student student
UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
ESCOLA DE ENGENHARIA
ENGENHARIA DE CONTROLE E AUTOMAÇÃO

MAXWELL RODRIGUES DA SILVA - 282162

APLICAÇÃO E AVALIAÇÃO DE MACHINE LEARNING EM DISPOSITIVOS EMBARCADOS

Porto Alegre
2025

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
ESCOLA DE ENGENHARIA
ENGENHARIA DE CONTROLE E AUTOMAÇÃO

MAXWELL RODRIGUES DA SILVA - 282162

APLICAÇÃO E AVALIAÇÃO DE MACHINE LEARNING EM DISPOSITIVOS EMBARCADOS

Trabalho de Conclusão de Curso submetido à COMGRAD/CCA da UFRGS como parte dos requisitos para a obtenção do título de *Bacharel em Engenharia de Controle e Automação.*

studentOrientador:
Prof. Dr. Marcelo Götz

Porto Alegre
2025

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
ESCOLA DE ENGENHARIA
ENGENHARIA DE CONTROLE E AUTOMAÇÃO

MAXWELL RODRIGUES DA SILVA - 282162

APLICAÇÃO E AVALIAÇÃO DE MACHINE LEARNING EM DISPOSITIVOS EMBARCADOS

Este Trabalho de Conclusão de Curso foi julgado adequado para a obtenção dos créditos da Atividade de *Trabalho de Conclusão de Curso CCA - II* e aprovado em sua forma final studentpelo Orientador e Banca Examinadora abaixo.

studentOrientador:
Prof. Dr. Marcelo Götz, UFRGS
Doutor pela Universidade Federal do Rio Grande do Sul – Porto Alegre, Brasil

Banca Examinadora:

Prof. Dr. Marcelo Götz, UFRGS
Doutor pela Universidade Federal do Rio Grande do Sul – Porto Alegre, Brasil

Prof. Dr. Rafael Antônio Comparsi Laranja, UFRGS
Doutor pela Universidade Federal do Rio Grande do Sul – Porto Alegre, Brasil

Prof. Dr. Valner João Brusamarello, UFRGS
Doutor pela Universidade Federal de Santa Catarina – Florianópolis, Brasil

---

Fabiano Disconzi Wildner
Coordenador de Curso
Eng. de Controle e Automação

Porto Alegre
Junho - 2025

DEDICATÓRIA

Dedico este trabalho à minha mãe e ao meu pai que sempre me apoiaram, ao Bruno, o melhor irmão que alguém poderia ter, a todos meus queridos amigos que alegram minha vida e em especial à Jaque, minha esposa, pela dedicação e apoio em todos os momentos difíceis, por sempre estar ao meu lado e pelo nosso amor. Sou grato por ter o privilégio de compartilhar um tempo e uma existência junto à ela. São estas relações de amizade e afeto que dão sentido à vida.

AGRADECIMENTOS

À Universidade Federal do Rio Grande do Sul (UFRGS), pela oportunidade de estudar em uma das instituições mais renomadas da América Latina e do mundo, de forma gratuita.

A todos que lutam e lutaram pela implementação da Lei de Cotas no Brasil, um marco para a construção de um país mais justo e que possibilitou minha formação acadêmica.

Ao Laboratório de Metalurgia Física (LAMEF), por ter me dado a oportunidade de iniciar minha trajetória profissional, onde tive a chance de aprender e crescer.

À minha liderança na TK Elevadores, pela compreensão e flexibilidade diante dos horários desafiadores do curso de Engenharia da UFRGS, conciliados com as demandas do trabalho.

E, por fim, ao corpo docente da Escola de Engenharia desta Universidade, que, mesmo utilizando métodos por vezes desafiadores e pouco ortodoxos, contribuiu significativamente para minha formação como profissional.

RESUMO

Com a escalada da demanda por dispositivos da Internet das Coisas (IoT) e exigências de inteligência embarcada, o paradigma TinyML surge como uma solução promissora. Este trabalho tem como objetivo avaliar o estado atual da aplicação de soluções baseadas em aprendizado de máquina a sistemas embarcados, com foco em dispositivos caracterizados por recursos computacionais limitados. A pesquisa abrange a análise de ferramentas de desenvolvimento TinyML, levantamento de plataformas disponíveis e elaboração de um fluxo de trabalho ponta-a-ponta para o desenvolvimento de soluções. A hipótese é que técnicas de TinyML podem substituir ou complementar arquiteturas IoT tradicionais, proporcionando maior eficiência e menor latência em aplicações críticas que utilizem inteligência artificial. Ademais, ao final, uma proposta de trabalho prático é apresentada, utilizando todo o resultado da pesquisa em uma solução aplicada de aprendizado de máquina embarcado.

Palavras-chave: TinyML, aprendizado de máquina, computação de borda, internet das coisas, sistemas embarcados

ABSTRACT

With the growing demand for Internet of Things (IoT) devices and the increasing need for embedded intelligence, the TinyML paradigm emerges as a promising solution. This work aims to evaluate the current state of applying machine learning-based solutions to embedded systems, focusing on devices characterized by limited computational resources. The research includes an analysis of TinyML development tools, a review of available platforms, and the design of an end-to-end workflow for solution development. The hypothesis is that TinyML techniques can replace or complement traditional IoT architectures, offering greater efficiency and lower latency in critical applications using artificial intelligence. Furthermore, a practical work proposal is presented at the end, leveraging all research findings in an applied embedded machine learning solution.

Palavras-chave: TinyML, edge computing, embedded systems, internet of things, machine learning

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

O avanço da tecnologia e a popularização dos dispositivos conectados têm impulsionado o desenvolvimento de soluções inteligentes em diversas áreas. Nesse cenário, a Internet das Coisas (IoT) destaca-se como um dos principais motores da transformação digital, permitindo a integração de sensores, atuadores e sistemas embarcados em aplicações que vão da saúde à indústria. No entanto, desafios como a latência na comunicação com a nuvem e a limitação de recursos computacionais em dispositivos embarcados exigem novas abordagens para garantir eficiência e desempenho em tempo real (**Xavier2022**).

O número de dispositivos pertencentes à IoT tem apresentado um crescimento constante durante a terceira década do século XXI (**Sinha2024**). Esses dispositivos estão sendo implementados em diversas áreas, como saúde (**iotSaude**), agronomia (**iotAgricultura**) e indústria (**iotIndustria**). Esse cenário reforça a importância de estudar soluções eficientes para IoT, já que seu impacto se estende a áreas essenciais do cotidiano e do desenvolvimento econômico.

Uma limitação dos dispositivos IoT é a latência do envio de dados para centros de processamento remotos. Mesmo os melhores serviços de nuvem alcançam latências de até 28 ms (**latencia**), variando conforme horário e tráfego de dados. Essa latência é um desafio em aplicações que exigem decisões em tempo real, como controle crítico, automação industrial e dispositivos de saúde (**Oladokun2025**).

A tecnologia de aprendizado de máquina aplicada a sistemas embarcados, conhecida como TinyML, permite realizar o processamento local dos dados no próprio dispositivo. Esse paradigma, associado à computação de borda, reduz a dependência de processamento remoto e melhora significativamente a latência, com reduções de 35% a 40% conforme indicado em (**edge**), tornando-a mais adequada para aplicações em tempo real.

Avaliar o estado atual de desenvolvimento do TinyML e um fluxo de trabalho para o desenvolvimento de soluções utilizando dispositivos embarcados pode contribuir para superar os desafios relacionados à dependência de conectividade. Além disso, tal abordagem viabiliza o desenvolvimento de sistemas mais robustos, eficientes e adaptados a aplicações críticas ou em áreas remotas. Com base nisso, este trabalho propõe-se a investigar, de forma estruturada, até que ponto as soluções baseadas em TinyML podem, atualmente, substituir ou complementar arquiteturas IoT tradicionais — conforme detalhado nos objetivos específicos a seguir.

1.1 Objetivos gerais

Este trabalho tem como objetivo avaliar a viabilidade de aplicar soluções baseadas em aprendizado de máquina a sistemas embarcados, com foco em dispositivos caracterizados por recursos computacionais limitados.

1.2 Objetivos específicos

Os objetivos específicos deste trabalho são os seguintes:

1. Pesquisar e avaliar as ferramentas de desenvolvimento TinyML disponíveis, considerando critérios como facilidade de uso e integração com as plataformas mais populares;

2. Realizar o levantamento e a análise das plataformas disponíveis para projetos TinyML, considerando aspectos como desempenho, consumo energético e custo;

3. Elaborar um fluxo de trabalho ponta-a-ponta para desenvolvimento de soluções baseadas em técnicas TinyML, de forma a ilustrar o processo;

4. Desenvolver e aplicar, como solução proposta, um modelo TinyML para classificação de resíduos sólidos, comparando seu desempenho com alternativas tradicionais.

1.3 Hipótese

A aplicação de técnicas de TinyML em dispositivos embarcados com recursos computacionais limitados é viável e prática nos dias atuais, devido à disponibilidade crescente de ferramentas de desenvolvimento acessíveis, plataformas de hardware compatíveis e métodos eficientes de otimização de modelos que permitem o processamento local com baixo consumo energético.

1.4 Estrutura do trabalho

Este trabalho está estruturado em cinco capítulos, incluindo esta introdução. O segundo capítulo apresenta uma revisão bibliográfica sobre TinyML e dispositivos embarcados, abordando os principais conceitos, desafios e trabalhos relacionados. O terceiro capítulo fornece a fundamentação teórica necessária para o desenvolvimento da solução proposta, detalhando as etapas do fluxo de trabalho de desenvolvimento de soluções TinyML. O quarto capítulo apresenta a proposta de trabalho, descrevendo a solução prática a ser desenvolvida e o planejamento das etapas do projeto. Por fim, o quinto capítulo conclui o trabalho, apresentando considerações finais e definições para o trabalho futuro.

2 REVISÃO BIBLIOGRÁFICA

2.1 TinyML e Dispositivos Embarcados

TinyML é uma subárea do aprendizado de máquina dedicada à implementação de modelos de Aprendizado de Máquina em dispositivos embarcados com recursos computacionais limitados, como microcontroladores e sensores. Esses dispositivos, como o Arduino Nano 33 BLE Sense, Raspberry Pi, ESP32 e placas STM32, são amplamente utilizados em aplicações de IoT e automação, onde a eficiência energética e a baixa latência são fatores críticos. Os desafios técnicos incluem a limitação de memória, capacidade de processamento e consumo de energia desses dispositivos, o que exige técnicas específicas de otimização de modelos e algoritmos para garantir que as aplicações funcionem de maneira eficiente e eficaz.

2.2 Trabalhos Relacionados e Abordagens na Literatura

Diversos estudos têm demonstrado o potencial do TinyML em aplicações de diferentes áreas, especialmente aquelas que exigem soluções embarcadas com baixo consumo de energia e capacidade de processamento local. Os trabalhos apresentados a seguir utilizam técnicas, ferramentas e plataformas voltadas à implementação eficiente de modelos de aprendizado de máquina em dispositivos com recursos limitados.

Por exemplo, (**Wulnye2024**) propõe uma solução aplicada à agricultura de precisão, utilizando imagens capturadas por um módulo de câmera VGA acoplada a um Arduino Nano 33 BLE Sense para detecção de avarias em folhas de milho. O modelo treinado possui tamanho de 321 kB e alcança uma acurácia (proporção de acertos do modelo) de 94,56%, demonstrando a viabilidade de aplicações visuais embarcadas com alto desempenho em campo. Ademais, o trabalho utilizou bancos de dados públicos de alta confiabilidade, como o repositório Harvard Dataverse (**harvard_dataverse**) para treinamento do modelo aliado à plataforma Edge Impulse (**edge_impulse**).

Outro exemplo é o estudo de (**Yap2024**), que apresenta uma solução simples para detecção de anomalias em um ventilador. Seu trabalho consiste em um giroscópio alinhado a um Arduino Nano 33 BLE Sense. O modelo de apenas 17 kB é capaz de detectar comportamentos anormais com 99,23% de acurácia. Para o treino do modelo, o autor realizou a coleta de dados empiricamente em dois cenários diferentes: um ventilador em funcionamento normal e outro com uma falha simulada com a quebra parcial de uma das hélices. O modelo foi treinado utilizando a plataforma LiteR, anteriormente conhecida como TensorFlow Lite (**litert2024**).

Dispositivos embarcados são em geral compactos e eficientes, podem realizar tarefas computacionais específicas por um custo de espaço e energia reduzidos. Neste sentido,

soluções TinyML podem ser ferramentas práticas para uso cotidiano, como mostra o estudo de (**Mellit2025**) utilizando sensor de infra-vermelho e um Arduino Portento H7 para detecção de danos em painéis fotovoltaicos, trabalhando na plataforma Edge Impulse e adquirindo os dados utilizando drone em fazendas algerianas, o modelo treinado alcançou uma acurácia de 98% e é apresentado no formato de pistola portátil com tela amigável para o usuário.

Nesses trabalhos citados, observa-se um fluxo de desenvolvimento recorrente para soluções baseadas em TinyML. As etapas principais de um projeto desse tipo podem ser resumidas da seguinte forma:

1. **Coleta de dados:** Captura de dados relevantes para o problema proposto, ou uso de repositórios públicos com conjuntos de dados apropriados;
2. **Pré-processamento:** Limpeza e preparação dos dados, incluindo normalização, remoção de ruídos, rotularização, seleção de características relevantes e aumento do númer de dados utilizando IA (data augmentation), se necessário;
3. **Escolha e treinamento do modelo:** Uso de plataformas como TensorFlow ou Edge Impulse para treinar modelos de aprendizado de máquina com os dados já preparados;
4. **Quantização do modelo:** Redução do tamanho e da complexidade do modelo, tornando-o mais eficiente para execução em dispositivos embarcados com recursos limitados;
5. **Validação do modelo:** Avaliação do desempenho do modelo utilizando dados de validação, verificando sua precisão, robustez e capacidade de generalização;
6. **Ajuste fino:** Modificações em hiperparâmetros, arquitetura ou técnicas de regularização para aprimorar o desempenho do modelo;
7. **Implementação:** Integração do modelo quantizado no dispositivo embarcado, com testes em ambiente real e adaptação ao hardware utilizado.

A escolha do modelo e a quantização do modelo são peças chaves na elaboração do projeto. Modelos de redes neurais como Redes Neurais Convolucionais (RNC) são frequentemente utilizados em aplicações de visão computacional, por exemplo. A quantização é particularmente importante em aplicações TinyML, pois trata-se de uma técnica que reduz a precisão dos pesos do modelo, permitindo que ele ocupe menos espaço e consuma menos recursos computacionais, sem comprometer significativamente o desempenho.

2.3 Considerações Finais da Revisão

Durante a revisão bibliográfica foi observado que projetos TinyML podem ser executados em algumas etapas principais. Dentro de cada uma destas etapas decisões importantes são tomadas, como a escolha do modelo, a plataforma de desenvolvimento e a forma de coleta dos dados. As vastas opções de plataformas disponíveis como TensorFlor e Edge Impulse, bem como a variedade de dispositivos IoT, permitem uma vasta gama de soluções em Aprendizado de Máquina embarcado.

3 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo, serão abordados todas as etapas do fluxo de trabalho de desenvolvimento de soluções TinyML, explorando nuances e escolhas importantes a serem feitas em cada passo.

3.1 Aquisição e Preparação dos Dados

O primeiro passo para o desenvolvimento de soluções TinyML é a coleta dos dados para treinamento. Esta etapa é fundamental, pois a quantidade e a qualidade dos dados influenciam diretamente a eficácia do modelo de aprendizado de máquina. Não há um número mínimo universal de dados necessários, nem um número mágico que funcione em todos os casos. Em geral, quanto mais representativos forem os dados e em maior volume estiverem disponíveis, melhor será o desempenho do modelo (**datasetSize**). Se a quantidade de dados não for suficiente, existem técnicas como *Aumento de Dados* que consiste em gerar novas amostras a partir dos dados existentes, aplicando transformações como rotação, escala, adição de ruído, entre outras (**Whang2023**).

Os métodos mais comuns de aquisição de dados podem ser divididos em duas grandes categorias: dados coletados empiricamente, realizando a captura de informação diretamente do ambiente ou realizando ensaios. Este método possui a vantagem de estar alinhado com a aplicação específica, o que leva a modelos mais eficazes (**specificDomain**), mas pode ser custoso e demorado. A outra categoria é a utilização de bancos de dados públicos, que podem ser encontrados em repositórios como o Kaggle (**kaggle**) ou Harvard Dataverse (**harvard_dataverse**). Esses bancos de dados são frequentemente utilizados para treinamento de modelos e podem acelerar o processo de desenvolvimento, mas é importante garantir que os dados sejam representativos do problema específico a ser resolvido.

Após a coleta dos dados, é necessário realizar uma etapa de curadoria para remoção de ruídos, inconsistências, duplicatas e outras anomalias que podem afetar o treinamento do modelo. Ao limpar dados para treinamento de modelos, deve-se utilizar ferramentas próprias para aplicações em Aprendizado de Máquina (**Whang2023**). Quando aplicável, os dados também devem ser rotulados cuidadosamente, garantindo que cada amostra possua uma anotação precisa e consistente com o objetivo da tarefa de aprendizagem. Além disso, é necessário normalizar os dados, isto é, ajustar as escalas das amostras de forma que tenham tamanho, resolução, frequência e demais características semelhantes. Com isso, garantimos que o modelo não seja enviesado por dados com características muito distintas, aprendendo características irrelevantes (**Ahmed2022**).

3.2 Desenvolvimento e Treinamento de Modelos

O desenvolvimento da solução começa com a escolha do tipo de modelo de aprendizado de máquina. Modelos de aprendizado de máquina são algoritmos que aprendem padrões a partir dos dados fornecidos, e sua escolha depende do tipo de problema a ser resolvido. Atualmente há uma variedade de modelos disponíveis, a Tabela 1 apresenta alguns dos modelos mais comuns utilizados em TinyML e suas principais funções (**tinyModels**).

Tabela 1: *Modelos de aprendizado de máquina e suas principais funções no contexto de TinyML.*

Modelo	Função Principal
Redes Neurais Convolucionais (RNC)	Classificação de imagens
Redes Neurais Densas (RND)	Reconhecimento de palavras
Redes de Memória de Longo Prazo (RMLP)	Processamento de fala
Redes Neurais Recorrentes (RNR)	Processamento sequencial

Fonte: *Do autor*

Para o cumprimento do objetivo específico 4, o modelo a ser desenvolvido será do tipo Rede Neural Convolucional (RNC), pois esse tipo de arquitetura é amplamente utilizado em tarefas de classificação de imagens em aplicações TinyML (**tinyModels**). A escolha detalhada da arquitetura, como a utilização de variantes compactas (por exemplo, TinyYOLO ou outras), será definida durante a implementação prática no TCC-II, levando em conta os requisitos do projeto e as limitações do hardware selecionado.

Uma arquitetura de rede neural é construída basicamente por camadas de neurônios. Neurônios, neste contexto, são basicamente unidades de funções matemáticas que recebem valores de entradas, realiza uma soma ponderada, efetua uma operação matemática chamada de Função de Ativação e então direciona este resultado para a saída. Essa saída pode ser um neurônio de uma próxima camada ou a saída final do modelo. A Figura 1 mostra um simples exemplo.

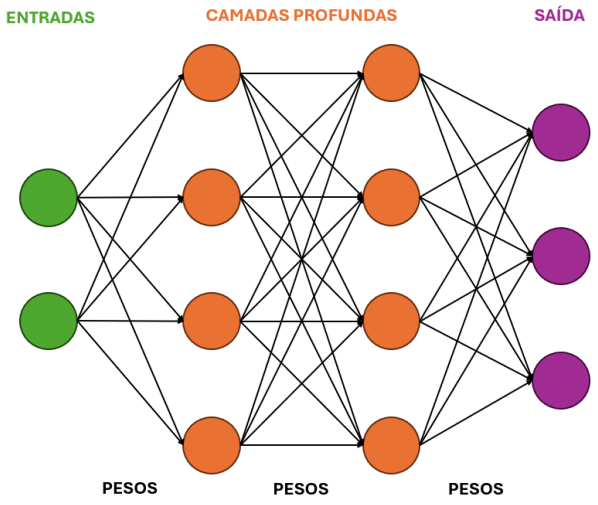

Figura 1: *Exemplo de uma rede neural simples.*

Fonte: *Do autor*

Os pesos exibidos na Figura 1 são os pesos da soma ponderada que são ajustados em cada neurônio durante o treinamento do modelo. Para realizar o ajuste destes pesos, durante o treinamento é utilizado um algoritmo de otimização, este programa trata-se de uma técnica matemática que busca minimizar a diferença entre a saída prevista pelo modelo e a saída real dos dados de treinamento. Conforme descreve (**comparaAlg**), o algoritmo mais comum utilizado para este fim é o *Gradiente Descendente Estocástico* (GDE), que ajusta os pesos da rede neural iterativamente, utilizando o gradiente da função de perda em relação aos pesos. Porém, ainda conforme (**comparaAlg**), outros algoritmos como Adam e RMSprop também são amplamente utilizados.

Ademais, é importante considerar qual Função de Custo utilizar, pois é através dela que o algoritmo de otimização avalia o desempenho do modelo e determina como reajustar os pesos para melhorar o aprendizado. Há várias funções de custo que podem ser escolhidas para o treinamento do modelo. Esta escolha deve ser efetuada levando em consideração qual tipo de problema o aprendizado de máquina visa resolver (**Wang2022**).

Tabela 2: *Principais funções de perda para diferentes categorias de aprendizado de máquina.*

Categoria	**Funções de Perda**
Problema de classificação	Perda 0-1, Perda Perceptron, Perda Logarítmica
Problema de regressão	Perda Quadrática, Perda Absoluta, Perda de Huber
Aprendizado não supervisionado	Erro Quadrático, Erro de Distância, Erro de Reconstrução

Fonte: Adaptado de (**Wang2022**)

Na definição do treinamento da rede, existem alguns ajustes chamados de hiperparâmetros. Os hiperparâmetros incluem número de épocas, tamanho do lote e taxa de aprendizado. Cada época é definida como uma passagem completa por todos os dados de treinamento. O tamanho do lote é o número de amostras que o modelo processará antes de atualizar os pesos. A taxa de aprendizado é o quão rápido os pesos do modelo são atualizados.

Na prática, o treinamento de modelos de aprendizado de máquina é desenvolvido utilizando ferramentas próprias para este fim. Com base em (**Kallimani2024**), podemos citar LiteRT, desenvolvido pela Google para dispositivos Android, iOS, Linux e microcontroladores, assim como μTensor desenvolvido pela ARM e Edge Impulse desenvolvido por Zach Shelby e Jan Jongboom.

3.3 Quantização para compactação de Modelos

Um importante passo no desenvolvimento de soluções TinyML é a quantização do modelo. A etapa de quantização, segundo (**quantization**), literalmente compacta o modelo, reduzindo seu tamanho para que possa ser executado em dispositivos embarcados. A realização da quantização é feita através do mapeamento dos pesos do modelo de ponto flutuante de 32 bits, por exemplo, para inteiros de 8 ou 4 bits. A quantização de modelos em contexto de TinyML é essencial para que a solução consuma uma quantidade menor de recursos. Contudo, a escolha do tipo de quantização deve ser feita com cautela, pois impacta diretamente a precisão e acurácia do modelo.

3.4 Validação, Ajuste Fino e Métricas de Desempenho

A validação dos modelos de aprendizado de máquina consiste em avaliar se o modelo está desempenhando adequadamente sua função. Como apresentado na Seção 2.2, uma prática comum é reservar de 20% a 30% dos dados coletados para testes, após o treinamento. Com base nesses testes, são calculadas métricas de desempenho. As principais métricas para avaliação de classificadores binários, segundo (**Rainio2024**), são: **acurácia** (ACU), **sensibilidade** (SEN), **especificidade** (ESP), **precisão** (PRE) e **medida F1** (F1), definidas nas Equações 1, 2, 3, 4 e 5, respectivamente.

$$\text{ACU} = \frac{\text{VP} + \text{VN}}{\text{VP} + \text{VN} + \text{FP} + \text{FN}} \quad \in [0, 1], \tag{1}$$

$$\text{SEN} = \text{Rec.} = \frac{\text{VP}}{\text{VP} + \text{FN}} \quad \in [0, 1], \tag{2}$$

$$\text{ESP} = \frac{\text{VN}}{\text{VN} + \text{FP}} \quad \in [0, 1], \tag{3}$$

$$\text{PRE} = \frac{\text{VP}}{\text{VP} + \text{FP}} \quad \in [0, 1]. \tag{4}$$

A medida F1 é definida como a média harmônica entre a precisão e a sensibilidade.

$$\text{F1} = 2 \cdot \frac{\text{PRE} \cdot \text{SEN}}{\text{PRE} + \text{SEN}} \tag{5}$$

Onde:

- **VP**: Verdadeiros Positivos, número de instâncias corretamente classificadas como positivas.
- **VN**: Verdadeiros Negativos, número de instâncias corretamente classificadas como negativas.
- **FP**: Falsos Positivos, número de instâncias incorretamente classificadas como positivas.
- **FN**: Falsos Negativos, número de instâncias incorretamente classificadas como negativas.

Ademais, existem algumas métricas com apelo mais visual, como é o caso do gráfico denominado Característica de Operação do Receptor (COR). Um gráfico COR é obtido a partir da variação do Limiar de Decisão, calculando especificidade e sensibilidade e então plotando num gráfico. A Figura 2 exibe um exemplo de curva COR, um comparativo entre o desempenho de dois modelos.

Um gráfico COR de um modelo ideal seria composto por uma reta vertical de (1,0) até (1,1) e então uma reta horizontal de (1,1) até (0,1), indicando máxima sensibilidade e especificidade. Ainda sobre o gráfico COR, uma métrica para desempenho é a Área Sobre a Curva (ASC). No caso ideal, ASC = 1. Desta forma, quanto mais tender ao ponto (1,1) o gráfico COR, melhor o modelo. Da mesma forma, quanto mais elevado o indicador ASC, melhor.

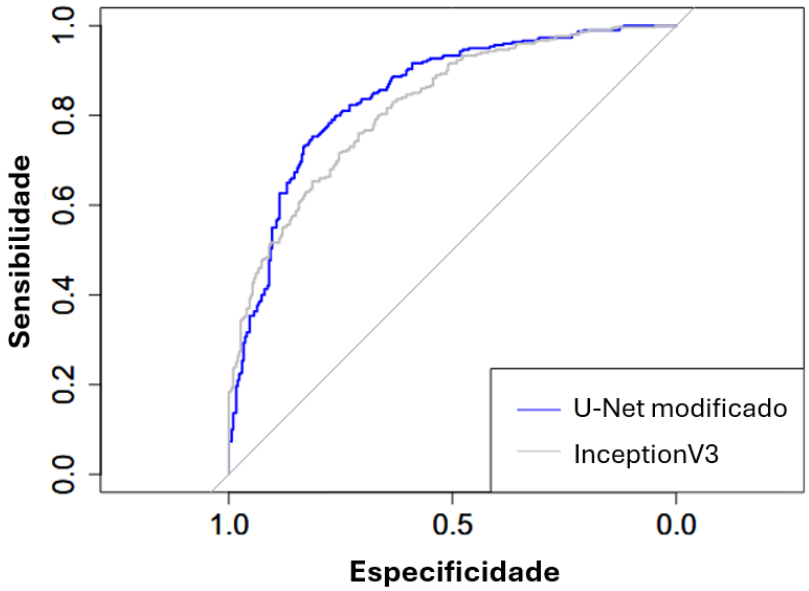

Figura 2: *Exemplo de um gráfico COR.*

Fonte: Adaptado de (**Rainio2024**)

Ainda sobre métricas visuais, temos a Matriz de Confusão. Esta matriz relaciona os valores previstos pelo modelo com os valores reais. A Figura 3 mostra um exemplo deste tipo de métrica.

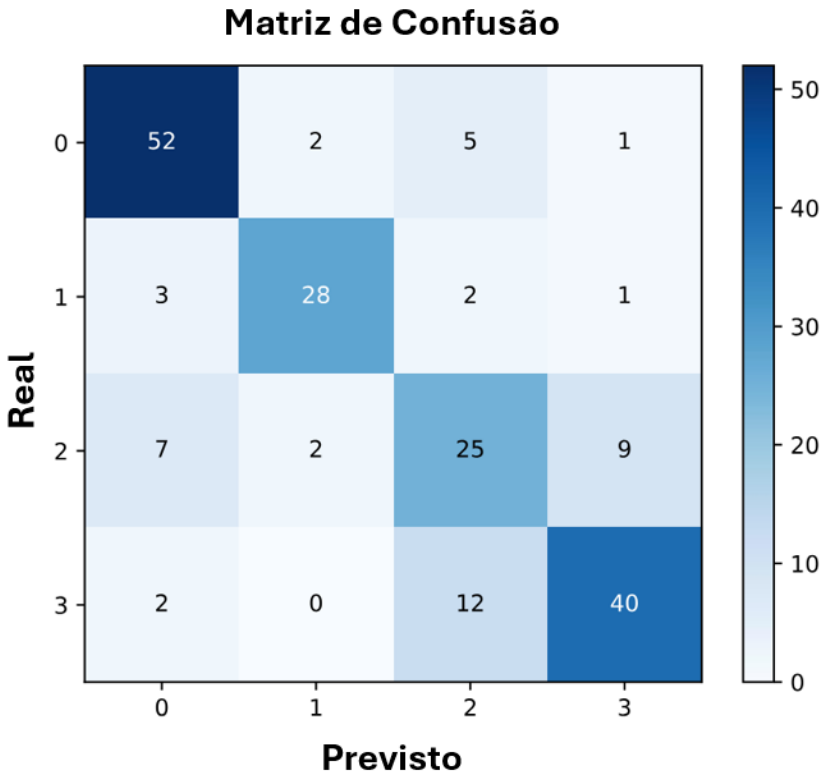

Figura 3: *Exemplo de uma Matriz de Confusão.*

Fonte: Adaptado de (**Alshammari2024**)

No caso de um modelo ideal, todos os valores da matriz de confusão estariam na

diagonal principal da matriz, indicando que o modelo acerta 100% das predições.

Finalmente, caso as métricas de desempenho do modelo não forem satisfatórias, ajustes nos hiperparâmetros podem ser feitos a fim de melhorar a resposta do modelo. Esta etapa é denominada Ajuste Fino e será aprofundada na segunda parte deste trabalho, Trabalho de Conclusão II.

3.5 Implantação em Dispositivos Embarcados

Há vários dispositivos no mercado que podem trabalhar com soluções TinyML. (**Oliveira2024**) realiza uma pesquisa dos principais dispositivos no mercado comparando suas configurações. Estas comparações são exibidas na Tabela 3.

Tabela 3: *Especificações de dispositivos TinyML de baixo e alto desempenho.*

Categoria	Placa	Custo (US$)	Corrente (A)	RAM (MB)	Arquitetura	Freq. (MHz)
Low-end	Arduino Nano BLE 33	∼25	0.5	0.256	32-bit	64
	Adafruit EdgeBadge	∼36	1.0	0.192	32-bit	120
	ESP32 DevKitC	∼10	0.5	0.512	32-bit	240
	Raspberry Pi Pico W	∼7	0.5	0.256	32-bit	133
	STM32F746	∼15	0.5	0.5	32-bit	216
	Sipeed Maix Bit	∼45	1.0	8.0	64-bit	600
	SparkFun Edge	∼17	0.5	0.384	32-bit	96
High-end	Banana Pi M2 Zero	∼20	2.0	512	32-bit	1200
	Orange Pi Zero 3	∼35	2.0	1000	64-bit	1500
	Raspberry Pi Zero W	∼15	1.2	512	32-bit	1000
	Raspberry Pi Zero 2 W	∼20	1.2	512	64-bit	1000
	Raspberry Pi 3 model B	∼35	2.5	1000	64-bit	1200
	Raspberry Pi 4 model B	∼35	3.0	4000	64-bit	1800
	Raspberry Pi 5	∼56	5.0	4000	64-bit	2400
	NVIDIA Jetson Nano	∼99	5.0	4000	64-bit	1430

Fonte: Adaptado de (**Oliveira2024**)

É fundamental que a plataforma escolhida e o modelo desenvolvido sejam compatíveis, ou seja, suas configurações e requisitos devem ser alinhados para evitar desperdício de recursos computacionais e garantir uma solução final eficiente. Além disso, é essencial considerar o consumo de energia e garantir que a escolha atenda às restrições do projeto.

A implantação do modelo final varia conforme a plataforma de desenvolvimento escolhida. Principais ferramentas como TensorFlow e Edge Impulse permitem gerar bibliotecas contendo o modelo treinado quantizado completo, prontas para integração no código do dispositivo embarcado, conforme descrito em (**edgeDocs**) e (**tensorDocs**). Ambas oferecem suporte a diversas linguagens de programação, como C++ e Python, garantindo flexibilidade no desenvolvimento e na integração com diferentes tipos de hardware.

Um comparativo entre as ferramentas TensorFlow e Edge Impulse pode ser visto em (**EssanoahArthur2024**), que compara o desempenho de dois modelos que servem ao mesmo propósito, que neste caso é identificação de doenças em folhas de milho.

As Tabelas 4 e 5 mostram uma comparação entre os consumos de recursos computacionais e uma comparação das métricas de desempenho, respectivamente.

Tabela 4: *Comparação de desempenho entre TensorFlow e Edge Impulse.*

Métrica	TensorFlow	Edge Impulse
RAM	890,5 kB	726,6 kB
Flash	374,4 kB	344,7 kB
Latência	84,99 ms	76,48 ms

Fonte: Adaptado de (**EssanoahArthur2024**)

Tabela 5: *Comparação de métricas de desempenho entre TensorFlow e Edge Impulse.*

Métrica	TensorFlow	Edge Impulse
Acurácia no treinamento	97%	96%
Acurácia na validação	96,54%	95%
Acurácia no teste	96,38%	94,84%

Fonte: Adaptado de (**EssanoahArthur2024**)

Por fim, a escolha definitiva do dispositivo embarcado e da plataforma de desenvolvimento será realizada no Trabalho de Conclusão II, considerando critérios como custo, facilidade de uso, compatibilidade com o modelo treinado e requisitos de desempenho.

4 PROPOSTA DE TRABALHO

Neste capítulo será apresentada a proposta de trabalho para o TCC-II, com base nos conceitos e técnicas discutidos nos capítulos anteriores. O objetivo é desenvolver uma solução real de aprendizado de máquina embarcado utilizando o fluxo de trabalho proposto.

4.1 Descrição da proposta

Em continuidade aos objetivos traçados neste trabalho, a proposta do TCC-II é aplicar, na prática, o fluxo de desenvolvimento TinyML para resolver um problema real de classificação de resíduos sólidos. O projeto busca consolidar os conhecimentos adquiridos ao longo do trabalho, conectando teoria e prática, e evidenciar, de forma aplicada, as vantagens do aprendizado de máquina embarcado. Além disso, será realizada uma comparação entre a solução TinyML desenvolvida e abordagens tradicionais de aprendizado de máquina, de modo a destacar os diferenciais e limitações de cada abordagem no contexto do problema proposto.

A proposta de trabalho consiste no desenvolvimento de um modelo de aprendizado de máquina embarcado para categorização de lixo em 12 categorias diferentes:

- Pilhas e baterias;
- Orgânico;
- Vidro marrom;
- Vidro transparente;
- Vidro verde;
- Papelão;
- Roupas;
- Metal;
- Papel;
- Plástico;
- Calçados;
- Rejeitos.

Esta proposta foi definida por tratar-se de um tema altamente relevante para o meio ambiente e pela ampla disponibilidade de dados em repositórios públicos, o que facilita a condução do fluxo de trabalho. O modelo será treinado utilizando a plataforma TensorFlow, visto que se trata de uma solução gratuita e de código aberto, além de apresentar resultados superiores de acurácia e desempenho conforme discutido na Seção 3.5. A coleta dos dados será realizada utilizando o repositório de dados públicos do Kaggle (**kaggle**), que disponibiliza um conjunto de dados com imagens de lixo categorizadas, facilitando o treinamento do modelo.

Durante a execução do TCC-II, a escolha do conjunto de dados foi revisada. Inicialmente, havia sido planejada a utilização de um banco de dados público do Kaggle (**kaggle**) para o treinamento do modelo de classificação de resíduos. Contudo, optou-se pela adoção do conjunto de dados *RealWaste* (**realwaste2023**), disponibilizado pelo UCI Machine Learning Repository (*UCI*, 2024), o qual está associado a um artigo publicado, possibilitando omparações diretas entre o modelo desenvolvido neste trabalho e o apresentado no estudo de referência. Os detalhes sobre o novo conjunto de dados são descritos na Seção 5.1.

4.2 Etapas do trabalho

O trabalho será desenvolvido em etapas, baseada no fluxo de trabalho apresentado no Seção 3.5 com a adição de alguns pontos. Ressalta-se que o fluxo poderá sofrer ajustes ao longo do TCC-II conforme os resultados e necessidades identificados durante as experimentações. As etapas são apresentadas a seguir com suas respectivas atribuições:

1. **Coleta de dados:** A primeira etapa consiste em obter imagens de lixo categorizadas a partir de repositórios públicos. Essa abordagem garante acesso a dados variados e já organizados em classes, essenciais para o treinamento de modelos de aprendizado de máquina. A diversidade do conjunto de dados é crucial para que o modelo possa generalizar bem a diferentes tipos de resíduos.

2. **Pré-processamento:** Após a coleta, as imagens passam por uma etapa de limpeza e normalização, que inclui o redimensionamento para tamanhos uniformes e o aumento de dados (*data augmentation*). Essas técnicas visam melhorar a qualidade do conjunto de dados, aumentar sua robustez e reduzir possíveis vieses durante o treinamento.

3. **Desenvolvimento e treinamento do modelo:** Nesta etapa, será implementada uma Rede Neural Convolucional (RNC) utilizando TensorFlow. Essa arquitetura é ideal para classificação de imagens, permitindo que o modelo aprenda padrões visuais complexos necessários para identificar diferentes categorias de lixo.

4. **Escolha da plataforma de hardware:** O hardware ideal para execução do modelo será avaliado com base na complexidade do programa e nos requisitos de desempenho, consumo energético e custo. Essa etapa garante que o dispositivo escolhido seja compatível com as restrições de um sistema embarcado.

5. **Quantização do modelo:** Técnicas de quantização serão aplicadas para reduzir o tamanho e os requisitos computacionais do modelo, permitindo sua execução eficiente em dispositivos embarcados. Essa etapa é fundamental para garantir a viabilidade da solução TinyML.

6. **Validação e ajuste fino:** O modelo treinado será avaliado utilizando métricas como acurácia, precisão e *F1-score*. Com base nos resultados obtidos, ajustes serão realizados para melhorar o desempenho do modelo, corrigindo possíveis falhas de generalização ou subaproveitamento.

7. **Implantação:** Após a validação, o modelo quantizado será integrado em um microcontrolador compatível. Testes práticos serão realizados para avaliar o desempenho da solução em um ambiente real, verificando sua eficiência e tempo de resposta.

8. **Avaliação dos resultados:** Nesta etapa, serão realizados comparativos entre as métricas de desempenho elencadas na Seção 3.4 da solução TinyML desenvolvida e aplicações de inteligência artificial tradicionais. O objetivo é destacar as vantagens em termos de eficiência, latência, custo e desempenho.

9. **Escrita do Trabalho de Conclusão II:** A escrita do relatório será uma atividade contínua, desenvolvida em paralelo às demais etapas do cronograma. Isso assegura que todos os passos sejam documentados detalhadamente, facilitando a organização e o cumprimento dos prazos acadêmicos.

Demais detalhes de cada etapa serão discutidos e decididos durante o desenvolvimento do trabalho, considerando as especificidades do modelo e do dispositivo embarcado escolhido.

5 DESENVOLVIMENTO

Este capítulo apresenta a execução prática do fluxo de trabalho proposto no TCC-I, abrangendo todas as etapas do desenvolvimento da solução TinyML para classificação de resíduos sólidos. São detalhados o processo de seleção e preparação do conjunto de dados, a implementação e o treinamento do modelo em ambiente de nuvem, as técnicas de quantização aplicadas e a implantação da rede neural em um dispositivo embarcado.

5.1 Coleta e seleção do banco de dados

Inicialmente, havia sido planejada a utilização de um conjunto de dados público do *Kaggle* (**kaggle**), contendo 12 categorias distintas de resíduos sólidos. No entanto, optou-se por empregar o conjunto de dados *RealWaste* (**realwaste2023**), proveniente do UCI Machine Learning Repository, em virtude de estar associado ao artigo *A Novel Real-Life Data Set for Landfill Waste Classification Using Deep Learning* (**rwPaper**). Essa associação permite a realização de comparações diretas entre o modelo desenvolvido neste trabalho e o apresentado no estudo de referência, fortalecendo a análise dos resultados.

O conjunto *RealWaste* é composto por imagens reais de resíduos coletados em aterros sanitários, classificadas conforme o tipo de material predominante. Os rótulos atribuídos às imagens representam a categoria principal. A Tabela 6 apresenta as categorias e a quantidade de amostras disponíveis em cada uma delas.

Tabela 6: *Categorias e quantidade de imagens no conjunto de dados RealWaste.*

Categoria	**Quantidade de imagens**
Papelão	461
Resíduos orgânicos alimentares	411
Vidro	420
Metal	790
Lixo diverso	495
Papel	500
Plástico	921
Têxteis	318
Vegetação	436
Total	**4.752**

O total de 4.752 imagens fornece um conjunto de dados de tamanho moderado, adequado para o treinamento de redes neurais convolucionais compactas voltadas a

aplicações embarcadas. O dataset apresenta boa diversidade de materiais, o que contribui para a generalização do modelo em cenários reais.

A Figura 4 apresenta exemplos de imagens pertencentes ao conjunto de dados *RealWaste*, ilustrando algumas das categorias utilizadas no treinamento do modelo. As amostras demonstram a variabilidade visual entre diferentes tipos de materiais, evidenciando as diferenças de textura, cor e formato que o modelo deve aprender a distinguir durante o processo de classificação.

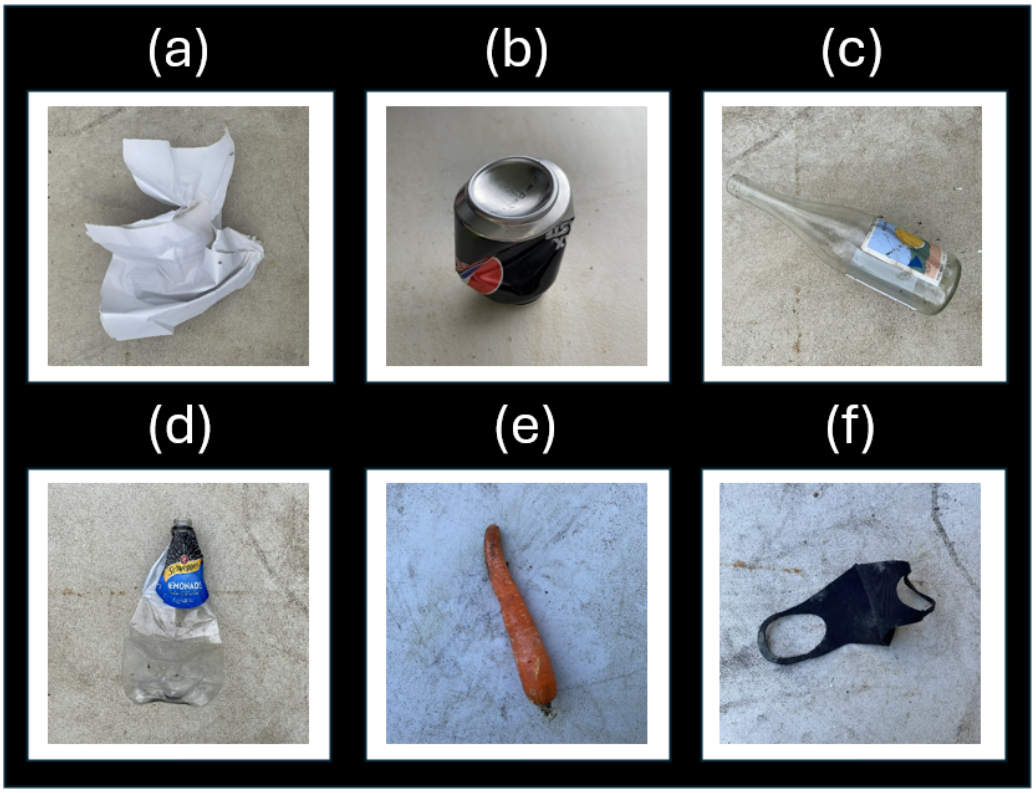

Figura 4: *Exemplos de imagens do conjunto de dados RealWaste: (a) papel; (b) metal; (c) vidro; (d) plástico; (e) resíduo orgânico alimentar; (f) resíduo têxtil.*

Fonte: Adaptado de (**realwaste2023**)

5.2 Pré-processamento dos dados

O conjunto de dados *RealWaste* já se apresenta homogêneo quanto à resolução (522×522 px) e à organização em subpastas por classe, o que dispensou etapas extensas de limpeza, remoção de ruído ou correções manuais. Assim, adotou-se um pré-processamento propositalmente simples, com dois objetivos principais: (i) adequar o tamanho das imagens à entrada da rede e (ii) balancear as classes para reduzir vieses no treinamento.

As etapas executadas foram:

1. **Redimensionamento para a entrada do modelo.** Durante o carregamento, cada imagem foi redimensionada para 224×224 px, tamanho compatível com arquiteturas convolucionais compactas e com as restrições de memória/processamento do alvo embarcado. Esse redimensionamento é realizado diretamente no script ao chamar `load_img(..., target_size=(224,224))`, garantindo que todo o conjunto balanceado seja gravado já no tamanho esperado pelo modelo.

2. **Balanceamento via aumento de dados (data augmentation).** Após contabilizar as imagens por classe, definiu-se uma meta por classe igual ao maior valor entre a classe mais populosa do conjunto. Na prática, isso faz com que todas as classes sejam expandidas até o tamanho da classe majoritária, que neste caso é de 921. Para gerar amostras sintéticas preservando a semântica, aplicaram-se apenas transformações geométricas e fotométricas leves com *ImageDataGenerator*: rotação ($\pm 20°$), deslocamentos horizontais e verticais (até 20%), cisalhamento (0,15), *zoom* (0,15), espelhamento horizontal e variação de brilho entre 0,7 e 1,3. O espelhamento vertical foi evitado por potencialmente alterar a plausibilidade física de algumas cenas. Limitou-se a no máximo 10 variações por imagem base, reduzindo redundância. As imagens originais redimensionadas foram copiadas para o diretório de saída e, em seguida, as variações foram geradas até atingir a meta por classe.

Todo o procedimento está implementado em `metadados/aumento_de_dados.py`, que mantém a estrutura de diretórios por classe, grava o conjunto balanceado em disco e facilita a reprodução dos resultados.

5.3 Desenvolvimento e treinamento do modelo

Esta seção descreve a implementação prática do modelo de classificação, desde a preparação do fluxo de dados (*pipeline*) até o treinamento com ajuste fino. A implementação foi conduzida em ambiente de nuvem (Google Colab) utilizando TensorFlow/Keras. Essa escolha apoia-se na facilidade de acesso (basta um navegador), disponibilidade gratuita — ainda que limitada — de CPU/GPU/TPU e integração direta com Google Drive para armazenamento e compartilhamento de artefatos.

5.3.1 Fluxo de dados

Os conjuntos de treino e validação foram criados a partir do diretório base do *RealWaste*, empregando TensorFlow, com divisão estratificada aproximada de 80/20 e semente 123 para reprodutibilidade. As imagens foram redimensionadas para 224×224 px e carregadas em **lotes** de 64 amostras.

5.3.2 Arquitetura e estratégia de treinamento

Utilizou-se a **MobileNetV2** como extratora de atributos base (`include_top = False`, `weights = 'imagenet'`, `alpha = 1`), por ser uma arquitetura leve, adequada ao contexto TinyML. Sobre essa base adicionaram-se camadas finais para classificação multiclasse:

- `GlobalAveragePooling2D`;
- `Dropout`(0,2) para regularização;
- `Dense`(128, ReLU);
- `Dense`(#classes, Softmax).

A estratégia de treinamento adotou duas fases principais:

1. **Pré-treino**: base congelada (`base_model.trainable = False`) por 5 épocas, para estabilizar as camadas adicionadas;
2. **Ajuste fino**: base liberada parcialmente, mantendo congeladas todas as camadas exceto as **80 últimas**, permitindo especialização progressiva dos filtros superiores à tarefa.

5.3.3 Função de perda, otimizador e mecanismos de controle (callbacks)

O modelo foi compilado com `loss = 'sparse_categorical_crossentropy'` e métrica principal foi a acurácia. Como otimizador empregou-se **AdamW** com taxa de aprendizado inicial de 1×10^{-4} e *decaimento de peso* (*weight decay*) de 1×10^{-5}.

Para maior robustez frente a eventuais desequilíbrios residuais, utilizaram-se **pesos por classe** obtidos via `sklearn.utils.compute_class_weight` sobre os **rótulos** do conjunto de treino.

Dois mecanismos de controle (*callbacks*) acompanharam o processo de otimização:

- **Parada antecipada** (*EarlyStopping*) monitorando `val_loss`, com `patience = 10` e restauração dos melhores pesos;
- **Redução da taxa em platô** (*ReduceLROnPlateau*) com `factor = 0,5`, `patience = 3`, `min_lr = 1e-6`, reduzindo a taxa de aprendizado quando a validação estagna.

O treinamento completo foi limitado a **150 épocas** e interrompido pela parada antecipada quando apropriado.

5.3.4 Resultados do treino

Durante o treinamento, foram registradas as curvas de **acurácia** e **perda** para treino e validação, respectivamente. A Figura 5 ilustra a evolução dessas métricas e o ponto de parada determinado pela parada antecipada.

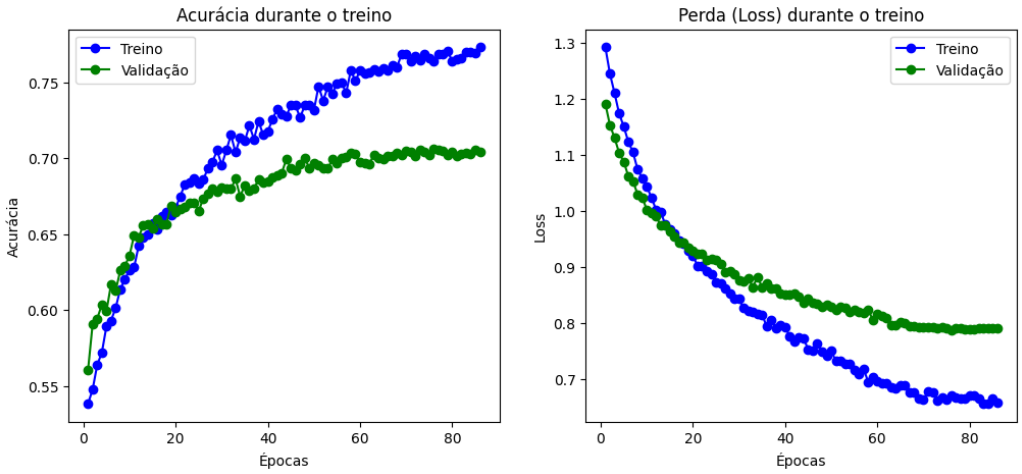

Figura 5: *Curvas de acurácia e perda por época (treino e validação).*

Fonte: Do autor

Ao final, o modelo em ponto flutuante foi salvo para posterior conversão e quantização (Seção 5.4).

5.4 Quantização do modelo

Nesta etapa, o modelo treinado em ponto flutuante foi convertido para o formato TensorFlow Lite com **quantização pós-treinamento** para **INT8**, visando reduzir tamanho do arquivo do modelo, uso de memória e latência de inferência, tornando-o adequado a dispositivos embarcados com recursos limitados.

5.4.1 Abordagem adotada

Empregou-se **quantização estática** com calibração via *conjunto representativo*. Os passos implementados no código Python foram:

1. Criar o conversor:

```
converter = tf.lite.TFLiteConverter.from_keras_model(model)
```

2. Ativar otimizações:

```
converter.optimizations = [tf.lite.Optimize.DEFAULT]
```

3. Definir gerador representativo (100 lotes):

```
def representative_data_gen():
    for _ in range(100):
        imgs, _ = next(train_ds_iter)
        yield [tf.cast(imgs, tf.float32)]
converter.representative_dataset = representative_data_gen
```

4. Restringir operações a inteiros:

```
converter.target_spec.supported_ops =
   [tf.lite.OpsSet.TFLITE_BUILTINS_INT8]
```

5. Fixar tipos de entrada/saída:

```
converter.inference_input_type  = tf.uint8
converter.inference_output_type = tf.uint8
```

6. Converter e salvar:

```
tflite_model = converter.convert()
with open(tflite_model_path, "wb") as f:
    f.write(tflite_model)
```

5.4.2 Conjunto representativo

O gerador percorre lotes reais do fluxo de dados (pré-processados para 224×224) e fornece tensores `float32` ao conversor. Durante a calibração, o TensorFlow Lite estima **escala** e **ponto-zero** de cada tensor (pesos e ativações), permitindo mapear eficientemente valores originais para o intervalo inteiro $[0, 255]$ sem perda severa de informação estatística.

5.4.3 Validação do modelo quantizado

O arquivo quantizado foi carregado via `tf.lite.Interpreter`. Para a avaliação, os tensores foram alocados e os metadados de entrada/saída inspecionados; as imagens foram convertidas condicionalmente para `uint8` conforme o tipo requerido pelo tensor de entrada; em seguida, realizou-se a inferência com `invoke()` e coletaram-se as distribuições de probabilidades por classe. A partir dessas pontuações, foram calculadas a acurácia, a precisão ponderada, a sensibilidade ponderada, o F1 ponderado, além da matriz de confusão normalizada e das curvas COR em esquema multiclasse.

Em síntese, a quantização INT8 permitiu obter um modelo compacto e eficiente sem alterar o fluxo principal de treinamento, viabilizando sua execução futura em dispositivos embarcados conforme discutido nas próximas seções.

5.5 Escolha e preparação do hardware

Nesta seção é apresentada a justificativa da escolha da plataforma de execução do modelo e a preparação realizada para torná-la apta à inferência embarcada. Optou-se por utilizar uma **Raspberry Pi 3 Model B+** principalmente pela disponibilidade física imediata no início do projeto, eliminando custos e atrasos de aquisição. Somam-se a isso: o equilíbrio entre recursos computacionais (CPU quad-core 64-bit e 1 GB de RAM) e simplicidade de uso; a ampla comunidade e maturidade do ecossistema (documentação consolidada, suporte direto a Python, TensorFlow Lite e bibliotecas de captura de vídeo e GPIO); o suporte nativo a câmera para aquisição direta das imagens de teste; e a compatibilidade com o fluxo de quantização INT8 (uso do `tflite_runtime`) sem necessidade de aceleração especializada. Esses fatores tornam a placa adequada como alvo intermediário: suficientemente capaz para validar latência e robustez do modelo convolucional quantizado, mas ainda representativa de um ambiente de recursos restritos frente a soluções de maior porte.

A Tabela 7 resume as principais especificações técnicas empregadas no protótipo.

Tabela 7: *Especificações principais da Raspberry Pi 3 Model B+.*

Característica	**Especificação**
Processador	Broadcom BCM2837B0, Cortex-A53 (ARMv8) 64-bit, 1,4 GHz
Memória RAM	1 GB LPDDR2 SDRAM
Conectividade	Wi-Fi 2,4/5 GHz 802.11 b/g/n/ac; Bluetooth 4.2 BLE
Portas USB	4 USB 2.0
GPIO	40 pinos (compatível com versões anteriores)
Saída de vídeo	1 porta HDMI
Periféricos	Conectores DSI (display) e CSI (câmera)
Áudio/Vídeo analógico	Conector P2 4 polos (áudio estéreo + vídeo composto)
Armazenamento	Slot microSD (SO e dados)
Alimentação	5 V / 2,5 A (mínimo) via micro USB

Fonte: Adaptado de (**rpi3**).

5.5.1 Preparação e configuração inicial

Os passos principais para preparar o dispositivo para a execução do modelo foram:

1. **Gravação do sistema operacional**: uso do Raspberry Pi Imager para instalar Raspberry Pi OS Lite (versão 64-bit), disponível no site oficial (**rpi3**);
2. **Acesso remoto**: habilitação de `SSH` e definição de hostname específico para identificação na rede local;
3. **Atualização de pacotes**: execução de atualização e cuidado em manter versão de Python compatível com dependências de inferência (TensorFlow Lite requer armv8 64-bit);
4. **Instalação das dependências**: instalação de `python3-pip`, bibliotecas de suporte (`numpy`, `tflite-runtime` ou `tensorflow-lite`);
5. **Habilitação de periféricos**: ativação opcional do suporte à câmera via `raspi-config`;
6. **Validação do ambiente**: teste de carregamento do modelo quantizado, verificação de memória disponível, tempo médio de inferência e temperatura sob carga leve.

5.6 Implementação e demonstração prática

Esta subseção descreve a execução prática do modelo quantizado em uma Raspberry Pi 3 Model B+, conforme especificado na Seção 5.5. O objetivo é demonstrar a captura de imagem, pré-processamento mínimo e inferência usando o `tflite_runtime`, validando o fluxo ponta-a-ponta em ambiente embarcado.

5.6.1 Fluxo de inferência

Assumem-se as seguintes bibliotecas importadas e o caminho do modelo definidos previamente:

```
import cv2, time; import numpy as np
from tflite_runtime.interpreter import Interpreter
MODEL_PATH = "mobilenet_test.tflite"
```

Os passos principais executados pelo script são:

1. Captura de um quadro da câmera.

```
cap = cv2.VideoCapture(0); ret, frame = cap.read(); cap.release()
```

2. Armazenamento da imagem original para registro e auditoria (`foto_teste.jpg`).

```
cv2.imwrite("foto_teste.jpg", frame)
```

3. Redimensionamento para 224×224 px (entrada do modelo).[1]

```
rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB); img = cv2.resize(rgb, (224,
    224), interpolation=cv2.INTER_AREA)
```

4. Expansão da dimensão de lote (shape `(1,224,224,3)`).

```
input_data = np.expand_dims(img, axis=0).astype(np.uint8)
```

5. Carregamento do arquivo `.tflite`, alocação de tensores e leitura dos descritores de I/O.

```
interpreter = Interpreter(model_path=MODEL_PATH);
    interpreter.allocate_tensors(); inp =
    interpreter.get_input_details()[0]; out =
    interpreter.get_output_details()[0]
```

6. Escrita dos dados de entrada e invocação da inferência (com medição simples de latência).

```
interpreter.set_tensor(inp['index'], input_data); start = time.time();
    interpreter.invoke(); latency_ms = (time.time() - start) * 1000
```

7. Leitura das probabilidades de saída e seleção da classe de maior valor (*argmax*).

```
output = interpreter.get_tensor(out['index'])[0]; pred_index =
    int(np.argmax(output))
```

[1] MobileNetV2 tradicionalmente utiliza normalização para [0,1] ou centragem em [-1,1]; o modelo quantizado foi calibrado diretamente em `uint8`, permitindo uso direto do frame bruto.

6 RESULTADOS E DISCUSSÃO

Neste capítulo, apresentamos e discutimos os resultados experimentais do estudo, analisando o desempenho do modelo nas principais métricas, a eficiência computacional da inferência em dispositivo embarcado e as limitações e implicações práticas dos achados à luz dos objetivos e da literatura.

6.1 Avaliação de desempenho

Esta seção sumariza o desempenho do classificador com base no roteiro executado no Google Colab, avaliando o modelo quantizado em INT8 sobre o conjunto de validação. São reportadas métricas globais (médias ponderadas), a matriz de confusão normalizada por classe e as curvas COR multiclasse.

6.1.1 Métricas globais

As métricas foram computadas via `scikit-learn` diretamente sobre as predições do intérprete TFLite:

- **Acurácia** (ACU): proporção de acertos no conjunto de validação;
- **Precisão ponderada** (PRE): média ponderada da precisão por classe, ponderada pelo suporte (número de amostras por classe);
- **Sensibilidade ponderada** (SEN): média ponderada da sensibilidade por classe, ponderada pelo suporte;
- **F1 ponderado**: média ponderada do F1 por classe, combinando precisão e sensibilidade.

Os resultados consolidados para o conjunto de validação são os exibidos na Tabela 8.

Tabela 8: *Métricas globais (validação) — modelo TFLite INT8.*

Métrica	Valor
Acurácia	0,68
Precisão (ponderada)	0,69
Sensibilidade (ponderada)	0,68
F1-score (ponderado)	0,68

Fonte: Do autor

6.1.2 Relatório de classificação por classe

O relatório por classe (precisão, sensibilidade e F1 por rótulo) é apresentado na Tabela 9. Os valores refletem o comportamento do classificador por categoria específica, auxiliando na identificação de classes com maior taxa de confusão.

Tabela 9: *Relatório de classificação por classe (validação).*

extbfClasse	**Precisão**	**Sensibilidade**	**F1-score**
Metal	0,67	0,58	0,62
Orgânicos	0,64	0,84	0,73
Papel	0,70	0,67	0,69
Papelão	0,70	0,72	0,71
Plástico	0,69	0,50	0,58
Rejeito	0,50	0,45	0,47
Têxtil	0,57	0,72	0,64
Vegetação	0,81	0,93	0,87
Vidro	0,88	0,74	0,80

Fonte: Do autor

6.1.3 Matriz de confusão

A Figura 6 apresenta a **matriz de confusão normalizada por classe** (em %), permitindo visualizar padrões de acerto/erro entre categorias. As diagonais indicam acertos; valores fora da diagonal revelam confusões recorrentes.

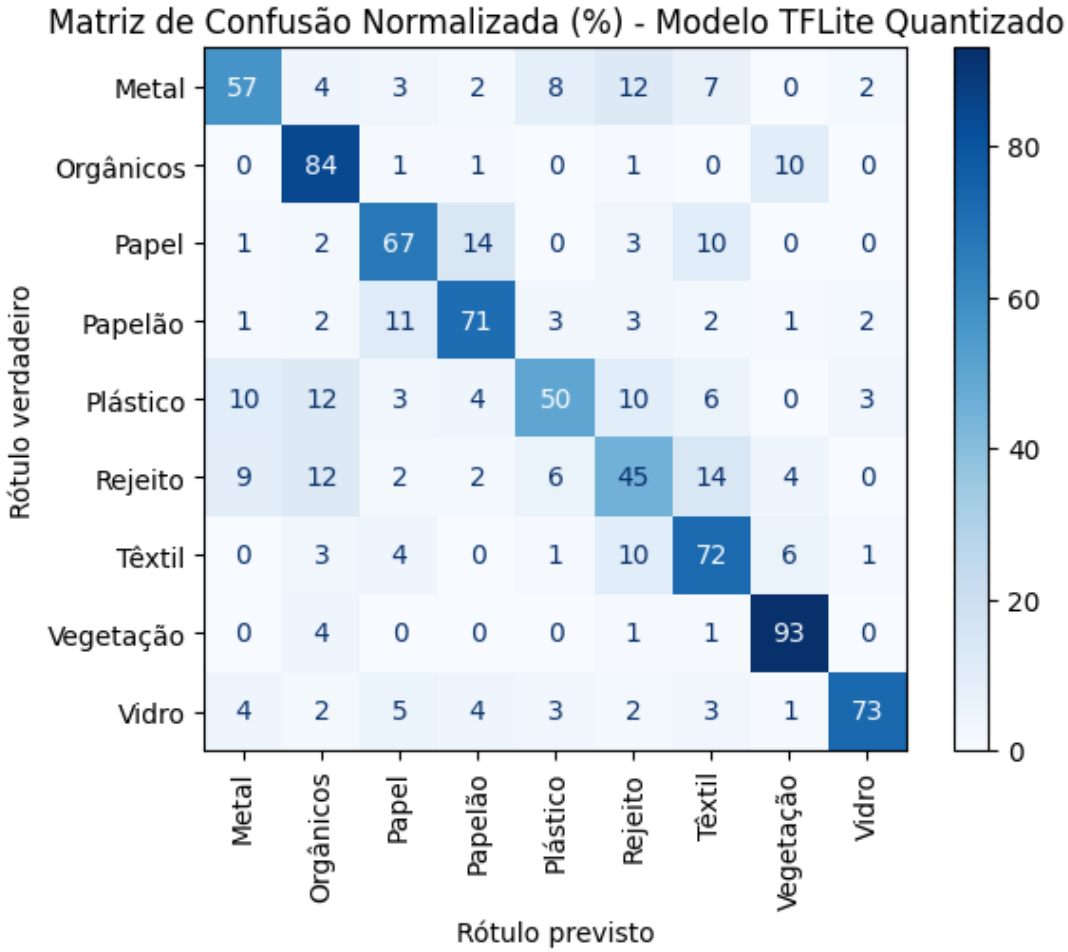

Figura 6: *Matriz de confusão normalizada (%).*

Fonte: Do autor

6.1.4 Curvas COR multiclasse

A Figura 7 mostra as **curvas COR por classe** e as respectivas áreas sob a curva (AUC), calculadas a partir dos escores de saída do modelo para cada rótulo. Em cenários multiclasse, cada curva contrasta o desempenho *um-versus-resto* de uma classe específica.

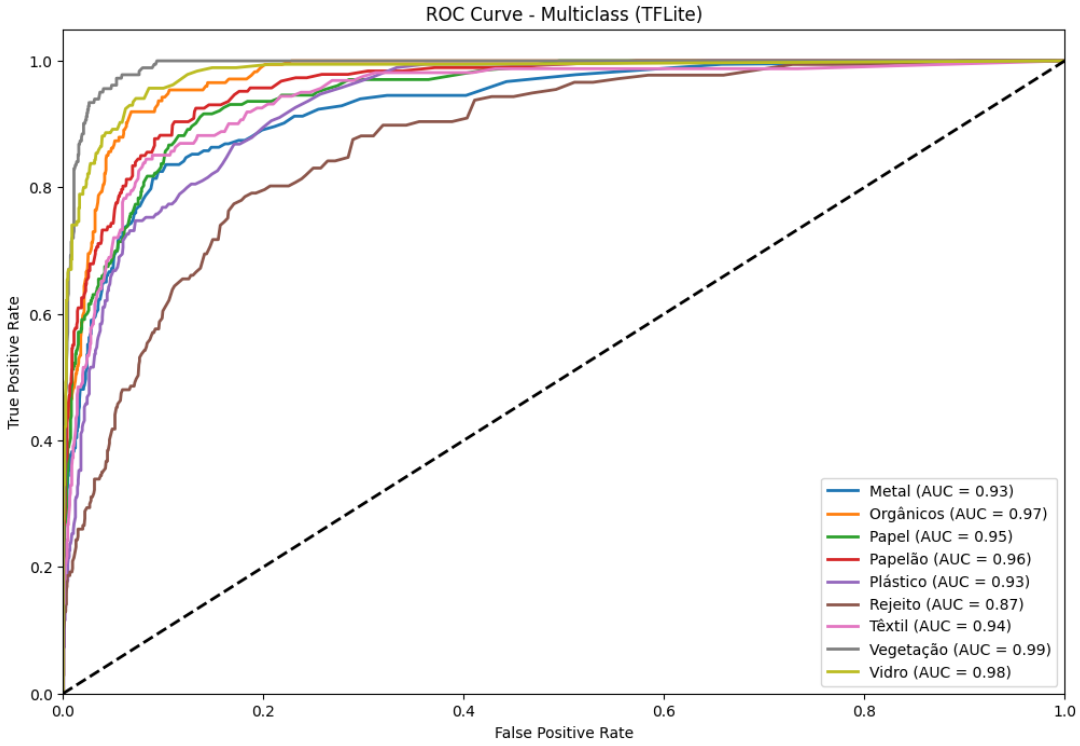

Figura 7: *Curvas COR por classe e respectivas AUCs.*

Fonte: Do autor

6.1.5 Discussão

Em linhas gerais, as médias ponderadas (PRE, SEN, F1) refletem o impacto do *suporte* de cada classe; assim, classes com menos amostras tendem a apresentar maior variância nas métricas individuais, influenciando menos o agregado ponderado. A leitura conjunta da matriz de confusão e das curvas COR permite identificar pares de classes mais propensos à confusão e orientar possíveis ajustes: coleta adicional para classes minoritárias, refinamento de aumento de dados direcionado e, se necessário, afrouxar ou ajustar limiares de decisão em cenários operacionais específicos.

6.2 Avaliação e análise de eficiência computacional

A eficiência computacional do sistema foi avaliada considerando três aspectos principais: (i) tempo médio de inferência por imagem, (ii) uso de CPU durante a execução do modelo e (iii) consumo de memória RAM. Todos os ensaios foram realizados utilizando a montagem descrita a seguir.

6.2.1 Montagem experimental

A Figura **??** apresenta a estrutura física utilizada para todos os ensaios. O suporte foi construído com tubos de PVC, permitindo fixar a webcam USB na posição superior, perpendicular ao plano de deposição dos resíduos. O Raspberry Pi utilizado para realizar

as inferências foi instalado diretamente na estrutura, garantindo estabilidade e reduzindo interferências externas. Essa montagem assegurou distância fixa entre câmera e objeto, iluminação uniforme e reprodutibilidade dos testes.

Figura 8: *Montagem utilizada nos ensaios, composta por tubos de PVC, presilhas de fixação, webcam USB e Raspberry Pi.*

Fonte: autor.

6.2.2 Resultados sobre o tempo de inferência

Durante os ensaios, observou-se que o sistema manteve tempos de inferência estáveis, próximos de 200 ms por imagem. A Figura **??** apresenta um exemplo do ensaio com um objeto metálico, onde o tempo de inferência registrado foi de aproximadamente 204 ms. Esse valor é representativo do comportamento geral do modelo durante todo o período de testes.

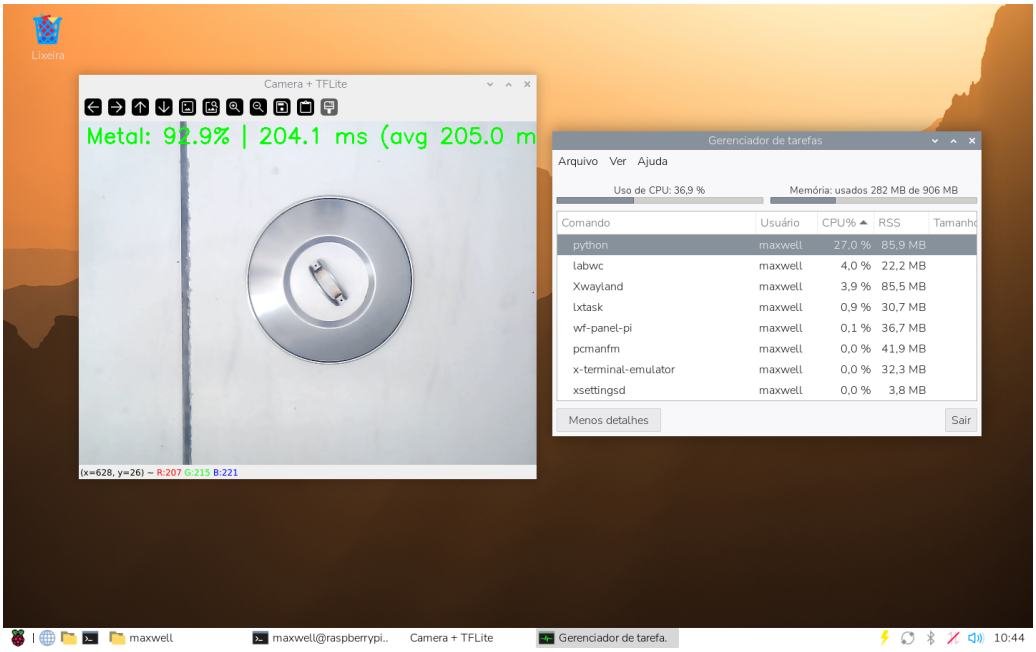

Figura 9: *Ensaio com objeto metálico, mostrando a predição da classe e o tempo de inferência.*

Fonte: autor.

Também foi avaliado o desempenho com resíduos da classe papel. A Figura **??** mostra que o sistema apresentou tempo de inferência da ordem de 220 ms, valor compatível com o observado nos demais ensaios.

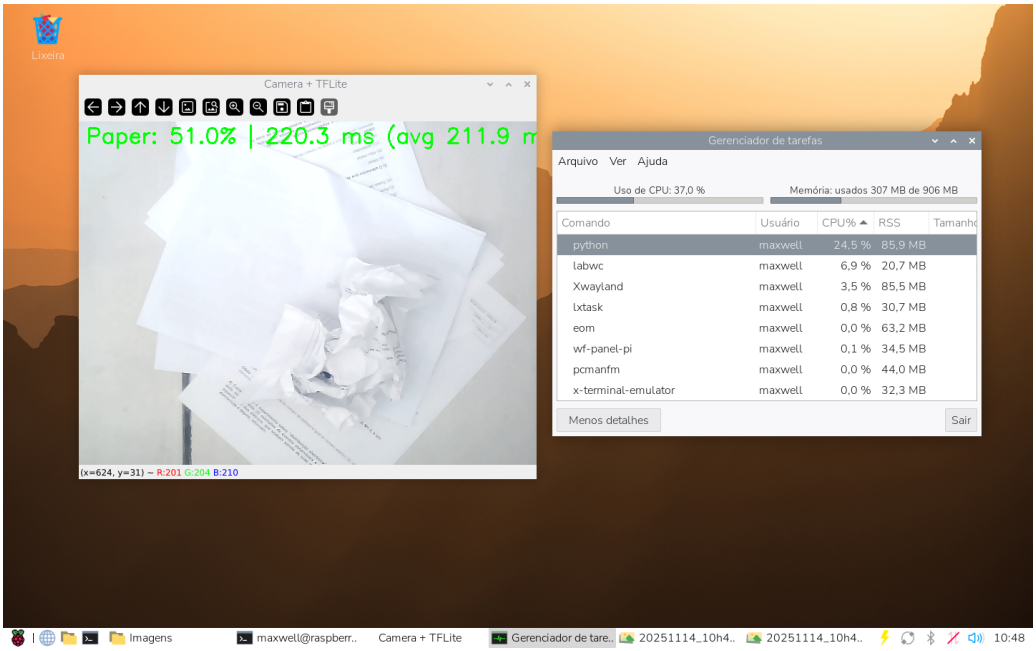

Figura 10: *Ensaio com papel, ilustrando o tempo de inferência registrado e a classificação produzida pelo modelo.*

Fonte: autor.

A Figura **??** corresponde ao ensaio com papelão, com tempo de inferência em torno

de 210 ms, mantendo o padrão observado nos demais testes e confirmando estabilidade temporal do modelo.

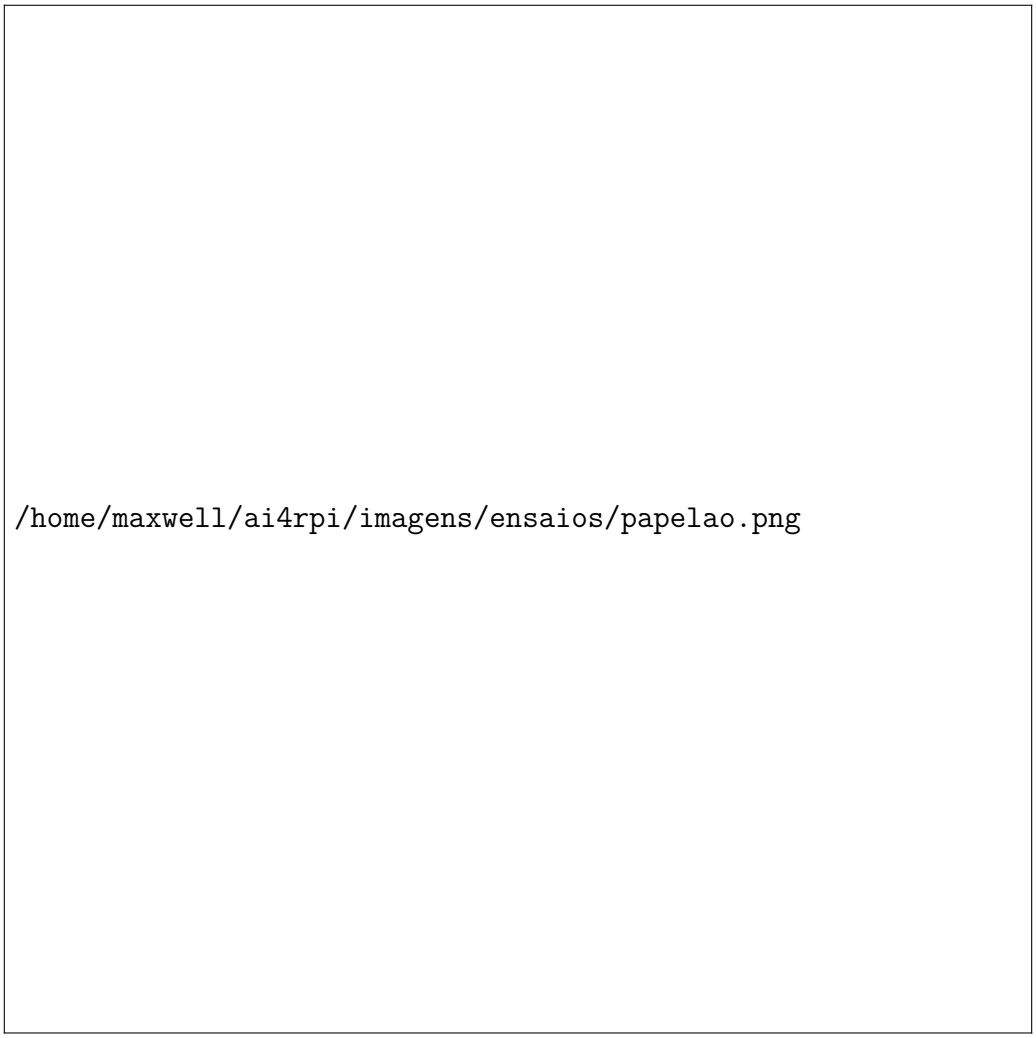

Figura 11: *Ensaio com papelão, exibindo o tempo de inferência e a predição realizada.*

Fonte: autor.

Por fim, a Figura **??** mostra o desempenho com material têxtil, novamente mantendo tempos de inferência próximos de 210 ms.

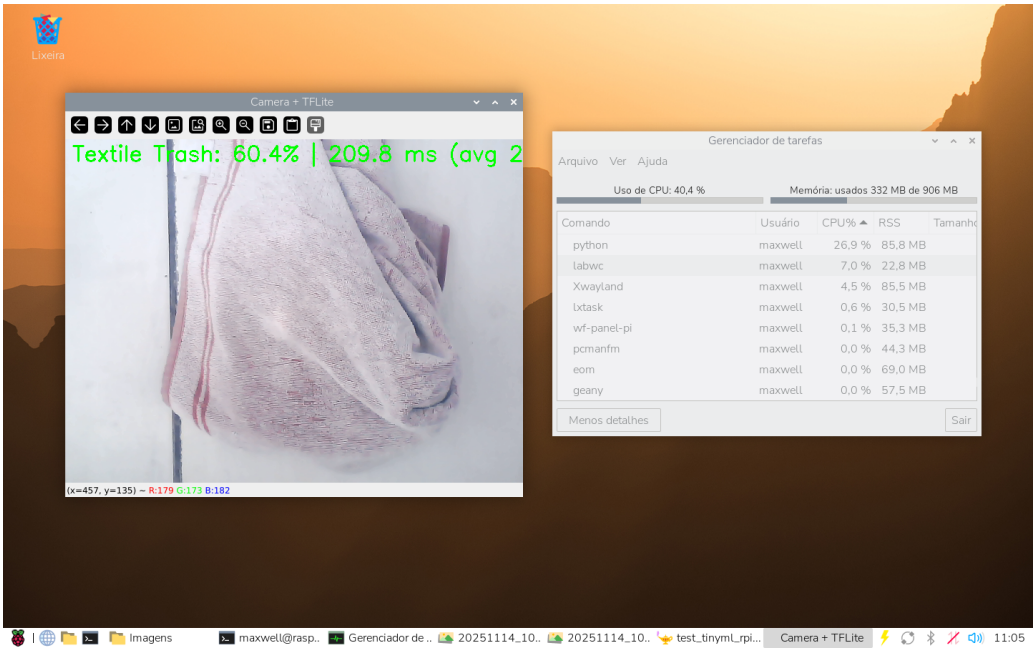

Figura 12: *Ensaio com resíduo têxtil, evidenciando tempo de inferência e classificação final.*

Fonte: autor.

6.2.3 Uso de CPU e memória RAM

Em todos os experimentos, o uso de CPU observado no processo `python` permaneceu entre 25% e 27%. Esse valor indica utilização praticamente completa de um dos quatro núcleos do Raspberry Pi, o que é esperado, pois o TensorFlow Lite, em sua configuração padrão, executa inferências utilizando apenas um núcleo de processamento.

O consumo de memória RAM apresentou valores entre 85 MB e 90 MB para o processo responsável pela inferência, enquanto o uso total do sistema permaneceu entre 280 MB e 330 MB. Esses valores confirmam que a aplicação opera de forma estável, sem risco de esgotamento dos recursos físicos do dispositivo.

6.2.4 Síntese dos resultados

Os resultados obtidos permitem concluir que o modelo MobileNetV2 quantizado executado via TensorFlow Lite apresenta desempenho adequado para operação embarcada em tempo real, com:

- latência média entre 205 ms e 215 ms por inferência;
- uso estável de um único núcleo de CPU (aprox. 27%);
- baixo consumo de memória RAM (cerca de 85 MB);
- estabilidade entre diferentes classes de resíduos.

Esses valores estão alinhados com trabalhos similares na literatura e confirmam que a solução proposta atende aos requisitos de desempenho definidos para o protótipo.

6.3 Limitações e sugestões de melhoria

7 CONCLUSÕES

O presente trabalho busca avaliar o estado atual da tecnologia e os desafios da aplicação de técnicas de aprendizado de máquina embarcado, o TinyML, em dispositivos com recursos computacionais limitados. A partir de uma revisão bibliográfica detalhada, foi possível compreender o contexto atual do TinyML e identificar como a latência, o consumo energético e a dependência de conectividade ainda são obstáculos relevantes para a adoção de soluções inteligentes em larga escala. Nesse cenário, o TinyML surge como uma alternativa promissora, permitindo o processamento local dos dados e reduzindo a necessidade de comunicação constante com a nuvem.

No decorrer deste trabalho, os três primeiros objetivos específicos propostos foram plenamente atingidos: realizou-se a pesquisa e avaliação das ferramentas TinyML, o levantamento e análise das plataformas disponíveis, bem como a elaboração de um fluxo de trabalho ponta-a-ponta para soluções TinyML. O quarto objetivo, referente ao desenvolvimento e aplicação prática do modelo TinyML para classificação de resíduos sólidos, será melhor abordado e concluído no TCC-II.

Durante o desenvolvimento do trabalho, foram analisadas as principais ferramentas e plataformas disponíveis para projetos TinyML, considerando critérios como facilidade de uso, integração com hardware e desempenho. Observou-se que o ecossistema de desenvolvimento está em rápida evolução, com soluções cada vez mais acessíveis tanto do ponto de vista de software quanto de hardware. A análise comparativa entre plataformas e dispositivos demonstrou que já é possível implementar modelos de aprendizado de máquina eficientes em microcontroladores e placas de baixo custo, ampliando o leque de aplicações práticas.

Além disso, foi elaborado um fluxo de trabalho ponta-a-ponta para o desenvolvimento de soluções TinyML, detalhando desde a aquisição e preparação dos dados até a quantização, validação e implantação do modelo em dispositivos embarcados. Destacou-se a importância da escolha adequada dos modelos, das técnicas de otimização e das métricas de avaliação, bem como a necessidade de ajustes finos para garantir o melhor desempenho possível dentro das limitações impostas pelo hardware.

Por fim, observa-se que a aplicação de TinyML em sistemas embarcados apresenta potencial para superar desafios atuais da IoT, especialmente em cenários críticos ou remotos. O trabalho prático proposto para o TCC-II buscará consolidar os conhecimentos adquiridos, aplicando o fluxo de trabalho desenvolvido em uma solução real de aprendizado de máquina embarcado, com o objetivo de validar na prática os conceitos discutidos e contribuir para o avanço da área.