



Universidade Federal da Bahia
Escola Politécnica
Colegiado do Curso de Eng. Elétrica



Maurício Taffarel Barreto da Silva

Aprendizado de Máquina em Microcontroladores Utilizando TinyML: Exploração, Otimização e Portabilidade

Orientador: Prof. Dr. Paulo César Machado de Abreu Farias

Salvador-BA – Brasil
15 de dezembro de 2023

Maurício Taffarel Barreto da Silva

Aprendizado de Máquina em Microcontroladores Utilizando TinyML: Exploração, Otimização e Portabilidade

Projeto apresentado ao Curso de Graduação em Engenharia Elétrica da Universidade Federal da Bahia como parte dos requisitos para a obtenção do grau de Engenheiro Eletricista.

Orientador: Prof. Dr. Paulo César Machado de Abreu Farias

Salvador-BA – Brasil

15 de dezembro de 2023

Maurício Taffarel Barreto da Silva

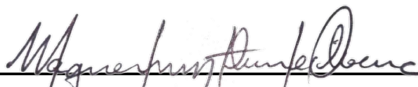
Aprendizado de Máquina em Microcontroladores Utilizando TinyML: Exploração, Otimização e Portabilidade

Projeto apresentado ao Curso de Graduação em Engenharia Elétrica da Universidade Federal da Bahia como parte dos requisitos para a obtenção do grau de Engenheiro(a) Eletricista.

Trabalho aprovado. Salvador-Ba – Brasil, 15 de dezembro de 2023:



Orientador: Prof. Dr. Paulo César Machado de
Abreu Farias



Prof. Dr. Wagner Luiz Alves de Oliveira



Prof. Dr. Tiago Trindade Ribeiro

Salvador-Ba – Brasil
15 de dezembro de 2023

*Este trabalho é dedicado a todos aqueles que acreditam no poder do conhecimento para
criar um mundo melhor.*

Agradecimentos

Agradeço, em primeiro lugar, à minha família pela educação e suporte: aos meus pais, Nilza e Josué, à minha irmã, Maressa, e à minha avó, Maria. Jamais teria concluído este curso sem o apoio deles.

À minha namorada, Nicolle, por estar ao meu lado em todos os momentos desta longa trajetória do curso, pelo suporte, compreensão e paciência.

Aos meus colegas de turma que caminharam junto comigo em cada etapa desta formação: Alípio, Bia, Claudelino, Coutinho, Handley, Isaac, Josias, Lucas, Nathane, Mariana, Staffa, Thácio e Veloso. Gostaria de agradecer em especial à Breno e Geraldo, pelos eternos momentos de estudos e parceria.

Aos mestres Aninha, Castilho, Edson, Felipe, Lu, Paulo César e Toinho por toda paciência e ensinamentos que me permitiram crescer como Engenheiro Eletricista.

Por fim, ao meu orientador, professor Dr. Paulo César Machado de Abreu Farias, pelas suas orientações valiosas.

*“Toda criança começa
como um cientista nato”.*
(Carl Sagan)

Resumo

A técnica TinyML refere-se ao conjunto de abordagens que viabilizam a implementação de algoritmos de aprendizado de máquina em dispositivos com recursos computacionais e capacidade de memória restritos, como sistemas embarcados. Este trabalho abordou duas maneiras de implementar tais técnicas como otimização e compactação de modelos, explorando diferentes tecnologias. Além disso, foram apresentados detalhes específicos relacionados a essa abordagem do TinyML no processo de desenvolvimento, com ênfase na portabilidade e escalabilidade. A avaliação da solução proposta permitirá analisar o impacto e a eficácia do uso do TinyML na implementação de sistemas de aprendizado de máquina em microcontroladores com recursos limitados.

Palavras-chave: TinyML, Inteligência Artificial, Sistemas Embarcados, Portabilidade.

Abstract

The TinyML technique refers to the set of approaches that enable the implementation of machine learning algorithms on devices with restricted computational resources and memory capacity, such as embedded systems. This work addressed two ways to implement such techniques as optimizations and model compression, exploring different technologies. Additionally, specific details related to this approach to TinyML in the development process were presented, with an emphasis on portability and scalability. The evaluation of the proposed solution will allow analyzing the impact and effectiveness of using TinyML in implementing machine learning systems on microcontrollers with limited resources.

Keywords: TinyML, Artificial Intelligence, Embedded Systems, Portability.

Lista de ilustrações

Fig. 1 – Estrutura da Inteligência Artificial	27
Fig. 2 – Estrutura do neurônio artificial	28
Fig. 3 – Classificação de entradas para $n = 2$	30
Fig. 4 – Dados para treinamento e validação	36
Fig. 5 – Estrutura do modelo proposto	37
Fig. 6 – Exemplo de áudio obtido	41
Fig. 7 – Dados coletados no Edge Impulse	42
Fig. 8 – Arquitetura do modelo para o Edge Impulse	43
Fig. 9 – Implantação no Arduino Nano 33 BLE Sense	43
Fig. 10 – Treinamento e Validação	45
Fig. 11 – Novas predições	45
Fig. 12 – Quantização de faixa dinâmica	46
Fig. 13 – Quantização com amostras representativas	46
Fig. 14 – Características de Energia do Mel-Filterbank Energy (MFE)	47
Fig. 15 – Ponderação dos pesos da Fast Fourier Transform (FFT)	47
Fig. 16 – Fonte: Edge Impulse	48
Fig. 17 – Matriz de confusão	48
Fig. 18 – Exploração de dados de treinamento	49
Fig. 19 – Estimativa de Desempenho no Arduino Nano 33 BLE Sense	50
Fig. 20 – Simulando ambiente real	51
Fig. 21 – Visão geral do desempenho	51

Lista de tabelas

Tab. 1 – Comparação do desempenho para diferentes estratégias de otimização diretamente no TensorFlow Lite	46
---	----

Lista de abreviaturas e siglas

ARM	Advanced RISC Machine
API	Application Programming Interface
CNN	Convolutional Neural Network
FAR	False Acceptance Rate
FRR	False Rejection Rate
FFT	Fast Fourier Transform
IA	Inteligência Artificial
IoT	Internet of Things
MFE	Mel-Filterbank Energy
RAIL	Responsible AI Licenses
SoC	System on Chip
t-SNE	t-Distributed Stochastic Neighbor Embedding

Lista de símbolos

ω	Letra grega ômega
o	Letra grega ômicron
θ	Letra grega theta
ϕ	Letra grega phi
η	Letra grega eta
\in	Pertence

Sumário

1	INTRODUÇÃO	25
2	FUNDAMENTAÇÃO TEÓRICA	27
2.1	Aprendizado de Máquina	27
2.2	Algoritmos Tradicionais vs. Aprendizado de Máquina	27
2.3	O neurônio artificial: Perceptron	28
2.4	Processo de aprendizagem de uma neurônio artificial	30
2.5	Ética em Aprendizado de Máquina	31
2.6	Aprendizado de máquina em Sistemas Embarcados	33
2.7	Quantização	34
2.7.0.1	Quantização de faixa dinâmica	34
2.7.0.2	Quantização com amostra representativa	34
2.7.0.3	Quantização inteira completa	34
3	MATERIAIS E MÉTODOS	35
3.1	TensorFlow Lite	35
3.1.1	Criação de um modelo simples utilizando Python e Keras	36
3.1.2	Conversão para um modelo TensorFlow Lite	37
3.1.3	Otimizações de modelo	38
3.1.4	Programando o microcontrolador ESP-32s	39
3.1.5	Conclusões	39
3.2	Edge Impulse	40
3.2.1	Obtenção dos dados de áudio para treinamento	40
3.2.2	Extração de características	41
3.2.3	Treinamento do Impulse	42
3.2.4	Implementando no microcontrolador	43
3.2.5	Conclusões	44
4	RESULTADOS E DISCUSSÕES	45
4.1	TensorFlow Lite no ESP-32s	45
4.1.1	Treinamento	45
4.1.2	Conversão e Otimização	46
4.2	Edge Impulse no Arduino Nano BLE 33 Sense	47
4.2.1	Extração de características	47
4.2.2	Treinamento	48
4.2.3	Implantação do firmware	52

5	CONCLUSÕES	53
	REFERÊNCIAS	55

1 Introdução

Machine Learning, ou Aprendizado de Máquina, é uma área da inteligência artificial que visa desenvolver algoritmos e técnicas computacionais capazes de tomar decisões e realizar tarefas complexas com base em conjuntos de dados.

Reconhecimento de padrões, previsão de eventos, classificação de dados e até automação de processos são alguns dos usos de Machine Learning em diversas áreas. Os algoritmos de Machine Learning tradicionalmente eram desenvolvidos e usados em computadores mais robustos capazes de lidar com grandes quantidades de dados e processamento.

No entanto, recentemente uma técnica de Machine Learning que tem crescido bastante é o TinyML que permite a implementação de algoritmos em dispositivos que não tem tantos recursos computacionais e capacidade de memória. Isto foi possível graças as técnicas de otimização e compactação de modelos, somado a isso também existe a preocupação com o consumo energético, pois os sistemas de TinyML geralmente estão associados a problemas com sistemas embarcados e aplicações de IoT em que o gasto de energia é uma desafio.

Assim, o TinyML se mostra uma tema relevante para o desenvolvimento de novas tecnologias para os próximos anos principalmente para áreas de Internet das Coisas (IoT) e Sistemas Embarcados com a capacidade de realizar inferências em ambientes isolados e tomar decisões autônomas.

O TinyML permite a implementação de modelos de *Machine Learning* (ML) em microcontroladores de baixo desempenho e com restrições de memória ([ARM-TINYML, 2021](#)). Além disso, esses sistemas consomem apenas alguns miliwatts de energia, no máximo ([RAY, 2022](#)). Isso significa que é possível executar tarefas de ML em dispositivos pequenos e com recursos limitados, abrindo um leque de possibilidades para aplicações de Inteligência Artificial em ambientes de borda. ([BAO et al., 2021](#)) ([UTVMSYSTEM, 2020](#)).

O TinyML apresenta-se como uma solução para enfrentar vários desafios encontrados na implementação de sistemas de Machine Learning em aplicações industriais ([RAY, 2022](#)):

- **Consumo elevado de energia:** O TinyML permite a execução eficiente de modelos de Machine Learning em microcontroladores com baixo consumo de energia, possibilitando o desenvolvimento de aplicações com maior eficiência energética.
- **Problemas com privacidade de dados:** Ao processar os dados localmente nos dispositivos de borda, o TinyML reduz a necessidade de enviar informações sensíveis para serviços na nuvem, proporcionando maior controle e segurança dos dados.

- **Dependência de conexão:** Com o TinyML, os modelos de Machine Learning podem ser implantados diretamente nos dispositivos, permitindo que as aplicações funcionem independentemente de uma conexão constante com a internet, garantindo maior disponibilidade e confiabilidade.
- **Latência:** A execução local dos modelos de ML no TinyML reduz a latência, uma vez que as inferências são feitas no próprio dispositivo, evitando a necessidade de enviar dados para servidores remotos e aguardar o processamento.

Essas vantagens do TinyML tornam-no uma escolha promissora para a implementação de soluções de Machine Learning em ambientes industriais e também no contexto da Internet das Coisas (IoT). A capacidade de executar modelos de Machine Learning em dispositivos de borda com recursos limitados oferece benefícios significativos para a IoT ([RAY, 2022](#)).

O objetivo geral deste trabalho é explorar as possibilidades e apresentar uma visão geral sobre o TinyML. Desta forma, espera-se como objetivos específicos abordar temas como os desafios enfrentados ao implantar modelos de ML em dispositivos de baixo consumo, os benefícios e aplicações potenciais dessa tecnologia, as principais técnicas para implementação de TinyML.

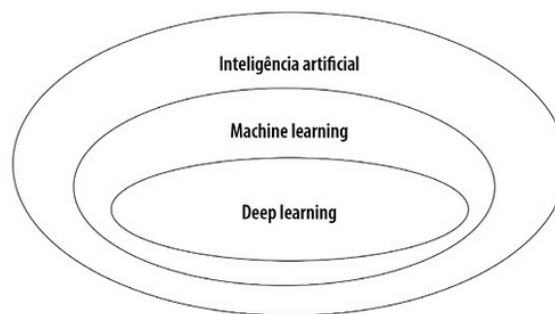
2 Fundamentação Teórica

A inteligência artificial pode ser definida por um conjunto de algoritmos que realizam função que tipicamente são atribuídas à inteligência humana e permitem que máquinas executem tarefas complexas, como percepção cognitiva, tomada de decisão e comunicação (SCHIEFERDECKER; GROßMANN; SCHNEIDER, 2019). Esta área tem se tornado cada vez mais relevante em todo o mundo. Aplicações como assistentes virtuais, carros autônomos, reconhecimento facial e sistemas de recomendação são apenas alguns exemplos desse avanço tecnológico (YOUNIS, 2023) (YANG et al., 2020).

2.1 Aprendizado de Máquina

O aprendizado de máquinas é um subcampo da inteligência artificial (vide Figura 1) que permite às máquinas tomar decisões e resolver problemas complexos sem a necessidade de programação explícita (BAE et al., 2019). Por meio de algoritmos especializados, as máquinas têm a capacidade de processar dados, realizando uma análise que lhes permite aprender padrões ou executar tarefas específicas, sem a necessidade de serem programadas explicitamente para cada uma dessas finalidades.

Fig. 1 – Estrutura da Inteligência Artificial



Fonte: (TAULLI, 2019)

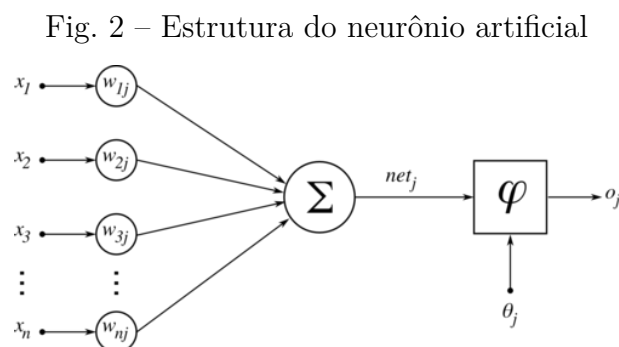
2.2 Algoritmos Tradicionais vs. Aprendizado de Máquina

Algoritmos tradicionais são baseados em regras programadas manualmente para realizar tarefas específicas. Por outro lado, o aprendizado de máquina permite que as máquinas aprendam com os dados e se adaptem a diferentes situações, permitindo tomadas de decisão mais precisas e personalizadas. Existem três categorias principais de aprendizado de máquinas (SHARMA; SINGH; SINGH, 2019):

- **Aprendizado supervisionado:** o algoritmo aprende a partir de um conjunto de exemplos rotulados, onde os dados de entrada estão associados a um rótulo de saída correspondente.
- **Aprendizado não supervisionado:** o algoritmo aprende a partir de um conjunto de dados não rotulados, identificando padrões e estruturas sem a utilização de rótulos ou informações externas.
- **Aprendizado por reforço:** o algoritmo aprende a tomar decisões e realizar ações com base em um sistema de recompensas e punições, maximizando assim a recompensa total ao longo do tempo.

2.3 O neurônio artificial: Perceptron

Inspirado no que se sabia até o momento sobre o cérebro humano e o modelo do neurônio biológico, foram realizadas diversas tentativas de realizá-lo matematicamente. O primeiro modelo que se tem registro é o dos pesquisadores McCulloch e Pitts no ano de 1943 (SCHMIDHUBER, 2015). Mais tarde, inspirado nas pesquisas destes pesquisadores, em 1958, o cientista Frank Rosenblatt desenvolveu o que o mesmo denominou como Perceptron (ROSENBLATT, 1958). Este modelo realiza uma soma ponderada de diversas entradas e calcula a saída através de uma função de ativação, a Figura 2 apresenta a estrutura de um Perceptron.



Disponível em: <https://commons.wikimedia.org/wiki/File:ArtificialNeuronModel.png>

Os impulsos elétricos dos neurônios no modelo artificial são representados pelas entradas $x_1, x_2, x_3, \dots, x_n$. Há entradas que excitam mais ou menos o neurônio artificial. A influência de cada entrada é determinada pelos pesos sinápticos $\omega_{1j}, \omega_{2j}, \omega_{3j}, \dots, \omega_{nj}$ onde j se refere ao neurônio de estudo e n ao peso do terminal de entrada de mesmo índice, estes pesos podem ter diferentes intensidades (valor do peso), assim como sua característica excitatória ou inibitória (através do sinal do peso). O somatório das entradas com seus respectivos pesos sinápticos, serão aplicadas na função de ativação φ , adicionando também

como argumento, um limiar de ativação θ_j para então produzir a saída do neurônio o_j , que é a resposta deste neurônio devido a todas as entradas.

Assim, utilizando como base a Figura 2, pode-se estabelecer equações que modelam um neurônio artificial:

$$\begin{aligned} net_j(t) &= \sum_{i=1}^n w_{ij}(t) \cdot x_i(t) \\ o_j(t) &= \varphi(net_j(t), \theta_j(t)) \end{aligned} \quad (2.1)$$

Algumas simplificações podem ser feitas para facilitar a implementação computacional. O limiar de ativação $\theta_j(t)$ pode incorporado ao somatório $net_j(t)$ da equação (2.1) na função de ativação para que $o_j(t)$ seja função apenas de $net_j(t)$, sendo ainda possível definir este limiar como se fosse uma das entradas da rede neural associada a seu respectivo um peso sináptico unitário ω_0 .

$$net'_j(t) := \theta_j(t) + \sum_{i=1}^n w_{ij}(t) \cdot x_i(t) = \omega_{0j} \cdot 1 + \sum_{i=1}^n w_{ij}(t) \cdot x_i(t) \quad (2.2)$$

$$net'_j(t) := \sum_{i=0}^n w_{ij}(t) \cdot x_i(t) \quad (2.3)$$

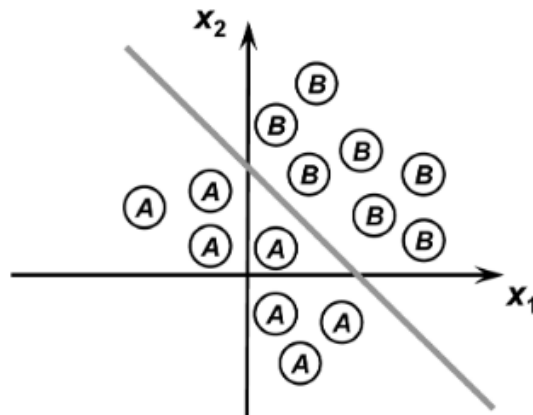
Geralmente a função de ativação que determina a saída do neurônio artificial, ela é determinada com base em diversos critérios. Uma opção usual pode ser fazer $o(t) \in [-1, 1]$. Neste caso, o Perceptron é utilizado para reconhecimento de padrões e classificação dedados dentre duas classes de amostras, com sua saída apresentando duas possibilidades de valores: verdadeiro ou falso, o que pode ser representado pela função degrau ou degrau bipolar, esta última expressa da seguinte forma:

$$o_j(t) = \varphi(net_j(t)) = \begin{cases} 1, & \text{se } net_j(t) \geq 0 \\ -1, & \text{se } net_j(t) < 0 \end{cases} \quad (2.4)$$

Neste caso, o limite da condição ou *threshold* é definido quando $net'_j(t) = 0$, ou seja, quando $\theta_j(t) + \sum_{i=1}^n w_{ij}(t) \cdot x_i(t) = 0$. Ao considerar $n = 2$, fazendo o Perceptron ter duas entradas, obtemos o seguinte limite de condição:

$$x_1(t) \cdot \omega_{1j}(t) + x_2(t) \cdot \omega_{2j}(t) + \theta_j(t) = 0 \quad (2.5)$$

Deste modo, uma vez atribuído os pesos sinápticos ω_{ij} assim como o limiar de ativação $\theta_j(t)$, é possível perceber que a fronteira de decisão do Perceptron é uma equação linear, podendo separar classes de amostras para novas entradas $x_n(t)$ (desde que sejam separáveis). Um exemplo para duas entradas está indicado na Figura 3.

Fig. 3 – Classificação de entradas para $n = 2$ 

Fonte: (SILVA, 2010)

2.4 Processo de aprendizagem de uma neurônio artificial

O aprendizado de uma rede neural pode ser compreendido ao observarmos o processo de treinamento de um Perceptron em um contexto de aprendizado supervisionado. O processo se dá através do ajuste de pesos sinápticos. Tomando um conjunto de amostras de entradas e de saídas, diz-se que uma rede neural aprendeu quando é capaz de determinar os pesos sinápticos e o limiar de ativação de tal forma que para todas (ou para quase todas) as entradas de amostras, as saídas determinadas pelo neurônio artificial são iguais às saídas de amostra. O processo de aprendizagem de uma rede neural é chamado de treinamento. A regra de aprendizado de Hebb (SILVA, 2010) é assim descrita:

When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased (HEBB... , 1950).

Donald Hebb, 1949

Esta é conhecida como a regra de aprendizado de Hebb (SILVA, 2010). O processo de aprendizado pode ser realizado seguindo estes princípios, em que pode-se implementar um algoritmo que incrementa (ou decrementa) o peso e o limiar de ativação um valor proporcional à diferença entre a saída obtida e a saída esperada e aos sinais de entrada caso estas saídas tenham valores diferentes. Este processo é repetido até que todas as amostras (ou boa parte delas) para o treinamento tenham as respostas de saídas iguais às respostas esperadas. Matematicamente, utilizando a equação (2.3), o processo de aprendizagem pode ser definido em notação vetorial na equação (2.6).

$$\begin{cases} \omega_i^{atual} = \omega_i^{anterior} + \eta \cdot (\mathbf{v}^k - \mathbf{o}^k) \cdot \mathbf{x}_i^k \\ \theta^{atual} = \theta^{anterior} + \eta \cdot (\mathbf{v}^k - \mathbf{o}^k) \cdot 1 \end{cases} \quad (2.6)$$

Nota-se a presença de um fator η no incremento (ou decremento) dos pesos e limiar de ativação, o qual é denominado taxa de aprendizagem, que define a velocidade de convergência do treinamento: quanto maior esta taxa, mais rápido a rede aprenderá e viceversa. Contudo, com um η muito grande pode causar instabilidades no processo de treinamento, normalmente a taxa de aprendizagem da rede é escolhida entre o intervalo $0 < \eta < 1$. As variáveis \mathbf{v}_k e \mathbf{o}_k se referem a valores desejados e valores obtidos com os pesos sinápticos atuais para a amostra k , respectivamente. Através das simplificações sintetizadas na equação (2.3), chega-se à notação computacional representada nas equações (2.7) e (2.8).

$$\boldsymbol{\omega} \leftarrow \boldsymbol{\omega} + \eta \cdot (\mathbf{v}^k - \mathbf{o}^k) \cdot \mathbf{x}^k \quad (2.7)$$

$$\boldsymbol{\omega} = \begin{bmatrix} \theta \\ \omega_1 \\ \omega_2 \\ \vdots \\ \omega_n \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \quad (2.8)$$

Desta forma, a implementação do computacional pode ser feita de acordo com o simples algoritmo 1 (SILVA, 2010).

2.5 Ética em Aprendizado de Máquina

Com a crescente utilização da inteligência artificial e do aprendizado de máquina em diversas áreas, é fundamental garantir que essas tecnologias sejam éticas e imparciais. É preciso tomar cuidado para que os algoritmos não perpetuem preconceitos, discriminação ou desigualdades existentes na sociedade (SICART; SHKLOVSKI; JONES, 2022). Outros aspectos éticos cruciais incluem o consentimento informado, a privacidade e segurança dos dados. A auditabilidade também é uma preocupação fundamental na área de aprendizado de máquina.

Para combater esses problemas, é importante que os desenvolvedores e pesquisadores adotem práticas de transparência e responsabilidade. Eles devem garantir que os dados de treinamento sejam representativos da diversidade da população e que sejam corrigidos vieses de dados existentes. Além disso, a colaboração interdisciplinar entre profissionais de

Algorithm 1 Treinamento de um Perceptron (Adaptado e desenvolvido (TAFFAREL, 2022))

Require: x_k, v_k
Ensure: $y = f(x')$

$x \leftarrow [x_1, x_2, x_3, \dots, x_n]$ ▷ Obter entradas de treinamento
 $v \leftarrow [v_1, v_2, v_3, \dots, v_n]$ ▷ Obter saídas desejadas para as respectivas entradas
 $w \leftarrow [0, 0, \dots, 0]$ ▷ Inicia o vetor w com valores iniciais
 $\eta \leftarrow 0.1$ ▷ Define um valor para taxa de aprendizagem
 $e \leftarrow 0$ ▷ Define variável contadora de épocas
 $aprendeu \leftarrow false$

while not aprendeu **do**
 $y \leftarrow o(w, x)$ ▷ Calcula as saídas com base nos pesos sinápticos atuais
if $y \neq v$ **then** ▷ Se não resultar em respostas feis
 $w \leftarrow f(w, x, \eta)$ ▷ Ajusta pesos sinápticos
else
 $aprendeu \leftarrow true$
end if
 $e \leftarrow e + 1$ ▷ Incrementa a época (pode ser usado para limitar iterações máximas)
end while

ética, direito, sociologia e outras áreas é essencial para abordar questões éticas complexas relacionadas ao aprendizado de máquina.

Uma iniciativa tomada pela Comissão Europeia foi a criação das Diretrizes de Ética para a Inteligência Artificial Confiável pelo Grupo de Especialistas de Alto Nível em Inteligência Artificial (IA). Estas diretrizes estabelecem um conjunto de 7 principais requisitos que os sistemas de IA devem atender para serem considerados confiáveis. Uma lista de avaliação específica foi desenvolvida para auxiliar na verificação da aplicação de cada um desses requisitos (ETHICS..., 2019).

Devido às possibilidades de uso prejudicial das tecnologias baseadas em inteligência artificial, as empresas também adotaram a iniciativa de utilizar licenças responsáveis para IA (RESPONSIBLE..., 2023). Licenças Responsáveis de IA - *Responsible IA Licenses* (Responsible AI Licenses (RAIL)) - (FAQ..., 2023) possibilitam os desenvolvedores a restringir o uso de sua tecnologia de IA, a fim de prevenir aplicações irresponsáveis e prejudiciais (CONTRACTOR et al., 2022).

2.6 Aprendizado de máquina em Sistemas Embarcados

Nos últimos anos, houve avanços significativos na eficiência dos algoritmos de aprendizado de máquina. Agora é possível executá-los em dispositivos menores e com menor consumo energético, ao invés de depender exclusivamente de redes neurais complexas e computadores poderosos. (AJANI; IMOIZE; ATAYERO, 2021)

Um sistema embarcado é qualquer sistema que esteja embutido ou incorporado dentro de outros sistemas elétricos ou mecânicos. Esses sistemas embutidos são projetados para realizar tarefas específicas e estão presentes em uma variedade de dispositivos, como eletrodomésticos, sistemas de controle industrial e até mesmo em dispositivos médicos. No entanto, quando se trata de integração de aprendizado de máquina nesses sistemas, o tamanho e a potência limitada dos dispositivos embarcados apresentam desafios significativos (AJANI; IMOIZE; ATAYERO, 2021).

O aprendizado de máquina tradicionalmente é realizado em sistemas computacionais com maior capacidade de processamento e memória. No entanto, com os avanços na área de Tiny Machine Learning, ou simplesmente TinyML, é possível implantar modelos de aprendizado de máquina em dispositivos embarcados com recursos restritos de energia, memória e processamento. O TinyML pode ser definido grosso modo através dos limites de potência energética: geralmente costuma-se utilizar 1mW ou menos para dizer-se que um dispositivo está dentro do escopo do TinyML (WARDEN, 2020) (JOHNNY, 2021).

Essa abordagem de aprendizado de máquina em sistemas embarcados traz uma série de benefícios e aplicações. Em termos de benefícios, a utilização de TinyML permite que sistemas embarcados realizem previsões em tempo real e automaticamente, sem a necessidade de uma conexão constante com a nuvem. Além disso, o TinyML também oferece maior privacidade nos aplicativos de aprendizado de máquina, pois reduz a necessidade de enviar grandes quantidades de informações para a nuvem além de ser um agente para desenvolvimento de tecnologias mais sustentáveis. (NOLAN, 2023) (PRAKASH et al., 2023)

Em relação às aplicações, a área de TinyML tem despertado um grande interesse e vem sendo considerada uma tecnologia disruptiva que acelerará a adoção generalizada do aprendizado de máquina em diversas indústrias e na sociedade como um todo. Pode-se listar algumas aplicações como: reconhecimento de imagem, reconhecimento de fala, processamento de linguagem de sinais, detecção de expressões faciais, reconhecimento de gestos manuais, localização de palavras-chave, agricultura de precisão, monitoramento de tartarugas marinhas, detecção de tosse, monitoramento ecológico, controle de tráfego, previsão ambiental, análise de sintomas respiratórios, monitoramento de veículos autônomos e ativação por voz (RAY, 2022).

2.7 Quantização

Uma forma de compactar o modelo é quantizar os pesos, isto permite reduzir o tamanho do modelo sem degradar tanto a sua precisão. Existem duas formas de quantização: quantização pós-treinamento e treinamento consciente de quantização. A primeira estratégia de quantização é feita após o treinamento (mais fácil de se usar) e a outra durante o treinamento, o qual é realizado considerando a quantização em tempo de inferência, o que gera modelos mais precisos ([TENSORFLOWLITE, 2023](#)).

Existem diversos tipos de quantização que podem ser usados. A escolha está intimamente ligada com o dispositivo alvo a ser implementado a rede neural. Usando como base os tipos de quantização utilizados nas principais bibliotecas computacionais, três formas bastante utilizadas são descritas na sequência.

2.7.0.1 Quantização de faixa dinâmica

Com esta estratégia quantiza apenas os pesos de ponto flutuante para inteiro para 8 bits de precisão, o que permite uma redução do uso da memória e na computação envolvida no processo. Isto pode ser utilizado em arquiteturas de processadores mais simples, pois você consegue implementar as redes com representações de apenas 8 bits.

2.7.0.2 Quantização com amostra representativa

Através de uma amostra de pesos, pode-se obter maior compatibilidade entre dispositivos, além de compactar mais os modelos reduzindo o uso de memória e tendo a possibilidade de melhoria de latência. Para este método um conjunto de amostras representativas é fornecido entre um intervalo.

2.7.0.3 Quantização inteira completa

É possível também, impor uma quantização inteira completa para entrada e saída. Isto pode ser utilizado para maior compatibilidade em microcontroladores de 8 bits, por exemplo.

3 Materiais e Métodos

Para o desenvolvimento de modelos de TinyML, é essencial ter ferramentas adequadas que permitam treinar e implementar redes neurais em dispositivos com recursos limitados. Há várias opções disponíveis para desenvolvimento em TinyML, incluindo bibliotecas de software e plataformas específicas. Algumas dessas ferramentas são:

1. **TensorFlow Lite**: Biblioteca móvel que permite a execução de modelos TensorFlow em dispositivos móveis, incorporados e Internet of Things (IoT). Ele oferece inferência de aprendizado de máquina no dispositivo com baixa latência e tamanho binário reduzido. Isso permite o suporte a dispositivos ainda mais limitados, como microcontroladores ([TFLITE, 2023](#)).
2. **uTensor**: Framework de inferência de aprendizado de máquina com a proposta de ser extremamente leve, construído sobre o Tensorflow e otimizado para alvos Advanced RISC Machine (ARM) ([UTENSOR, 2023](#)).
3. **EdgeImpulse**: Plataforma de desenvolvimento para soluções de machine learning em sistemas embarcados, a qual permite o treinamento de estruturas e desenvolvimento diretamente do navegador com pouco uso de codificação ([IMPULSE, 2023](#)).
4. **NanoEdge AI Studio**: Plataforma de desenvolvimento para soluções de TinyML para STM32. Ela oferece suporte para treinamento de modelos de aprendizado de máquina e implementação em dispositivos com recursos limitados ([NANOEDGE-AISTUDIO, 2023](#)).

PyTorch Mobile, Embedded Learning Library (ELL), STM32Cube.AI, ITVM e CMSIS-NN são outras opções de frameworks e bibliotecas disponíveis para o desenvolvimento em TinyML ([RAY, 2022](#)). Neste trabalho serão investigadas implementações de redes neurais para as plataformas TensorFlow Lite e EdgeImpulse.

3.1 TensorFlow Lite

A ferramenta TensorFlow foi utilizada neste trabalho, pois boa parte das demais ferramentas ou a utilizam ou tem um funcionamento muito semelhante, o TensorFlow Lite é uma biblioteca que compõe o ecossistema do TensorFlow e permite a implementação de modelos de machine learning para diferentes dispositivos móveis e de borda.

Neste trabalho foi utilizado o TensorFlow Lite para microcontroladores, o qual é construído de tal forma que se exija pouco espaço em memória e não requer suporte a

sistemas operacionais, bibliotecas C ou C++ padrão nem a alocação de memória dinâmica. Requer o C++11 e uma plataforma de 32 bits, sendo amplamente testado em *Arm Cortex-M Series* (TENSORFLOW, 2023).

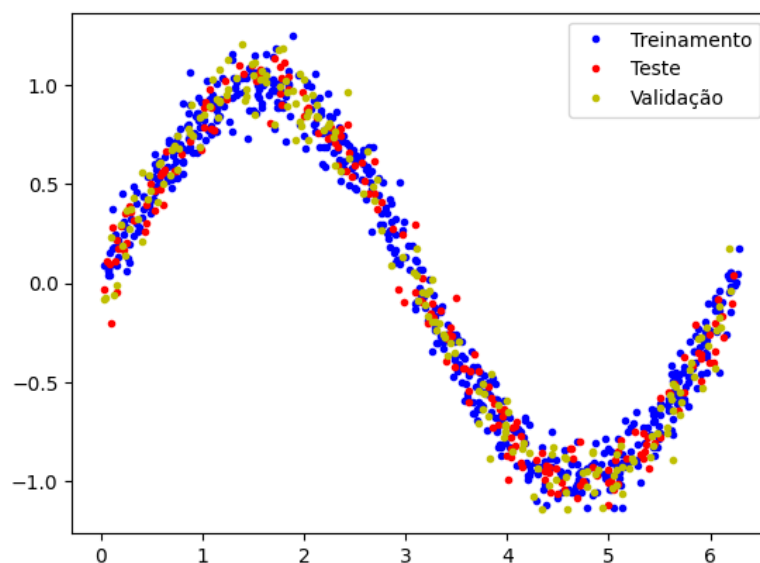
Além disso, o desenvolvimento de uma rede neural pode ser feito utilizando bibliotecas tradicionais e linguagens de alto nível, se aproximando do desenvolvimento de redes neurais convencionais.

3.1.1 Criação de um modelo simples utilizando Python e Keras

O modelo em questão se trata de um exemplo apresentado na documentação do TensorFlow Lite denominado *Hello World* com algumas adaptações. Se trata de um modelo de uma entrada e uma saída, para avaliar o valor de $\text{sen}(x)$ para $x \in [0, 2\pi]$. Embora seja um modelo simples, espera-se que ele possa demonstrar os conceitos fundamentais das etapas de implantação de redes em sistemas embarcados. Foram utilizados para compor a base de dados, um conjunto de 1000 amostras gerados através de uma distribuição uniforme entre $x = [0, 2\pi]$ e com um erro gaussiano com média 0 variância 0,1 apresentado na figura 4, divididos da seguinte maneira:

1. 60% utilizados para treinamento;
2. 20% utilizados para validação;
3. 20% utilizados para teste.

Fig. 4 – Dados para treinamento e validação



Fonte: Autoria própria

Para se construir a rede, foi utilizada a biblioteca Keras, a qual disponibiliza uma interface com o Python para desenvolvimento de redes neurais artificiais, capaz de rodar em cima de TensorFlow. A rede utilizada, assim como os parâmetros para compilação do modelo, estão apresentados no código 3.1 enquanto a visualização está representada na figura 5.

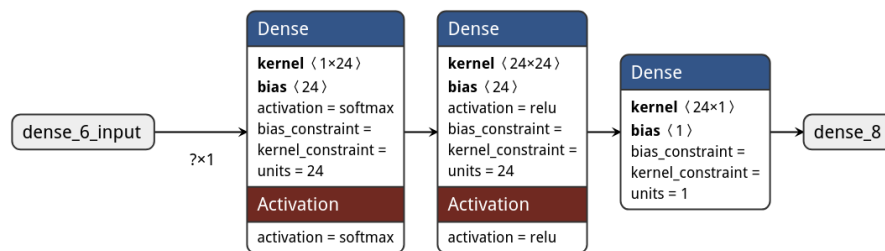
```

1 # Create a model
2 model = tf.keras.Sequential()
3 model.add(layers.Dense(24, activation='softmax', input_shape=(1,)))
4 model.add(layers.Dense(24, activation='relu'))
5 model.add(layers.Dense(1))
6
7 # Add optimizer, loss function, and metrics to model and compile it
8 model.compile(optimizer='rmsprop', loss='mae', metrics=['mae'])

```

Código 3.1 – Criação do modelo

Fig. 5 – Estrutura do modelo proposto



Fonte: Autoria própria (Renderizado por <https://netron.app/>)

3.1.2 Conversão para um modelo TensorFlow Lite

Após a realização do treinamento, o próximo passo é obter o modelo para ser carregado no microcontrolador. Para esta tarefa, a Application Programming Interface (API) do Tensorflow fornece o método `tf.lite.TFLiteConverter` para ser usado juntamente com um modelo *Keras* como mostrado no código 3.2.

```

1 # Convert Keras model to a tflite model
2 converter = tf.lite.TFLiteConverter.from_keras_model(model)
3 tflite_model = converter.convert()

```

Código 3.2 – Conversão para .tflite

Para implementar este modelo sem depender de sistemas de arquivos nos microcontroladores alvo, o mesmo é convertido também para uma representação em código C. O método utilizado para isso é o `convert_bytes_to_c_source` da `tensorflow.lite.python.util`. Este método retorna uma *string* representando um vetor de *char* constante C a partir do binário do modelo. O código 3.3 mostra um exemplo do uso deste método.

```

1 from tensorflow.lite.python.util import convert_bytes_to_c_source
2
3 source_text, header_text =
4     convert_bytes_to_c_source(tflite_model, "sine_model")
5 with open('sine_model.h', 'w') as file:
6     file.write(header_text)
7 with open('sine_model.cc', 'w') as file:
8     file.write(source_text)

```

Código 3.3 – Conversão para representação em bytes C

3.1.3 Otimizações de modelo

Para este trabalho, será considerada a estratégia de quantização pós treinamento com dois tipos de otimização. A quantização de faixa dinâmica quantiza apenas os pesos de ponto flutuante para inteiro para 8 bits de precisão e pode ser utilizada definindo o atributo `converter.optimizations` para `tf.lite.Optimize.DEFAULT` conforme mostrado no código 3.4.

```

1 import tensorflow as tf
2 converter = tf.lite.TFLiteConverter.from_saved_model(saved_model_dir)
3 converter.optimizations = [tf.lite.Optimize.DEFAULT]
4 tflite_quant_model = converter.convert()

```

Código 3.4 – Quantização de faixa dinâmica

Também foi utilizada a quantização através de uma amostra representativa, possibilitando compactar mais os modelos reduzindo o uso de memória e tendo a possibilidade de melhoria de latência. Para este método um conjunto de amostras representativas é fornecido conforme o código 3.5.

```

1 def representative_data_gen():
2     num_samples = 100
3     for _ in range(num_samples):
4         # Gere um valor de entrada entre 0 e 2pi
5         input_value = np.random.uniform(0, 2 * np.pi)
6         # Crie um array numpy com o valor de entrada
7         input_array = np.array([input_value], dtype=np.float32)
8         yield [input_array]
9 # Converter modelo Keras para tflite
10 converter = tf.lite.TFLiteConverter.from_keras_model(model)
11 converter.optimizations = [tf.lite.Optimize.DEFAULT]
12 converter.representative_dataset = representative_data_gen
13 tflite_model = converter.convert()

```

Código 3.5 – Quantização inteira completa

3.1.4 Programando o microcontrolador ESP-32s

De posse do modelo treinado, otimizado e convertido para C, a implementação foi realizada utilizando um microcontrolador ESP-32s. O suporte oficial à arquitetura, a popularidade deste microcontrolador e a ampla comunidade fizeram deste dispositivo uma escolha ideal para executar o modelo *Hello World* treinado anteriormente.

Para utilizar o interpretador TensorFlow para realizar as inferências no dispositivo, foi utilizada uma biblioteca *EloquentTinyML*, desenvolvida para a *Framework* Arduino. Desta forma foi estabelecido também um código de fácil entendimento com o uso da biblioteca TensorFlow abstraída. O código 3.6 apresenta o uso do método `predict()` para realizar as inferências.

```
1 #include <EloquentTinyML.h>
2 #include "sine_model.cc"
3
4 Eloquent::TinyML::TfLite<NUMBER_OF_INPUTS, NUMBER_OF_OUTPUTS,
5     TENSOR_ARENA_SIZE> ml;
6
7 void setup() {
8     ml.begin(sine_model);
9     unsigned long tic, toc, total_latency = 0, latency = 0;
10    for(int i=0; i<=NUMBER_OF_POINTS_INFERRING; i++) {
11        float x = 2*PI*i/NUMBER_OF_POINTS_INFERRING; float y = sin(x);
12        float input[1] = { x };
13        tic = micros(); float predicted = ml.predict(input);
14        toc = micros(); total_latency += toc - tic;
15        Serial.print(x,8); Serial.print(","); Serial.println(predicted,8);
16    }
17    latency = total_latency / NUMBER_OF_POINTS_INFERRING;
18    Serial.println("Latencia media da inferencia: ");
19    Serial.println(latency);
20 }
21 void loop() {}
```

Código 3.6 – Inferências no microcontrolador ESP-32s

3.1.5 Conclusões

Através deste exemplo, todo o processo de treinamento, otimização, conversão e inferência foram abordados além dos conceitos básicos deste processo. Os resultados serão discutidos no capítulo a seguir. Todo o processo gerou como produto um *workflow* e pode ser encontrado no repositório do GitHub <<https://github.com/taffarel55/graduation-thesis>>.

3.2 Edge Impulse

Uma outra ferramenta promissora para o desenvolvimento de modelos de machine learning em sistemas embarcados é o Edge Impulse <<https://edgeimpulse.com/>>. Através da plataforma de IA em Edge, é possível coletar e armazenar grandes quantidades de dados, treinar modelos e otimizar da mesma forma que o *workflow* do capítulo anterior.

Deste modo, a ferramenta se mostra de grande utilidade no desenvolvimento de novos produtos e soluções, pois permite com que modelos sejam implementados de forma mais rápida com colaboração entre equipe e portabilidade facilitada para diversos dispositivos.

O modelo desenvolvido foi capaz de reconhecer um determinado som ou uma frase, tratando-se de uma classificação de áudio. Para esta tarefa um microcontrolador com uma entrada de sinal analógico é necessário, razão pela qual foi utilizando outro dispositivo, o **Arduino Nano 33 BLE Sense Rev2** que contém uma série de sensores inclusive um microfone ultracompacto de baixo consumo (MP34DT06J). O dispositivo utiliza um System on Chip (SoC) nRF52840 que é construído em cima de um processador ARM® Cortex™-M4 com unidade de ponto flutuante a 64 MHz.

3.2.1 Obtenção dos dados de áudio para treinamento

O objetivo foi construir um modelo capaz de detectar a frase "Olá mundo", para isso, o primeiro passo é a obtenção de uma quantidade de áudios para treinamento. Foi levantado um conjunto de 50min de áudio para classificação de 3 tipos:

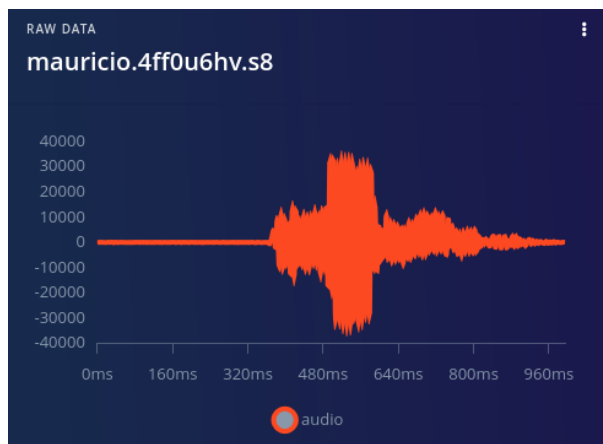
1. Rótulo: **ola-mundo** (1000 amostras de 1s de "Olá mundo")
2. Rótulo: **desconhecido** (1000 amostras de 1s de palavras desconhecidas)
3. Rótulo: **ruído** (1000 amostras de 1s de ruídos)

Os dados de classificação par "Olá mundo" foram obtidos através da gravação de cerca de 20 pessoas voluntárias de sexo e idades distintas com diferentes formas de entonação, velocidade e pronúncia. Esta base de dados foi levantada por mim e está disponível abertamente em <<https://github.com/taffarel55/graduation-thesis/tree/main/Edge%20Impulse/datasets>>. A figura 6 mostra o exemplo de um áudio capturado.

Para ruído foram gravados diferentes tipos de sinais aleatórios:

- Ruído branco: Densidade espectral de potência do sinal é constante para todas as frequências do sinal.

Fig. 6 – Exemplo de áudio obtido



Fonte: Edge Impulse

- Ruído rosa: Densidade espectral de potência do sinal é inversamente proporcional à frequências do sinal.
- Ruído marrom: Densidade espectral de potência do sinal é inversamente proporcional à frequências do sinal ao quadrado.

Além destes, foram usados ruídos típicos de ambientes como: restaurantes, shoppings, aeroporto, metrô, cozinha, conversas de fundo além de áudios silenciosos.

Por último, os áudios de palavras desconhecidas reunia um conjunto de palavras e frases diferentes de "Olá mundo". Estes dados foram obtidos do Edge Impulse e uma cópia pode ser obtida no repositório <<https://github.com/taffarel55/graduation-thesis>>. Todos os áudios tiveram uma frequência de amostragem de $16kHz$. Para subir os dados na plataforma, foi utilizado o Edge Impulse CLI conforme o código 3.7.

```
1 $ edge-impulse-uploader --clean --info-file info.labels
```

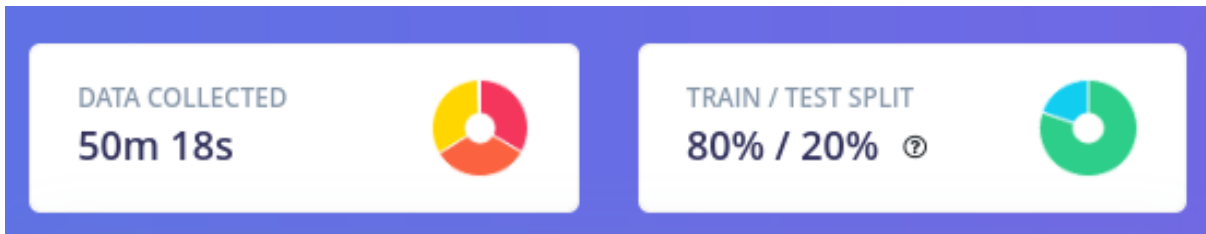
Código 3.7 – Subindo os dados para o Edge Impulse

Os dados foram categorizados e divididos em dados para treinamento e teste ficando aproximadamente $13min$ e $23s$ para treinamento e $3min$ e $17s$ para cada tipo de áudio conforme a figura 7.

3.2.2 Extração de características

Existem diversos blocos prontos para uso na plataforma para extração de características através da DSP, todos eles são código aberto (<<https://github.com/edgeimpulse/processing-blocks>>). Bloco utilizado para este modelo foi o *Mel-filterbank energy features* (MFE). Este sistema avalia o espectro do sinal e realiza uma conversão das frequências para a escala Mel.

Fig. 7 – Dados coletados no Edge Impulse



Fonte: Edge Impulse

Os seguintes parâmetros para este bloco foram utilizados:

1. Largura do frame: 0,025s
2. Passo do frame: 0,01s
3. Número de filtros do banco: 41
4. Número de pontos da FFT: 512
5. Normalização de ruído: $-100dB$

3.2.3 Treinamento do Impulse

A arquitetura do modelo é uma Convolutional Neural Network (CNN) composta por camadas de convolução, *pooling*, *dropout* e camadas densas. A última camada usa a função *softmax* para a classificação em várias classes. No código 3.8 podem ser vistos mais detalhes da arquitetura assim como a figura 8. Esta configuração foi obtida através de uma sugestão da plataforma Edge Impulse, faz uma estimativa e sugere uma configuração, além disso, na plataforma existe a possibilidade de encontrar um modelo utilizando técnicas de *AutoML* em que diversos modelos são investigados com variações nos seus hiperparâmetros.

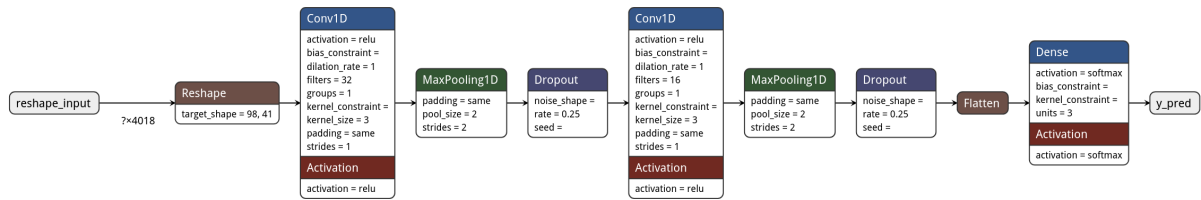
```

1 model = Sequential()
2 model.add(Reshape((int(input_length / 41), 41), input_shape=(
   input_length, )))
3 model.add(Conv1D(32, kernel_size=3, padding='same', activation='relu'))
4 model.add(MaxPooling1D(pool_size=2, strides=2, padding='same'))
5 model.add(Dropout(0.25))
6 model.add(Conv1D(16, kernel_size=3, padding='same', activation='relu'))
7 model.add(MaxPooling1D(pool_size=2, strides=2, padding='same'))
8 model.add(Dropout(0.25))
9 model.add(Flatten())
10 model.add(Dense(classes, name='y_pred', activation='softmax'))

```

Código 3.8 – Arquitetura do modelo para o Edge Impulse

Fig. 8 – Arquitetura do modelo para o Edge Impulse

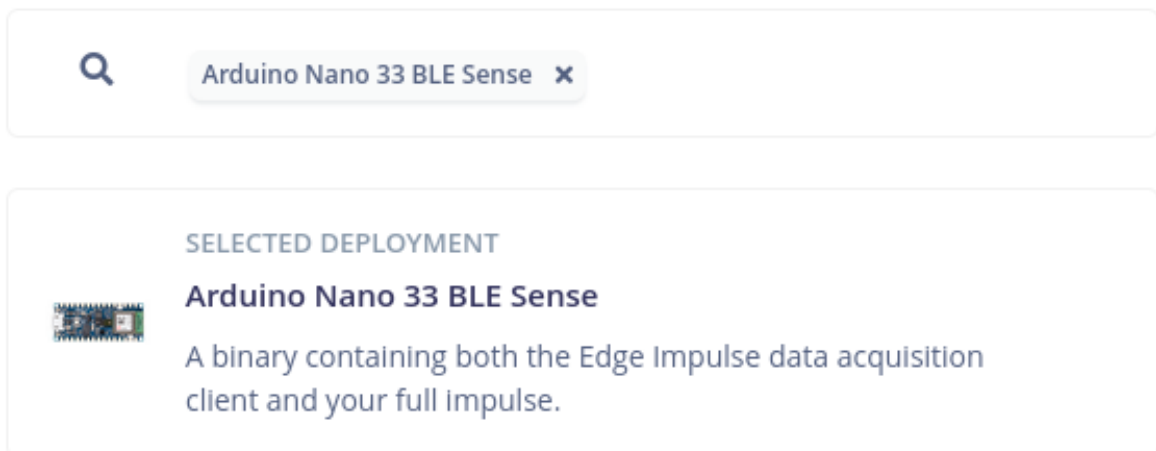


Fonte: Edge Impulse

3.2.4 Implementando no microcontrolador

Para a implementação no microcontrolador, é possível escolher entre um modelo otimizado (*int8*) ou um modelo não otimizado (*float32*). Há diferentes formas de implementação: gerando uma biblioteca C/C++ autocontida com o modelo treinado e o interpretador TensorFlow Lite (framework é utilizada pelo Edge Impulse), uma biblioteca Arduino com o modelo treinado e o binário. A figura 9 mostra a seleção do ambiente de implantação para o dispositivo Arduino Nano 33 BLE Sense.

Fig. 9 – Implantação no Arduino Nano 33 BLE Sense



Fonte: Edge Impulse

```
1 $./flash_linux.sh
2
3 Finding Arduino Mbed core...
4 Finding Arduino Mbed OK
5 Finding Arduino Nano 33 BLE...
6 Finding Arduino Nano 33 BLE OK
7 Flashing board...
8
9 Device      : nRF52840-QIAA
10 Version     : Arduino Bootloader (SAM-BA extended) 2.0 [Arduino:IKXYZ]
11 Address     : 0x0
12 Pages      : 256
13 Page Size  : 4096 bytes
14 Total Size : 1024KB
15 [...]
16
17 Done in 0.001 seconds
18 Write 367080 bytes to flash (90 pages)
19 [=====] 100% (90/90 pages)
20 Done in 14.077 seconds
21 New upload port: /dev/ttyACM0 (serial)
```

Código 3.9 – Gravando o firmware no Arduino Nano 33 BLE

3.2.5 Conclusões

Através deste exemplo utilizando o Edge Impulse, todo o processo de obtenção de dados, extração de características, treinamento, conversão e otimização e inferência no microcontrolador também foram abordados. Os resultados serão discutidos no capítulo a seguir. Todos os dados para repetir o que foi feito neste capítulo podem ser encontrados no repositório do GitHub <<https://github.com/taffarel55/graduation-thesis>>.

4 Resultados e Discussões

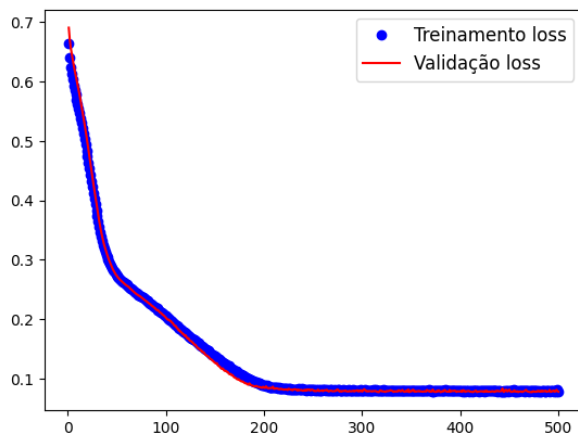
Segue a representação dos resultados obtidos em cada etapa do processo para as diferentes abordagens de desenvolvimento de modelos em dispositivos.

4.1 TensorFlow Lite no ESP-32s

4.1.1 Treinamento

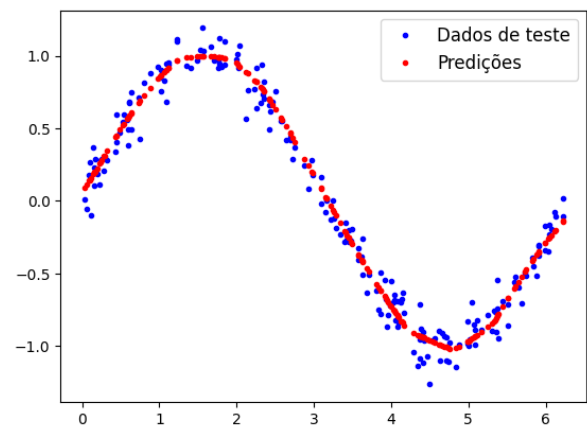
Os resultados do treinamento realizados com o Python e Keras podem ser vistos nas figuras 10 e 11. Da figura 10, percebe-se que o modelo atingiu um baixo *loss* (diferença entre o valor de saída da rede e o esperado para determinadas entradas), inferior a 0,09 a partir de 200 épocas de treinamento. Na figura 11 é possível verificar que a curva de predições se situa entre os dados de teste e a sua forma se aproxima de uma função seno.

Fig. 10 – Treinamento e Validação



Fonte: Autoria própria

Fig. 11 – Novas predições



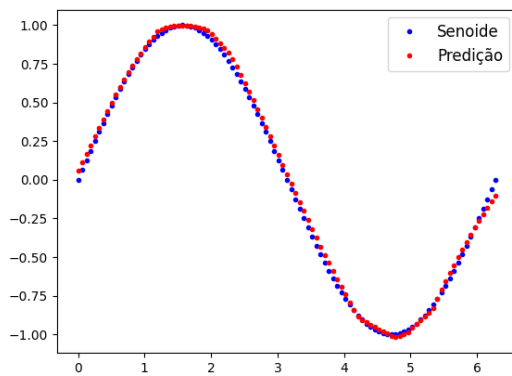
Fonte: Autoria própria

O resultado do treinamento se deu de forma satisfatória, simulando inclusive condições reais de dados de teste com ruídos.

4.1.2 Conversão e Otimização

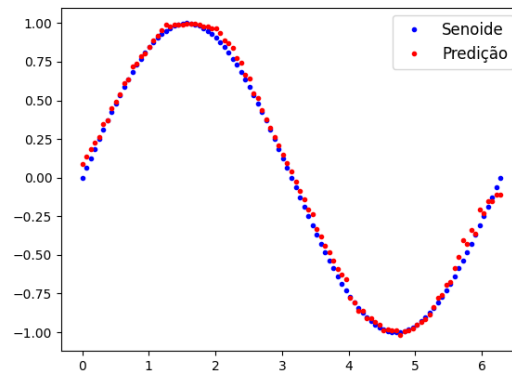
A conversão e otimização foram realizadas utilizando duas estratégias: Quantização de faixa dinâmica e Quantização inteira completa. Para a Quantização de faixa dinâmica foi observado na figura 12, uma boa aproximação com o valor real da função seno, no entanto, é possível perceber algumas regiões com uma certa discrepância, com semelhança de uma curva linearizada por partes. A figura 13 apresenta mais desvios em relação à função seno real, isto era esperado, pois diferentemente da quantização de faixa dinâmica que quantiza apenas os pesos, o modelo foi avaliado usando quantização completa e toda a matemática interna é quantizada.

Fig. 12 – Quantização de faixa dinâmica



Fonte: Autoria própria

Fig. 13 – Quantização com amostras representativas



Fonte: Autoria própria

Mesmo com a pequena perda de precisão do modelo pela otimização, a rede ainda pode ser utilizada. A tabela 1 exibe alguns dados que não podem ser estimados através das figuras como tamanho do modelo, desvio padrão, erro absoluto máximo e latência média para cada inferência.

Tab. 1 – Comparação do desempenho para diferentes estratégias de otimização diretamente no TensorFlow Lite

Otimização	Tamanho (kB)	Desvio padrão	Erro máximo	Latência média (μs)
DEFAULT	4764	0,026722	0,10139	136
REPRESENTATIVE_100	3624	0,032042	0,13240	791

Fonte: Produzido pelo autor

Da tabela 1, o tipo de otimização denominado *DEFAULT* se trata de uma quantização utilizando faixa dinâmica (8 bits) e o tipo de otimização *REPRESENTATIVE_100*, uma quantização utilizando 100 amostras representativas. Nota-se que a diferença entre os modelos treinados foi de pouco mais de $1kB$ no tamanho do modelo (uma redução de

23,93%), no entanto houve um aumento no desvio padrão e no erro máximo, além um aumento significativo na latência das inferências (581,62%).

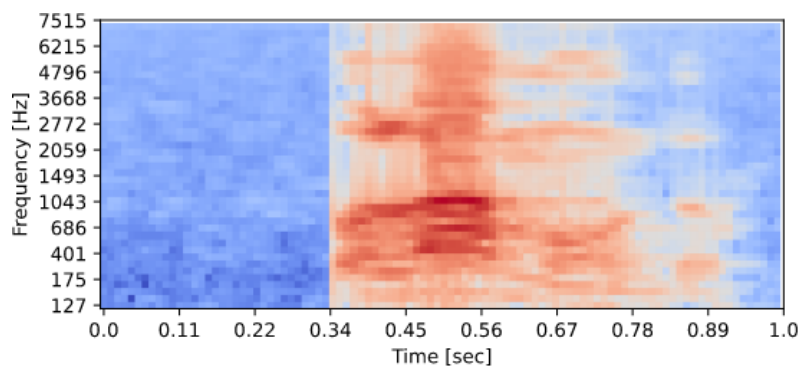
4.2 Edge Impulse no Arduino Nano BLE 33 Sense

Assim como foi feito para o TensorFlow, segue a representação dos resultados obtidos em cada etapa do processo para as diferentes abordagens de desenvolvimento de modelos em dispositivos. No Edge Impulse a quantização é feita pós treinamento e é possível obter uma quantização de faixa dinâmica de 8 bits ou um modelo não quantizado com uma representação de 32 bits de ponto flutuante.

4.2.1 Extração de características

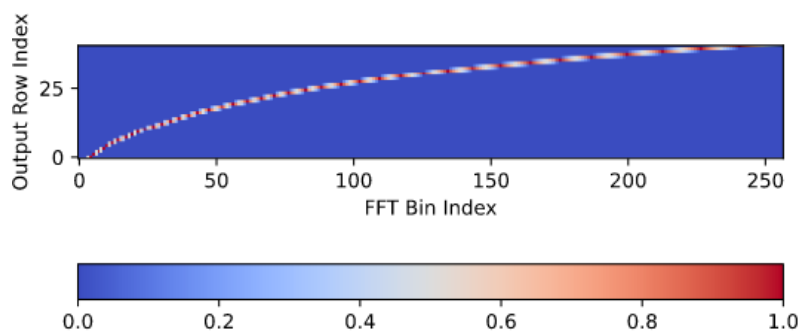
Nesta etapa foram geradas as características mais relevantes para o processamento de dados. Nas figuras 14 e 15 estão representadas as características de energia do filtro MFE e a densidade espectral de potência e a FFT de uma amostra de áudio qualquer, estes resultados são usados para extrair informações sobre o áudio e reduzir a dimensão.

Fig. 14 – Características de Energia do MFE



Fonte: Edge Impulse

Fig. 15 – Ponderação dos pesos da FFT



Fonte: Edge Impulse

Utilizando o algoritmo t-Distributed Stochastic Neighbor Embedding ([t-SNE](#)) (Método estatístico para visualizar dados de alta dimensão) para representação dos dados após a extração de características (neste caso, uma redução na dimensão para visualização em um mapa bidimensional), pode-se ver na figura 4.2.1 diferentes regiões que separam os dados. Importante notar que o ruído se sobrepõe sob os outros dados além de possuir um conjunto de dados mais afastados.

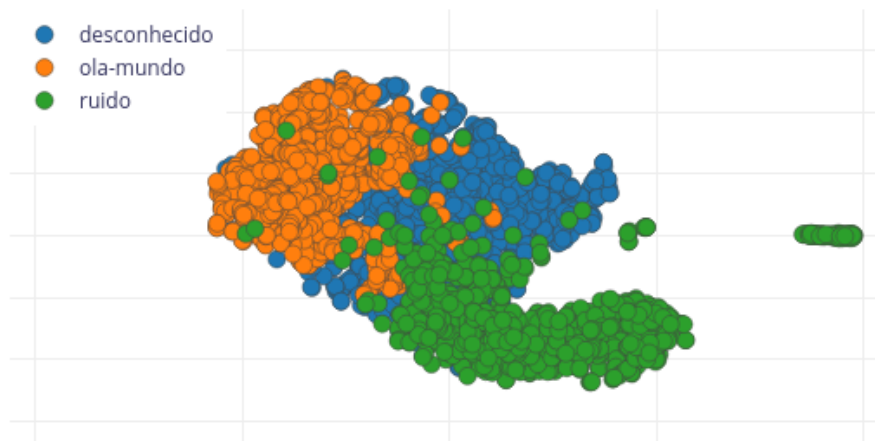


Fig. 16 – Fonte: Edge Impulse

Fonte: Edge Impulse

4.2.2 Treinamento

Após o treinamento da rede neural, os dados de testes foram usados para avaliar a eficácia do modelo. A figura 17 apresenta a matriz de confusão do modelo.

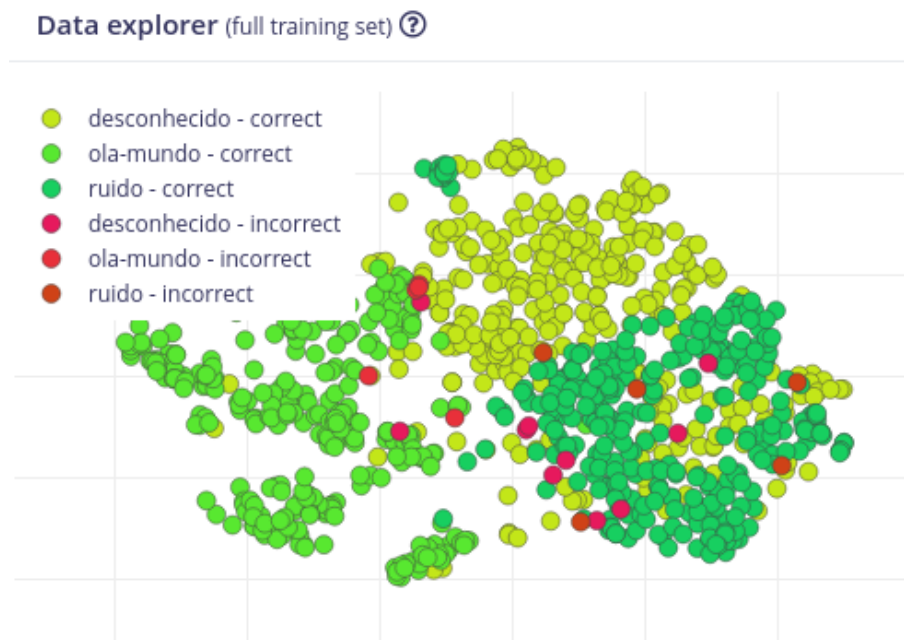
Fig. 17 – Matriz de confusão



Fonte: Edge Impulse

Uma outra exploração de dados foi realizada afim de visualizar como o modelo está classificando os dados de treinamento. Nota-se que boa parte dos dados foram classificados apresentando apenas alguns pontos (21) com classificação incorreta. A figura 18 apresenta tais resultados.

Fig. 18 – Exploração de dados de treinamento



Fonte: Edge Impulse

Além disso, é possível estimar, através do Edge Impulse, o desempenho do modelo no dispositivo escolhido. A figura 19 apresenta as estimativas de inferência, uso máximo de memória RAM e uso da memória FLASH. Para o caso do uso modelo diretamente utilizando o TensorFlow apresentado anteriormente, as colunas *Tamanho* e *Latência média* da tabela 1 estão relacionadas respectivamente com o *Flash Usage* e *Inferencing Time*, com uma observação: O tempo de inferência apresentado pelo EdgeImpulse é apenas ao tempo que o modelo utiliza para realizar a classificação a partir das entradas após o bloco de extração de parâmetros

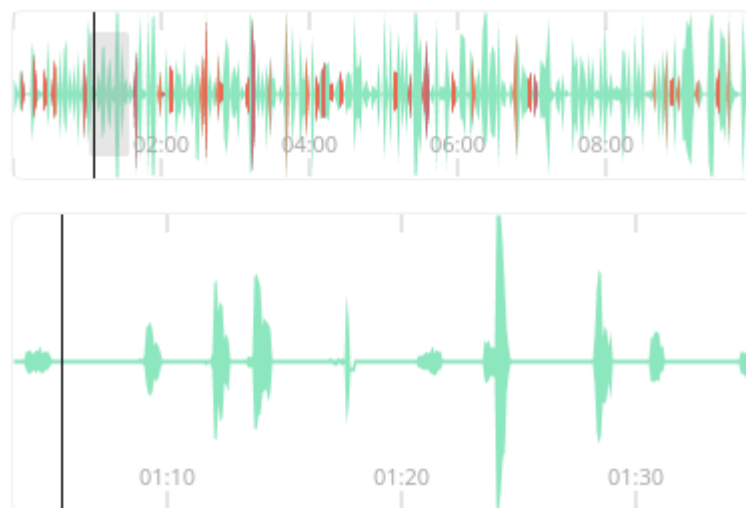
Fig. 19 – Estimativa de Desempenho no Arduino Nano 33 BLE Sense



Fonte: Edge Impulse

Um outro recurso interessante apresentado na figura 20 é a possibilidade de testar como o modelo responde a detecção de eventos simulando alguns minutos de áudio de dados reais ou dados gerados a partir dos dados de teste com dados de ruído como ruído de fundo. Na figura 20 é apresentado um áudio com 10 minutos de duração e um corte de aproximadamente 45 segundos de duração com dados gerados a partir da base de dados. Para este caso, áudios de **desconhecido** e **ola-mundo** foram somados com os dados ruídos para tentar avaliar um ambiente mais real. Nos momentos em que o modelo previu o áudio verdadeiro que estava combinado com ruído, o áudio gerado fica verde, enquanto nos momentos que a predição foi errada, o áudio fica vermelho.

Fig. 20 – Simulando ambiente real



Fonte: Edge Impulse

É possível perceber que há diversos falsos positivos e falsos negativos. A figura 21 sumariza as taxas False Acceptance Rate (**FAR**) (taxa de erros do modelo em realizar uma predição de olá mundo), False Rejection Rate (**FRR**) (taxa de erros do modelo em realizar uma predição de olá mundo errada), além da quantidade de falsos positivos, falsos negativos, verdadeiros positivos e falsos positivos.

Fig. 21 – Visão geral do desempenho

LABEL	FAR ②	FRR ②	TRUE POSITIVES ②	FALSE POSITIVES ②	TRUE NEGATIVES ②	FALSE NEGATIVES ②
ola-mundo	1.1%	62.3%	23	2	179	38
desconhecido ②	10.6%	74.6%	15	20	169	44

Fonte: Edge Impulse

4.2.3 Implantação do firmware

Após a implantação do firmware, o comando apresentado no código 4.1 executa o modelo em modo contínuo e realiza as respectivas predições.

```
1 $ edge-impulse-run-impulse --continuous
2 Edge Impulse impulse runner v1.22.0
3 [SER] Connecting to /dev/ttyACM0
4 [SER] Serial is connected, trying to read config...
5 [SER] Retrieved configuration
6 [SER] Device is running AT command version 1.8.0
7 [SER] Started inferencing, press CTRL+C to stop...
8
9 Starting inferencing, press 'b' to break
10 Predictions (DSP: 74 ms., Classification: 39 ms., Anomaly: 0 ms.):
11     desconhecido: 0.01172
12     ola-mundo: 0.00391
13     ruído: 0.98438 # <- Prediction: ruído
14 Predictions (DSP: 74 ms., Classification: 39 ms., Anomaly: 0 ms.):
15     desconhecido: 0.15234
16     ola-mundo: 0.01562
17     ruído: 0.82812 # <- Prediction: ruído
18 Predictions (DSP: 74 ms., Classification: 39 ms., Anomaly: 0 ms.):
19     desconhecido: 0.00391
20     ola-mundo: 0.99609 # <- Prediction: ola-mundo
21     ruído: 0.00000
22 Predictions (DSP: 74 ms., Classification: 39 ms., Anomaly: 0 ms.):
23     desconhecido: 0.99609 # <- Prediction: desconhecido
24     ola-mundo: 0.00000
25     ruído: 0.00391
26 Predictions (DSP: 74 ms., Classification: 39 ms., Anomaly: 0 ms.):
27     desconhecido: 0.00000
28     ola-mundo: 0.99609 # <- Prediction: ola-mundo
29     ruído: 0.00000
```

Código 4.1 – Testando o firmware no Arduino Nano 33 BLE

5 Conclusões

Neste trabalho, pode-se perceber a importância de contar com ferramentas adequadas para o desenvolvimento de modelos de TinyML. Essas ferramentas fornecem recursos específicos para treinar e implementar redes neurais em dispositivos com recursos limitados, como microcontroladores para sistemas embarcados, além de outros dispositivos. A ferramenta TensorFlow Lite é especialmente relevante devido à sua popularidade e ampla adoção na comunidade de desenvolvimento de modelos em TinyML. O Edge Impulse também é uma escolha que acelera o processo de desenvolvimento e implantação de soluções de Aprendizado de Máquina em sistemas embarcados.

Desta forma, espera-se que este trabalho sirva como fundamento para investigações futuras, onde os conceitos e ferramentas apresentados possam ser utilizados como alicerce para a pesquisa de aspectos mais específicos no campo de TinyML.

Referências

- AJANI, T. S.; IMOIZE, A. L.; ATAYERO, A. A. An overview of machine learning within embedded and mobile devices—optimizations and applications. 06 2021. Disponível em: <<https://doi.org/10.3390/s21134412>>. Citado na página 33.
- ARM-TINYML. *TinyML Brings AI to Smallest Arm Devices* — *arm.com*. 2021. <<https://www.arm.com/blogs/blueprint/tinyml>>. [Acessado 24-Jun-2023]. Citado na página 25.
- BAE, H. et al. Multi-robot path planning method using reinforcement learning. 07 2019. Disponível em: <<https://doi.org/10.3390/app9153057>>. Citado na página 27.
- BAO, W. et al. Edge computing-based joint client selection and networking scheme for federated learning in vehicular IoT. *China Communications*, Institute of Electrical and Electronics Engineers (IEEE), v. 18, n. 6, p. 39–52, jun. 2021. Disponível em: <<https://doi.org/10.23919/jcc.2021.06.004>>. Citado na página 25.
- CONTRACTOR, D. et al. *Behavioral use licensing for responsible AI*. New York, NY, USA: ACM, 2022. Disponível em: <<https://doi.org/10.1145/3531146.3533143>>. Citado na página 32.
- ETHICS guidelines for trustworthy AI. 2019. [Acessado 24-Nov-2023]. Disponível em: <<https://web.archive.org/web/2023111221626/https://digital-strategy.ec.europa.eu/en/library/ethics-guidelines-trustworthy-ai>>. Citado na página 32.
- FAQ — Responsible AI Licenses (RAIL). 2023. [Acessado 24-Nov-2023]. Disponível em: <<https://www.licenses.ai/faq-2>>. Citado na página 32.
- HEBB, D. O. The organization of behavior: A neuropsychological theory. New York: John Wiley and Sons, Inc., 1949. 335 p. \$4.00. *Science Education*, Wiley, v. 34, n. 5, p. 336–337, dez. 1950. Disponível em: <<https://doi.org/10.1002/sci.37303405110>>. Citado na página 30.
- IMPULSE, E. *For beginners* — *web.archive.org*. 2023. <<https://web.archive.org/web/20231115234805/https://docs.edgeimpulse.com/docs/readme/for-beginners>>. [Acessado 15-Nov-2023]. Citado na página 35.
- JOHNNY, F. K. F. *CMSIS-NN & Optimizations for Edge AI*. 2021. Presentation at TinyML Talks. Sweden Area Group. Citado na página 33.
- NANOEDGEAISTUDIO. *NanoEdgeAIStudio - STMICROELECTRONICS* — *web.archive.org*. 2023. <<https://web.archive.org/web/20231115235431/https://www.st.com/en/development-tools/nanoedgeaistudio.html>>. [Acessado 15-Nov-2023]. Citado na página 35.
- NOLAN, A. *The next big thing in machine learning is tiny* | *Thoughtworks*. 2023. [Acessado 22-Nov-2023]. Disponível em: <<https://web.archive.org/web/20231114182604/https://www.thoughtworks.com/en-us/insights/blog/machine-learning-and-ai/tinyml-the-next-big-thing>>. Citado na página 33.

PRAKASH, S. et al. Is TinyML sustainable? *Communications of the ACM*, Association for Computing Machinery (ACM), v. 66, n. 11, p. 68–77, out. 2023. Disponível em: <<https://doi.org/10.1145/3608473>>. Citado na página 33.

RAY, P. P. A review on TinyML: State-of-the-art and prospects. *Journal of King Saud University - Computer and Information Sciences*, Elsevier BV, v. 34, n. 4, p. 1595–1623, abr. 2022. Disponível em: <<https://doi.org/10.1016/j.jksuci.2021.11.019>>. Citado 4 vezes nas páginas 25, 26, 33 e 35.

RESPONSIBLE AI License. 2023. [Acessado 24-Nov-2023]. Disponível em: <<https://web.archive.org/web/20231111224311/https://docs.edgeimpulse.com/page/responsible-ai-license>>. Citado na página 32.

ROSENBLATT, F. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychol. Rev.*, American Psychological Association (APA), v. 65, n. 6, p. 386–408, nov. 1958. Citado na página 28.

SCHIEFERDECKER, I.; GROßMANN, J.; SCHNEIDER, M. How to safeguard ai. 09 2019. Disponível em: <<https://doi.org/10.14361/9783839447192-015>>. Citado na página 27.

SCHMIDHUBER, J. Deep learning in neural networks: An overview. *Neural Networks*, Elsevier BV, v. 61, p. 85–117, jan. 2015. Disponível em: <<https://doi.org/10.1016/j.neunet.2014.09.003>>. Citado na página 28.

SHARMA, S.; SINGH, G.; SINGH, D. N. Role and performance of different traditional classification and nature-inspired computing techniques in major research areas. 06 2019. Disponível em: <<https://doi.org/10.4108/eai.13-7-2018.158419>>. Citado na página 27.

SICART, M.; SHKLOVSKI, I.; JONES, M. Can machine learning be moral? 01 2022. Disponível em: <<https://doi.org/10.48550/arxiv.2201.06921>>. Citado na página 31.

SILVA, I. *Introdução à Inteligência Artificial: Uma abordagem não técnica - Redes Neurais Artificiais Para Engenharia e Ciências Aplicadas. Fundamentos Teóricos e Aspectos Práticos*. [S.l.]: Artliber Editora, 2010. Citado 2 vezes nas páginas 30 e 31.

TAFFAREL, M. *Perceptron: Implementação em R e em Javascript*. 2022. [Acessado 24-Nov-2023]. Disponível em: <<https://github.com/taffarel55/perceptron>>. Citado na página 32.

TAULLI, T. *Introdução à Inteligência Artificial: Uma abordagem não técnica*. [S.l.]: Novatec Editora, 2019. Citado na página 27.

TENSORFLOW. *TensorFlow Lite para microcontroladores — web.archive.org*. 2023. <<https://web.archive.org/web/20231125151634/https://www.tensorflow.org/lite/microcontrollers?hl=pt-br>>. [Acessado 25-Nov-2023]. Citado na página 36.

TENSORFLOWLITE. *Treinamento consciente de quantização | TensorFlow Model Optimization — web.archive.org*. 2023. <https://web.archive.org/web/20231125200428/https://www.tensorflow.org/model_optimization/guide/quantization/training?hl=pt-br>. [Acessado 25-Nov-2023]. Citado na página 34.

- TFLITE. *TensorFlow Lite / ML para dispositivos móveis e do Edge* — *web.archive.org*. 2023. [Acessado 11-Nov-2023]. Disponível em: <<https://web.archive.org/web/20231115234136/https://www.tensorflow.org/lite?hl=pt-br>>. Citado na página 35.
- UTENSOR. *GitHub - uTensor/uTensor: TinyML AI inference library* — *github.com*. 2023. <<https://github.com/uTensor/uTensor>>. [Acessado 15-Nov-2023]. Citado na página 35.
- UTVMSYSTEM. *TinyML - How TVM is Taming Tiny* — *tvm.apache.org*. 2020. <<https://tvm.apache.org/2020/06/04/tinyml-how-tvm-is-taming-tiny>>. [Accessed 24-Jun-2023]. Citado na página 25.
- WARDEN, P. *Tiny ML*. Sebastopol, CA: O'Reilly Media, 2020. Citado na página 33.
- YANG, Y. et al. Multi-source transfer learning via ensemble approach for initial diagnosis of alzheimer's disease. 01 2020. Disponível em: <<https://doi.org/10.1109/jtehm.2020.2984601>>. Citado na página 27.
- YOUNIS, R. M. A survey on using machine learning and deep learning based iris recognition. 01 2023. Disponível em: <<https://doi.org/10.25007/ajnu.v12n1a1677>>. Citado na página 27.