, referentes às duas possíveis classificações do pavimento (perfeito ou imperfeito). Nas duas primeiras camadas totalmente conectadas foi utilizada a função de ativação *ReLU*. Na última camada foi utilizada a função de ativação sigmoide, para classificação. A rede possui 25.813.802 parâmetros, sendo todos treináveis. A Figura 4.1 ilustra a estrutura da rede.

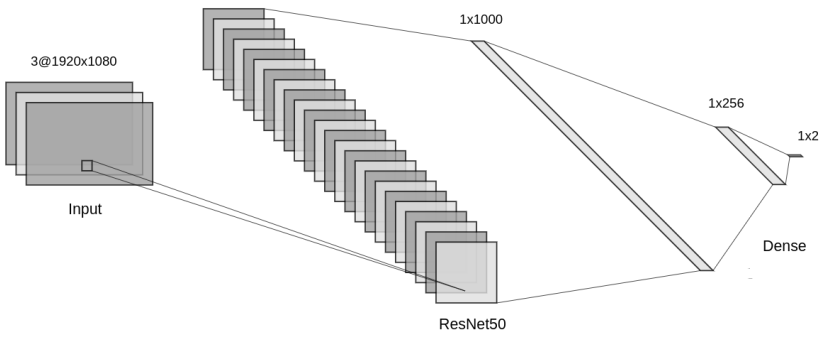

Figura 4.1: Estrutura do modelo de IA de imagem.

4.1.1 Treinamento do modelo

Para o treinamento da rede, a função de perda utilizada foi a de entropia cruzada. O otimizador foi o SGD, com *learning rate* de 0,001 e *momentum* de 0,9. O parâmetro *momentum* é uma técnica para acelerar o gradiente descendente, armazenando as direções tomadas durante o algoritmo em um vetor. [39]

Para a construção do *dataset* de treinamento, foram utilizados vídeos de pesquisas de anos anteriores. Foi desenvolvido um código para extrair *frames* dentro desses vídeos em intervalos igualmente espaçados. Então, classificava-se manualmente esses *frames* entre perfeito ou imperfeito. Cada frame possuía dimensão de 1920x1080 pixels. A Figura 4.2 ilustra exemplos de *frames* perfeitos e imperfeitos selecionados para o primeiro treinamento.

Figura 4.2: Exemplos de *frames* de pavimento perfeito e imperfeito. Imagens cedidas pela CNT.

Primeiro treinamento

Inicialmente, o *dataset* montado continha frames pouco espaçados temporalmente. Com isso, observou-se que alguns *frames* eram muito semelhantes entre si. O desempenho da IA nesse primeiro treinamento foi muito alto, alcançando precisão superior a 90% com poucas épocas, conforme mostrado na Figura 4.3. Entretanto, verificou-se que o modelo não estava demonstrando uma boa capacidade de generalização para *frames* não incluídos no conjunto de treinamento. Por conseguinte, constatou-se que a rede não estava efetivamente aprendendo, mas sim memorizando os *frames* que eram muito similares.

```
Epoch 1/5
225/225 - 301s - loss: 0.1247 - accuracy: 0.9594 - val_loss: 0.1080 - val_accuracy: 0.9650 - 301s/epoch - 1s/step
Epoch 2/5
225/225 - 300s - loss: 0.0961 - accuracy: 0.9689 - val_loss: 0.1080 - val_accuracy: 0.9600 - 300s/epoch - 1s/step
Epoch 3/5
225/225 - 300s - loss: 0.0695 - accuracy: 0.9833 - val_loss: 0.0848 - val_accuracy: 0.9600 - 300s/epoch - 1s/step
Epoch 4/5
225/225 - 315s - loss: 0.0526 - accuracy: 0.9922 - val_loss: 0.0769 - val_accuracy: 0.9750 - 315s/epoch - 1s/step
Epoch 5/5
225/225 - 300s - loss: 0.0450 - accuracy: 0.9922 - val_loss: 0.0747 - val_accuracy: 0.9750 - 300s/epoch - 1s/step
<keras.callbacks.History at 0x7d28d4137610>
```

Figura 4.3: Resultados do primeiro treinamento do modelo de IA de imagem.

A partir desses resultados, foi implementado um algoritmo para calcular o nível de semelhança entre dois *frames*, utilizando o método SSIM [40], com o objetivo de excluir *frames* muito semelhantes do conjunto de treinamento. Além disso, o intervalo entre *frames* extraídos dos vídeos foi alterado para um *frame* a cada 5 segundos.

Segundo treinamento

Com essas modificações, foi construído um novo dataset. Esse novo dataset era constituído por 1450 imagens de pavimento perfeito e 1450 imperfeito, sendo 1000 para treinamento, 150 para validação e 300 para teste em cada classe. As imagens do conjunto de teste foram extraídas de vídeos não utilizados durante o treinamento. O modelo foi treinado com este novo dataset e os resultados obtidos foram os seguintes:

- **Precisão de treinamento: $> 95\%$**
- **Precisão de validação: 86%**
- **Precisão de teste: 90,33%**

As Figuras 4.4 a 4.6 ilustram o custo, a precisão e a matriz de confusão associados ao modelo, respectivamente.

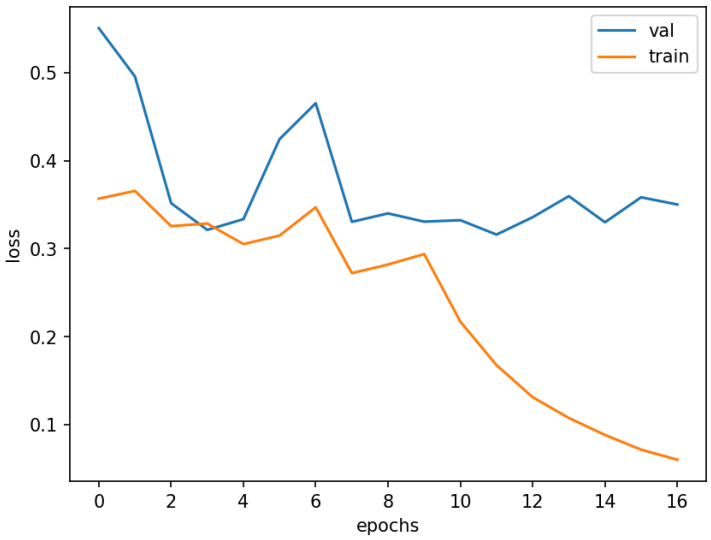

Figura 4.4: Custo do segundo treinamento do modelo de IA de imagem.

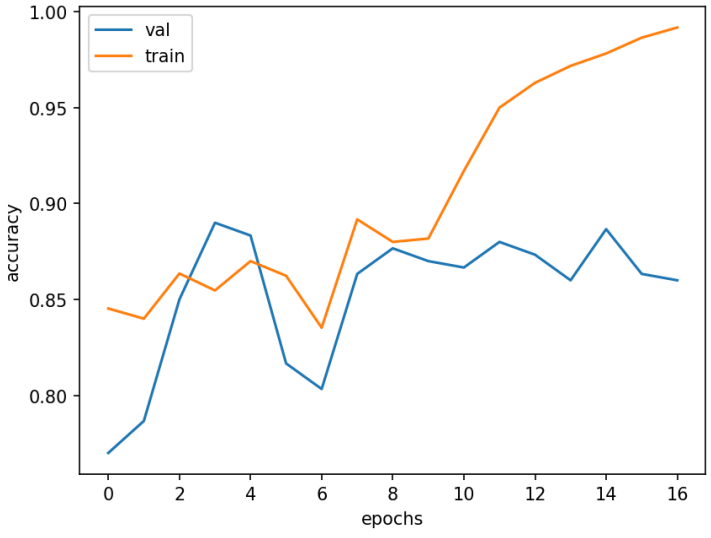

Figura 4.5: Precisão do segundo treinamento do modelo de IA de imagem.

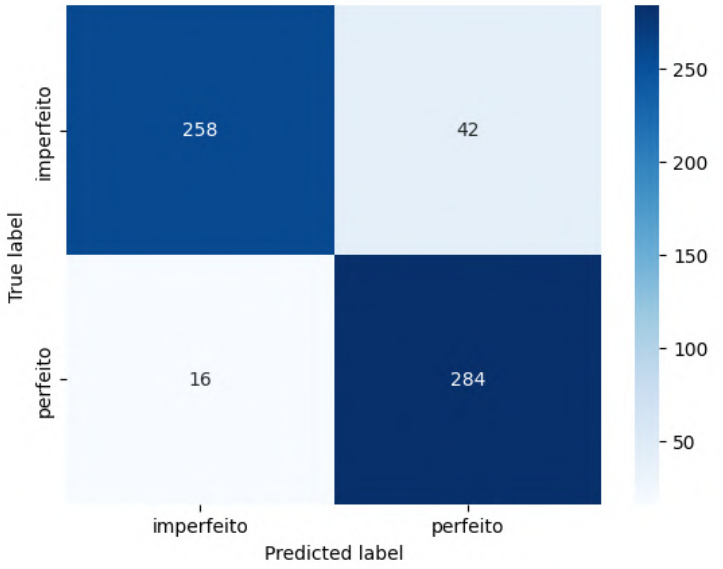

Figura 4.6: Matriz de confusão para o segundo treinamento do modelo de IA de imagem.

Terceiro treinamento

Ao realizar uma análise mais minuciosa dos dados de treinamento, observou-se que algumas imagens englobavam partes maiores do ambiente externo.

Isso ocorria devido à ausência de um protocolo estabelecido para o posicionamento da câmera no veículo, resultando, portanto, em variações nos ângulos em determinados posicionamentos. As Figuras 4.7 a 4.8 ilustram exemplos de *frames* contendo mais informações relacionadas ao contexto em torno do pavimento.

Figura 4.7: Exemplo de *frame* englobando partes do ambiente externo. Imagem cedida pela CNT.

Concluiu-se que isto poderia estar prejudicando a aprendizagem do modelo desenvolvido, visto que sua ênfase deveria ser na assimilação de características do pavimento, e

Figura 4.8: Exemplo de *frame* englobando partes do ambiente externo. Imagem cedida pela CNT.

não do ambiente externo. Com isso, foram feitos testes realizando cortes na parte superior das imagens e treinando novamente a rede. Foram testados três tipos de cortes. Um de 25% da altura da imagem, outro de 35% e outro de 50%. As Figuras 4.9 a 4.10 ilustram as mesmas imagens das Figuras 4.7 a 4.8 após a realização do corte de 35% da altura.

Figura 4.9: *Frame* da Figura 4.7 após a realização de corte. Imagem cedida pela CNT.

Figura 4.10: *Frame* da Figura 4.8 após a realização de corte. Imagem cedida pela CNT.

Além disso, após a realização desses cortes nas imagens, foi necessário reavaliar manualmente o conjunto de dados de treinamento, para certificar que partes relevantes da imagem não tinham sido cortadas. O modelo foi treinado novamente com esses três níveis

de corte, e o que apresentou melhor resultado foi o de 25% da altura da imagem. Os resultados obtidos associados a este corte foram os seguintes:

- **Precisão de treinamento: $> 95\%$**
- **Precisão de validação: 90%**
- **Precisão de teste: 91,67%**

As Figuras 4.11 a 4.13 ilustram o custo, a precisão e a matriz de confusão associados ao modelo, respectivamente.

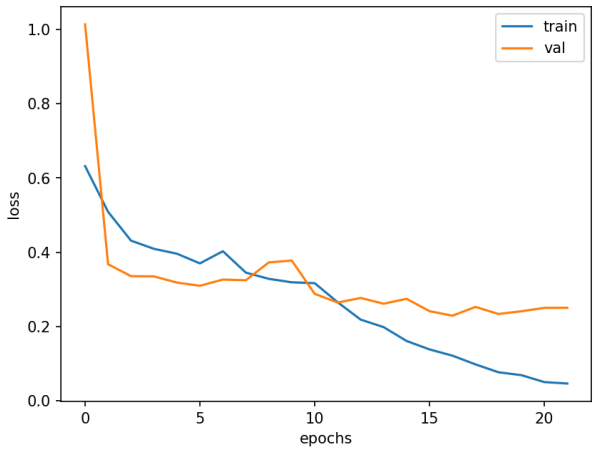

Figura 4.11: Custo do terceiro treinamento do modelo de IA de imagem.

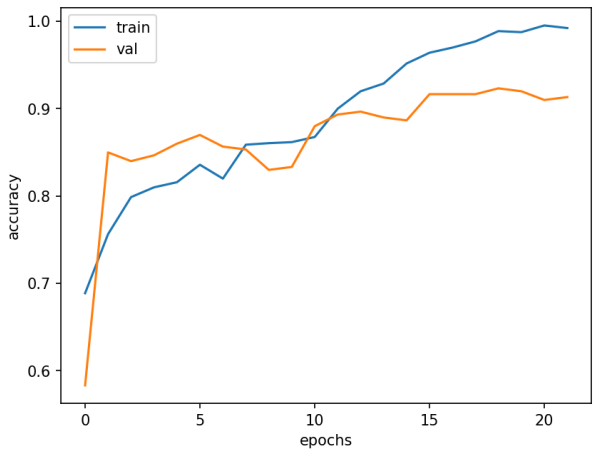

Figura 4.12: Precisão do terceiro treinamento do modelo de IA de imagem.

Observou-se que o modelo teve um leve ganho de precisão. O fato de este ganho não ser tão significativo sugere uma possibilidade de que o modelo já está próximo de um limite de aprendizagem.

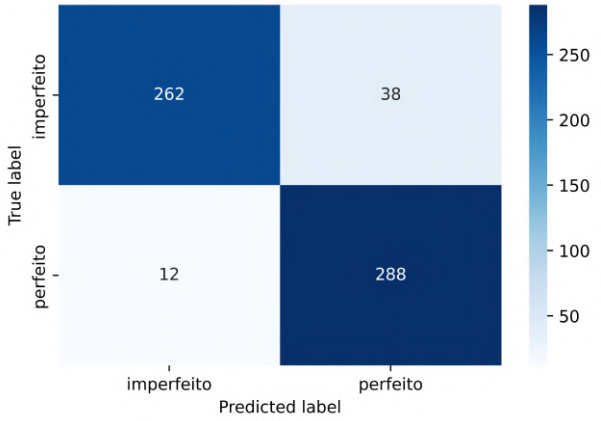

Figura 4.13: Matriz de confusão para o terceiro treinamento do modelo de IA de imagem.

Quarto treinamento

Foi feita uma análise dos *feature maps* do conjunto de dados. A partir desta análise, detectou-se um viés na classificação do modelo. Observou-se que o modelo estava detectando caracteres presentes na imagem associados às coordenadas do veículo, e utilizando essas informações na aprendizagem. A Figura 4.14 ilustra um exemplo de imagem contida nos *feature maps*, destacando a influência dos caracteres na parte inferior da imagem na aprendizagem do modelo.

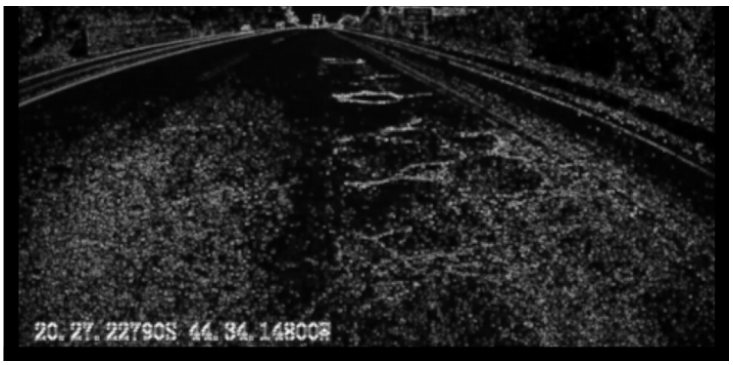

Figura 4.14: Influência de caracteres na aprendizagem do modelo. Imagem cedida pela CNT.

Para solucionar isso, foi feito um novo corte na parte inferior das imagens para remover essas informações. Então, o modelo foi treinado novamente com esse novo conjunto de dados. Os resultados obtidos foram os seguintes:

- **Precisão de treinamento: $> 95\%$**
- **Precisão de validação: 90%**

- **Precisão de teste: 90,33%**

As Figuras 4.15 a 4.17 ilustram o custo, a precisão e a matriz de confusão associados ao modelo, respectivamente.

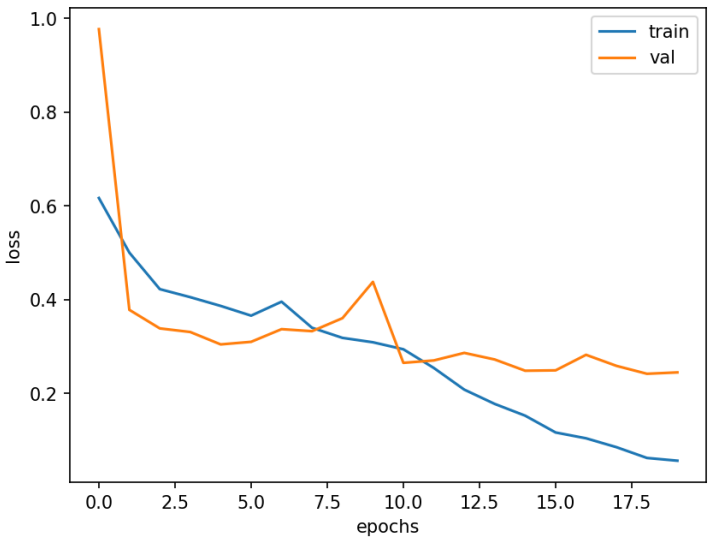

Figura 4.15: Custo do quarto treinamento do modelo de IA de imagem.

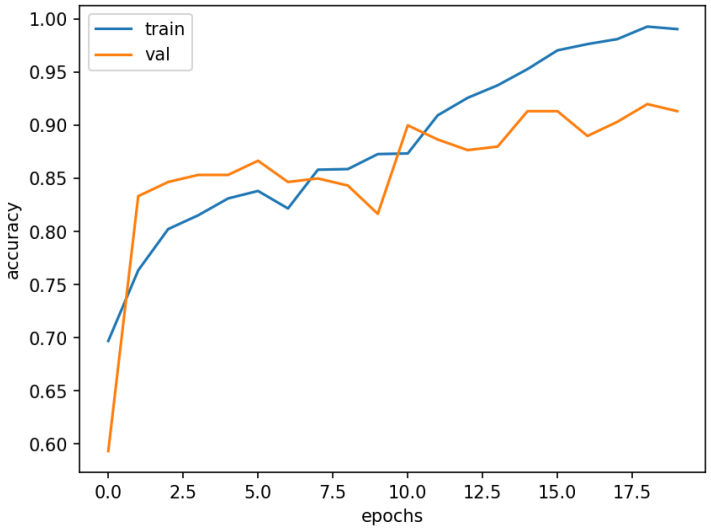

Figura 4.16: Precisão do quarto treinamento do modelo de IA de imagem.

Os resultados mostraram que, de fato, os caracteres estavam influenciando negativamente o modelo, aumentando artificialmente a precisão. Porém, essa interferência não foi significativa, de forma que a precisão de teste manteve-se acima de 90%.

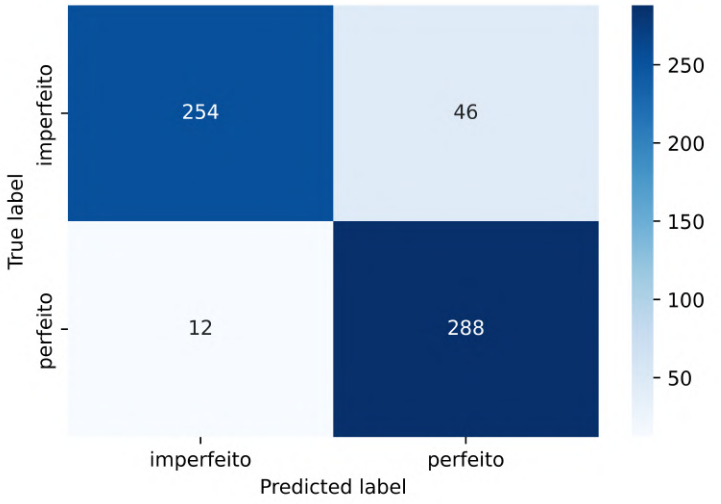

Figura 4.17: Matriz de confusão para o quarto treinamento do modelo de IA de imagem.

Os resultados obtidos neste último treinamento constituem a base conclusiva para esta pesquisa, relativamente à IA de imagem.

4.2 IA do sensor

Este modelo de IA tem como objetivo classificar os dados coletados pelo sensor entre duas classes associadas à condição do pavimento (perfeito e imperfeito). Para o desenvolvimento deste modelo, utilizou-se um outro modelo de IA como referência, disponível em [41]. O modelo base é uma aplicação de IA voltada para identificação de padrões de movimento do ser humano, a partir da classificação de dados coletados por acelerômetro. Esses dados foram obtidos em experimentos feitos na Universidade de Fordham, descritos em [42]. Este modelo classifica os sinais em 6 classes possíveis associadas ao movimento humano: andando, correndo, subindo escada, descendo escada, sentando e parado em pé. O classificador de pavimento desenvolvido neste trabalho utiliza a mesma estrutura do modelo em [41], alterando-se a quantidade de neurônios na última camada de 6 para 2, referentes às duas classificações possíveis para o pavimento.

O modelo implementado é constituído por duas camadas convolucionais seguidas de duas camadas totalmente conectadas. A primeira camada convolucional possui 16 filtros e a segunda 32. A primeira e a segunda camadas totalmente conectadas possuem, respectivamente, 64 e 2 neurônios. Nas duas camadas convolucionais e na primeira camada totalmente conectada é utilizada a função de ativação *ReLU*. Na última camada, é utilizada a função de ativação sigmoide. Além disso, nas camadas convolucionais é realizado um *padding* de zeros. A rede possui 408178 parâmetros, sendo todos treináveis.

Conforme detalhado na Seção 3.1.3, o sensor utilizado coleta informações de aceleração e velocidade angular em 6 eixos. Os dados desses 6 eixos são fornecidos como entrada para o modelo de IA. Por conta disso, diferentemente do que ocorreu com a IA de imagem, em que se dispunha de um conjunto de dados provenientes de anos anteriores para realizar o treinamento, para este modelo, foi necessário construir o conjunto de dados integralmente do início. Dessa forma, foi necessário realizar saídas de campo com o sensor implementado no interior do veículo para a construção do *dataset*. Neste capítulo, são abordadas apenas características estruturais do modelo, reservando-se a abordagem do treinamento do modelo para o Capítulo 5, onde também será detalhado o processo de coleta de dados em campo.

Antes de serem fornecidos ao modelo, os dados são submetidos a uma etapa de pré-processamento. Nesta etapa, as amostras são separadas em janelas de tamanho fixo, e essas janelas são fornecidas como entrada para a rede. Há uma etapa de rotulação prévia, em que cada amostra é rotulada como perfeita ou imperfeita. Com isso, cada janela de amostras é classificada como perfeita ou imperfeita com base na predominância do rótulo na amostragem. A definição para o tamanho da janela e o processo de classificação das amostras serão mais detalhados no Capítulo 5. A Figura 4.18 ilustra a estrutura da rede para uma janela constituída por 64 amostras.

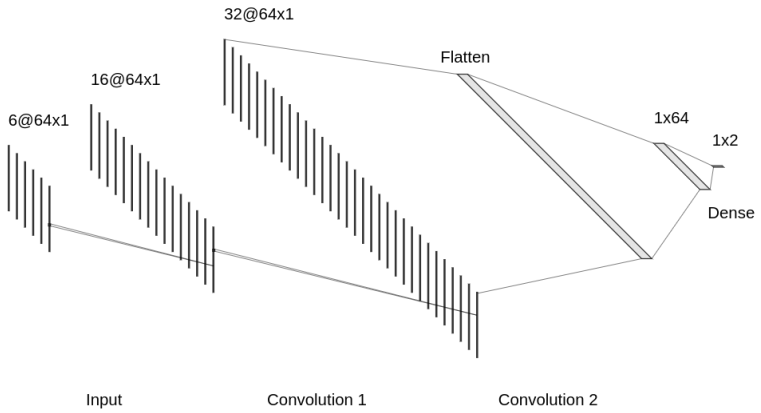

Figura 4.18: Estrutura do modelo de IA do sensor.

5 Testes e Resultados

Neste capítulo, serão apresentados os testes práticos feitos com o protótipo e os resultados obtidos com os modelos de IA. Foram realizadas duas saídas de campo. A primeira para testar o funcionamento geral do *hardware* e a segunda para avaliar o desempenho dos dois modelos de IA.

5.1 Saída de campo para testar funcionamento do *hardware*

Após a implementação do *hardware*, foi realizada uma saída de campo para testar seu funcionamento e definir alguns parâmetros, como a taxa de amostragem do sensor. Neste momento, nenhum dos modelos de IA havia sido desenvolvido ainda. Portanto, a finalidade da saída era avaliar exclusivamente o desempenho do *hardware*. Nesta saída, ainda não havia sido implementada a lógica de comunicação Bluetooth com o tablet, portanto, o módulo Bluetooth não foi utilizado. Este protótipo era constituído pelo microcontrolador, pelo *datalogger*, pelo sensor e pela bateria externa. A Figura 5.1 ilustra o protótipo.

O veículo utilizado foi um Jeep Renegade. Não foram registradas fotos da instalação do *hardware* no veículo. No entanto, o posicionamento do *hardware* dentro do veículo é mostrado na Figura 5.2. Buscou-se fixar o conjunto em uma superfície plana dentro do carro, e o local indicado na Figura 5.2 foi o que mais se adequou. A orientação dos eixos do sensor é mostrada na Figura 5.3.

A rota percorrida durou aproximadamente 20 minutos, sendo constituída por um trecho de aproximadamente 11 km de via pavimentada e um trecho de 700 metros de via não pavimentada. As diferenças nos sinais capturados em cada tipo de via são perceptíveis e serão mostradas nesta seção.

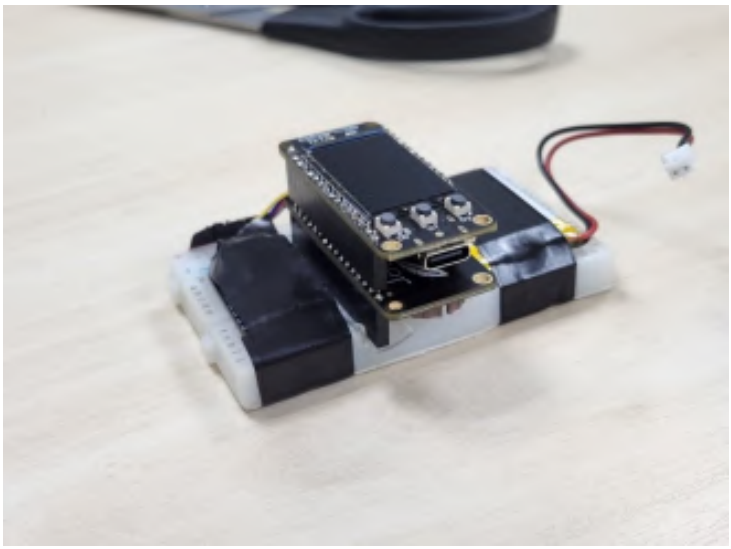

Figura 5.1: Protótipo utilizado na primeira saída de campo.

Figura 5.2: Posicionamento do *hardware* dentro do veículo na primeira saída de campo.

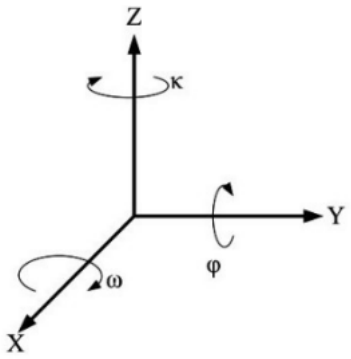

Figura 5.3: Orientação dos eixos do sensor.

5.1.1 Dados obtidos

A Figura 5.4 ilustra os dados brutos coletados pelo sensor durante o trajeto, sem qualquer tipo de filtragem. A Figura 5.5 ilustra os espectros referentes aos 6 eixos do acelerômetro e giroscópio. Neste teste, foi definida uma taxa de amostragem de 833 Hz para o sensor, a fim de possibilitar a realização da análise em frequência para uma faixa maior do espectro, tendo em vista o objetivo de determinar a taxa de amostragem ideal para o sensor.

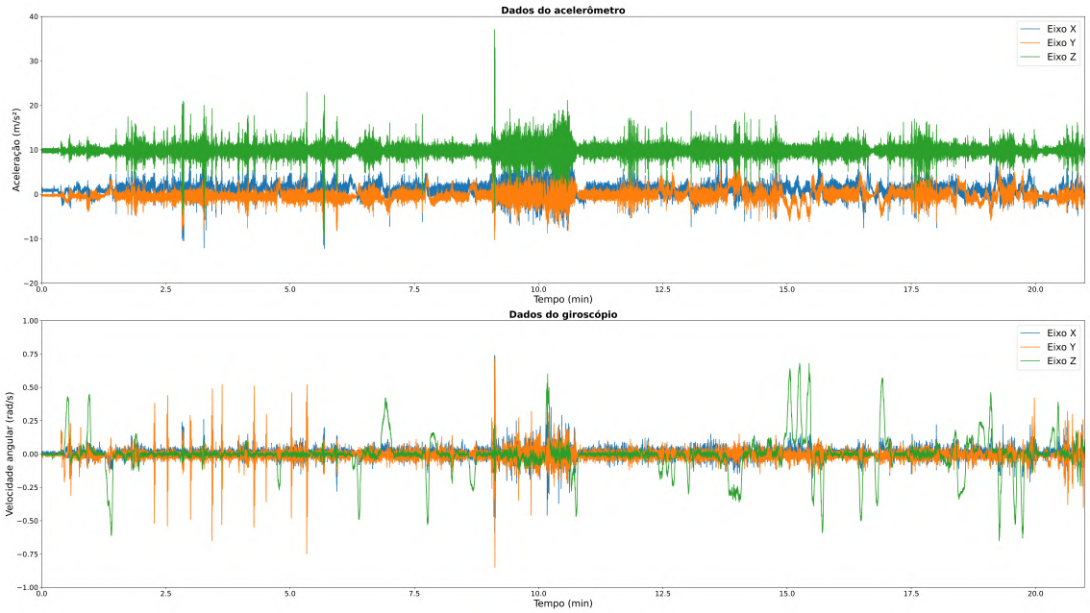

Figura 5.4: Dados brutos coletados pelo sensor na primeira saída de campo.

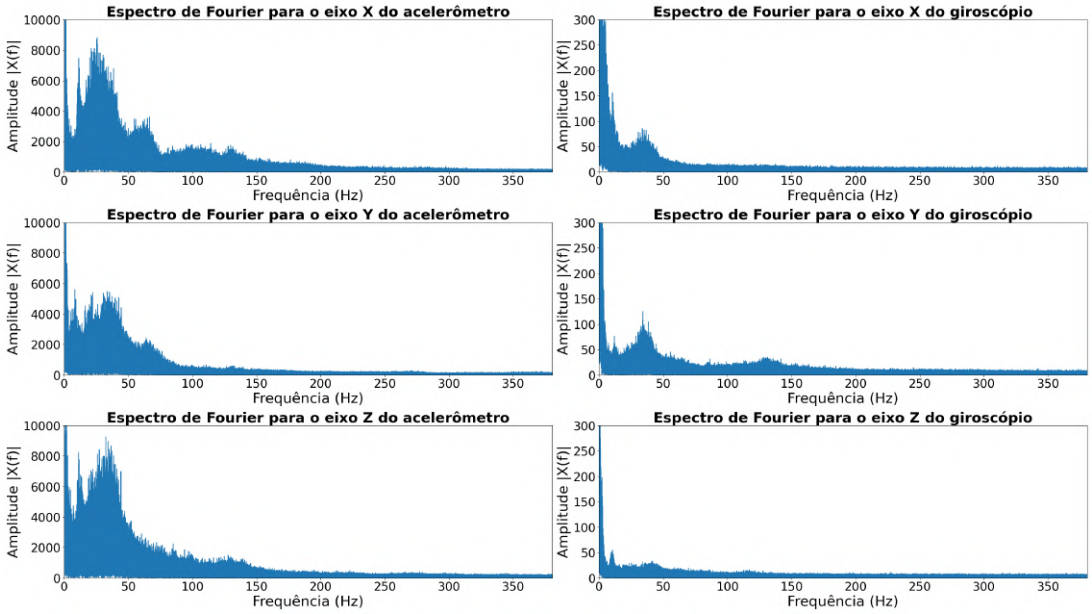

Figura 5.5: Respostas em frequência dos dados coletados pelo sensor na primeira saída de campo.

Através da análise da Figura 5.5, é possível identificar a presença de componentes de frequência superiores a 50 Hz de forma significativa em praticamente todos os eixos. No entanto, ao realizar filtragens no sinal, constatou-se que essas altas frequências não constituíam uma parte relevante do sinal, e sim, ruído.

Para chegar a essa conclusão, foram realizadas filtragens sucessivas para diferentes frequências de corte e observando o aspecto do sinal. Utilizou-se um filtro FIR passa-baixas de 7ª ordem. Durante esses testes, verificou-se que, para frequências de corte abaixo de 30 Hz, o sinal começava a sofrer distorções significativas. Portanto, concluiu-se que a parte mais relevante do sinal estava concentrada em frequências de até 30 Hz. Com isso, a taxa de amostragem do sensor foi definida em 104 Hz.

Além disso, nesta saída de campo foi possível verificar padrões no sinal associados a irregularidades no pavimento ou características da via.

A Figura 5.6 ilustra o comportamento do sinal durante a passagem do veículo por um quebra-mola.

A Figura 5.7 ilustra o comportamento do sinal em um instante em que o veículo passa por remendos e tachões.

As Figuras 5.8 a 5.9 ilustram as diferenças relacionadas à pavimentação. A Figura 5.8 ilustra a diferença percebida no sinal entre uma via pavimentada e uma não pavimentada. A Figura 5.9 ilustra a diferença percebida no sinal entre um pavimento liso e um pavimento rugoso.

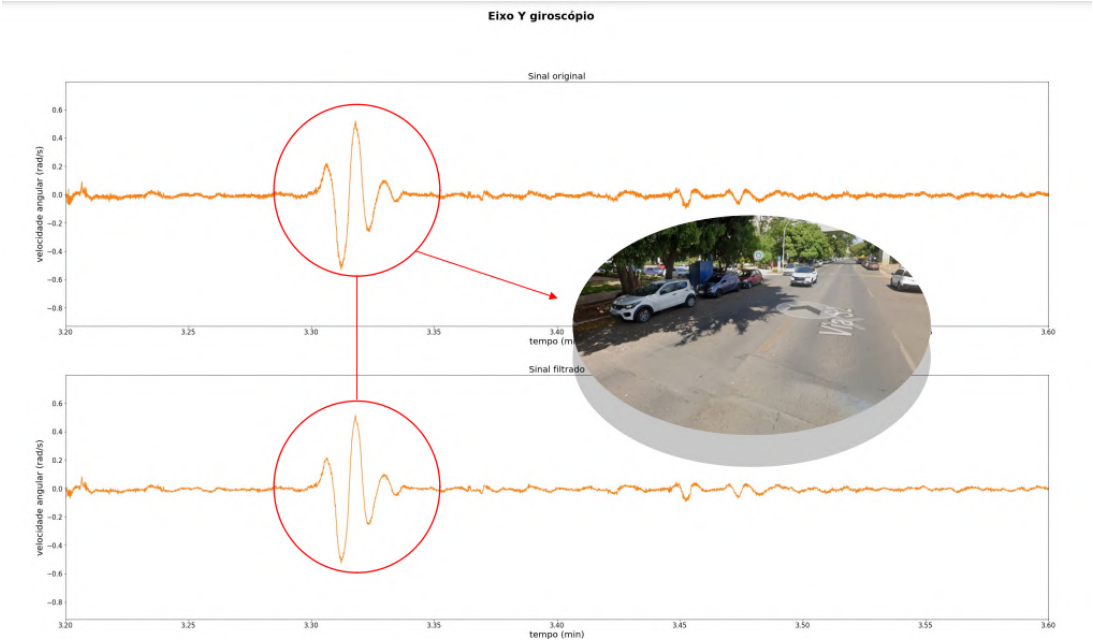

Figura 5.6: Sinal captado pelo sensor durante a passagem do veículo por um quebra-mola.

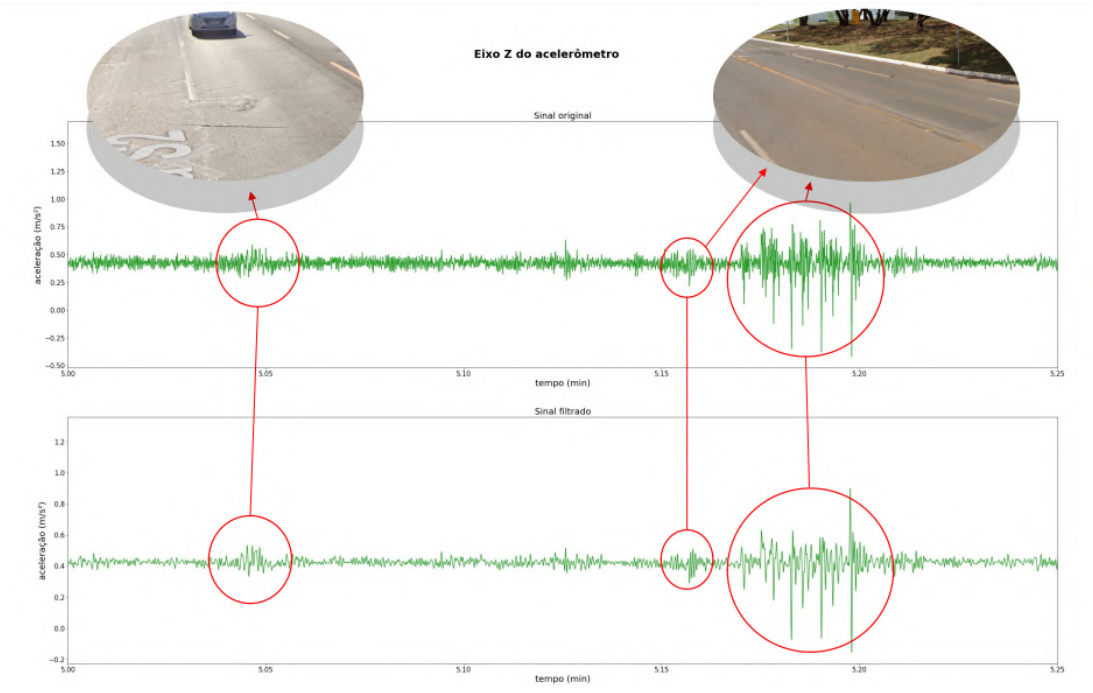

Figura 5.7: Sinal captado pelo sensor em instante em que o veículo passa por remendos e tachões.

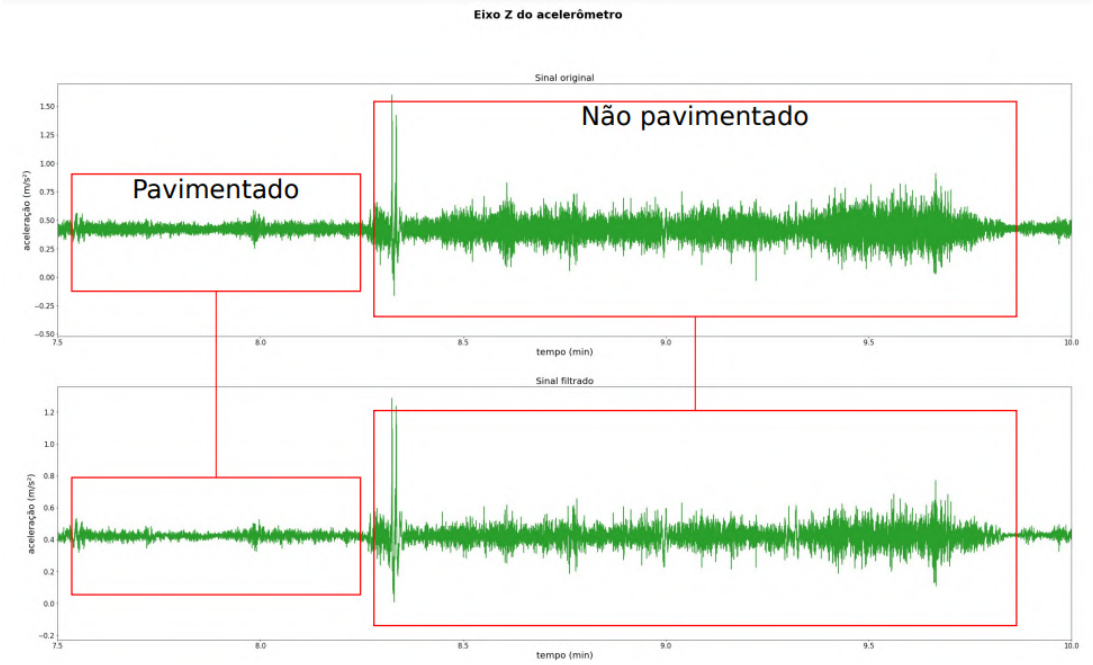

Figura 5.8: Sinal captado pelo sensor em região pavimentada e não pavimentada.

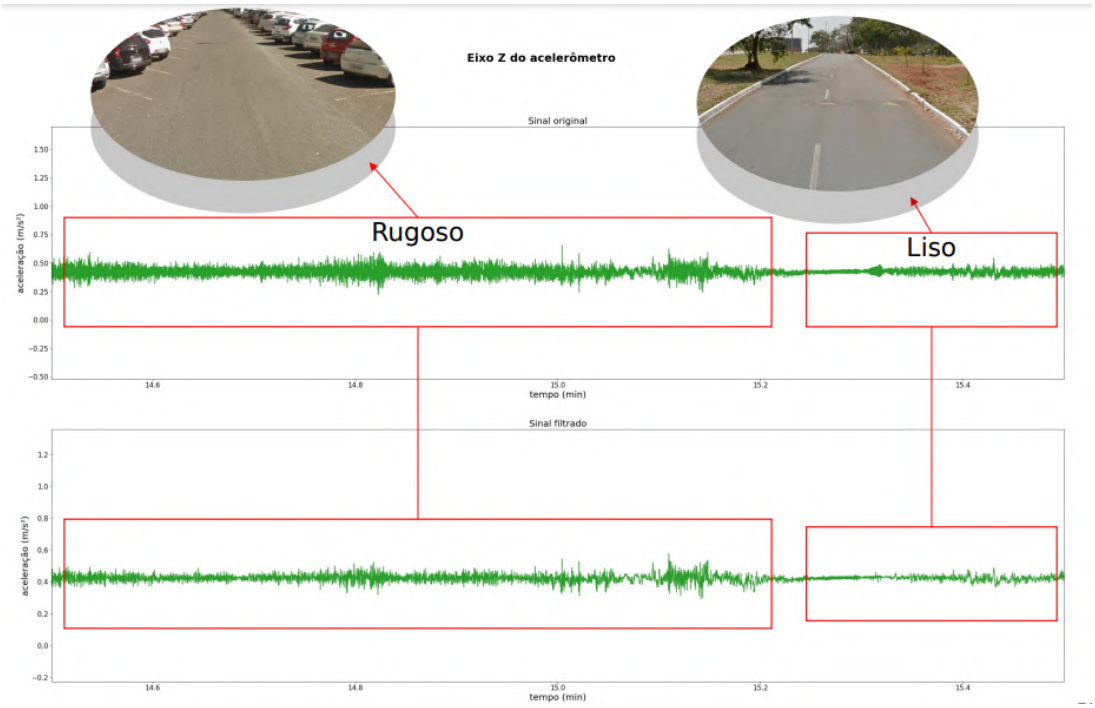

Figura 5.9: Sinal captado pelo sensor em regiões de pavimento liso e rugoso.

5.2 Saída de campo para avaliar modelos de IA

Após a realização dos testes com o *hardware* e a implementação dos dois modelos de IA, foi realizada outra saída de campo com o intuito de avaliar o desempenho dos referidos modelos de IA. Esta saída serviu para treinar o modelo de IA do sensor e testar o modelo de IA de imagem, que já havia sido treinado previamente.

Nesta saída, foi levado a campo o protótipo final do *hardware*, contendo o microcontrolador, *datalogger*, sensor e módulo Bluetooth. A alimentação do protótipo foi feita via cabo USB. O veículo foi o mesmo da primeira saída, um Jeep Renegade. Nesta saída, o protótipo foi posicionado no painel do veículo, devido à superfície de contato ser maior, possibilitando uma fixação mais efetiva do conjunto. Além disso, desejava-se posicionar o conjunto em um local de maior incidência da luz do Sol, com o objetivo de testar a resistência do equipamento ao calor. A orientação dos eixos foi a mesma que na primeira saída, indicada na Figura 5.3. A instalação do conjunto no interior do veículo é mostrada na Figura 5.10.

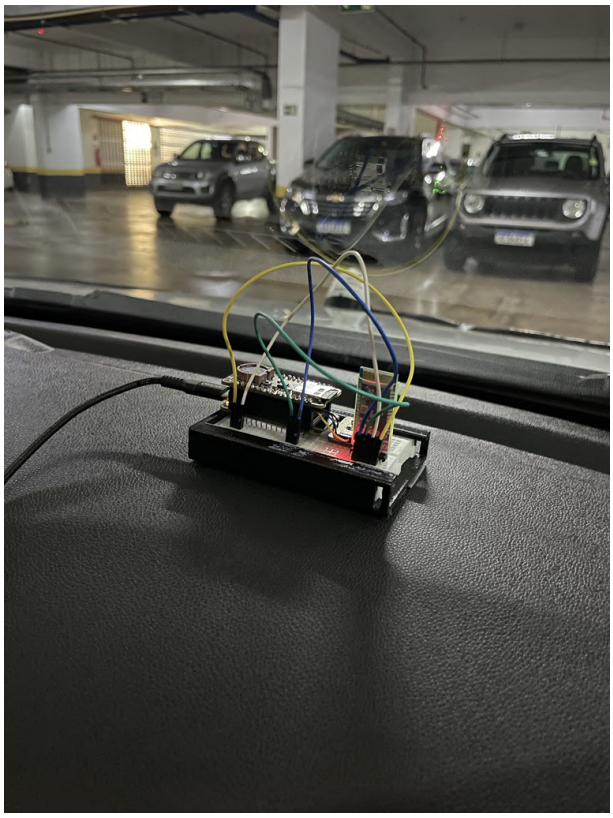

Figura 5.10: Instalação do protótipo dentro do veículo na segunda saída de campo.

5.2.1 Treinamento e resultados do modelo de IA do sensor

Assim como no modelo de IA de imagem, a função de perda utilizada foi a de entropia cruzada e o otimizador foi o SGD, com *learning rate* de 0,001 e *momentum* de 0,9. Para o treinamento do modelo, foram fornecidos os dados originais coletados pelo sensor em seus 6 eixos, sem qualquer tipo de filtragem em frequência.

Seleção das amostras

Para criar o *dataset*, o procedimento consistia em analisar o vídeo gravado durante a saída de campo e selecionar trechos de pavimento perfeito e imperfeito. Então, a etapa seguinte consistia em analisar o sinal para verificar se, de fato, essas oscilações associadas a características do pavimento estavam presentes. Essa etapa era necessária pelo fato de que, eventualmente, o veículo poderia não ter passado por cima de um defeito. Neste caso, o sinal estaria sendo classificado como imperfeito, quando na verdade é perfeito. A ideia era mapear cada trecho selecionado no vídeo com as amostras correspondentes no sinal.

Para a análise no sinal, eram observados principalmente o eixo Z do acelerômetro e o eixo Y do giroscópio. Nesses dois eixos, as oscilações associadas à trepidação no pavimento eram mais evidentes. A Figura 5.11 ilustra um exemplo de um segmento do sinal contendo amostras perfeitas e imperfeitas.

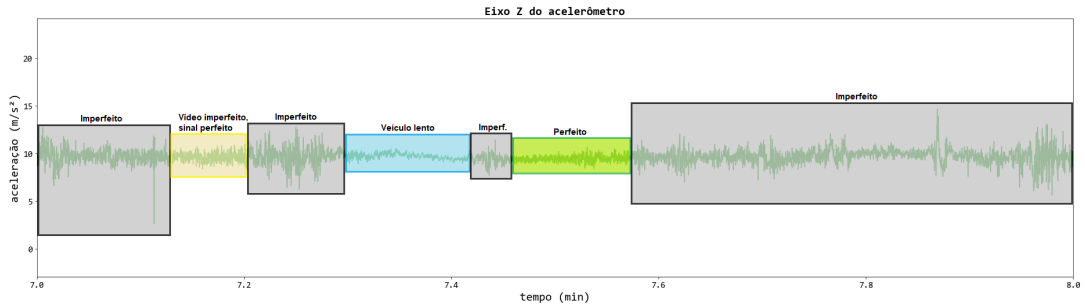

Figura 5.11: Exemplo de segmento do sinal contendo amostras perfeitas e imperfeitas.

Então, as amostras selecionadas eram rotuladas como perfeitas ou imperfeitas. A rotulação das amostras é feita de forma automatizada por meio de um algoritmo desenvolvido em Python. O algoritmo concatena todos os dados em um mesmo *dataframe*. São fornecidos para o algoritmo um arquivo contendo os instantes de tempo de início e fim correspondentes às amostras perfeitas selecionadas, e outro para as imperfeitas. Por meio da multiplicação do tempo pela frequência de amostragem, são obtidos os índices das amostras. Com isso, o algoritmo percorre todo o *dataframe* rotulando as amostras cujos índices estiverem dentro desses intervalos estabelecidos.

Após a seleção das amostras, foi iniciado o treinamento do modelo. O treinamento foi dividido em duas partes. A primeira parte do treinamento foi realizada com um *dataset* reduzido, com o objetivo de avaliar a influência dos hiperparâmetros e de diferentes tamanhos de janelas de amostras no desempenho do modelo. A segunda parte foi realizada com o *dataset* completo, tendo em vista obter os resultados definitivos do modelo.

Primeira etapa do treinamento do modelo

Nesta etapa, foram variados alguns hiperparâmetros e testados diferentes tamanhos de janelas. Observou-se que o treinamento deste modelo demandava um número maior de épocas para convergir em comparação com o modelo de IA de imagem, necessitando aproximadamente entre 90 a 200 épocas. Conforme abordado na Seção 4.2, o tamanho da janela corresponde à quantidade de amostras que compõem cada entrada fornecida para a rede. Portanto, a ideia inicial era definir uma janela de amostras cuja extensão fosse equivalente à distância entre eixos do veículo. Para o Jeep Renegade, a distância entre eixos é de aproximadamente 2,57 metros. [43]

Considerando uma velocidade média de 60 km/h e uma frequência de amostragem média do sensor de 109 Hz, pode-se calcular a distância percorrida em termos da quantidade de amostras coletadas pela expressão mostrada na Equação 5.1, em que N é a quantidade de amostras, ΔS é a distância percorrida em metros, v é a velocidade do veículo em m/s e f_s é a taxa de amostragem do sensor.

$$N = \frac{\Delta S}{v} \cdot f_s \tag{5.1}$$

Portanto, uma distância de 2,57 metros corresponde a uma janela de aproximadamente 17 amostras. Porém, esse valor é um número ímpar e primo. Para a realização das operações de multiplicação nos filtros convolucionais, em termos de custo computacional, é preferível que o tamanho da entrada seja uma potência de 2. Por isso, optou-se por definir uma janela de 32 amostras, que é a potência de 2 mais próxima acima de 17. Uma janela de 32 amostras corresponde à aproximadamente 4,89 metros percorridos, com o veículo a 60 km/h.

Para fins de comparação, testou-se também uma janela de 64 amostras, correspondente a uma distância de aproximadamente 9,78 metros, com o veículo a 60 km/h. Dessa forma, foram feitos testes com janelas associadas a deslocamentos de 5 e 10 metros, aproximadamente. O propósito de realizar esses testes fundamenta-se na consideração de que o veículo não estará constantemente a uma velocidade de 60 km/h. Para velocidades menores, pode acontecer de o veículo percorrer uma distância menor que a estabelecida como a mínima (2,57 metros) para o intervalo de uma janela.

Nesta primeira etapa do treinamento, foi utilizado um *dataset* com **34608 amostras**, sendo **19831 perfeitas (57,3%)** e **14777 imperfeitas (42,7%)**. O modelo foi treinado inicialmente para uma janela de 32 amostras, e os resultados obtidos foram os seguintes:

Resultados do treinamento para janela de 32 amostras

- **Tamanho da janela: 32 amostras**
- **Precisão de treinamento: 84,52%**
- **Precisão de validação: 85,19%**
- **Precisão de teste: 82,82%**

As Figuras 5.12 a 5.14 ilustram o custo, a precisão e a matriz de confusão associados ao modelo, respectivamente.

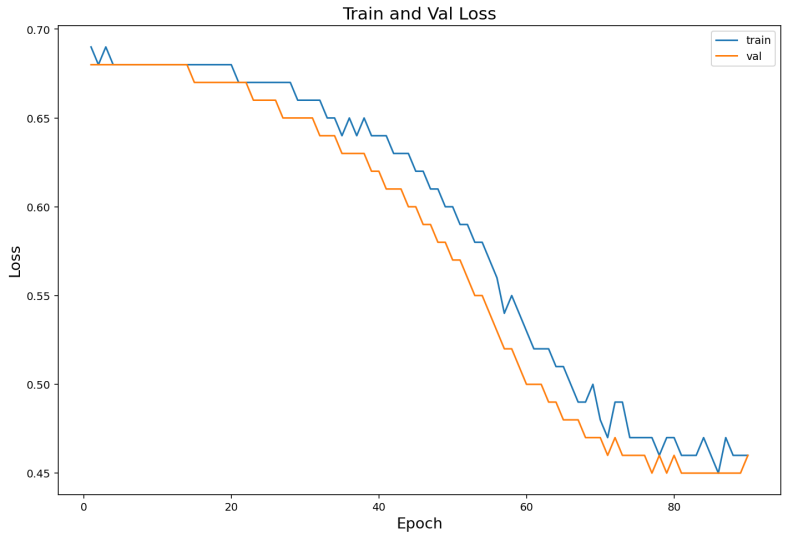

Figura 5.12: Custo do treinamento do modelo de IA do sensor para janela de 32 amostras.

Figura 5.13: Precisão do treinamento do modelo de IA do sensor para janela de 32 amostras.

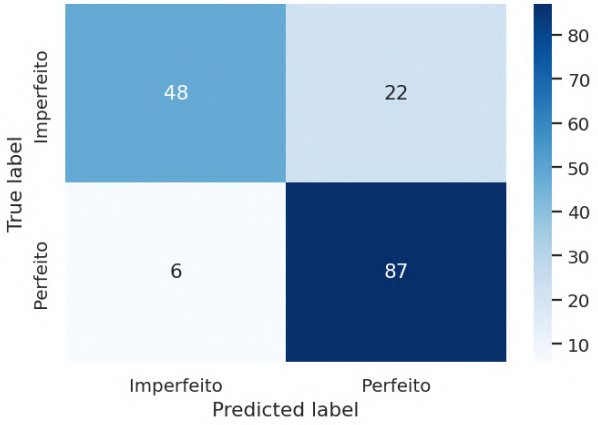

Figura 5.14: Matriz de confusão para o treinamento do modelo de IA do sensor com janela de 32 amostras.

Então, aumentando-se o tamanho da janela para 64 amostras e mantendo todos os outros parâmetros iguais, o modelo foi treinado novamente. Os resultados obtidos foram os seguintes:

Resultados do treinamento para janela de 64 amostras

- **Tamanho da janela: 64 amostras**
- **Precisão de treinamento: 85,19%**
- **Precisão de validação: 87,65%**

- **Precisão de teste: 87,65%**

As Figuras 5.15 a 5.17 ilustram o custo, a precisão e a matriz de confusão associados ao modelo, respectivamente.

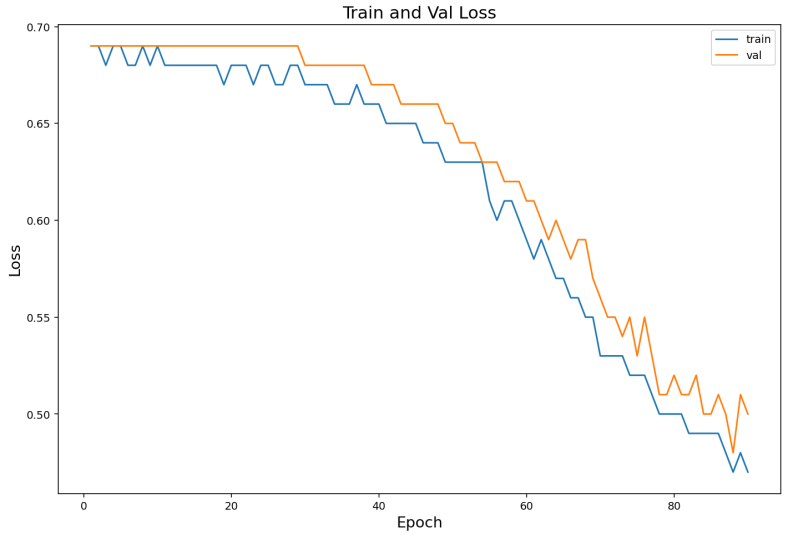

Figura 5.15: Custo do treinamento do modelo de IA do sensor para janela de 64 amostras.

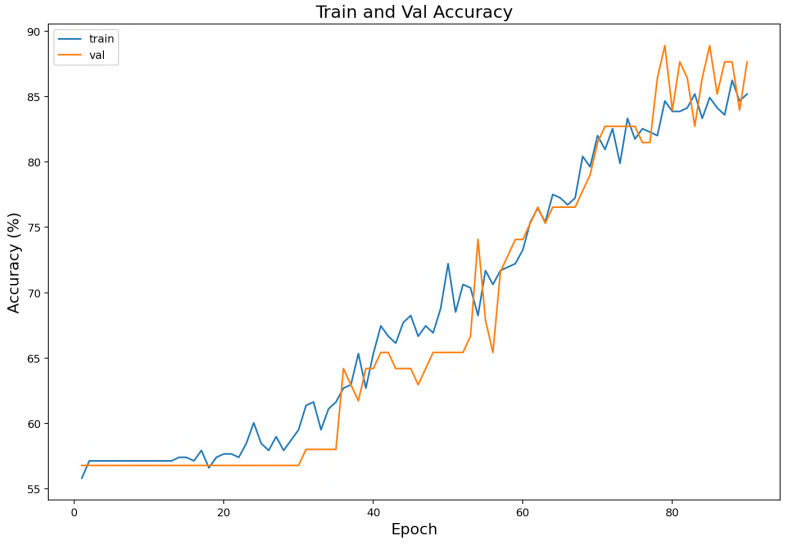

Figura 5.16: Precisão do treinamento do modelo de IA do sensor para janela de 64 amostras.

Observa-se que o desempenho do modelo para janelas de 64 amostras é levemente superior que com janelas de 32 amostras, especialmente na classificação de amostras imperfeitas, como pode ser visto na Figura 5.17. Dessa maneira, optou-se por estabelecer o tamanho da janela em 64 amostras para os treinamentos seguintes.

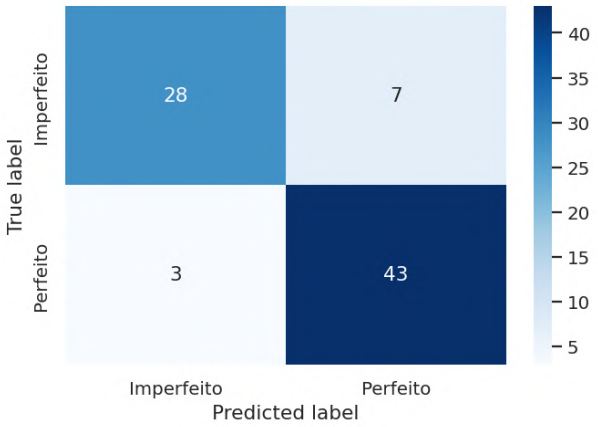

Figura 5.17: Matriz de confusão para o treinamento do modelo de IA do sensor com janela de 64 amostras.

Segunda etapa do treinamento do modelo

Na segunda etapa do treinamento, foi utilizado o *dataset* completo, composto por **78439 amostras**, sendo **24992 perfeitas (31,9%)** e **53447 imperfeitas (68,1%)**. A extensão percorrida durante a saída de campo consistia predominantemente em trechos de pavimentos imperfeitos. Por isso, o *dataset* completo estava consideravelmente desbalanceado. No entanto, todo esse conjunto de dados foi fornecido para a rede para verificar seu desempenho. Os resultados obtidos foram os seguintes:

Resultados do treinamento para o *dataset* completo

- **Tamanho da janela: 64 amostras**
- **Precisão de treinamento: 87%**
- **Precisão de validação: 83,11%**
- **Precisão de teste: 85,33%**

As Figuras 5.18 a 5.20 ilustram o custo, a precisão e a matriz de confusão associados ao modelo, respectivamente.

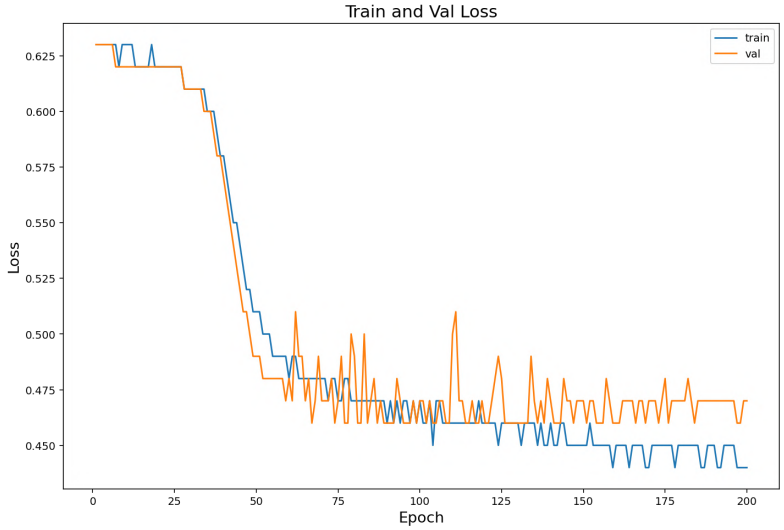

Figura 5.18: Custo do treinamento do modelo de IA do sensor para o *dataset* completo.

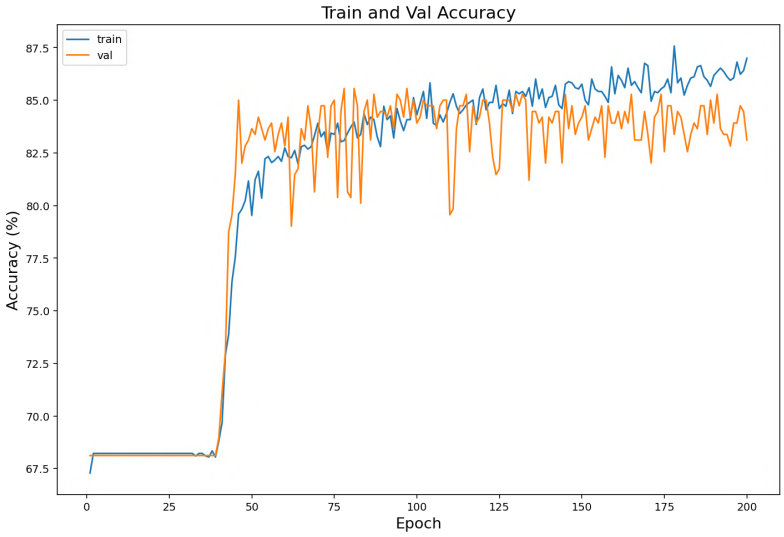

Figura 5.19: Precisão do treinamento do modelo de IA do sensor para o *dataset* completo.

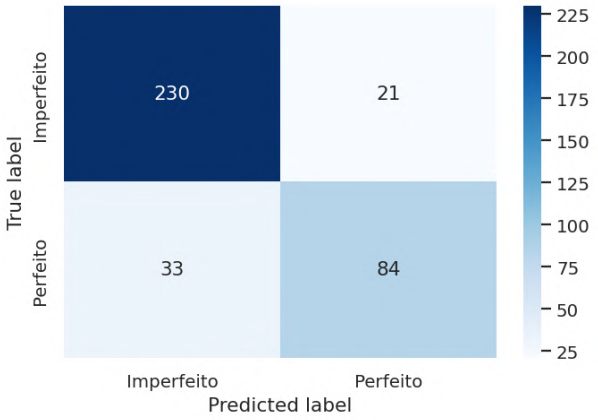

Figura 5.20: Matriz de confusão para o treinamento do modelo de IA do sensor com o *dataset* completo.

Observa-se que a precisão caiu um pouco, principalmente na classificação de amostras perfeitas, possivelmente por conta da maior proporção de amostras imperfeitas no *dataset* de treinamento.

Analisando de forma minuciosa o *dataset*, verificou-se que alguns trechos classificados como imperfeitos continham amostras perfeitas e vice-versa. Então, foi feito um refinamento manual do *dataset*, com o intuito de eliminar as amostras classificadas de forma incorreta e balancear mais o *dataset*. Após esse refinamento, o novo *dataset* constituiu-se de **55144 amostras**, sendo **27332 perfeitas (49,6%)** e **27812 imperfeitas (50,4%)**. Os resultados obtidos neste segundo treinamento foram os seguintes:

Resultados do treinamento após o refinamento do *dataset*

- **Tamanho da janela: 64 amostras**
- **Precisão de treinamento: 89,29%**
- **Precisão de validação: 91,09%**
- **Precisão de teste: 89,96%**

As Figuras 5.21 a 5.23 ilustram o custo, a precisão e a matriz de confusão associados ao modelo, respectivamente.

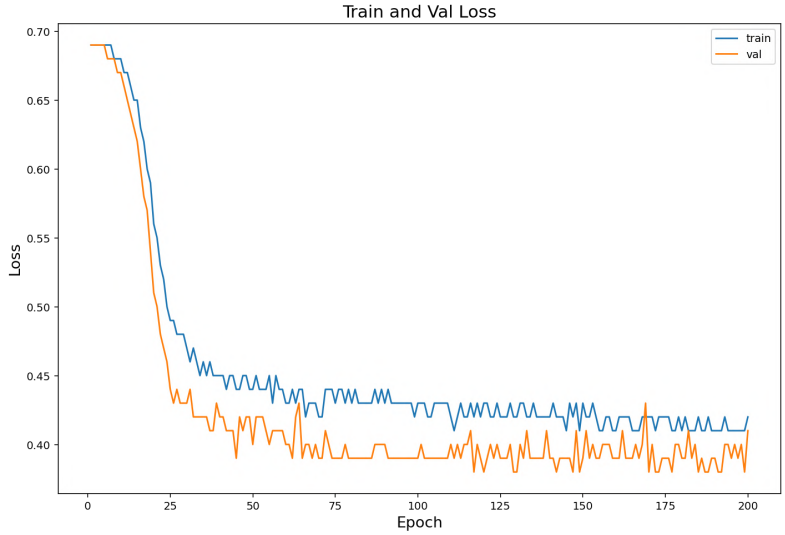

Figura 5.21: Custo do treinamento do modelo de IA do sensor após o refinamento do *dataset*.

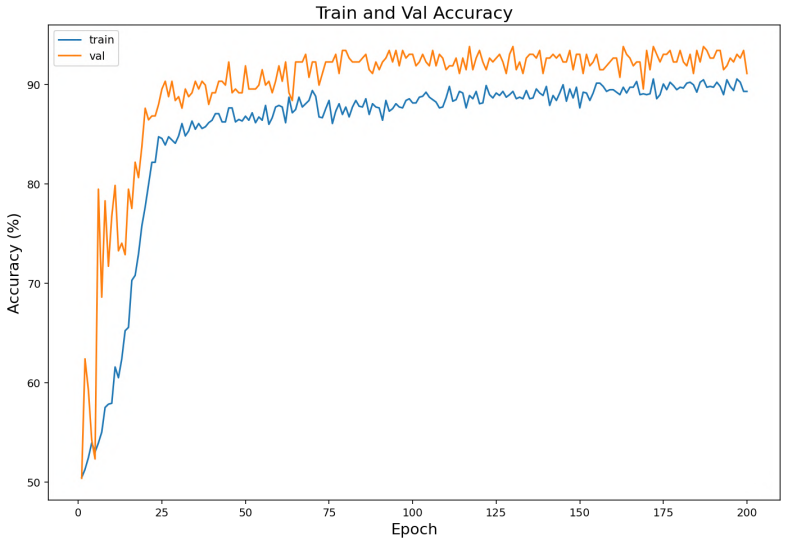

Figura 5.22: Precisão do treinamento do modelo de IA do sensor após o refinamento do *dataset*.

Após este refinamento do *dataset*, observa-se que a precisão do modelo aumentou consideravelmente, atingindo um nível considerado satisfatório.

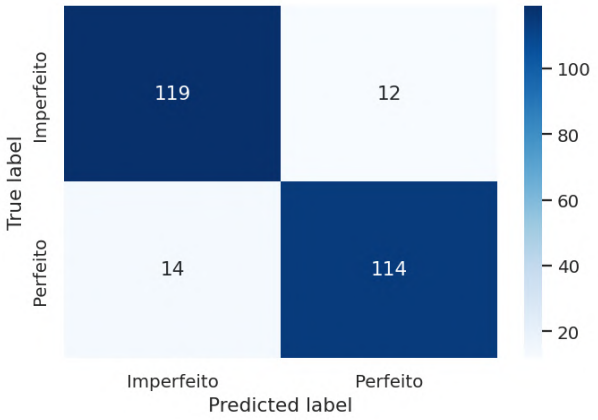

Figura 5.23: Matriz de confusão para o treinamento do modelo de IA do sensor após o refinamento do *dataset*.

Por fim, foi realizado um tratamento nos dados, com o intuito de remover eventuais *offsets* presentes em algum dos eixos. Como até o momento ainda não há um protocolo estabelecido para o posicionamento do protótipo no interior do veículo, é provável que este posicionamento não seja idêntico em testes futuros, gerando *offsets* diferentes nos dados. O tratamento realizado consistiu em subtrair o sinal original pela sua média. Então, o modelo foi treinado novamente e os resultados obtidos foram os seguintes:

Resultados do treinamento após a remoção do *offset* dos dados

- **Tamanho da janela: 64 amostras**
- **Precisão de treinamento: 91,87%**
- **Precisão de validação: 92,64%**
- **Precisão de teste: 90,35%**

As Figuras 5.24 a 5.26 ilustram o custo, a precisão e a matriz de confusão associados ao modelo, respectivamente.

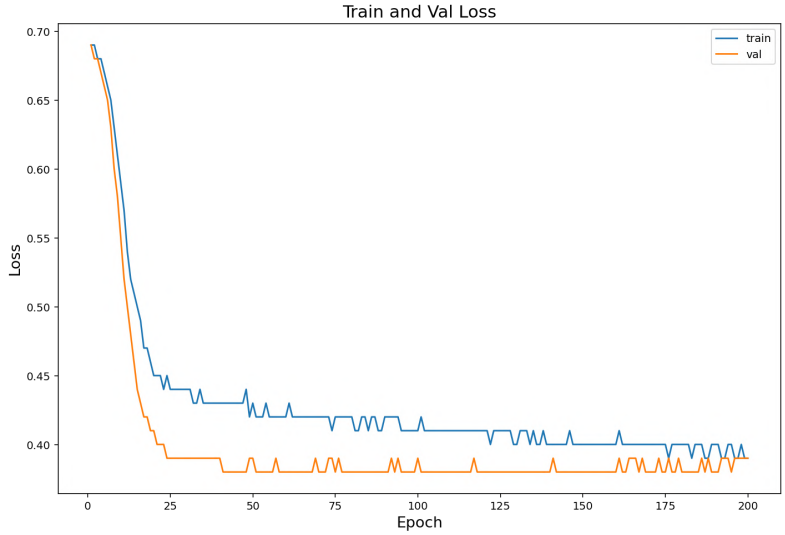

Figura 5.24: Custo do treinamento do modelo de IA do sensor após a remoção do *offset*.

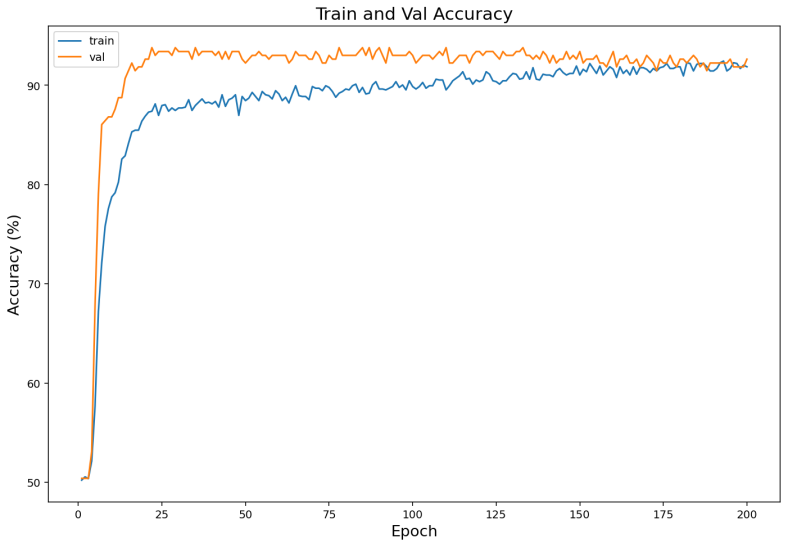

Figura 5.25: Precisão do treinamento do modelo de IA do sensor após a remoção do *offset*.

Após a remoção do *offset* dos dados, observa-se que a precisão aumentou ainda mais. Além disso, a velocidade de convergência do treinamento aumentou consideravelmente, como pode ser visto nas Figuras 5.24 a 5.25.

Para fins de visualização, foram separadas duas janelas presentes no conjunto de teste, uma classificada pelo modelo como perfeita e a outra como imperfeita. As janelas são mostradas nas Figuras 5.27 a 5.28, com relação ao eixo Z do acelerômetro.

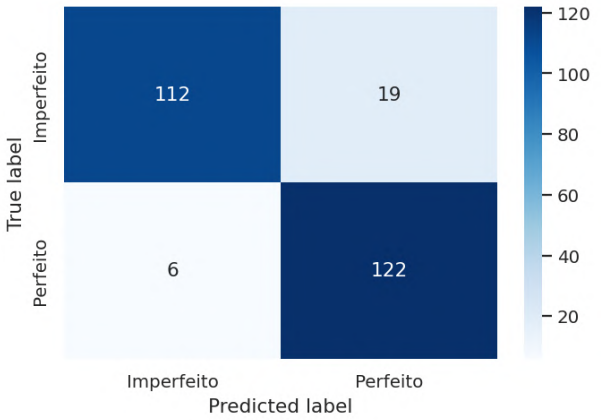

Figura 5.26: Matriz de confusão para o treinamento do modelo de IA do sensor após a remoção do *offset*.

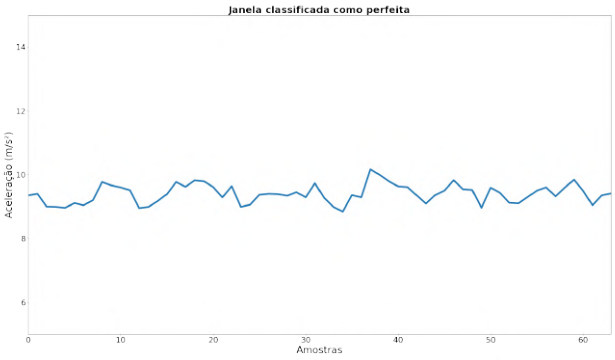

Figura 5.27: Exemplo de janela classificada como perfeita pelo modelo.

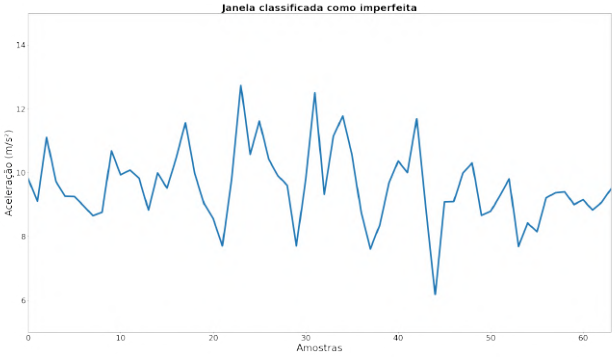

Figura 5.28: Exemplo de janela classificada como imperfeita pelo modelo.

Os resultados obtidos neste último treinamento constituem a base conclusiva para esta pesquisa, relativamente à IA do sensor. Na seção seguinte, será descrito como o modelo de IA de imagem foi testado durante a saída de campo.

5.2.2 Teste com o modelo de IA de imagem

As imagens coletadas nesta saída não foram utilizadas para treinamento do modelo, sendo reservadas exclusivamente para teste. As imagens foram extraídas do vídeo da saída a uma taxa de 2 frames por segundo. Então, foram selecionados e rotulados manualmente 500 frames, sendo 250 perfeitos e 250 imperfeitos. Os frames foram fornecidos ao modelo para realizar a classificação. Os resultados obtidos foram os seguintes:

- **Precisão: 87,2%**

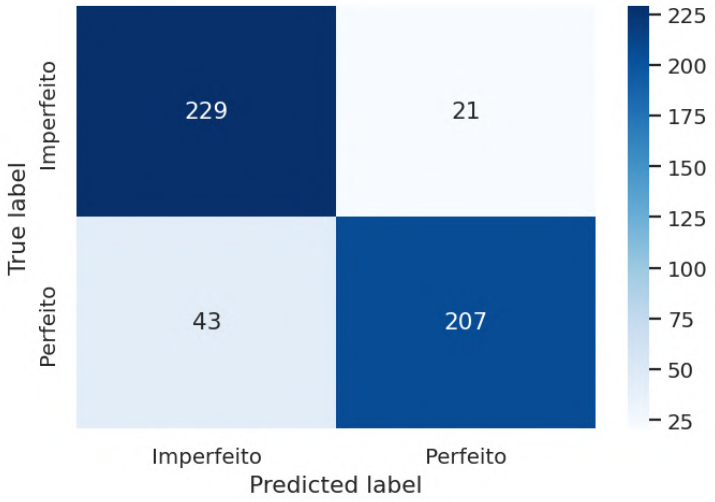

Figura 5.29: Matriz de confusão para o teste da IA de imagem na saída de campo.

Observa-se que a precisão caiu um pouco em relação à precisão de teste do último treinamento realizado. No entanto, este valor ainda é considerado satisfatório. Vale ressaltar também que esta saída de campo foi realizada em um horário mais tardio, sendo possível que as condições de iluminação do ambiente tenham exercido influência na classificação do modelo.

As Figuras 5.30 a 5.31 ilustram exemplos de frames classificados pelo modelo.

5.2.3 Resultados finais

Ao final do trabalho e de todos os testes realizados, foram obtidos os resultados finais apresentados na Tabela 5.1.

Figura 5.30: Exemplo de frame classificado como perfeito.

Figura 5.31: Exemplo de frame classificado como imperfeito.

Tabela 5.1: Resultados finais de ambos modelos de IA.

Precisão para o modelo de IA de imagem	**Precisão para o modelo de IA do sensor**
90,33%	90,35%

6 Conclusão e Trabalhos Futuros

Neste trabalho, foi possível integrar as áreas de sistemas embarcados, processamento de sinais e Inteligência Artificial no âmbito da Engenharia Elétrica. Conforme descrito na Seção 1.1, o escopo deste estudo consistiu no desenvolvimento de um classificador binário para as condições do pavimento.

O primeiro objetivo do trabalho, descrito na Seção 1.1.2, consistia em desenvolver um protótipo de *hardware* embarcado utilizando microcontrolador e sensor acelerômetro/giroscópio. Conforme detalhado na Seção 3.2, este objetivo foi alcançado, comprovando a viabilidade do projeto. Os outros dois objetivos consistiam em desenvolver modelos de IA para realizar a classificação do pavimento. Um com base em imagens, e outro com base em dados do sensor. Os testes e resultados apresentados nos Capítulos 4 e 5 corroboram a consecução destes objetivos também, com um desempenho satisfatório de ambos modelos.

Contudo, é importante ressaltar que este é um trabalho inicial. O treinamento de ambos modelos foi feito com um *dataset* pequeno. E, com relação ao modelo do sensor, o treinamento ocorreu em um contexto bastante específico. Para futuros trabalhos, é válido testar a capacidade de generalização dos modelos.

Como perspectivas relevantes para futuros estudos, sugere-se correlacionar os resultados de ambos modelos, para gerar um resultado final mais preciso acerca da condição do pavimento. Além disso, uma outra sugestão para trabalhos futuros consiste em realizar a classificação em um número maior de categorias, separando a classe de imperfeitos por tipo de defeito.

Referências

[1] *I2C Pressure Sensor | Pa6DC*. https://appmeas.co.uk/products/pressure-sensors/i2c-pressure-sensor/. Acessado em: 24/09/2023. ix, 4

[2] Zelenovsky, Ricardo: *T10 A02 v1 - I2C com MSP430*, Set. 2020. https://youtu.be/-h50bUcfZ5I. ix, 4, 5

[3] Lathi, Bhagwandas Pannalal: *Sinais e Sistemas Lineares - 2ª Ed.* Bookman, 2006. ix, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 19

[4] Brandão, Eric: *Proc. de sinais - aula 42: Transformada discreta de fourier - definição*, Dez. 2020. https://youtu.be/B7OVlVCyJPc?si=uacDF8wD_YRUDF79. ix, 16

[5] *Perceptron: A Simple yet Mighty Machine Learning Algorithm*. https://medium.com/@cprasenjit32/perceptron-a-simple-yet-mighty-machine-learning-algorithm-9ff6b7d86a71. Acessado em: 30/10/2023. ix, 18

[6] *Sigmoid function*. https://en.wikipedia.org/wiki/Sigmoid_function. Acessado em: 13/11/2023. ix, 19

[7] *Tangente hiperbólica*. https://pt.wikipedia.org/wiki/Tangente_hiperbólica. Acessado em: 13/11/2023. ix, 19

[8] Furtado, Maria Inês Vasconcellos: *Redes neurais artificiais: uma abordagem para sala de aula*. Ponta Grossa, PR. Atena Editora, página 19, 2019. ix, 18, 20, 21

[9] Online, Stanford: *Stanford CS229: Machine Learning - Linear Regression and Gradient Descent | Lecture 2 (Autumn 2018)*, Abril 2020. https://youtu.be/4b4MUYve_U8?si=Jged3Jpss9sXMqgz. ix, 21, 22

[10] Mitesh M. Khapra: *CS7015 (Deep Learning): Lecture 11*. http://www.cse.iitm.ac.in/~miteshk/CS7015/Slides/Teaching/pdf/Lecture11.pdf. Acessado em: 12/11/2023. ix, 24, 25

[11] Zhang, Aston, Zachary C. Lipton, Mu Li e Alexander J. Smola: *Dive into Deep Learning*. Cambridge University Press, 2023. https://D2L.ai. ix, 24, 26

[12] *CNN | Introduction to Pooling Layer*. https://www.geeksforgeeks.org/cnn-introduction-to-pooling-layer/. Acessado em: 12/11/2023. ix, 26

[13] He, Kaiming, Xiangyu Zhang, Shaoqing Ren e Jian Sun: *Deep residual learning for image recognition*. Em *Proceedings of the IEEE conference on computer vision and pattern recognition*, páginas 770–778, 2016. ix, 27

[14] *Adafruit ESP32-S2 Feather - 4 MB Flash + 2 MB PSRAM - STEMMA QT / Qwiic.* https://www.adafruit.com/product/5000#technical-details. Acessado em: 14/11/2023. ix, 28

[15] *Adalogger FeatherWing - RTC + SD Add-on For All Feather Boards.* https://www.adafruit.com/product/2922. Acessado em: 15/11/2023. ix, 29

[16] *Adafruit ISM330DHCX - 6 DoF IMU - Accelerometer and Gyroscope - STEMMA QT / Qwiic.* https://www.adafruit.com/product/4502. Acessado em: 15/11/2023. ix, 30

[17] *Módulo Bluetooth 3.0 JDY-31.* https://www.eletrogate.com/modulo-bluetooth-30-jdy-31. Acessado em: 19/11/2023. ix, 31

[18] *Lithium Ion Polymer Battery - 3.7v 500mAh.* https://www.adafruit.com/product/1578. Acessado em: 15/11/2023. ix, 31, 32

[19] CNT; SEST; SENAT (2022) Pesquisa CNT de Rodovias 2022: relatório gerencial. 2022. Confederação Nacional do Transporte; Serviço Social do Transporte; Serviço Nacional de Aprendizagem do Transporte, 229p, Brasília. 1

[20] Bannatyne, RASS e Greg Viot: *Introduction to microcontrollers.* Em *WESCON/97 Conference Proceedings*, páginas 564–574. IEEE, 1997. 3

[21] Gridling, Gunther e Bettina Weiss: *Introduction to microcontrollers.* Vienna University of Technology Institute of Computer Engineering Embedded Computing Systems Group, 2007. 3, 4, 5, 6

[22] Zelenovsky, Ricardo: *T12 A01 v1 - UART do MSP430*, Out. 2020. https://youtu.be/RZhe0-guQss. 5

[23] McLeod, Robert M: *The generalized Riemann integral*, volume 20. American Mathematical Soc., 1980. 12

[24] Brandão, Eric: *Proc. de sinais - aula 43: Fast Fourier Transform (FFT)*, Dez. 2020. https://youtu.be/yvw5VwID2W4?si=qV7x-8puNe1AjPJo. 17

[25] Moreland, Kenneth e Edward Angel: *The FFT on a GPU.* Em *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, páginas 112–119, 2003. 17

[26] Rosenblatt, Frank: *The perceptron: a probabilistic model for information storage and organization in the brain.* Psychological review, 65(6):386, 1958. 18

[27] Karsoliya, Saurabh: *Approximating number of hidden layer neurons in multiple hidden layer BPNN architecture.* International Journal of Engineering Trends and Technology, 3(6):714–717, 2012. 20

[28] Bhatt, Bhavesh: *Log Loss or Cross-Entropy Cost Function in Logistic Regression*, Abril 2019. https://youtu.be/MztgenIfGgM?si=icRzC74giVkq-bBK. 23

[29] Online, Stanford: *Lecture 12 - Backprop and Improving Neural Networks | Stanford CS229: Machine Learning (Autumn 2018)*, Abril 2020. https://youtu.be/zUazLXZZA2U?si=MUj4KuvMfiV6RQOB. 23

[30] Haykin, Simon: *Redes Neurais: Princípios e prática*. Bookman Editora, 2001. 23

[31] LeCun, Yann, Yoshua Bengio e Geoffrey Hinton: *Deep learning*. nature, 521(7553):436–444, 2015. 27

[32] Espressif Systems: *ESP32-S2 Family Datasheet*, 2020. https://cdn-learn.adafruit.com/assets/assets/000/096/705/original/esp32-s2_datasheet_en.pdf?1604350607. 28

[33] *CircuitPython*. https://circuitpython.org/. Acessado em: 15/11/2023. 29

[34] *Adafruit LSM6DSOX 6 DoF Accelerometer and Gyroscope - STEMMA QT / Qwiic*. https://www.adafruit.com/product/4438. Acessado em: 15/11/2023. 30

[35] *JDY-31 Bluetooth Module*, 2019. http://myosuploads3.banggood.com/products/20190129/20190129043725SKUA87502.pdf. 31

[36] *STEMMA QT / Qwiic JST SH 4-Pin Cable - 50mm Long*. https://www.adafruit.com/product/4399. Acessado em: 19/11/2023. 32

[37] *Flutter documentation*. https://docs.flutter.dev/. Acessado em: 19/11/2023. 37

[38] *RESNET50*. https://pytorch.org/vision/main/models/generated/torchvision.models.resnet50.html. Acessado em: 22/11/2023. 41

[39] Sutskever, Ilya, James Martens, George Dahl e Geoffrey Hinton: *On the importance of initialization and momentum in deep learning*. Em *International conference on machine learning*, páginas 1139–1147. PMLR, 2013. 42

[40] Rouse, David M e Sheila S Hemami: *Understanding and simplifying the structural similarity metric*. Em *2008 15th IEEE international conference on image processing*, páginas 1188–1191. IEEE, 2008. 43

[41] Kant, Laxmi: *Human Activity Recognition Using Accelerometer Data*, Set. 2019. https://github.com/laxmimerit/Human-Activity-Recognition-Using-Accelerometer-Data-and-CNN. 50

[42] Kwapisz, Jennifer R, Gary M Weiss e Samuel A Moore: *Activity recognition using cell phone accelerometers*. ACM SigKDD Explorations Newsletter, 12(2):74–82, 2011. 50

[43] *Jeep Renegade 1.3 T270 2024: Ficha Técnica*. https://www.icarros.com.br/jeep/renegade/ficha-tecnica. Acessado em: 07/12/2023. 60