UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO DE JOINVILLE
CURSO DE ENGENHARIA MECATRÔNICA.

ROMILSON FRANÇA FILHO

UTILIZAÇÃO DE TINYML PARA AVALIAÇÃO DE TESTE DE ESTANQUEIDADE

Joinville
2022

ROMILSON FRANÇA FILHO

UTILIZAÇÃO DE TINYML PARA AVALIAÇÃO DE TESTE DE ESTANQUEIDADE

Trabalho de Conclusão de Curso apresentado como requisito parcial para obtenção do título de bacharel em Engenharia Mecatrônica no curso de Engenharia Mecatrônica, da Universidade Federal de Santa Catarina, Centro Tecnológico de Joinville.

Orientador: Prof. Dr. Milton Evangelista de Oliveira Filho.

Joinville
2022

RESUMO

Machine learning (ML), é um ramo da inteligência artificial no qual tem como base o desenvolvimento de algoritmos que melhoram seu desempenho automaticamente com a experiência. Com isso, sistemas que aprendem a analisar e fazer inferências sobre diversos dados de forma automática, tendem a gerar eficiência e precisão em seu processo. Sistemas complexos que utilizam da ML requerem muitos gastos (computacionais, econômicos e de estrutura), motivo de sua aplicação não ser generalizada. O TinyML visa compactar ao máximo todas as dependências necessárias para aplicação da ML em todos as áreas e necessidades. Esses benefícios, aliados com o baixo custo energético, computacional e de projeto, oriundos das técnicas de TinyML, viabilizam a implementação de sistemas inteligentes em qualquer cenário. Tendo isso em mente, o objetivo deste trabalho é confeccionar um sistema de avaliação de peças, ao serem submetidas a testes de estanqueidade, utilizando-se dos princípios de TinyML. Tendo em vista a recente aplicação dessa tecnologia, não se conseguiu atingir o resultado esperado, porém serão exibidos os resultados e discussões de cada etapa do desenvolvimento da aplicação.

Palavras-chave: Inteligência Artificial. TinyML. Teste de Estanqueidade.

LISTA DE FIGURAS

LISTA DE QUADROS

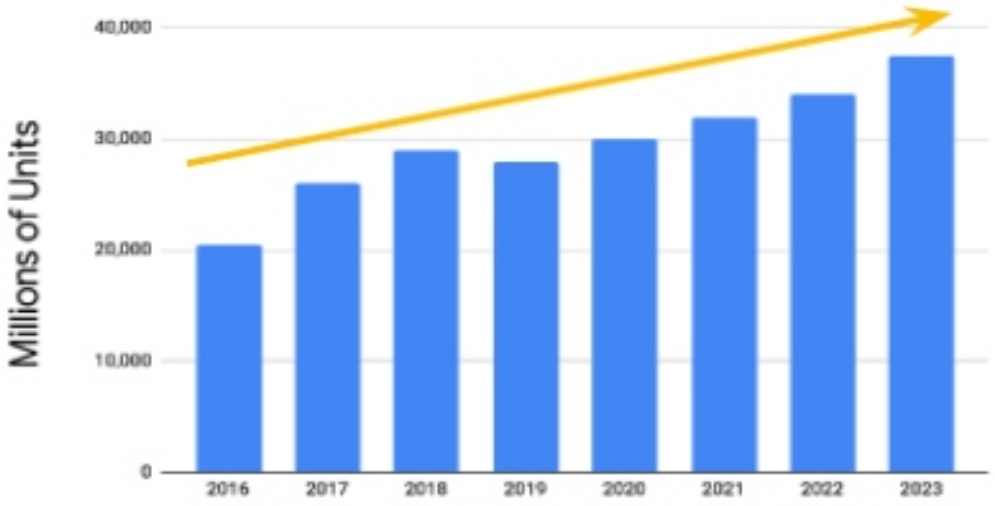
LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

O objetivo do *TinyML* (Tiny, do inglês traduzido para o português, pequeno; e ML abreviação de um ramo da IA denominado *machine learning*) é utilizar microprocessadores com capacidade limitada em processamento e memória, visando redução de custo e economia de energia. Com a solidificação destes conceitos, a utilização desse método pode-se expandir ainda mais comparada a presente realidade. A Figura 1 exibe um gráfico que mostra a evolução das vendas de *Microcontroller Unit* (MCU) durantes os últimos anos.

Figura 1 – Vendas de MCUs de 2016 a 2023



Fonte: Reddi (2020).

É esperado que em 2023 a venda por ano de MCU seja de 35 bilhões. Outro dado importante é a redução no custo final de um MCU, cotado para ser vendido em 2023 a 0,55 dólares.

Segundo Reddi (2020), microprocessadores executam diversas funções como comando de motores, leitura de sensores e muito mais. Porém, muito dos periféricos necessários para a realização destas funções não são integrados no mesmo chip, sendo sua função principal a de executor. A aplicação dos microprocessadores se encaixa em tarefas de execução com maior processamento, como gráficos de jogos, além de realizar diversos seguimentos de tarefas para a mesma aplicação.

Os microcontroladores, por sua vez, realizam tarefas específicas e simples com baixo processamento, pois periféricos e unidade de processamento estão um único encapsulamento. Esses periféricos são memórias EEPROM, Flash, SRAM, temporizadores, comparadores, conversores e muito mais (KENSHIMA, 2021).

Pode-ser observar nitidamente que a escolha do uso de um microprocessador ou microcontrolador não é de difícil análise, pois, se encaixam em demandas totalmente distintas.

Entretanto, com o surgimento de ferramentas que compactam os recursos necessários para desenvolvimentos de tecnologias *Internet of Things* (IoT), aplicações que requerem muito processamento podem ser desenvolvidas com ferramentas de baixo custo. O TinyML se baseia nisso: explorar o uso da machine learning em aplicações simples e específicas, otimizando consumo, espaço e dinheiro.

Diversos projetos já foram implementados usando TinyML nas áreas da agricultura, saúde e indústria.

A-) Agricultores na África, têm a possibilidade de cuidar de suas lavouras, com o uso de um App para smartphones que sem o uso da Internet auxilia no descobrimento e tratamento de pragas para suas plantações (RAMCHARAN, 2020).

B-) Um sistema de prevenção de larvas de mosquitos que transmitem Dengue, Zika entre outros, usa da TinyML para prever quando agitar águas paradas de maneira eficiente para evitar proliferação dos mosquitos e ter maior rendimento de energia (KAPUR, 2020).

Dado o potencial de aplicações do TinyML, este trabalho tem como objetivo o projeto e desenvolvimento de um sistema de avaliação de peças submetidas a testes de estanqueidade. Em muitas empresas, esse tipo de teste é feito ao imergir o produto em um reservatório de água e analisar a olho nu, o surgimento de bolhas no tanque decorrentes de frechas ou descontinuidades de uma peça em locais que deveriam estar isolados à vácuo (PIAZZETTA, 2017). O sistema substituirá a avaliação humana e irá garantir precisão e agilidade no processo produtivo.

1.1 OBJETIVOS

1.1.1 Objetivo Geral

Aplicação da tecnologia TinyML para avaliação de peças ao serem submetidas a testes de estanqueidade.

1.1.2 Objetivos Específicos

- Avaliar a aplicação de TinyML para desenvolvimento de projetos futuros;
- Mostrar cenários para a utilização de machine learning;
- Automatizar avaliações de testes.

2 FUNDAMENTAÇÃO TEÓRICA

Neste Capítulo serão abordados conceitos sobre machine learning, como desenvolver TinyML, desafios e aplicações da TinyML, e testes de estanqueidade. Desta forma pretende-se dar uma visão e compreensão geral do projeto.

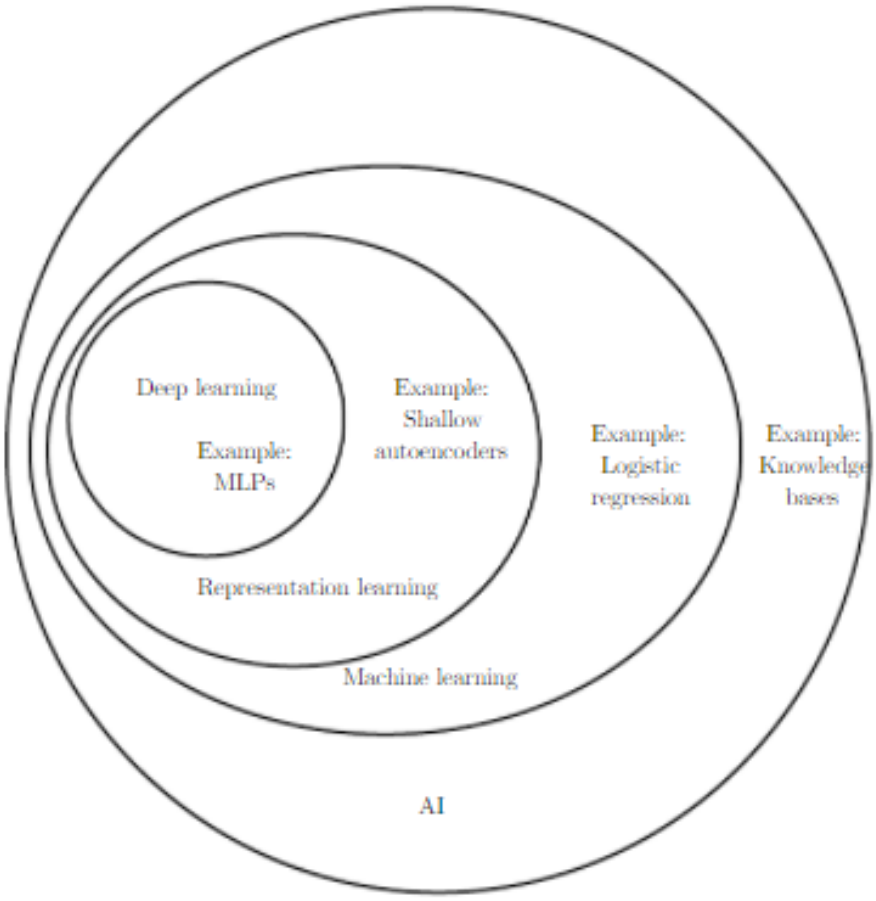
2.1 MACHINE LEARNING

De acordo com Jordan e Mitchel (2015) machine learning, é uma área de estudo que visa o aprendizado automático das máquinas com as experiências que ela tiver em seu tempo de uso. Por exemplo, utilizando ML podemos ensinar a maquina o que é um gato, assim como se ensina a uma criança; ou em ensinar como fazer um veículo autônomo parar para um pedestre atravessar a rua, assim como um instrutor de direção ensina seu aluno sobre os princípios e leis de trânsito.

Ou seja, machine learning é uma coleção de técnicas estatísticas para construir modelos matemáticos que podem fazer inferências a partir de amostras de dados (conhecido como conjunto de treinamento). Machine learning faz parte da inteligência artificial porém com um objetivo concreto: adaptar-se a um ambiente em mudança (RIBEIRO; GROLINGER; CAPRETZ, 2015).

Duas questões alicerçam o que é o ML e sua evolução: como construir sistemas computacionais que automaticamente melhoram por meio da experiência? Quais são as leis estatísticas, computacionais, informacionais e teóricas que governam todos esses sistemas de aprendizado incluindo computadores, humanos e organizações? (PIAZZETTA, 2017).A Figura 2 ilustra a categorização da machine learning dentro da área da inteligência artificial.

Figura 2 – Categorização das áreas de inteligência artificial



Fonte: Goodfellow e Courville (2016, p.9).

Nos primórdios da era computacional, já se pensava na ideia de tornar estas máquinas inovadoras em sistemas inteligentes (GOODFELLOW; COURVILLE, 2016). Hoje em dia, a utilização da inteligência artificial (IA) está muito difundida com aplicações práticas. Atualmente, existem softwares inteligentes para automatizar trabalhos rotineiros, entender conversas, falas, imagens, realizar diagnósticos de medicina.

Esses exemplos citados, somados com desafios da realidade humana (como jogos, soluções matemáticas) foram problemas que a IA deu resultados de imediato após seu desenvolvimento. Apenas com uma lista de regras matemáticas, o computador se tornava mais eficiente que o ser humano (GOODFELLOW; COURVILLE, 2016).

Ironicamente, tarefas fáceis para os seres humanos podem ser difíceis para serem solucionadas pela IA. Por exemplo, tarefas intuitivas e automáticas do ser humano, como reconhecer palavras ditas de uma maneira não tão clara porém carregada de um contexto. A vida cotidiana de um ser humano requer uma quantidade imensa de conhecimento, dados e cálculos, e sendo estes em sua maioria subjetivos, a transferência precisa deste tipo de conhecimento se torna de grande complexidade (GOODFELLOW; COURVILLE, 2016).

A Machine Learning atua diretamente nesses problemas: treinar o sistema para agir, reconhecer e avaliar dados não tão claros e decifráveis analiticamente. Para as diversas aplicações existentes da ML, existem técnicas/algoritmos que se encaixam de maneira mais eficiente a determinadas aplicações, como mostrado na Figura 2, onde em cada sub área da inteligência artificial temos um exemplo de algoritmo.

Na Figura 2, um exemplo de algoritmo utilizado em ML é apresentado: a Regressão Logística. Com esse algoritmo, pode-se recomendar um parto cesariana. Outro algoritmo, Naive Bayes, pode separar e-mails legítimos de spam. O modelo de ML a ser utilizado nesse projeto é o de Redes Neurais Artificiais (RNA), que será discutido nas seções subsequentes junto com seus esforços computacionais e demanda de energia no seu processamento (GOODFELLOW; COURVILLE, 2016).

2.1.1 Redes Neurais Artificiais

Em 1958, segundo LI Xinbo Chen e Zong (2016), Frank Rosenblatt propôs o conceito de Perceptron e uma teoria sobre o funcionamento dos neurônios no cérebro humano. Alicerçado nesses estudos, surgiu um novo campo da inteligência artificial, chamado redes neurais artificiais.

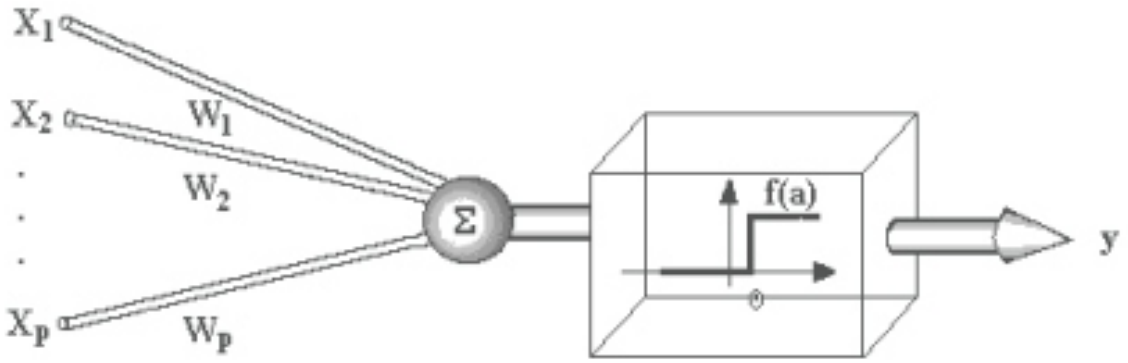
Diversas aplicações se beneficiam do uso da RNA para solução de problemas, porém a principal é o reconhecimento de padrões (RADOWITZ, 2009). Do ponto de vista humano, o reconhecimento de um padrão, seja ele qual for, compreende a técnica pela qual uma pessoa, uma vez havendo aprendido a reconhecer determinado assunto, poderá reconhecê-lo outra vez, mesmo que ocorra variações em relação ao primeiro modelo aprendido.

Por exemplo, se uma pessoa é ensinada a reconhecer o que é um cachorro por meio da exibição de uma raça A, ela ainda saberá que se trata do mesmo animal, caso se depare com um cachorro da raça B, que pode apresentar características físicas diferentes. Da mesma forma, essa capacidade de aprender e fazer reconhecimento de pessoas e objetos também funciona com as RNAs (RADOWITZ, 2009).

Uma RNA é formada por um conjunto de neurônios artificais conectados. As propriedades da rede são determinadas pela sua topologia e pelas propriedades dos neurônios (RUSSEL; NORVIG, 2013). As RNAs são compostas por nós ou unidades conectadas por ligações direcionadas. Cada ligação tem um peso numérico associado a ela, que determina a força e o sinal da conexão.

O processo computacional envolvido com a rede neural é a de concentrar diversas entradas oriundas de outros neurônios, combinadas por somas ponderadas, assim produzindo uma única entrada a partir das demais. A Figura 3 exemplifica o funcionamento de um neurônio.

Figura 3 – Representação do funcionamento de um neurônio



Fonte: Radowitz (2009, p.10).

A função básica de um neurônio é formar uma saída de acordo com as entradas e os pesos associados. A combinação dessas entradas por meio dos pesos, mais a produção de um estado de ativação por meio de uma função, determinam a saída. Os três componentes essenciais de um sistema computacional baseado em redes neurais artificiais são a função de ativação, a arquitetura e a regra geral de treinamento (RADOWITZ, 2009).

Como não basta conectar neurônios para que eles forneçam um resultado útil é necessário um método para treiná-los.

O treinamento das RNAs se baseia na habilidade da rede em modificar seu comportamento mediante eventos externos que disponibilizam um conjunto de entradas, o qual a rede busca um padrão. Após a execução consistente e correta deste aprendizado, a rede torna-se capaz de compor similaridades e generalizar situações que ainda não foram aprendidas (RADOWITZ, 2009).

2.1.2 Gastos computacionais e energéticos

A aplicação de redes neurais artificiais proporcionou, em meado dos anos noventa, que Yann Lecun desenvolvesse com sucesso um sistema que reconhecia cheques manuscritos (LI XINBO CHEN; ZONG, 2016). No entanto, o tempo de treinamento de sua rede neural era de aproximadamente três dias. O que mostra que o gasto e esforço computacional para realizar treinamentos em máquinas é elevado, e muitas vezes inviabilizando o uso de uma tecnologia tão poderosa.

Atualmente uma operação similar como a de Lecun demoraria cerca de segundos, porém o alto custo pela demanda de processamento e de energia continuam existindo. Devido a esses tais requisitos, o avanço de algoritmos e aplicativos baseados em RNA ainda possui empecilhos (LI XINBO CHEN; ZONG, 2016).

Avanços recentes em software e hardware, incluindo o uso de *graphics processing unit* (GPU) de alto rendimento para acelerar treinamento de rede neural, aliviaram este problema. Isto é, agora é possível treinar RNAs complexas em tempo razoável em hardwares relativamente baratos (LI XINBO CHEN; ZONG, 2016). Em contra-partida, o custo energético para acelerar os processos de treinamento e obter precisão sobre as inferências dadas pelo sistema, manteve-se alto e elevado.

A precisão com a qual uma rede neural executa, por exemplo, um reconhecimento de imagem ou som, é a sua principal métrica de eficiência. Desta forma, a comunidade de cientistas da área reconheceu a necessidade de implementação de RNAs que sejam precisas e energeticamente eficazes (LI XINBO CHEN; ZONG, 2016). Como resultado, em 2015 a *IEEE Rebooting Computing* lançou o *Low-Power Image Recognition Challenge*, uma iniciativa que visa promover métodos de design de classificação de imagens com eficiência energética.

2.2 TINYML

O conceito de TinyML centra na integração de sistemas baseados em machine learning dentro de pequenos objetos inteligentes suportados por microcontroladores. Tal integração abre portas para o desenvolvimento de novos aplicativos e serviços que não precisam de suporte constante de processamento e de baixo custo energético, proporcionando segurança de dados (SANCHEZ-IBORRA; SKARMETA, 2020).

Microcontroladores de baixo custo, antes ignoradas para aplicações inteligentes, vêm sendo estudadas para constituírem as unidades centrais de pequenas aplicações inteligentes. Essas pequenas unidades, são focadas em realizar funções específicas para economia de recursos tanto como memória, processamento e energia (SANCHEZ-IBORRA; SKARMETA, 2020).

A comunidade científica que estuda TinyML está impulsionando a integração do ML dentro de objetos econômicos e inteligentes baseados em MCU. Previsões apontam que o mercado de computação de ponta global alcançará 1,12 trilhão de dólares em 2023 e algumas empresas de renome como a *Ericsson*, já estão oferecendo soluções *TinyML-as-aService* (SANCHEZ-IBORRA; SKARMETA, 2020). Ao prosseguir com as discussões sobre TinyML, serão apresentados as vantagens, desafios e exemplos reais do uso desta tecnologia nova e emergente.

2.2.1 Vantagens

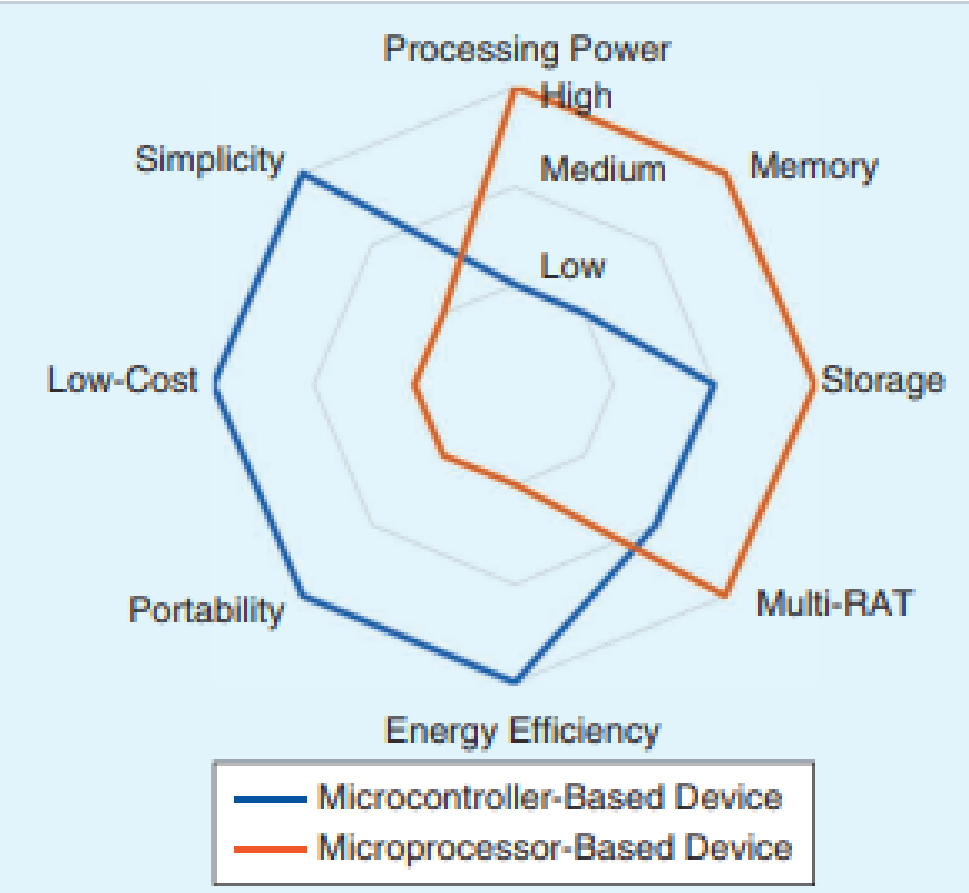
Para apresentação das vantagens do projeto, serão abordados os temas de eficiência energética, baixo custo e segurança.

2.2.1.1 Eficiência Energética

Este é o primeiro ponto, se não o mais importante que viabiliza a adoção do TinyML. Tendo em mente que, processadores poderosos e unidades de processamento gráfico exigem grande quantidade de energia, e sendo essas unidades, em outrora, as mais utilizadas em sistemas inteligentes, MCU apresenta consumo baixo, ainda que utilizando-se de de sua capacidade de processamento ao máximo.

Isso permite que aplicativos baseado em TinyML sejam colocados em quase todos os lugares, sem a necessidade de serem conectados à rede elétrica. A Figura 4 compara as principais características dos dispositivos alimentado por microcontrolador ou microprocessador (SANCHEZ-IBORRA; SKARMETA, 2020).

Figura 4 – Comparação de desempenho entre microcontroladores e microprocessadores



Fonte: Sanchez-Iborra e Skarmeta (2020, p.6).

2.2.1.2 Baixo Custo

A baixa complexidade e as definições otimizadas e simples de hardware para esses dispositivos, permitem que o custo de implementação por unidade produzida de um projeto envolvendo TinyML seja moderado, motivo que viabiliza e impulsiona a

rápida alavancada desse mercado. Segmentos que antes necessitavam de recursos caros ou até mesmo não possuíam esses recursos, atualmente podem usufruir de ferramentas de análise de dados e auxílio nas tomadas de decisão por um preço muito baixo (SANCHEZ-IBORRA; SKARMETA, 2020).

2.2.1.3 Segurança

Transmissões wireless exigem uma demanda de energia superior em comparação com processamento de dados. Com isso, limitar as atividades de comunicação se torna uma necessidade para tornar o sistema eficiente no quesito de energia. Desta forma, pode-se inferir que tal consequência seja um aspecto negativo, porém essa limitação permite aumentar a confiabilidade do sistema, bem como sua segurança e privacidade de dados (SANCHEZ-IBORRA; SKARMETA, 2020).

Aplicações de Big Data estão sujeitos a ataques por possuir um sistema com consumo de largura de banda que permite esses ataques, pois possuem um fluxo denso de dados passando por canais sem fio. Esses problemas são evitados executando o processamento de dados dentro do dispositivo, o que permite realizar menos transmissões com dados agregados e irrelevantes para um possível invasor (SANCHEZ-IBORRA; SKARMETA, 2020).

2.2.2 Desafios

Os desafios a serem tratados nesta seção se concentram na diversidade de dispositivos para a utilização de TinyML e o uso de microcontroladores para machine learning.

2.2.2.1 Diversidade de dispositivos

De antemão, a realidade de existir diversos dispositivos disponíveis para serem usados para desenvolvimento e confecção de aplicativos baseados em TinyML pode ser vista como um benefício. Mas essas unidades podem apresentar diferentes potências de consumo, capacidades de processamento, memória, capacidade de armazenamento, periféricos, comunicação com portas, protocolos, entre outras diversas configurações importantes para a elaboração de um projeto de sistemas embarcados.

Esse fato, dificulta o design de metodologias genéricas, tendo que ser adaptado o projeto para cada componente escolhido. Portanto, a implementação universal de frameworks de desenvolvimento frente a esta heterogeneidade não é uma tarefa fácil (SANCHEZ-IBORRA; SKARMETA, 2020).

2.2.2.2 MCU

Para que a integração de sistemas embarcados, pequenos e de baixo custo, com o aprendizado de máquinas seja efetiva, deve-se coordenar esforços para alcançar essa convergência.

A função dos fabricantes de MCU deve ser a de garantir que seus componentes sejam adequados para integração direta de ML, que permitirá um fácil desenvolvimento de aplicativos com o suporte de software no nível do dispositivo. A esse respeito, é importante que os futuros firmwares possam atingir altos níveis de confiabilidade e lidar com os principais aspectos, como segurança de dados no dispositivo e operações de transmissão otimizadas.

Um exemplo disso, é a utilização da ferramenta *Tensor Flow Lite* (FLOW, 2021), oriunda da ferramenta *Tensor Flow*, porém para sistemas com menores capacidades. *Tensor Flow Lite* requer um demanda computacional muito menor, pois é uma ferramenta que não precisa aprender e sim exercitar o que foi aprendido. Essas compactações são fundamentais para que o projeto em questão seja desenvolvido (SANCHEZ-IBORRA; SKARMETA, 2020).

Portanto, a adaptação das principais estruturas de ciência de dados aos requisitos dos MCUs é crucial para uma ampla expansão de TinyML. Não apenas bibliotecas de programação, mas também o ciclo de vida de ferramentas de gerenciamento são necessárias para simplificar o desenvolvimento de soluções, manuseio de dados.

2.2.3 Exemplos Reais

Ainda que esta área de desenvolvimento de inteligência artificial seja nova, muitos projetos já foram elaborados e testados mostrando o quão promissor é essa tecnologia. Além dos projetos citados na seção 1, pode-se encontrar na literatura outras aplicações que já fazem o uso dessa tecnologia e outras que podem se beneficiar da TinyML.

Por exemplo, a manutenção e o monitoramento de turbinas eólicas remotas podem ser bastante desafiadoras e demoradas. No entanto, se houvesse uma maneira de prever de forma proativa que a máquina teria problemas, pode-se fazer a manutenção preditiva antes de quaisquer falhas. Essa manutenção preditiva pode levar a economia de custo significativa devido a tempos de inatividade reduzido e melhor disponibilidade dos sistemas para maior confiabilidade no produto, o que leva a uma qualidade geral de serviço mais alta para usuários finais/clientes (REDDI, 2020).

Existem muitos aplicativos TinyML para manutenção preditiva. Por exemplo, uma startup australiana, *Ping Services*, introduziu um novo dispositivo IoT que inspeciona de forma contínua e autônoma uma turbina enquanto ela está funcionando.

Ao se fixar magneticamente à parte externa de qualquer turbina e analisar dados detalhados na borda e dados resumidos na nuvem, o dispositivo IoT pode alertar de forma eficiente e eficaz sobre quaisquer problemas potenciais antes que um problema surja dentro do turbina (LOMBARDO, 2020).

O TinyML também já está sendo usado para monitoramento ecológico e ambiental. Por exemplo, nos últimos 10 anos, a linha férrea *Siliguri-Jalapaiguri* na Índia teve mais de 200 colisões fatais com elefantes. Pesquisadores do Laboratório de Bioacústica Aplicada da Universidade Politécnica da Catalunha projetaram um sistema de sensor acústico e térmico inteligente usando modelos de aprendizado de máquina personalizados que funcionam com energia solar como um sistema de alerta precoce (SOLANA, 2020).

2.3 TESTE DE ESTANQUEIDADE

Ensaios Não Destrutivos (END) são definidos como métodos usados para testar uma peça, material ou sistema sem prejudicar sua futura usabilidade (PIAZZETTA, 2017). Intuitivamente, ensaios destrutivos provocam a perda da usabilidade do produto, como ensaios de tração para avaliação de propriedades mecânicas de uma peça. Para que a qualidade do um produto se mantenha íntegra, os END se utilizam de fenômenos físicos e a interação da peça com algum ambiente externo para assim obter parâmetros que possam inferir sobre a qualidade do produto (PIAZZETTA, 2017).

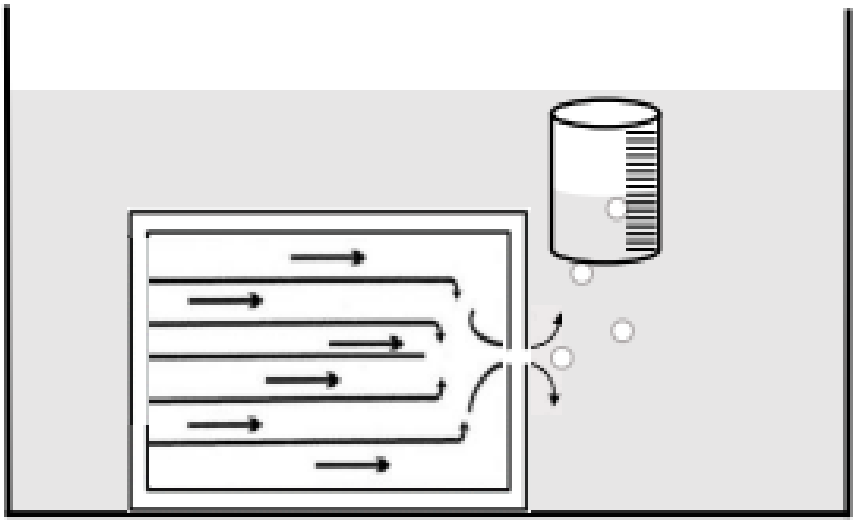
Um dos END mais recorrentes nas industrias é o teste de vazamento, o qual verifica a existência de perda de fluído por descontinuidades no material. O que se busca medir, controlar ou evitar, é a taxa de vazamento, definida como a quantidade de gás que atravessa uma seção em um determinado intervalo de tempo.

Sistemas isolados que contêm gases indicam a existência de vazamentos ao apresentar uma variação em sua pressão interna. Essa variação pode ser usada como determinante para se analisar a taxa de vazamento. Outros métodos têm sido usados para quantificar ou apenas sinalizar esses vazamentos. Um desses métodos, é a formação de bolhas quando objetos com taxas de vazamento superior a $10^{-5}\ Pa * m^3/s$ são submersos em água (PIAZZETTA, 2017).

O método da bolha tem como objetivo exibir visualmente o desprendimento de bolhas de uma sistema fechado em um tanque de ensaio, possibilitando a identificação do local do vazamento. Como o sistema de reconhecimento de bolhas, normalmente é realizado visualmente por humanos, esse método se encaixa em um método de análise qualitativa, ou seja, definir se a peça possui ou não vazamento. Existem porém, maneiras de determinar a taxa de vazamento afim de tornar o método quantitativo. A Figura 5 exibe a utilização de uma proveta graduada que auxilia na quantificação da taxa de vazamento. O recipiente recebe a quantidade de gás expelido pelo material

imerso e armazena o fluído gasoso para medição de sua quantidade.

Figura 5 – Utilização de proveta em testes de estanqueidade



Fonte: Piazzetta (2017, p.49).

Alguns fatores influenciam na formação de bolhas, auxiliando na identificação de tais. As propriedades físicas do líquido e do gás devem ser escolhidas para melhor adaptação do método. Ao se substituir a água por um líquido de tensão superficial mais baixa, com a mesma pressão aplicada, é possível a formação de uma bolha em uma descontinuidade de diâmetro três vezes menor do que com água. Somado a isso, ao utilizar um líquido com tensão superficial ainda mais baixa, pode-se detectar taxas de vazamentos que chegam até cem vezes menores do que o limite de detecção quando as bolhas são geradas pela água (PIAZZETTA, 2017).

3 METODOLOGIA

Esta seção aborda e detalha todos os procedimentos para a realização do trabalho proposto. Primeiramente será exposto a modelagem física do projeto: os hardwares utilizados, materiais escolhidos para confecção do modelo de teste e a montagem de todos os componentes que compõem o modelo físico por completo. Somado a isso, as técnicas de engenharia de software necessárias para a realização do trabalho, também serão abordadas e discutidas, sendo tais técnicas o viabilizador mais importante no que se refere a TinyML.

3.1 MODELO DE TESTE

Para obter-se um objeto de estudo da aplicação do TinyML em testes de estanqueidade, buscou-se realizar um modelo simples porém suficiente para aplicação dos conceitos de inteligência artificial para detecção de peças não conformes no que se refere a vazamento de fluídos. O tanque a ser preenchido com água, foi confeccionado com vidro, assemelhando-se a um aquário. As dimensões do tanque a ser utilizado são 20cm x 9cm x 12,5cm, como mostrado na Figura 6.

Figura 6 – Tanque utilizado para modelo físico (foto do autor).

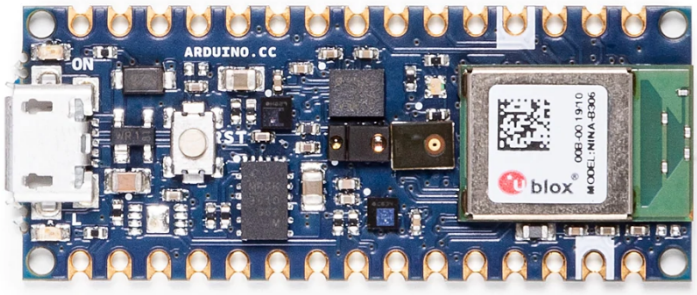
O corpo de teste escolhido para ser submetido ao estanque foi uma bola de tênis de mesa, que possuem um formato de fácil manuseio com diâmetro de 4cm.

3.1.1 Hardware

O hardware utilizado para inferência do teste de estanqueidade, ao apontar quais peças estão conformes no que se refere ao isolamento/vácuo, foi um arduino Nano 33 BLE Sense *with headers* (ARDUINO, 2022). A placa vem com uma série de sensores embutidos:

- Sensor inercial de 9 eixos: o que torna esta placa ideal para dispositivos vestíveis;
- sensor de umidade e temperatura: para obter medições precisas das condições ambientais;
- sensor barométrico: para fazer uma estação meteorológica simples;
- microfone: para capturar e analisar o som em tempo real;
- sensor de gesto, proximidade, cor da luz e intensidade da luz: para estimar a luminosidade da sala ou para detectar se alguém está se aproximando do quadro.

Figura 7 – Arduino Nano 33 BLE Sense.



Fonte: Arduino (2022).

O Arduino Nano 33 BLE Sense é uma evolução do Arduino Nano tradicional, mas com um processador muito mais poderoso, o nRF52840 da Nordic Semiconductors, uma CPU ARM® Cortex™-M4 de 32 bits rodando a 64 MHz. Esses adicionais permitem programas maiores do que com o Arduino Uno (ele tem 1 MB de memória de programa, 32 vezes maior), e com muito mais variáveis (a memória RAM é 128 vezes maior). O processador principal inclui outros recursos tais como emparelhamento Bluetooth® via NFC e modos de consumo de energia ultrabaixo (ARDUINO, 2022).

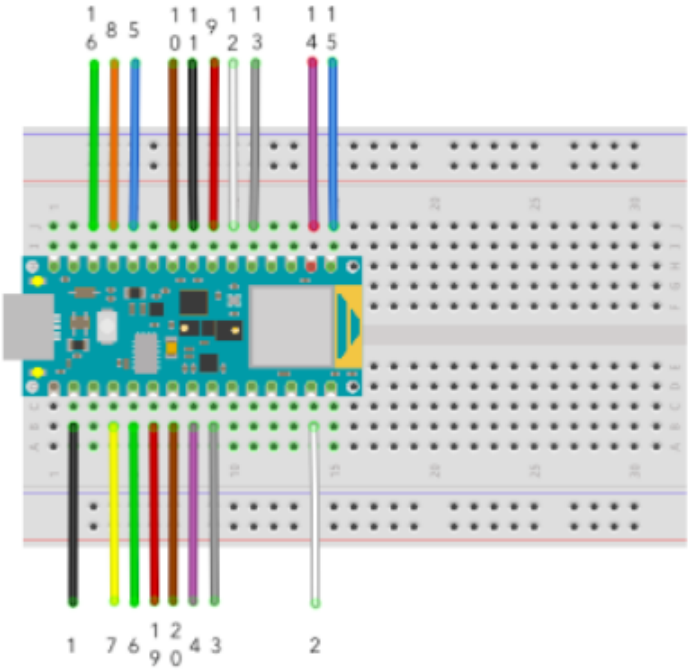
A principal característica desta placa, além da seleção de sensores, é a possibilidade de rodar aplicações de inteligência artificial utilizando TinyML. Esse kit possibilita a criação de modelos de aprendizado de máquina usando o TensorFlow™ Lite e carregá-los em sua memória usando o IDE do Arduino (ARDUINO, 2022).

Para obtenção das imagens capturadas foi utilizado o módulo OV7670, um módulo que permite a captura e armazenamento de imagens coloridas pelo Arduino, com uma taxa de atualização de até 30 frames por segundo, com resolução máxima de 640 x 480 Pixels (THOMSEN, 2013).

Para facilitar o acesso dos periféricos do kit, foi utilizado uma uma matriz de

contato, mais comumente chamada de protoboard, seguindo o esquema que pode ser visto na Figura 8.

Figura 8 – Montagem do hardware para a utilização do Arduino e o OV7670



Fonte: Reddi (2020).

A maioria dos exemplos disponíveis na internet, de TinyML utilizando Arduino em aplicações de detecção de imagem, usam como ferramenta de captura das imagens em tempo real o módulo OV7675, porém não foi possível sua aquisição devido à falta do módulo no mercado.

Tanto é que ministrantes de um curso sobre TinyML na plataforma EDX, e suportada pela Harvard, expõem a situação da falta do componente e fornecem a solução e as modificações necessárias para a utilização do módulo substituto OV7670 para aplicações de TinyML no Arduino Nano 33 BLE Sense. As modificações no hardware seguem as instruções conforme Figura 9, no qual apenas excluímos a utilização dos pinos 17/18.

Figura 9 – Tabela de pinos para ligar o OV7670.

Description	Camera Module Pin	Microcontroller Board Pin
VCC / 3.3V	1	3.3V
GND	2	GND
SIOC / SCL	3	SCL / A5
SIOD / SDA	4	SDA / A4
VSYNC / VS	5	D8
HREF / HS	6	A1
PCLK	7	A0
XCLK	8	D9
D7	9	D4
D6	10	D6
D5	11	D5
D4	12	D3
D3	13	D2
D2	14	D0/RX
D1 (may be labeled D0)	15	D1/TX
D0 (may be labeled D1)*	16	D10
NC	17	–
NC	18	–
PEN / RST	19	A2
PWDN / PDN	20	A3

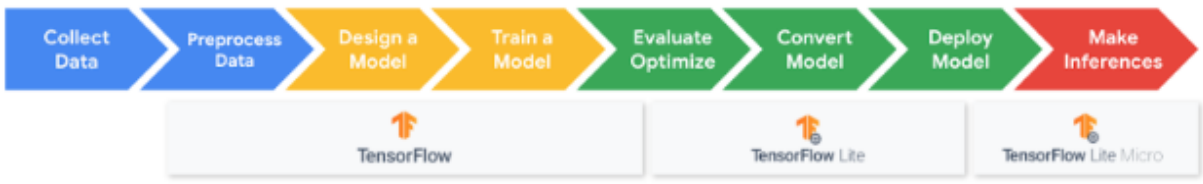
Fonte: Reddi (2020).

As modificações no software serão relatadas mais adiante na Seção 4.

3.1.2 Software

Na seção 3.2 será detalhado cada ciclo da construção de uma aplicação de TinyML. Porém o modo como cada passo é desenvolvido depende muito do ambiente a ser utilizado. A ferramenta TensorFlow é uma das principais bibliotecas de código aberto para desenvolver e criar modelos de ML (FLOW, 2021).O TensorFlow sempre possibilitou um caminho direto para a produção. Tanto em servidores quanto em dispositivos finais ou Web, o TensorFlow permite treinar e implantar modelos de maneira simplificada, independentemente da linguagem ou da plataforma utilizada (FLOW, 2021).

Figura 10 – Ciclo do TinyML e suas plataformas utilizadas.



Fonte: Reddi (2020).

Como mostrado na Figura 10, utilizando o TensorFLow o usuário consegue desenhar um modelo apropriado, treinar e avaliar o modelo no que se refere aos dados coletados, modelo proposto, entre outros. Para este projeto foi utilizado o Google Colab (COLAB, 2022), plataforma de desenvolvimento que consegue utilizar as funções do TensorFlow.

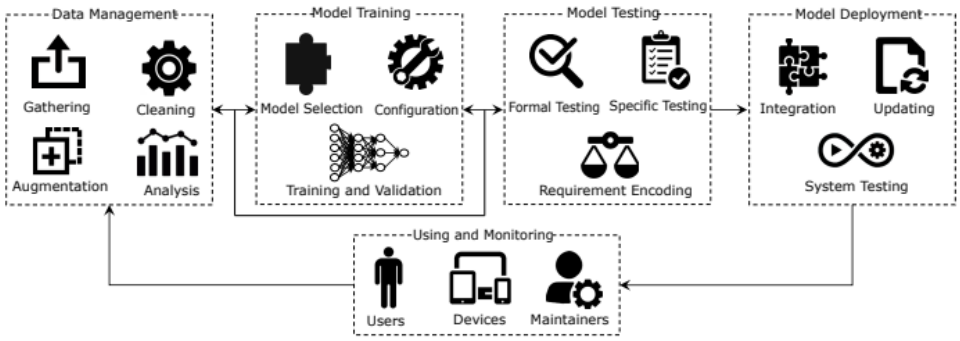
Para a parte de conversão do modelo para aplicações em smartphones e dispositivos similares, como mostrado na Figura 10,é representado pela fase em verde e utiliza-se a uma instância do Tensor Flow, o TensorFlow Lite.

E por último, para uso em microcontroladores, representado pela fase em vermelho da Figura 10, utiliza-se o TensorFlow Lite Micro, uma biblioteca para desenvolvimento em aplicações que necessitam compactação de memória, economia de energia, e tudo o que envolve o TinyML. O TensorFlow Lite para microcontroladores é escrito em C++ e requer uma plataforma de 32 bits (FLOW, 2021). Como a placa escolhida foi o Arduino Nano 33 BLE Sense, o ambiente de desenvolvimento utilizado será o Arduino IDE (ARDUINO, 2022), um ambiente com recursos que permitem que se manipule rapidamente placas de desenvolvimento de microcontroladores.

3.2 FLUXO DE DESENVOLVIMENTO DE UM UMA APLICAÇÃO DE TINYML

O desenvolvimento de um modelo de machine learning pode seguir uma metodologia constituída por etapas padronizadas que alicerçam a elaboração de qualquer aplicação que se utiliza dessa tecnologia. Este padrão de etapas é comumente chamado de ciclo de vida do machine learning.

Figura 11 – Ciclo de desenvolvimento para aplicações de ML.



Fonte: See (2021).

See (2021) se baseia da Figura 11 para explicar cada passo que constitui a obra completa da confecção de um modelo de machine learning, que pode ser usado para aplicações de TinyML porém com alguns processos intermediários que viabilizam sua utilização em equipamentos que consomem menos memória, energia, e com uma resposta rápida para o usuário.

Vale ressaltar que este ciclo foi utilizado para o desenvolvimento do modelo de validação de peças ao serem submetidas a testes de estanqueidade.

O primeiro estágio parte da definição do hardware a ser utilizado, pois no caso do TinyML, baixo espaço em memória, baixo consumo de energia, e uma boa resposta em tempos real (latência) são pré requisitos estabelecidos para todas as aplicações de inteligência artificial em microcontrolador. E com essas definições pode-se escolher o hardware a ser utilizado.

A plataforma para aplicações TinyML Edge Impulse disponibiliza uma lista de hardware suportados em seu ambiente de desenvolvimento, conforme Tabela 1.

Tabela 1 – Hardwares apropriados para TinyML

Hardwares
ST B-L475E-IOT01A (IoT Discovery Kit)
Arduino Nano 33 BLE Sense
Arduino Portenta H7 + Vision shield
Eta Compute ECM3532 AI Sensor
Eta Compute ECM3532 AI Vision
OpenMV Cam H7 Plus
Himax WE-I Plus
Nordic Semiconductor nRF52840 DK
Nordic Semiconductor nRF5340 DK
Nordic Semiconductor nRF9160 DK
Nordic Semiconductor Thingy:91
Silicon Labs Thunderboard Sense 2
Sony's Spresense
Syntiant TinyML Board
TI CC1352P LaunchPad
Raspberry Pi 4
NVIDIA Jetson Nano

Fonte: Impulse (2022).

Após a definição do hardware e sensores que serão compatíveis com o modelo a ser criado, inicia-se efetivamente no ciclo de vida da machine learning aplicada a modelos reduzidos.

3.2.1 Aquisição de dados

Com os requisitos de projeto já definidos, inicia-se a coleta de dados que auxilia na previsão e categorização das variáveis de interesse.

Este processo de coleta de dados, pode incluir tanto a ação ativa e autoral de adquirir dados, como a utilização de conjunto de informações já existentes e direcionadas para certas aplicações, como por exemplo a CIFAR-10, um banco de dados que consiste em 60.000 imagens coloridas 32x32 em 10 classes (carro,avião, gato), com 6.000 imagens por classe.

Existem 50.000 imagens de treinamento e 10.000 imagens de teste. Com este pacote de dados, é possível treinar um modelo a reconhecer as classes abrangidas neste conjunto (KRIZHEVSKY, 2009).

Em aplicações reais, como citado na seção 2.2.3, normalmente esses conjuntos de dados são criados pelo próprio desenvolvedor do modelo. A aplicação do teste de estanqueidade não foi diferente.

Geralmente, esse é um dos estágios mais demorados, pois pode ser difícil e demorado selecionar um conjunto de dados suficientemente grande para garantir a precisão do modelo, levando em consideração todas as variáveis no que se refere a classificação e reconhecimento inteligente.

Por exemplo, modelos que utilizam imagens para detectar ou classificar objetos precisam levar em consideração a luminosidade em suas inferências, ou seja, se o modelo for aplicado em um local aberto, imagens no período da noite devem ser parte da base de dados utilizadas para o treinamento do modelo de inteligência artificial (REDDI, 2020).

Após a coleta de dados, deve-se fazer a classificação destes dados para que a aplicação saiba como definir as amostras que constantemente serão inseridas em seu sistema. Em certas ocasiões essas classificações podem requirir certo domínio técnico, por exemplo, em um modelo de análise de resultados médicos.

Por último, sempre deve-se manter o refinamento da base de dados, pois a probabilidade de inserir amostras repetidas ou inúteis é grande e a possibilidade de faltar amostras que cubram todas as variáveis relacionadas àquela aplicação, também é grande.

Para o desenvolvimento do projeto em questão, foram utilizados 6 bolas de tênis de mesa com diferentes tipos de furos, para que ao serem mergulhadas em água, exibissem diferentes padrões de bolhas.

No momento da imersão destes diferentes corpos de prova, foram tomadas diversas fotos que constituíram a base de dados e a classificação de peças com deformidades. A base de dados relacionadas com peças não conformes foi classificada como NOK (Não OK) e possuía 94 arquivos.

Figura 12 – Peças com deformidades (foto do autor).

O mesmo foi feito para as peças conformes. As fotos foram tiradas quando imersas no tanque com água e constitui-se uma base de dados denominada OK com 89 arquivos.

Figura 13 – Peças em conformidade (foto do autor).

3.2.2 Preparação dos dados

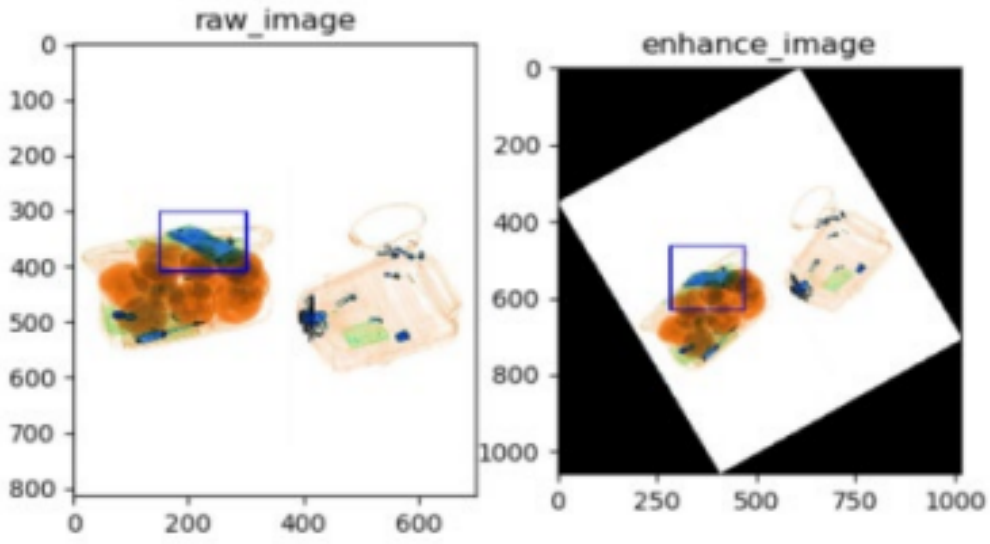
Com a base de dados pronta para alimentar a rede neural, deve-se preparar e extrair as informações que serão utilizadas para ensinar a máquina suas funções. Antes de fazer a extração destes dados um processo importante para tornar a base de dados eficiente é a *data augmentation*.

Data augmentation é uma estratégia de aumentar a base de dados buscando ampliar as possibilidades de como um dado pode ser apresentado (ZHU, 2020).

Por exemplo, se em uma aplicação que diferencie cachorros e gatos, caso a base de dados possua apenas imagens dos animais na vertical, a possibilidade da inferência falhar caso um cachorro esteja na horizontal é relevante.

Existem vários métodos de data augmentation: transformação geométrica, transformação de cor, transformação de reflexão rotativa, injeção de ruído, entre outros (ZHU, 2020). A Figura 14 exibe o resultado de um tipo de transformação e complemento de dados.

Figura 14 – Fotos de Raio-x após data augmentation.



Fonte: Zhu (2020).

Com a base de dados pronta para alimentar a rede, deve-se ter em mente que o mesmo dado que compõe o dataset, não será o mesmo que será colocado no modelo de aprendizado de máquina para ser inferido. Deve-se extrair algumas configurações que minimizem a quantidade de informações e facilite o reconhecimento da rede neural em detectar as informações propostas previamente.

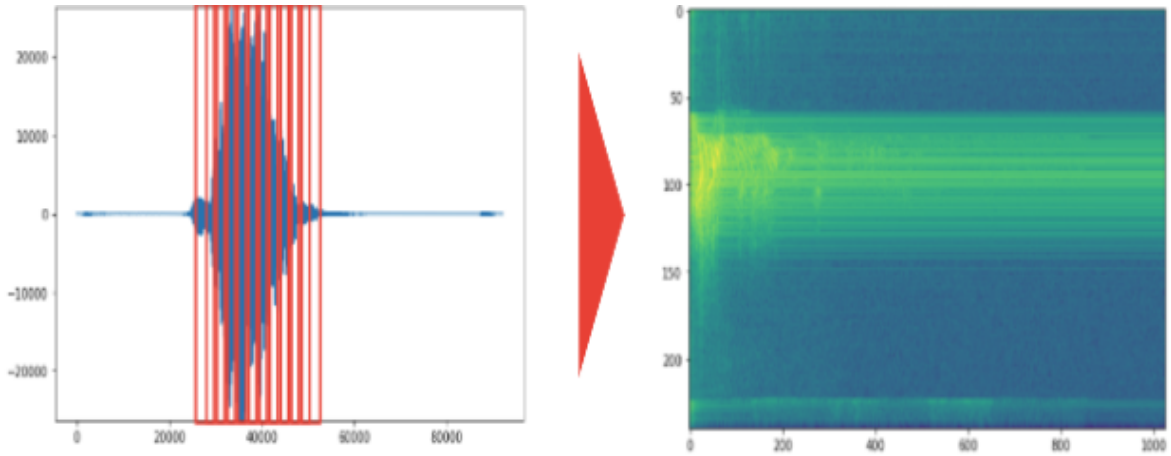
Por exemplo, um detector de imagem captura um sinal contínuo no tempo e o transforma para um dado de domínio da frequência. Essa transformação é feita usando-se a Transformada de Fourier, que possibilita uma análise discreta das informações

que chegam pelos sensores (REDDI, 2020).

Tendo em vista que o objetivo do TinyML é de efetuar as operações que possibilitam a inteligência artificial, porém com um gasto computacional baixo no que se refere à memória utilizada e ainda uma baixa latência, a solução é a utilização da transformada rápida de Fourier (FFT) localizada em uma biblioteca chamada KISS FFT que faz parte da interface chamada de TensorflowLite Micro, que seria a compactação de todos os métodos do machine learning para rodar em microcontroladores.

Aplicando-se a FFT, podemos analisar a intensidade (em cor) de cada frequência. Essa análise em cor chama-se Espectrograma.

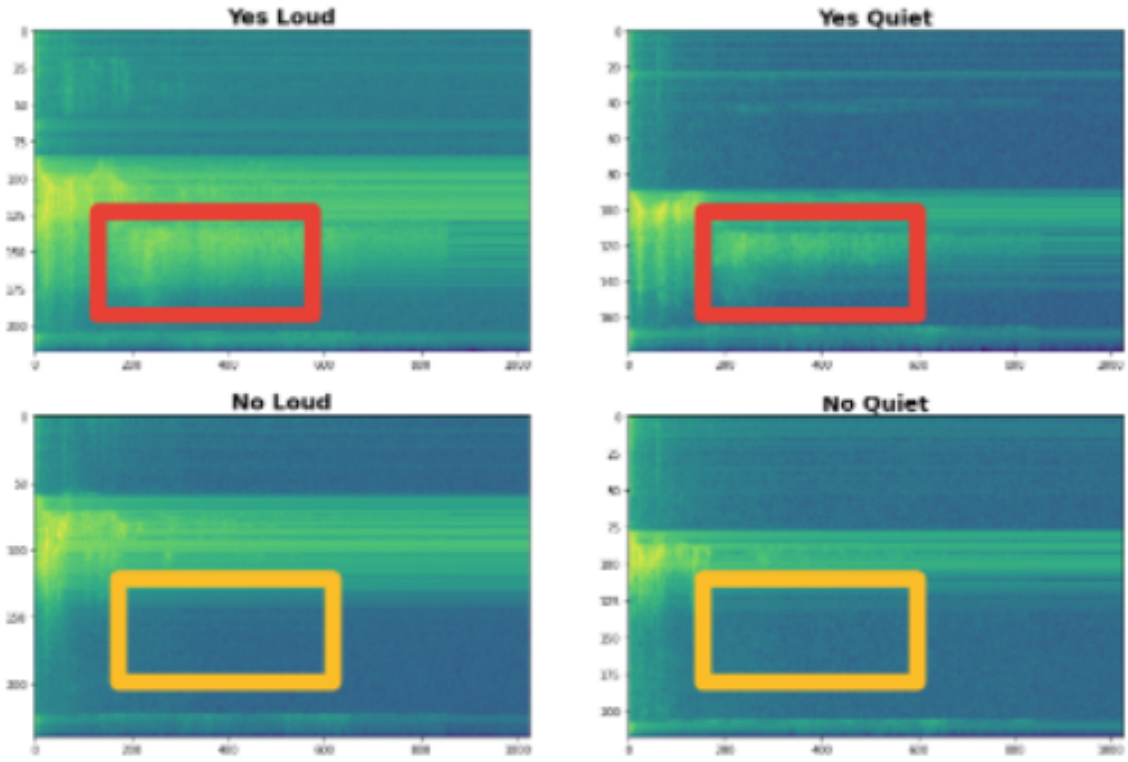
Figura 15 – Conversão de de um sinal de áudio para o espectograma.



Fonte: Reddi (2020).

Um espectrograma é um dado melhor de entrada para inferências em algumas aplicações de inteligências artificiais, pois visualmente a rede neural pode diferenciar, por exemplo, as frequências geradas ao se dizer "yes"e "no", como mostrado na Figura 16.

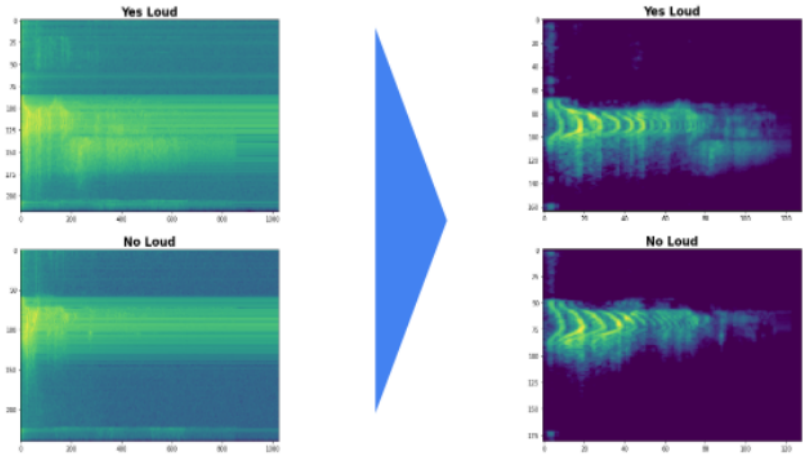
Figura 16 – Exibição de duas palavras distintas em um espectograma.



Fonte: Reddi (2020).

Ainda que, pela Figura 16, a inferência da rede neural para diferenciar sinais capturados por sensores seja facilitado pela utilização da FFT, ainda pode-se perceber que as imagens geradas ainda não são bem claras. De igual maneira que o ouvido humano por exemplo, ignora baixas frequências, pode-se aplicar o mesmo raciocínio para os espectrogramas. Ao aplicar um filtro chamado de Mel Filter Bank (REDDI, 2020), tem-se como saída Mel Frequency Cepstral Coefficient (MFCC), mostrado na Figura 17 .

Figura 17 – Exibição de duas palavras distintas após passagem por um filtro.

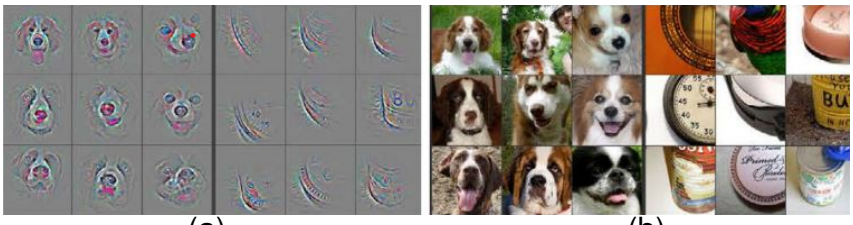


Fonte: Reddi (2020).

Com isso, a rede neural possui dados mais simples para fazer as inferências de uma maneira que consuma baixa memória, com uma resposta rápida e menor consumo de energia. Com a base de dados pronta para alimentar a rede, deve-se analisar qual modelo de inteligência de máquina será a melhor para o tipo de aplicação vigente.

No caso da classificação de imagens é utilizado redes convolucionais para a extração de informações relevantes para a classificação. A convolução é a aplicação de filtros que determinam que tipos de informações serão retidas e eliminadas.

Figura 18 – (a) Imagens convolucionadas, (b) imagens.



(a) (b)

Fonte: Reddi (2020).

A Figura 18 exibe imagens e seus respectivos resultados após a convolução. Com isso a rede neural trabalha com menos informações e mais eficaz para assim realizar a classificação. Existem diversas técnicas de aprendizado de máquina. Cabe então ao engenheiro/desenvolvedor TinyML avaliar os pós e contra de cada modelo e definir a técnica que melhor se aplica para o uso proposto (REDDI, 2020).

3.2.3 Desenvolvimento do Modelo

As etapas anteriores se concentram na preparação, extração e refinamento dos dados para serem aplicados em um modelo de aprendizagem de máquina. Ou seja, assim que um conjunto de dados viável estiver disponível, a etapa seguinte

é a adequação destes dados e a aplicação de um modelo que produza o melhor desempenho (REDDI, 2020).

A habilidade do desenvolvedor de aplicações envolvendo TinyML se concentra em adequar aplicações reais com as técnicas mais apropriadas para cada caso, selecionando o melhor modelo de treinamento para o conjunto de dados disponível para a situação real em que o sistema irá atuar.

Por exemplo, a regressão linear pode ser eficiente em situações onde a interpretabilidade é fundamental, porém está restrita a um conjunto de funções lineares. Por outro lado, as RNAs podem oferecer alto desempenho, pois são capazes de modelar a distribuição de dados não lineares de forma mais eficaz, mas são menos interpretáveis para o usuário (REDDI, 2020).

Vale ressaltar que após a definição do modelo, é preciso ajustar as configurações para um melhor desempenho. Como mostrado na Figura 11, esse ajuste é representado pelo ícone *Configuration*. Por exemplo, em uma regressão linear, podemos ajustar e definir um melhor modelo ao combinar da melhor forma possível o gradiente com sua taxa de aprendizado.

3.2.4 Validação do modelo

Para que o modelo desenvolvido atue dentro do esperado, deve-se treinar este modelo, ou seja, analisar como a rede neural classifica os dados de entrada que foram selecionados anteriormente na Seção 3.2.1.

Neste trabalho, uma rede neural processa todas as fotos obtidas no teste de estanqueidade como entradas e classifica as peças como OK ou não OK, assim a rede vai atribuindo pesos para os nós podendo assim classificar outras fotos além das fotos que fazem parte da base de dados.

Após o primeiro treinamento, para que a rede não tenha apenas um único contato com uma única base de dados, passamos para a parte de teste e validação para prevenir que o modelo não tenha apenas um bom desempenho restrito à uma base de dados.

Esse é o motivo pelo qual separamos o dados em dados de treinamento, teste e validação, e assim dar ao sistema uma maior gama de situações para melhor desempenho em seu papel (REDDI, 2020).

3.2.5 Deploy do modelo

O último estágio do ciclo envolve o *deploy*, ou seja, o *upload* do que foi desenvolvido com ajustes que o adaptem ao microcontrolador, para aplicações TinyML. Segundo Reddi (2020), essa é uma das fases mais importantes.

O tamanho do modelo precisa ser reduzido suficientemente para se encaixar

perfeitamente nos dispositivos de sistemas embarcados. Para isso, métodos como o *Pruning* e a quantização são aplicados.

Pruning, do inglês podar, tratando-se de inteligência artificial, é um método de reduzir excessos e o tamanho de modelos computacionais, tanto como RNA de treinamento, árvore de decisões, entre outros (RANJAN, 2021). Por exemplo, no caso de RNA, pode-se reduzir o número de camadas e/ou neurônios.

Existem algumas razões para a aplicação do Pruning, sendo elas: evitar o overfitting (quando um modelo se ajusta perfeitamente a um conjunto de dados mas se mostra ineficaz para prever novos resultados), criar versões mais compactas dos modelos computacionais com uma perda insignificante em sua precisão e reduzir a complexidade dos modelos e seu tempo de inferência (RANJAN, 2021).

Para uma melhor compactação do modelo e ajudar na redução computacional, no gasto de energia e na velocidade de inferência após o Pruning, pode-ser mesclar o método de quantização, o qual se baseia na redução de bits dos pesos da rede neural sem mudar a estrutura da rede, diferentemente do Pruning (KIM, 2021).

A quantização é um método que visa minimizar o consumo de energia e o tempo de operação da aplicação ao realizar operações matemáticas e de acumulação de dados (gerenciamento de memória). Este objetivo é alcançado reduzindo os parâmetros geralmente expressados por dados de 32-bits do tipo real, para um tipo significantemente menor (KIM, 2021). Essa conversão reducional pode ser feita para diversos tipos, entre eles: inteiro, ponto-fixo, escala logarítmica (KIM, 2021).

Existem duas formas de se aplicar a quantização: Quantização pós-treinamento (PTQ), e a quantização durante o treinamento (QAT). Embora o PTQ seja geralmente menos preciso do que QAT, é mais adequado para a fase de inferência porque pode ser realizado imediatamente com apenas parâmetros pré-treinados sem ajustes adicionais (KIM, 2021).

Depois de todos esses estágios, o monitoramento contínuo do modelo no ambiente de produção geralmente é necessário para garantir que ele esteja funcionando de forma eficaz. Se, após todas essas etapas, o desempenho do modelo for insatisfatório no ambiente de produção, o modelo deverá ser atualizado. Isso pode ser feito usando novos requisitos de projeto, dados ou técnicas de modelagem. Essa é a essência do ciclo de vida do aprendizado de máquina.

4 DESENVOLVIMENTO

Na presente seção, serão discutidas e apresentadas todas as operações realizadas para o desenvolvimento do trabalho, como também toda a parte prática vista na Seção 3.2. Ou seja, toda a parte teórica do ciclo de confecção de um sistema TinyML será detalhado na prática.

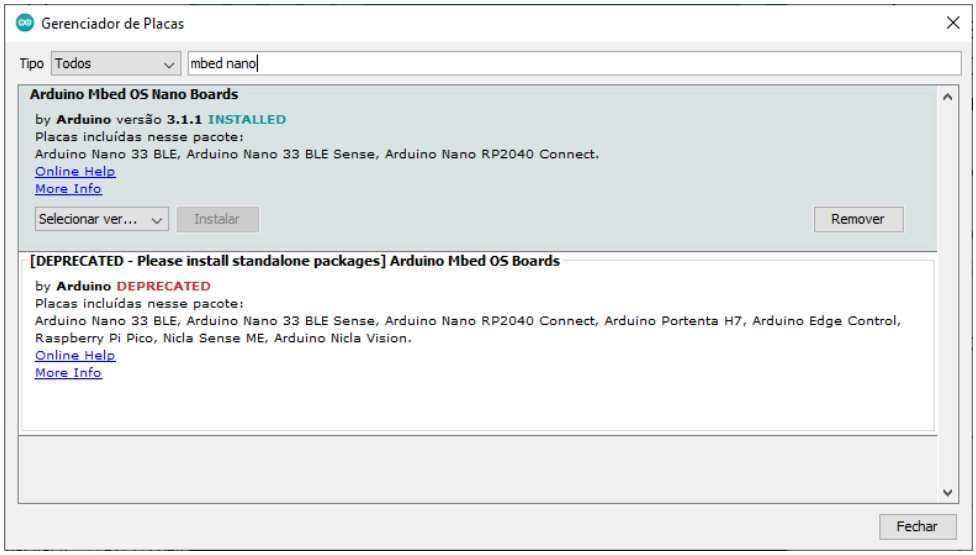
4.1 PREPARAÇÃO DO SOFTWARE PARA DESENVOLVIMENTO DO TINYML

O download do Arduino IDE por si só já traz diversas funcionalidades básicas de implementação em dispositivos/microcontroladores e placas de desenvolvimento convencionais. Para projetos que envolvem machine learning algumas ações adicionais devem ser realizadas para que a execução destes projetos seja bem sucedida.

Uma das principais vantagens que o Arduino IDE oferece é a portabilidade do código que você escreve para uma ou outra placa do mesmo padrão ou até mesmo na portabilidade de código para placas afiliadas.

O primeiro procedimento executado foi o download da placa Arduino Nano 33 BLE Sense. Na opção gerenciador de placas, busca-se a placa denominada *Arduino Mbed OS NANO Boards* e instala-se a versão 2.3.1 ou a mais atual, como mostrado na Figura 19.

Figura 19 – Placa Arduino Mbed OS NANO (fonte do autor).

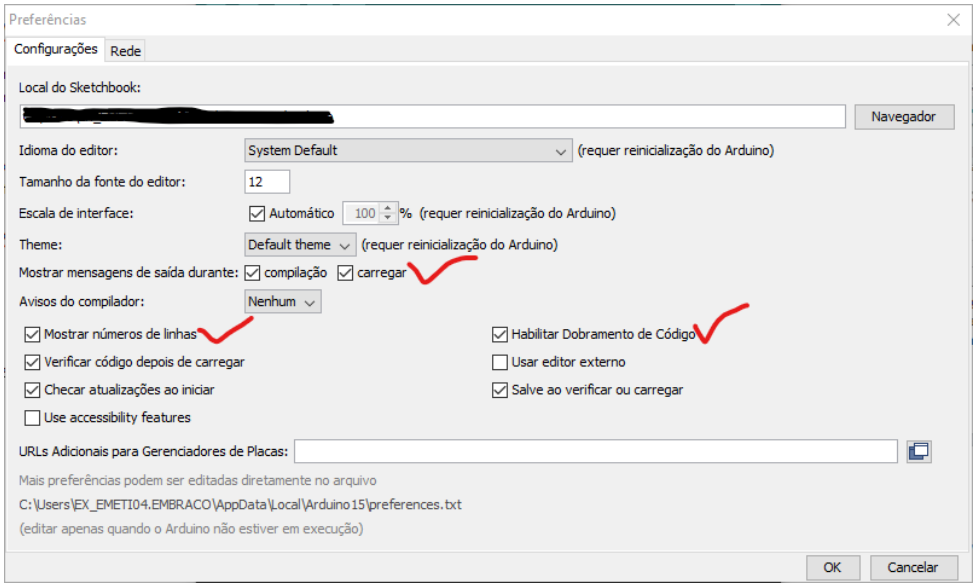


Outro recurso de grande valia disponível na plataforma IDE é a vasta variedade de bibliotecas que realizam diversas funções referentes à diversos hardwares, como interface com sensores, manipulação de determinados tipos de dados, etc. As bibliotecas utilizadas no projeto são:

- Arduino TensorFlowLite
- Harvard TinyMLx
- Arduino LSM9DS1
- ArduinoBLE

As bibliotecas acima permitem a utilização das funções do TensorFlow, auxiliam na comunicação da placa com o sensor LSM9DS1 ou I2C que realiza as leituras do acelerômetro, giroscópio e magnetômetro, entre outras funções importantes para o funcionamento do sistema TinyML. Por último, foi configurado alguns parâmetros que auxiliam na apuração de erros e visualização dos resultados. Ao entrar na janela preferências, deve-se deixar como marcado as seguintes opções conforme mostrado na Figura 20.

Figura 20 – Opções de visualização para depuração dos códigos (fonte do autor).

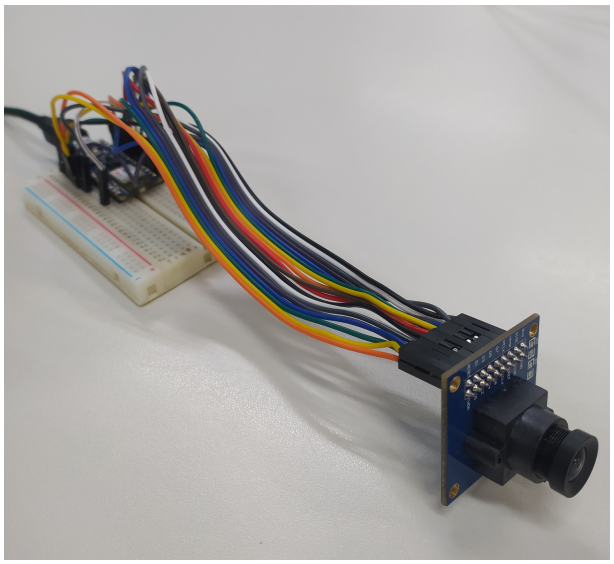


Isso permitirá uma maior facilidade ao encontrar erros de compilação, incompatibilidade, sintaxe e entre outros.

4.2 TESTES

Com o intuito de testar o hardware e sua funcionalidade, rodou-se alguns programas testes retirados do curso oferecido pela Harvard, *Deploynig TinyML* (REDDI, 2020). O hardware montado pode ser visto na Figura 21.

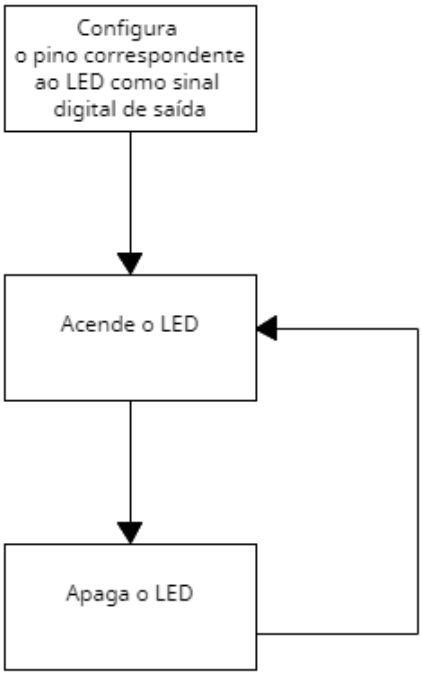
Figura 21 – Hardware preparado para utilização (fonte do autor).



O primeiro programa teste chamado de BLink, é retirado da própria plataforma Arduino IDE, que fornece alguns códigos de exemplos, tanto para a familiarização do usuário como também para teste de hardware.

O Blink, por si só, tem a funcionalidade de testar a comunicação entre o hardware e o ambiente de desenvolvimento, pois o código executa no Arduino a ação de acender e apagar um LED. O fluxograma exposto na Figura 22 exibe a estrutura do Blink. Após o upload do código no hardware, o sistema se comportou como esperado, o que comprova que a comunicação entre as portas do computador e a placa do Arduino estão funcionando da maneira correta.

Figura 22 – Fluxograma do código Blink (fonte do autor).



O seguinte teste a ser realizado, foi o de verificação da biblioteca TensorFlow Lite Micro, exemplo também tirado da plataforma Arduino IDE. O nome do arquivo executado no hardware é "Hello World".

O conjunto de códigos Hello World roda um modelo de rede neural que capta um valor qualquer para X, e faz uma previsão para seu valor dentro do ciclo geométrico, representado pelo seu seno, que resultaria um valor Y. Visualmente, o sinal do seno é uma curva suave periódica variando de -1 a 1. Este padrão se encaixa perfeitamente para controle de piscar de luzes.

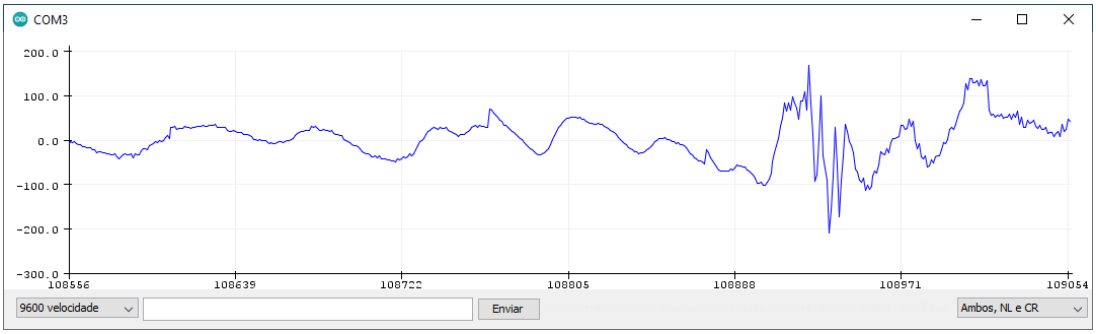
Embora o resultado visualmente não seja muito diferente do que visto no exemplo do Blink, pois a LED acenderá e apagará, é na estrutura do programa que comandará as ações do LED que reside a diferença.

Como visto, as RNAs aprendem padrões antes exposto à elas na etapa de treinamento, podendo assim fazer inferências de dados de entradas futuras. Apesar de ser uma operação básica, ela é válida para testar a gestão dos modelos de redes, as inferências, e as demais ações que compõe um código de redes neurais artificiais utilizando-se da biblioteca TensorFlow Lite Micro. Após o carregamento do código para o hardware, o sistema se comportou como esperado.

A próxima rodada de testes realizados foram para certificar a funcionalidade dos sensores existentes no Arduino Nano 33 Ble. Ainda que o único sensor utilizado foi o visual, ou seja, o módulo OV7670, é importante o teste dos outros sensores, para certificar que a placa em si e todos os seus componentes funcionam de maneira integral.

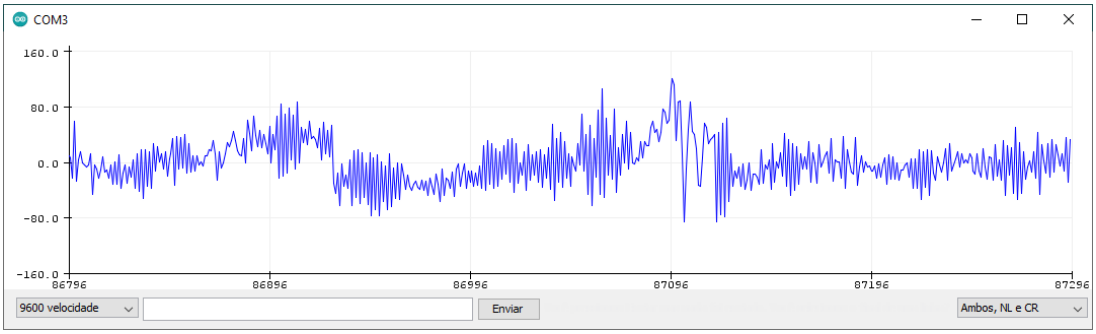
O primeiro sensor testado foi o microfone. O script rodado também é retirado de uma base de exemplos, porém dessa vez, é uma base advinda da biblioteca Harvard TinyMLx. O nome do arquivo rodado é encontrado por test microphone, e sua função principal é captar o som após enviar a palavra "click". No monitor serial e no plotter serial podemos ver a representação em forma de onda do que foi capturado pelo microfone do Arduino. A Figura 23, representa o que foi capturado pelo microfone, enquanto esse está em um escritório de TI (Tecnologia da Informação) de uma empresa de compressores em Joinville.

Figura 23 – Captura de um ruído ambiente pelo microfone (fonte do autor).



Já a Figura 24, representa a forma de onda de uma pessoa falando com o microfone perto de sua boca.

Figura 24 – Captura da fala de uma pessoa pelo microfone (fonte do autor).



Com os resultados obtidos acima, pode-se verificar que o sensor de microfone, corresponde satisfatóriamente. Também, fica certificado que o monitor e o plotter serial respondem bem às sáidas que o Arduino envia.

O próximo sensor testado foi a câmera, componente essencial do projeto, pois é pela análise das imagens coletadas através do OV7670 que será feita a inferência se a peça está boa ou não. Como no exemplo do microfone, o código é retirado do acervo de exemplos da biblioteca Harvard TinyML. O código permite a opção de captura que gera uma sequência de dígitos hexadecimais, com os quais, através de um código no Google Colab, pode-se ver a imagem capturada.

Figura 25 – Captura de imagem pelo módulo OV7670 (foto do autor).

Como o código de exemplo foi desenvolvido para o módulo OV7675, deve-se fazer uma mudança no código de teste da câmera. Na parte de inicialização da câmera, no campo no qual está escrito OV7675, deve-se mudar para 0V7670. O trecho do código em questão, encontra-se na Figura 26.

Figura 26 – Mudança no código de teste para o módulo OV7670 (fonte do autor).

```
// Initialize the OV7675 camera
void if(!Camera.begin(QCIF, RGB565, 1 OV7675)) {
    Serial.println("Failed to initialize camera");
    While(1);
}
```

Um teste adicional relacionado somente ao funcionamento da câmera foi executado para verificar o processamento e envio de informações em tempo real. As imagens mostradas pelo software Processing (PROCESSING, 2022) em certos intervalos de tempo continham algum ruído porém podia-se ver nitidamente o cenário apontado pelo OV7670.

4.3 DESENVOLVIMENTO DA APLICAÇÃO E CORRELAÇÃO COM O CICLO DE DESENVOLVIMENTO TINYML

Como visto na Seção 3.2.1, a aquisição de dados é o primeiro passo de desenvolvimento da aplicação que realizará a detecção/classificação das entradas lidas pelo o sensor. Visto que o cenário de teste de estanqueidade não varia no que se refere a variação de situações, o conjunto de dados formado não precisou ter uma quantidade excessiva de dados.

4.3.1 Preparação dos dados

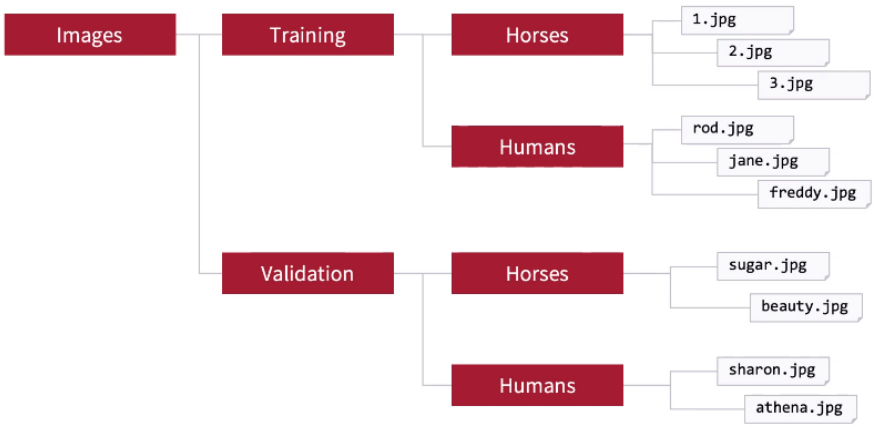
Para manter a assertividade das previsões feitas,deve-se padronizar o ambiente que se encontra o tanque, pois como mostrado na Figura 13, o aquário utilizado encontra-se acima de uma superfície branca. Sendo assim, deve-se manter esse padrão, ou caso o ambiente de teste varie, deve-se levar em consideração essas variações na criação da base de dados.

Os primeiros passos de desenvolvimento se concentram na ferramenta Google Colab. Esta ferramenta possibilita verificar a funcionalidade primária do sistema, como está sendo visto os dados de treinamento, como o modelo proposto se comporta após o treinamento e como está a assertividade da inferência do modelo em nuvem. Estas primeiras etapas estão baseadas na utilização da biblioteca TensorFlow.

Com isso, foi usado a base de dados já citadas, OK e NOK, e para manipulação das fotos no Colab, foram criadas pastas com os respectivos nomes no Google Drive, assim, com algumas funções do módulo OS (módulo para ler ou escrever um arquivo; manipular estruturas de diretórios; entre outros) do Python podemos manipular esses arquivos via seus diretórios para fornecer os dados à rede neural. Foi criado um objeto

de treinamento e validação para as condições OK e NOK. Assim podemos indicar ao nosso modelo o que é uma peça valida ou não, pela separação dos diretórios existentes.

Figura 27 – Classificação e separação das imagens.



Fonte: Reddi (2020).

O exemplo acima retrata como é feita a rotulação para que uma rede diferencie homens de cavalos. A mesma estratégia foi adotada neste trabalho. Foi criado um diretório para as imagens de treinamento e esse diretório contém uma subdivisão em duas pastas: uma rotulada como peças OK e outra NOK. Replica-se a mesma ideia para o diretório de validação.

Com a biblioteca TensorFlow é possível a utilização do parâmetro ImageDataGenerator para realizar as modificações necessárias e transformar um banco de dados simples em um banco de dados robusto e treinável.

Na Figura 28, podemos ver a utilização do ImageDataGenerator, onde utiliza-se a variável *rescale* para normalizar as imagens dividindo-as por 255. O próximo passo seria a utilização de funções que generalizem o banco de dados, metodologia chamada de Data Augmentation.

Como dito anteriormente, as possíveis variações dentro do cenário do teste de estanqueidade são, em quantidade, pequenas. Com isso, as fotos adicionadas ao banco já foram pensadas para abranger, por exemplo, a posição do aço utilizado para imergir a bolinha. De igual maneira, no código está comentado as operações de modificações para conhecimento em aplicações que sejam necessárias.

O método *rotation_range* rotaciona aleatoriamente todas as imagens do arquivo, para a esquerda ou direita. Pode-se perceber que é possível controlar o grau de rotação. Na imagem em questão, as imagens seriam rotacionadas em 40 graus.

As variáveis *shift_range* deslocam as imagens para evitar apenas o aprendizado de imagens centralizadas. Como a câmera da nossa aplicação estará fixa, e em processos industriais as posições também são fixas, este parâmetro não é necessário. O valor de 0.2 na linha comentada da variável em questão indica a porcentagem de

deslocamento da imagem.

Figura 28 – Código para aprimoramento do banco de dados (fonte do autor).

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(
      rescale=1./255,
      #rotation_range=40,
      #width_shift_range=0.2,
      #height_shift_range=0.2,
      #shear_range=0.2,
      #zoom_range=0.2,
      #horizontal_flip=True,
      #fill_mode='nearest'
      )

train_generator = train_datagen.flow_from_directory(
        '/content/drive/My Drive/NOK-OK',
        target_size=(100, 100),
        batch_size=128,
        class_mode='binary')

validation_datagen = ImageDataGenerator(rescale=1./255)

validation_generator = validation_datagen.flow_from_directory(
        '/content/drive/My Drive/NOK-OK-validation',
        target_size=(100, 100),
        class_mode='binary')
```

A variável *shear_range* realiza deslocamentos angulares chamados de *skewing*, essa função se diferencia do *rotation_scale*, pois a última realiza a rotação no eixo perpendicular à foto. A primeira é a combinação dos eixos horizontais e verticais paralelos à foto. *Zoom range*, amplia a foto fazendo com que parte da imagem original se perca. Ou seja, no caso de uma foto de humanos, a foto seria ampliada porém somente acima da cintura se manteria na foto. O parâmetro *Zoom_range* ajuda no reconhecimento de uma imagem, sem precisar analisar a imagem por completo.

Já a variável *horizontal_flip* faz o espelhamento horizontal da imagem. Por último, *fill_mode* preenche alguns pixels que ficaram vazios devido às mudanças realizadas, por exemplo, em um rotação. O parâmentro *nearest* preenhe os pixels vazios com os mesmos valores dos pixels vizinhos que possuem alguma informação.

Após a utilização do artifício ImageDataGenerator, podemos criar uma base de treinamento indicando o diretório das imagens. Pela variável *target_size*, consegue-se redefinir o tamanho das imagens após a normalização. O parâmetro *batch_size* separa as imagens em lotes.

A última variável, *class_mode*, define a quantidade de classificações de sua base de dados e também do sistema como um todo. Existem dois tipos: *binary* quando

existem dois tipos de classificação (no caso da aplicação em questão, OK e NOK) e *categorical* quando existem mais de duas categorias de classificação.

Essa operação é feita tanto para a base de dados de treinamento quanto para a base de dados de validação, garantido assim, que a melhoria se estenda para todas as fases de treinamento da rede neural a ser utilizada.

4.3.2 Desenvolvimento e treinamento do modelo

A rede neural escolhida para o modelo de classficação foi a CNN, do inglês *Convolutional Neural Network*, em português Redes Neurais Convolucionais. Segundo Lee et al. (2019), a CNN é um rede neural que provou ser usada com sucesso em vários campos (reconhecimento/classificação de imagem, reconhecimento de voz, tradução de idiomas).

A maior vantagem da utilização das CNN é a utilização de pesos e termos constantes em funções para a extração de elementos de imagem que facilitam o reconhecimento de imagens complexas. Esses elementos extraídos formam dados que realçam o brilho, linhas horizontais e verticais, entre outras formas complexas que compõem a construção de uma imagem (LEE et al., 2019).

A Figura 29 mostra como foi aplicado o CNN e como se estrutura em código uma rede neural convolucional.

Figura 29 – Código para definição de um modelo (fonte do autor).

```
model = tf.keras.models.Sequential([

    tf.keras.layers.Conv2D(8, (3,3), activation='relu', input_shape=(100, 100, 3)),
    tf.keras.layers.MaxPooling2D(2, 2),

    tf.keras.layers.Conv2D(16, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),

    tf.keras.layers.Conv2D(32, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),

    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),

    tf.keras.layers.Flatten(),

    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(32, activation='relu'),

    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

Cada camada da rede acima tem uma função específica. A junção dessas funções formam a definição da CNN. A camada de convolução, representada pela variável *tf.feras.layers.Conv2D*, é responsável por extrair o conteúdo da imagem, transformando em dados. Essa transformação ocorre por meio da aplicação de filtros que retêm as informações de interesse das imagens.

A camada de Pooling, apresentada como *tf.feras.layers.MaxPooling2D* recebe a informação filtrada pela etapa de convolução e simplifica essa informação, transformando os dados, normalmente recebidos em uma matriz, em um único valor. Após a passagem por 4 camadas do conjunto convolução-Polling, a informações é passada para uma camada totalmente conectada, *tf.feras.layers.Flatten*. Essa camada pega a saída das camadas anteriores, faz uma redução e as transforma em um único vetor que pode ser uma entrada para a próxima etapa.

As próximas duas camadas, *tf.feras.layers.Dense* fazem, respectivamente, as funções de: coletar a informação da camada anterior e aplicar pesos e termos constantes para fazer a previsão correta. E a última fornece ao usuário/programa a probabilidade final da inferência constatada diante das imagens.

Figura 30 – Código para compilação de um modelo (fonte do autor).

```
from tensorflow.keras.optimizers import RMSprop
optimizer = RMSprop(learning_rate=0.0001)
model.compile(loss='binary_crossentropy',
              optimizer=optimizer,
              metrics=['acc'])
```

Após a adição das camadas, deve-se compilar o modelo para que ele se torne efetivo. O método *compile* realiza essa função. O argumento *optimizer* é um dos argumentos necessários para se compilar um modelo. O escolhido para este caso foi o RMSprop, dado sua alta performance para aplicações de reconhecimento de imagem conforme citado e exposto por Zaheer e Shaziya (2019).

O objetivo das *loss_functions* é calcular a quantidade que um modelo deve procurar minimizar durante o treinamento. A biblioteca TF Keras recomenda a utilização da *binary_cross_entropy* para aplicações de classificação binária (1 ou 0).

Por ultimo, o argumento *metrics* é uma função usada para avaliar o desempenho do seu modelo. A função escolhida foi a *acc*, abreviação de *accuracy* que calcula a frequência de previsões corretas.

Figura 31 – Código para treinamento de um modelo (fonte do autor).

```
history = model.fit(
      train_generator,
      steps_per_epoch=8,
      epochs=2,
      verbose=1,
      validation_data=validation_generator)
```

Para o treinamento, utilizamos o método *fit* da nossa classe *model*, conforme

a Figura 31. Inclui-se as base de dados de treinamento e validação criadas pelo ImageDataGenerator e as variáveis que deve-se ter atenção são as: *steps_per_epochs* e *epochs*. *Epochs* representa quantos ciclos de treinamento serão realizados efetivamente, e *steps_per_epochs* quantos passos serão dados em um ciclo. Ao alterar essas variáveis, pode-se chegar a uma maior precisão do modelo.

4.3.3 Otimizações e deploy do modelo

Para realizar o deploy do modelo treinado para realizar as inferências, conforme a Seção 3.2.5, deve-se realizar algumas técnicas de otimização e conversão para que o modelo se encaixe no dispositivo final a ser utilizado. O primeiro método aplicado foi o de quantização durante o treinamento (QAT).

Figura 32 – Código para quantização de um modelo (fonte do autor).

```
! pip install -q tensorflow-model-optimization
import tensorflow_model_optimization as tfmot

quantize_model = tfmot.quantization.keras.quantize_model

# q_aware stands for for quantization aware.
q_aware_model = quantize_model(model)

# `quantize_model` requires a recompile.
q_aware_model.compile(loss='binary_crossentropy',
              optimizer=optimizer,
              metrics=['acc'])

q_aware_model.summary()


q_aware_model.fit(
      train_generator,
      steps_per_epoch=8,
      epochs=2,
      verbose=1,
      validation_data=validation_generator)
```

Os passos para realizar a quantização QAT são bem simples. A Figura 32 começa mostrando as dependências necessárias para a utilização. Na quarta linha efetiva-se a conversão para um modelo quantizado e, após isso é necessário a compilação do modelo e seu treinamento subsequente, assim como no treinamento de um modelo não quantizado. Vale ressaltar que neste exemplo foi realizado dois treinamentos, com fins explicativos, antes e durante a quantização. Porém, pode-se e se torna mais vantajoso realizar apenas durante a quantização.

Figura 33 – Código para otimização de um modelo (fonte do autor).

```
export_dir = 'saved_model'
tf.saved_model.save(q_aware_model, export_dir)

import pathlib
converter = tf.lite.TFLiteConverter.from_saved_model(export_dir)

converter.optimizations = [tf.lite.Optimize.DEFAULT]
tflite_model = converter.convert()
```

Os últimos passos seriam: realização da otimização e a conversão. O código acima, realiza a otimização simultaneamente com a conversão. Para iniciar a conversão, salvamos o modelo utilizando o argumento *saved_model*, que salva argumentos, parâmetros e arquivos em um formato comum para a utilização em diversas aplicações como microcontroladores, smartphones e etc. As três últimas linhas realizam a conversão e otimização.

Vale detalhar que a opção *tf.lite.Optimize.DEFAULT* pode ser substituída por *tf.lite.Optimize.FOR_SIZE* e *tf.lite.Optimize.FOR_LATENCY*, otimizando assim em tamanho e latência respectivamente. A opção DEFAULT realiza a otimização das duas maneiras, que é interessante e importante no caso da aplicação.

Como o desenvolvimento em microcontroladores é feito no Arduino IDE, deve-se por último gerar (realizar a conversão) um arquivo em .cc para sua utilização. Este processo é a conversão de TFLite para TFLite Micro.

Figura 34 – Código para conversão de um modelo (fonte do autor).

```
!apt-get update && apt-get -qq install xxd
MODEL_TFLITE = 'model.tflite'
MODEL_TFLITE_MICRO = 'model.tflite.cc'
!xxd -i {MODEL_TFLITE} > {MODEL_TFLITE_MICRO}
REPLACE_TEXT = MODEL_TFLITE.replace('/', '_').replace('.', '_')
!sed -i 's/'{REPLACE_TEXT}'/g_model/g' {MODEL_TFLITE_MICRO}
!cat {MODEL_TFLITE_MICRO}
```

Para converter o modelo quantizado do TensorFlow Lite em um arquivo fonte C que pode ser carregado pelo TensorFlow Lite Micro no Arduino, basta usar a ferramenta *xxd* para converter o arquivo .tflite em um arquivo .cc. A primeira linha da Figura 34 exibe o código que realiza algumas operações disponibilizando assim o uso da ferramenta. Este arquivo conterá um vetor com dados binários que formam o modelo convertido. A estrutura do código esperado pode ser vista na Figura 35.

Figura 35 – Modelo convertido (fonte do autor).

```
unsigned char g_model[] = {
  0x20, 0x00, 0x00, 0x00, 0x54, 0x46, 0x4c, 0x33, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x12, 0x00, 0x1c, 0x00, 0x04, 0x00, 0x08, 0x00, 0x0c, 0x00,
  0x10, 0x00, 0x14, 0x00, 0x00, 0x00, 0x18, 0x00, 0x12, 0x00, 0x00, 0x00,
  0x03, 0x00, 0x00, 0x00, 0x94, 0x48, 0x00, 0x00, 0x34, 0x42, 0x00, 0x00,
  0x1c, 0x42, 0x00, 0x00, 0x3c, 0x00, 0x00, 0x00, 0x04, 0x00, 0x00, 0x00,
  0x01, 0x00, 0x00, 0x00, 0x0c, 0x00, 0x00, 0x00, 0x08, 0x00, 0x0c, 0x00,
  0x04, 0x00, 0x08, 0x00, 0x08, 0x00, 0x00, 0x00, 0x08, 0x00, 0x00, 0x00,
  0x0b, 0x00, 0x00, 0x00, 0x13, 0x00, 0x00, 0x00, 0x6d, 0x69, 0x6e, 0x5f,
  0x72, 0x75, 0x6e, 0x74, 0x69, 0x6d, 0x65, 0x5f, 0x76, 0x65, 0x72, 0x73,
  0x69, 0x6f, 0x6e, 0x00, 0x0c, 0x00, 0x00, 0x00, 0xd4, 0x41, 0x00, 0x00,
  0xb4, 0x41, 0x00, 0x00, 0x24, 0x03, 0x00, 0x00, 0xf4, 0x02, 0x00, 0x00,
  0xec, 0x02, 0x00, 0x00, 0xe4, 0x02, 0x00, 0x00, 0xc4, 0x02, 0x00, 0x00,
  0xbc, 0x02, 0x00, 0x00, 0x2c, 0x00, 0x00, 0x00, 0x24, 0x00, 0x00, 0x00,
  0x1c, 0x00, 0x00, 0x00, 0x04, 0x00, 0x00, 0x00, 0x16, 0xbd, 0xff, 0xff,
  0x04, 0x00, 0x00, 0x00, 0x05, 0x00, 0x00, 0x00, 0x31, 0x2e, 0x35, 0x2e,
  0x30, 0x00, 0x00, 0x00, 0x94, 0xba, 0xff, 0xff, 0x98, 0xba, 0xff, 0xff,
  0x32, 0xbd, 0xff, 0xff, 0x04, 0x00, 0x00, 0x00, 0x80, 0x02, 0x00, 0x00,
  0xfa, 0xee, 0x28, 0xc4, 0xee, 0xfe, 0xcf, 0x0f, 0x1e, 0xf7, 0x1f, 0x06,
  0x0d, 0xed, 0xe9, 0x83, 0x5c, 0xc9, 0x18, 0xe3, 0xf9, 0x14, 0x28, 0x2a,
  0x09, 0xf2, 0x18, 0x34, 0x62, 0xea, 0xef, 0xd6, 0x36, 0xb7, 0x1e, 0xf7,
  0x3b, 0x22, 0x28, 0x39, 0xc2, 0x9d, 0xf1, 0x07, 0x5e, 0x0b, 0x1e, 0x2c,
  0x07, 0xdd, 0xfd, 0xc3, 0xd8, 0x4a, 0xf3, 0x28, 0xa7, 0x16, 0xd5, 0xf1,
  0xc3, 0x05, 0xfd, 0x27, 0xcc, 0xba, 0x1e, 0xcb, 0xd7, 0x3d, 0xd4, 0x29,
```

Com o arquivo .cpp gerado, utiliza-se do mesmo nos códigos que rodam no Arduino IDE. Este passo de desenvolvimento no IDE será abordado com maior abrangência na subseção seguinte.

4.3.3.1 Desenvolvimento no Arduino IDE

Para validar o deploy da aplicação para o Arduino, começou-se fazendo um teste com o código denominado *person detection*, disponível nos exemplos oferecidos pela Harvard. O teste se refere ao reconhecimento facial, em que quando um rosto de um ser humano é capturado pelo OV7060, o Arduino acende um led verde. Quando o módulo não reconhece o rosto humano o led fica azul.

Ao realizar o carregamento do programa ao Arduino não houveram problemas, de acordo com os *setups* mostrados nas Seçoes 4.1. E o funcionamento da aplicação também se mostrou satisfatório, com alguns delays em reconher a face humana. Porém, mesmo com os ruídos apresentados no teste do módulo OV7670, somado a não robustez do hardware, visto que os componentes estão sendo conectados via protoboard, o resultado do teste se mostrou positivo.

Conforme Reddi (2020), o nome de aplicações de reconhecimento de imagens é *Visual Wake Words*, que se resume na realização de alguma ação, quando um sensor de imagem reconhece uma situação pré-definida. No caso do código *person detection*, um led verde acende ao se deparar com um rosto. Para o teste de estanqueidade, foi utilizado parte do código da aplicação de detecção de pessoas.

Figura 36 – Declaração de variáveis (fonte do autor).

```
tflite::ErrorReporter* error_reporter = nullptr;
const tflite::Model* model = nullptr;
tflite::MicroInterpreter* interpreter = nullptr;
TfLiteTensor* input = nullptr;


constexpr int kTensorArenaSize = 136 * 1024;
static uint8_t tensor_arena[kTensorArenaSize];
```

Também, segundo Reddi (2020), o modelo de código para as aplicações *Visual Wake Words* é padronizado, mudando os valores das variáveis de memória, estrutura da rede neural e coisas específicas da aplicação em si. A Figura 36 apresenta as variáveis padrão deste tipo de aplicação, as quais servem, respectivamente para: gerar alertas de erros; salva o modelo da rede; variável para executar o modelo; variável de entrada de dados e as duas últimas se referem a locação de memória para entradas, saídas e outras operações.

Após as declarações, inicializamos o modelo conforme a Figura 37, com o método *GetModel*. É importante observar que é passado como parâmetro uma variável chamada *g_person_detect_model_data*, este arquivo é o modelo em .cpp gerado conforme a Figura 35. Então, deve-se subsituir o que está entre chaves no arquivo *g_person_detect_model_data* e no final deste arquivo, a variável que recebe o tamanho do arquivo também deve ser substituída pela variável do modelo gerada pela aplicação de estanqueidade.

Figura 37 – Carregar o modelo (fonte do autor).

```
model = tflite::GetModel(g_person_detect_model_data);
if (model->version() != TFLITE_SCHEMA_VERSION) {
  TF_LITE_REPORT_ERROR(error_reporter,
                       "Model provided is schema version %d not equal "
                       "to supported version %d.",
                       model->version(), TFLITE_SCHEMA_VERSION);
  return;
}
```

Após isso, é realizado uma checagem para ver se o o esquema do modelo é compatível com o que o TFLite Micro suporta. O próximo passo é definir as operações da sua rede via classe *Operator Support (OpsResolver)* que deve ser definida a partir do modelo contruído na primeira fase de implementação da rede neural.

Figura 38 – OpResolvers - Mecanismo para realizar operações do modelo (fonte do autor).

```
static tflite::MicroMutableOpResolver<5> micro_op_resolver;
micro_op_resolver.AddConv2D();
micro_op_resolver.AddFullyConnected();
micro_op_resolver.AddMaxPool2D();
micro_op_resolver.AddSoftmax();
micro_op_resolver.AddReshape();
```

A Figura 38, exibe os operadores específicos conforme definido na Figura 29. Caso o desenvolvedor não conheça os operadores equivalentes à sua rede neural, pode-se usar a opção *AllOpsResolver* porém obtém-se um gasto desnecessário de memória.

As últimas etapas da inicialização consistem na construção do Interpreter que carregará o modelo e na alocação efetiva da memória a ser utilizada. A última linha da Figura 39 exibe também a chamada do Interpreter para buscar o dado de entrada.

Figura 39 – Últimas etapas da fase de inicialização (fonte do autor).

```
static tflite::MicroInterpreter static_interpreter(
    model, micro_op_resolver, tensor_arena, kTensorArenaSize, error_reporter);
interpreter = &static_interpreter;


TfLiteStatus allocate_status = interpreter->AllocateTensors();
if (allocate_status != kTfLiteOk) {
  TF_LITE_REPORT_ERROR(error_reporter, "AllocateTensors() failed");
  return;
}

input = interpreter->input(0);
```

O restante do código é similar, pois as operações são basicamente as mesmas. As únicas alterações a serem feitas seria a troca do módulo OV7675 para o OV7670 no arquivo *arduino_image_provider.cpp* e executar a mudança de maneira identica à que se encontra na Figura 26. A resposta do sistema (mudança na cor do LED) também segue o padrão da aplicação de detecção de pessoas.

5 RESULTADOS

Na seção vigente, serão comentados os resultados de cada etapa, pois a avaliação do trabalho se concentra na soma dos resultados obtidos em cada etapa. Sendo assim, torna-se mais fácil a melhoria da aplicação como um todo ao encontrar os desafios separadamente em cada fase.

5.1 DEFINIÇÃO DO MODELO E OTIMIZAÇÕES

Os primeiros resultados observados no desenvolvimento, foram a geração exitosa da rede neural convolucional. Após a definição das camadas e compilação do modelo, pelo método *summary* aplicado na classe do modelo criado, pode-se ver a impressão das camadas, tipos de camadas e quantidade de parâmetros utilizados (informação importante para controle do tamanho da rede).

Figura 40 – Impressão do modelo criado (fonte do autor).

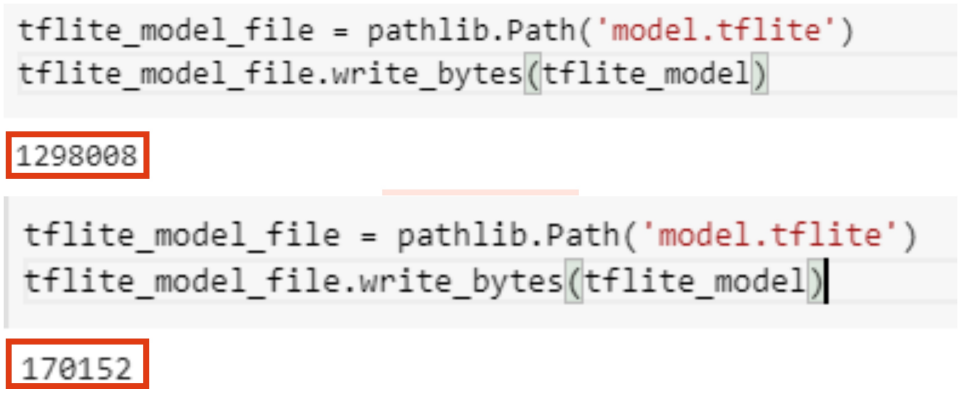
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 quantize_layer_1 (QuantizeL  (None, 100, 100, 3)      3
 ayer)

 quant_conv2d (QuantizeWrapp  (None, 98, 98, 8)        243
 erV2)

 quant_max_pooling2d (Quanti  (None, 49, 49, 8)        1
 zeWrapperV2)

 quant_conv2d_1 (QuantizeWra  (None, 47, 47, 16)       1203
 pperV2)

 quant_max_pooling2d_1 (Quan  (None, 23, 23, 16)       1
 tizeWrapperV2)

 quant_conv2d_2 (QuantizeWra  (None, 21, 21, 32)       4707
 pperV2)

 quant_max_pooling2d_2 (Quan  (None, 10, 10, 32)       1
 tizeWrapperV2)

 quant_conv2d_3 (QuantizeWra  (None, 8, 8, 64)         18627
 pperV2)

 quant_max_pooling2d_3 (Quan  (None, 4, 4, 64)         1
 tizeWrapperV2)

 quant_flatten (QuantizeWrap  (None, 1024)             1
 perV2)

 quant_dense (QuantizeWrappe  (None, 128)              131205
 rV2)

 quant_dense_1 (QuantizeWrap  (None, 32)               4133
 perV2)

 quant_dense_2 (QuantizeWrap  (None, 1)                38
 perV2)

=================================================================
```

A Figura 40 exibe a impressão do modelo já quantizado, o qual possui 4 camadas de convolução e de *pooling*. Ambas possuem o mesmo número de filtros começando em 8 filtros na primeira camada e dobrando seu valor nas camadas subsequentes.

Também utilizou-se uma camada totalmente conectada para unir as informações e 3 camadas que realizam a distribuição dos pesos para realizar a classificação da imagem.

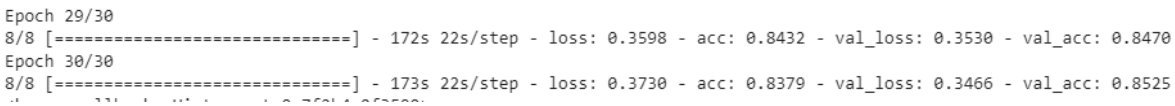
Pode-se observar o tamanho do modelo em bytes gerado e assim comparar se a conversão e otimizações surtiram o efeito esperado. Espera-se que após os procedimentos de otimização, o tamanho do modelo sofra uma redução de no mínimo 4 vezes do seu tamanho original. No caso real, conforme Figura 41, a redução foi de aproximadamente 7.5, sendo 1298008 bytes o tamanho do modelo sem a otimização e 170152 bytes com a otimização.

Figura 41 – Comparação de tamanhos entre um modelo otimizado e um não otimizado (fonte do autor).



O treinamento do modelo se comportou satisfatoriamente ao fixar a variável *epochs* com um valor elevado. Conforme a Figura 32, o valor de *epochs* se encontra baixo apenas para validação de compilação do código, pois o treinamento com um valor aproximadamente na ordem de 100, que foi utilizado oficialmente, demora em torno de 3 horas para se realizar.

Figura 42 – Precisão do modelo durante treinamento (fonte do autor).

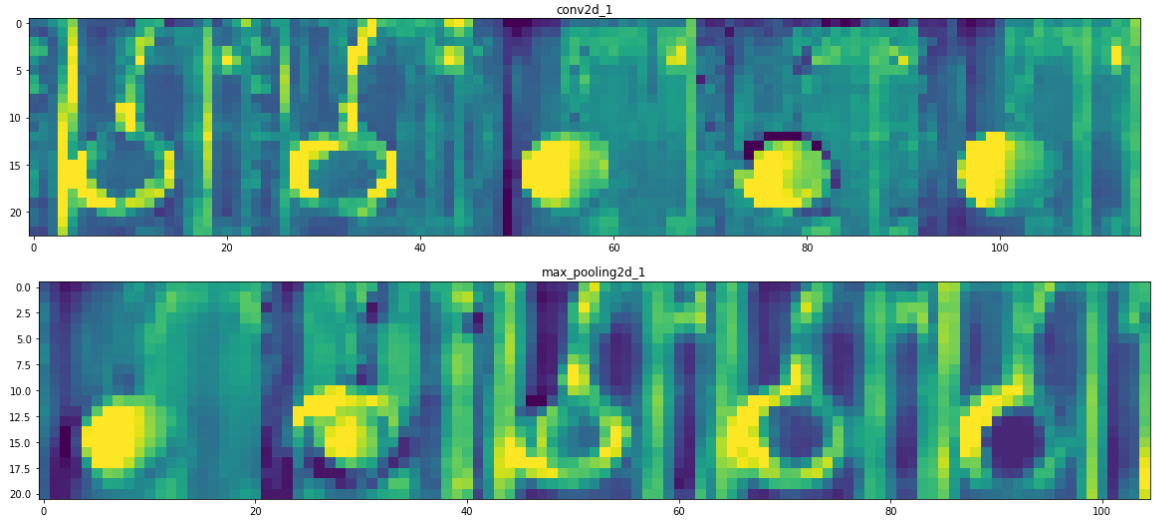


Na Figura 42 *loss acc* referem-se às perdas e precisão da assertividade do modelo em relação à base de dados de treinamento respectivamente. *Val loss* e *val acc*, referem-se também às perdas e precisão, porém da base de dados de validação. A Figura 42 mostra os resultados no trigésimo ciclo de treinamento, e seu desempenho aumento proporcionalmente ao números de ciclos determinado por

epochs, obviamente se a rede for desenvolvida de maneira eficiente para a aplicação. Sendo assim, utilizando o valor de *epochs* em 100, obteve-se *loss* muito próximo a zero e *acc* muito próximo a um.

Pode-se observar também o comportamento da convolução em cada camada, realizando assim a filtragem e extração dos dados que de fato alimentaram a rede.

Figura 43 – Dado após passagem pela segunda camada da rede (fonte do autor).



Pela Figura 43, é possível observar a composição do dado após a passagem pela segunda camada da rede neural. Com isso garante-se que a convolução realiza de forma efetiva a simplificação do dado a ser treinado e inferido.

5.2 INFERÊNCIA DO MODELO EM NUVEM

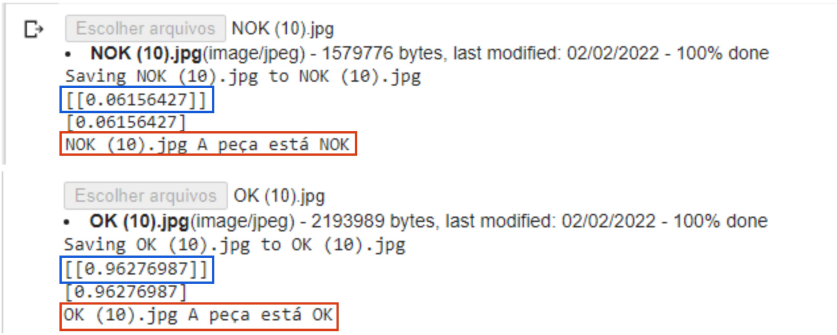
Com os valores de treinamento e validação dentro da meta esperada, o próximo passo antes de realizar a conversão para Tensor Flow Lite deve-se apurar como está a classificação do modelo em nuvem. O código que realiza a predição é exibido na Figura 44.

Figura 44 – Inferência em nuvem (fonte do autor).

```
import numpy as np
from google.colab import files
from keras.preprocessing import image

uploaded = files.upload()

for fn in uploaded.keys():

  # predicting images
  path = '/content/' + fn
  img = image.load_img(path, target_size=(100, 100))
  x = image.img_to_array(img)
  x = x / 255.0
  x = np.expand_dims(x, axis=0)

  image_tensor = np.vstack([x])
  classes = q_aware_model.predict(image_tensor)
  print(classes)
  print(classes[0])
  if classes[0]>0.5:
    print(fn + " A peça está OK")
  else:
    print(fn + " A peça está NOK")
```

Após o carregamento de todas dependências, carrega-se a quantidade e as fotos a serem classificadas pelo modelo. Em seguida normalizamos a imagem ao padrão proposto no ImageDataGeneration e usamos a função *predict* para gerar a probabilidade da peça ser OK ou NOK.

Figura 45 – Resultado da inferência em nuvem (fonte do autor).



Na Figura 45, o que se encontra destacado em azul é a probabilidade da imagem a ser analisada conter o ensaio de uma peça conforme. Como pode-se ver, o primeiro resultado foi obtido a partir de uma foto nomeada como NOK (10).jpg, sendo

tal uma imagem de uma peça não conforme e sua probabilidade para ser uma peça boa está perto de 6 porcento, classificando a foto como NOK.

Já no segundo resultado, é visto uma imagem nomeada como OK (10).jpg, sendo tal uma imagem de uma peça conforme e sua probabilidade resultando ser em torno de 96 porcento, sendo classificada então como uma peça OK.

5.3 INFERÊNCIA DO MODELO NO TENSOR FLOW LITE

Modelos obtidos a partir da conversão utilizando *TFLiteConverter* pode ser testado em Python utilizando *interpreters*.

Figura 46 – Interpreter para inferência de modelos em TFLite (fonte do autor).

```
interpreter = tf.lite.Interpreter(model_content=tflite_model)
interpreter.allocate_tensors()

input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()
```

Tflite_model pode ser salvo em um arquivo e carregado posteriormente, ou diretamente no Interpreter. Como o TensorFlow Lite pré-planeja as alocações de tensores (arrays de armazenamento de informações) para otimizar a inferência, o usuário precisa chamar a função *alocar tensors* antes de qualquer inferência.

O código para realizar a inferência no modelo TFLite foi parecido com o que se encontra na Figura 44. Apenas a parte da alocação dos tensores e inferência teve algumas alterações.

Figura 47 – Diferenças de predições entre TF e TFLite (fonte do autor).

```
image_tensor = np.vstack([x])
classes = q_aware_model.predict(image_tensor)
```

```
to_predict = np.vstack([x])
interpreter.set_tensor(input_details[0]['index'], to_predict)
interpreter.invoke()
classes = interpreter.get_tensor(output_details[0]['index'])
```

Na Figura 47, o código acima da linha azul pertence a parte do código da Figura 44. O que está abaixo da linha azul, refere-se a utilização do Interpreter. Depois de carregar o modelo, realiza-se a inferência. Para tal, é necessário obter detalhes dos tensores de entrada e saída para o modelo. Define-se o valor do tensor de entrada, invoca-se o modelo (via função *invoke*) e obtêm-se o valor do tensor de saída.

O resultado para este procedimento foi semelhante ao que se encontra na Figura 45, validando assim mais uma etapa do procedimento em converter a aplicação

de inteligência artificial para uma aplicação TinyML.

5.4 DEPLOY PARA TFLITE MICRO

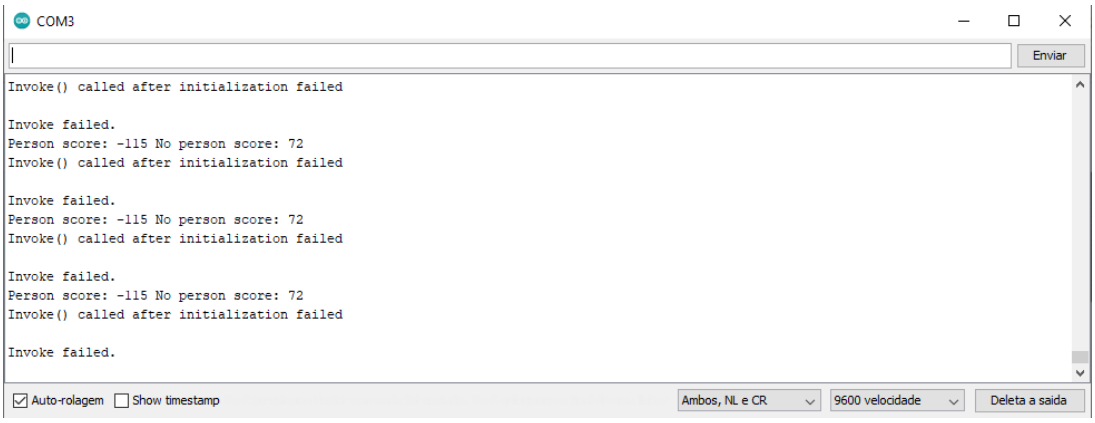
Após a realização dos ajustes descritos na Seção 4.3.3.1, foi feita a compilação do código. Não gerando erros, apresentou os seguintes resultados: o *sketch* (nome dado ao programa ou conjunto de programas a ser carregado em uma única aplicação) utilizou 32% do espaço de armazenamento de memória e variáveis globais 71% de memória dinâmica, deixando o restante para variáveis locais.

Como comparação, o código de detecção de pessoas possui um valor de 45% do espaço de armazenamento de memória e mesma porcentagem para memória dinâmica. Sendo assim, no que se diz respeito a dimensionamento da rede, otimização de memória, o código desenvolvido foi modelado satisfatoriamente.

Porém, na execução e inferência no dispositivo final, não obteve-se o resultado esperado. Na Figura 48 pode-se ver a mensagem de falha na chamada da função *Invoke*. Algumas pesquisas foram realizadas para entender a razão e possíveis saídas para a resolução do erro mostrado.

A maioria das propostas buscadas relatavam ações referentes à memória. Vale a ressalva de que o modelo da rede neural desenvolvido inicialmente era maior do que o mostrado na Figura 29. Diversos modelos foram testados, até que se chegou no modelo final. O primeiro modelo proposto possuía um tamanho aproximando de 2,5 Megabytes e o atual possui 0,17 Megabytes.

Figura 48 – Erro na inferência (fonte do autor).

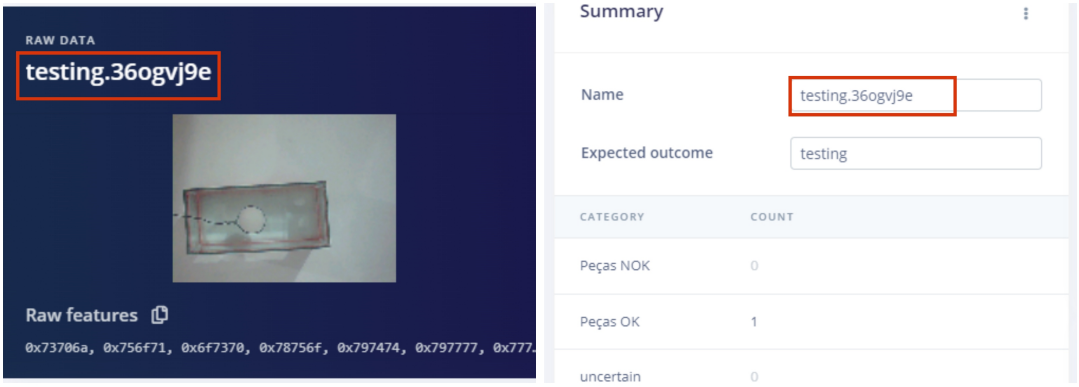


Ainda que essas mudanças foram realizadas, o resultado se manteve o mesmo: gerando o erro acima. Outras modificações foram realizadas, como mudança na alocação dos tensores (que gerenciam a memória) porém sem nenhum resultado efetivo.

Na tentativa de validar o modelo e se as imagens captadas pelo OV7670 seriam validadas pela rede, foi realizado um experimento na plataforma Edge Impuse,

no qual realiza-se a o desenvolvimento de uma aplicação TinyML de forma automática, apenas fornecendo a base de dados, configurando a rede e realizando os testes. Após o treinamento da rede, o fornecimento de imagens a serem analisadas foi de responsabilidade do OV7670.

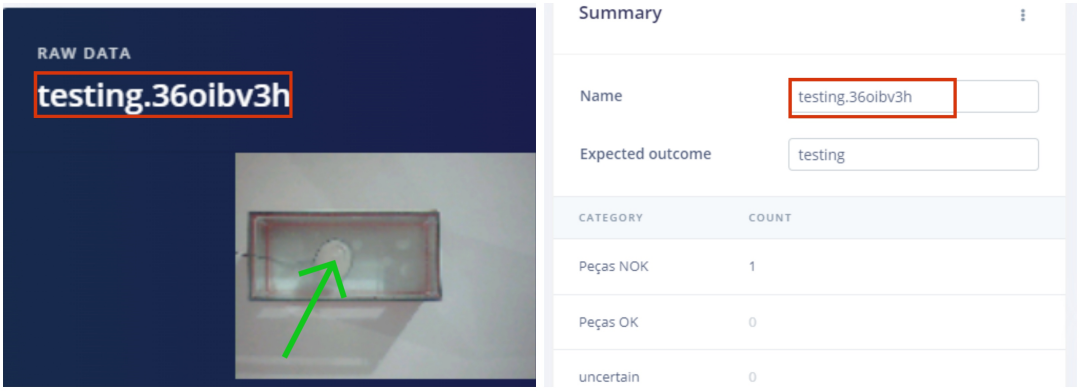
Figura 49 – Inferência de imagens captadas via OV7670 - Peça OK (fonte do autor).



A Figura 49, exibe a captação de uma imagem pelo módulo conectado ao Arduino. Esta imagem refere-se a uma peça em perfeitas condições. A direta da imagem, é possível a visualização da classificação realizada pelo modelo, indicando que é uma peça OK.

Já a Figura 50, apresenta uma peça não conforme. A seta em verde indica a bolha surgida pela imersão desta peça na água. Tal situação foi indicada como NOK pelo modelo.

Figura 50 – Inferência de imagens captadas via OV7670 - Peça NOK (fonta de autor).



Na mesma plataforma é realizada a conversão para o Arduino, porém a intenção do último teste feito é testar a eficacia do modelo ao classificar imagens oriundas do módulo. A garantia que o Arduino possui a capacidade computacional, energética e numérica para realizar classificações de imagens foi obtida pelo teste de detecção de pessoas realizado na Seção 4.3.3.1.

6 CONCLUSÕES

Tendo em vista todas as etapas do desenvolvimento do projeto e seus respectivos resultados, pode-se concluir que o cenário de aplicações TinyML é muito promissor, considerando a capacidade cada vez mais ampla de extração de dados de diversas atividades para o controle e supervisão de ações e processos no âmbito da Indústria 4.0.

Conclui-se também que o projeto de teste de estanqueidade via microcontrolador é viável, pois tanto em nuvem como em aplicações web, o modelo treinado atendeu satisfatoriamente ao que foi proposto. O resultado positivo, tanto da modelagem da base de dados, como o de todo o ciclo de criação do sistema antes de ser compilado no microcontrolador, foi possível de ser alcançado pelo suporte robusto encontrado em livros e sites para tal situações.

Entretanto, a efetiva funcionalidade de se usar o microcontrolador, garantido eficiência em memória, energia, dados e outros, não foi possível de se alcançar. Baseado no fato do trabalho ser realizado alicerçado no curso oferecido por Reddi (2020), a etapa de deploy do modelo foi abordada de uma maneira superficial, utilizando apenas um exemplo já desenvolvido para exemplificar.

Sendo assim, próximos trabalhos podem ser desenvolvidos para se alcançar um domínio generalizado na compilação de modelos de inteligência artificial em microcontroladores, analisando as diferenças quando se mudam os modelos de RNA e os diversos parâmetros utilizados. Tal estudo seria essencial pois complementaria o trabalho atual, disponibilizando assim um conhecimento completo de como desenvolver qualquer aplicação TinyML.

Complementando as sugestões de futuros trabalhos, propõe-se a integração dos microcontroladores com sistemas atuadores, ou seja, além de ter a funcionalidade de classificar e detectar, soma-se a habilidade em acionar sistemas, mecanismos baseados na inferências realizada.

REFERÊNCIAS

ARDUINO. **Arduino Nano 33 BLE Sense**. 2022. Disponível em: http://store-usa.arduino.cc/products/arduino-nano-33-ble-sense. Acesso em: 11 mai. 2022.

COLAB, G. **Conheça o Colab**. 2022. Disponível em: https://colab.research.google.com/?hl=pt_BR. Acesso em: 26 mai. 2022.

FLOW, T. **Tensor Flow**. 2021. Disponível em: https://www.tensorflow.org/lite. Acesso em: 11 set. 2021.

IAN GOODFELLOW, Y. B.; COURVILLE, A. **Deep Learning**. 2016. Disponível em: https://www.deeplearningbook.org/lecture_slides.html. Acesso em: 13 set. 2021.

IMPULSE, E. **Development Plataforms**. 2022. Disponível em: https://docs.edgeimpulse.com/docs/fully-supported-development-boards. Acesso em: 12 mai. 2021.

JORDAN, M. I.; MITCHEL, T. M. Machine learning: Trends, perspectives, and prospects. **Science**, v. 348, n. 11–13, p. 255–260, jul. 2015.

KAPUR, A. **Solar scare mosquito 2.0**. 2020. Disponível em: https://hackaday.io/project/174575-solar-scare-mosquito-20. Acesso em: 15 set. 2021.

KENSHIMA, G. **Diferenças entre microcontrolador e microprocessador**. 2021. Disponível em: https://www.filipeflop.com/blog/diferencas-entre-microcontrolador-e-microprocessador/. Acesso em: 12 set. 2021.

KIM, S. K. H. Linear domain-aware log-scale post-training quantization. In: IEEE, 1., 2021, Gangwon, Republic of Korea. **IEEE International Conference on Consumer Electronics-Asia (ICCE-Asia)**. Gangwon, Republic of Korea: IEEE, 2021. p. 1–3. Disponível em: https://ieeexplore.ieee.org/xpl/conhome/9641867/proceeding.

KRIZHEVSKY, A. **Learning Multiple Layers of Features from Tiny Images**. 2009. Disponível em: https://www.cs.toronto.edu/~kriz/cifar.html. Acesso em: 12 mai. 2021.

LEE, M.-Y. et al. The sparsity and activation analysis of compressed cnn networks in a hw cnn accelerator model. In: IEEE, 1., 2019, Jeju, South Korea. **International SoC Design Conference**. Jeju, South Korea: IEEE, 2019. p. 255–256. Disponível em: https://ieeexplore.ieee.org/xpl/conhome/9017212/proceeding.

DA LI XINBO CHEN, M. B.; ZONG, Z. Evaluating the energy efficiency of deep convolutional neural networks on cpus and gpus. In: IEEE, 1., 2016, Atlanta, GA, USA. **IEEE International Conferences on Big Data and Cloud Computing (BDCloud), Social Computing and Networking (SocialCom), Sustainable Computing and Communications (SustainCom) (BDCloud-SocialCom-SustainCom)**. Atlanta, GA, USA: IEEE, 2016. p. 477–484. Disponível em: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9166461.

LOMBARDO, T. **IoT device detects wind turbine faults in the field**. 2020. Disponível em: https://www.engineering.com/story/iot-device-detects-wind-turbine-faults-in-the-field. Acesso em: 12 set. 2021.

PIAZZETTA, G. R. **Avaliação de estanqueidade em vasos de pressão de pequeno porte com técnicas acústicas**. Dissertação de mestrado ao programa de Pós-Graduação em Engenharia Mecânica — Programa de Pós-Graduação em Engenharia Mecânica, Centro Tecnológico, Universidade Federal de Santa Catarina, 2017.

PROCESSING. **Processing**. 2022. Disponível em: https://processing.org/. Acesso em: 20 mai. 2022.

RADOWITZ, R. **Detecção de intrusão em redes móveis AD HOC, baseado em redes neurais artificiais**. Trabalho de Conclusão de Curso (Bacharel em Sistemas de Informação) — Bacharel em Sistemas de Informação, Centro Tecnológico Departamento de Informática e Estatística, Universidade Federal de Santa Catarina, 2009.

RAMCHARAN, A. **Transforming farmers' lives with just a mobile phone**. 2020. Disponível em: https://grow.google/intl/europe/story/transforming-farmers%E2%80%99-lives-with-just-a-mobile-phone. Acesso em: 13 set. 2021.

RANJAN, F. J. M. S. S. C. M. M. B. P. D. J. N. M. R. A comprehensive study on pre-pruning and post-pruning methods of decision tree classification algorithm. In: IEEE, 5., 2021, Tirunelveli, India. **5th International Conference on Trends in Electronics and Informatics (ICOEI)**. Tirunelveli, India: IEEE, 2021. p. 1339–1345. Disponível em: https://ieeexplore.ieee.org/xpl/conhome/9452735/proceeding.

REDDI, V. J. **Fundamentals of TinyML**. 2020. Disponível em: https://learning.edx.org/course/course-v1:HarvardX+TinyML1+3T2020/home. Acesso em: 3 set. 2021.

RIBEIRO, M.; GROLINGER, K.; CAPRETZ, M. A. Mlaas: Machine learning as a service. In: IEEE, 14., 2015, Miami, FL, USA. **2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)**. Miami, FL, USA: IEEE, 2015. p. 896–902. Disponível em: https://ieeexplore.ieee.org/document/7424435. Acesso em: 11 set. 2021.

RUSSEL, S.; NORVIG, P. **Inteligência Artificial**. 3. ed. Rio de Janeiro: Elsevier Editora Ltda., 2013.

SANCHEZ-IBORRA, R.; SKARMETA, A. F. Tinyml-enabled frugal smart objects: Challenges and opportunities. **IEEE Circuits and Systems Magazine**, v. 20, n. 3, p. 4–18, ago. 2020. Disponível em: https://ieeexplore.ieee.org/document/9166461. Acesso em: 10 sep. 2021.

SEE, C. Y. W. W. Y. Z. Z. Z. L. S. Y. L. J. Mlife: A lite framework for machine learning lifecycle initialization. In: IEEE, 8., 2021, Porto, Portugal. **IEEE 8th International Conference on Data Science and Advanced Analytics (DSAA)**. Porto, Portugal: IEEE, 2021. p. 1–2. Disponível em: https://ieeexplore.ieee.org/xpl/conhome/9564091/proceeding.

SOLANA, A. **Elephants vs trains: This is how AI helps ensure they don't collide**. 2020. Disponível em: https://www.zdnet.com/article/elephants-vs-trains-this-is-how-ai-helps-ensure-they-dont-collide/. Acesso em: 13 set. 2021.

THOMSEN, A. **Módulo câmera VGA OV7670**. 2013. Disponível em: https://www.filipeflop.com/blog/modulo-camera-vga-ov7670/. Acesso em: 5 mai. 2022.

ZAHEER, R.; SHAZIYA, H. 2019 third international conference on inventive systems and control (icisc). In: IEEE, 3., 2019, Coimbatore, India. **International Conference on Inventive Systems and Control**. Coimbatore, India: IEEE, 2019. p. 536–539. Disponível em: https://ieeexplore.ieee.org/xpl/conhome/9165557/proceeding.

ZHU, Y. L. L. Research on data augmentation for object detection based on x- ray security inspection picture. In: IEEE, 1., 2020, Dalian, China. **IEEE International Conference on Advances in Electrical Engineering and Computer Applications (AEECA)**. Dalian, China: IEEE, 2020. p. 219–222. Disponível em: https://ieeexplore.ieee.org/xpl/conhome/9574040/proceeding.