UNIVERSIDADE FEDERAL DO OESTE DA BAHIA
Centro Multidisciplinar de Bom Jesus da Lapa
Colegiado do Curso de Engenharia Elétrica

GABRIEL ROCHA BARRETO

METODOLOGIAS PARA IMPLEMENTAÇÃO DE ALGORITMOS DE *MACHINE LEARNING* EMBARCADO

Bom Jesus da Lapa–BA
Dezembro de 2023

UNIVERSIDADE FEDERAL DO OESTE DA BAHIA
Centro Multidisciplinar de Bom Jesus da Lapa
Colegiado do Curso de Engenharia Elétrica

Gabriel Rocha Barreto

METODOLOGIAS PARA IMPLEMENTAÇÃO DE ALGORITMOS DE *MACHINE LEARNING* EMBARCADO

Trabalho de Conclusão de Curso apresentado ao Colegiado do Curso de Engenharia Elétrica do Centro Multidisciplinar de Bom Jesus da Lapa da Universidade Federal do Oeste da Bahia, como requisito parcial para obtenção do título de Bacharel em Engenharia Elétrica.

Orientador: Prof. Dr. Manoel Messias Silva Junior

Bom Jesus da Lapa–BA
Dezembro de 2023

FICHA CATALOGRÁFICA

B273 Barreto, Gabriel Rocha

Metodologias para implementação de algoritmos de *Machine Learning* Embarcado. / Gabriel Rocha Barreto. – 2023.

68f.: il.

Orientador: Prof. Dr. Manoel Messias Silva Junior
TCC - Graduação em Engenharia Elétrica, Universidade Federal do Oeste da Bahia. Centro Multidisciplinar de Bom Jesus da Lapa - BA, 2023.

1. Redes Neurais. 2. Engenharia Elétrica. I. Silva Junior, Manoel Messias. II. Universidade Federal do Oeste da Bahia – Centro Multidisciplinar de Bom Jesus da Lapa - BA. III. Título.

CDD 006.32

Biblioteca Universitária de Bom Jesus da Lapa – UFOB

FOLHA DE APROVAÇÃO

GABRIEL ROCHA BARRETO

METODOLOGIAS PARA IMPLEMENTAÇÃO DE ALGORITMOS DE *MACHINE LEARNING* EMBARCADO

Trabalho de Conclusão de Curso foi aprovado como requisito parcial para obtenção do título de Bacharel em Engenharia Elétrica, aprovada em sua forma final pelo Colegiado do Curso de Engenharia Elétrica do Centro Multidisciplinar de Bom Jesus da Lapa da Universidade Federal do Oeste da Bahia.

Bom Jesus da Lapa, 15 de Dezembro de 2022

Prof. Dr. Manoel Messias Silva Junior (UFOB)
(Orientador)

Prof. Ma. Andressa Pereira Oliveira (UFOB)

Prof. Me. Ademário José de Carvalho Neto (UFOB)

Esta monografia é dedicada aos meus pais, pilares da minha formação como ser humano.

Agradecimentos

Agradeço primeiramente a Deus, o Grande Arquiteto do Universo. Sem ele não conseguiria realizar nada.

Agradeço a minha família, em especial aos meus pais Vera lucia Conceição Rocha e Salatiel Rodrigues Barreto, meus maiores incentivos ao longo de toda a trajetória durante a graduação. Por eles, eu persisti para chegar até aqui. Por todo apoio e compreensão durante momentos difíceis.

Aos meus irmãos Diego Rocha Barreto e Danilo Rocha Barreto, meus avós, tios e primos que contribuíram com apoios e incentivos.

Agradeço a toda a UFOB por prover os recursos necessários para minha formação.

A todos os técnicos, estagiários, vigilantes e demais servidores que, direta ou indiretamente contribuíram para a minha formação.

A todos os professores que trocaram experiências e saberes comigo, em especial ao professor Dr. Manoel Messias Silva Junior pela orientação, apoio e confiança, tanto nas disciplinas cursadas, como durante a realização deste trabalho.

"O insucesso é apenas uma oportunidade para recomeçar com mais inteligência."

(Henry Ford)

Resumo

O *Machine Learning* embarcado tem como principal benefício a extração de informações úteis a partir de dados brutos provenientes de sensores e câmeras, sem a necessidade de transmitir esses dados para servidores externos. Isso diminui o consumo de energia e banda, melhora a privacidade e a segurança dos dados, e possibilita uma resposta mais rápida e precisa do dispositivo. Além disso, o *Machine Learning* embarcado permite que o dispositivo se adapte ao seu ambiente e às preferências do usuário, tornando-o mais inteligente e personalizado. Um dispositivo embarcado pode identificar gestos, rostos, objetos, sons, fala, texto, emoções, entre outros, e usar essas informações para interagir com o usuário ou com outros dispositivos. Um exemplo de aplicação é o de uma câmera inteligente que pode reconhecer intrusos, animais ou veículos em uma propriedade e enviar alertas para o proprietário ou para as autoridades. No entanto, essa técnica também apresenta alguns desafios e limitações que devem ser levados em conta na hora de escolher uma plataforma adequada. Um dos principais desafios é o equilíbrio entre complexidade e eficiência do modelo de aprendizado de máquina. Modelos mais complexos podem oferecer uma maior precisão e generalização, mas também demandam mais recursos computacionais, como memória, processamento e energia. Por outro lado, modelos mais simples podem ser mais eficientes, mas também mais suscetíveis a erros e ruídos. Portanto, é necessário encontrar um ponto ótimo entre esses fatores, considerando as características e restrições do dispositivo embarcado. Outro desafio é o processo de treinamento e atualização do modelo de aprendizado de máquina. O treinamento requer uma grande quantidade de dados rotulados e um ambiente adequado para realizar os cálculos necessários. A atualização requer uma forma de enviar o novo modelo para o dispositivo ou permitir que ele aprenda com novos dados. Diante deste contexto, este trabalho tem como objetivo pesquisar as principais ferramentas que podem auxiliar no desenvolvimento de sistemas inteligentes. Como resultado, apresentou-se um exemplo prático baseado na plataforma *Integrated Development Environment*(STM32CubeIDE). Neste exemplo, empregou-se a plataforma STM32F429I DISC1STMicroelectronics para implementar o algoritmo.

Palavras-chave: Redes neurais artificiais, microcontrolador, $STM32F429I - DISC1$, *Machine Learning*

Abstract

Embedded Machine Learning's main benefit is the extraction of information. useful formations from raw data from sensors and cameras, without the need for need to transmit this data to external servers. This reduces consumption of power and bandwidth, improves data privacy and security, and enables faster and more accurate response from the device. Furthermore, Machine Learning embar- cado allows the device to adapt to its environment and user preferences, making it smarter and more personalized. An embedded device can identify gestures, faces, objects, sounds, speech, text, emotions, among others, and use this information to interact with the user or other devices. An example application is the of a smart camera that can recognize intruders, animals or vehicles in a property and send alerts to the owner or authorities. However, this technique also presents some challenges and limitations that must be taken into account when choosing a suitable platform. One of the main challenges is balance between complexity and efficiency of the machine learning model. More models complex systems can offer greater precision and generalization, but they also require more computational resources, such as memory, processing and energy. On the other hand, Simpler models can be more efficient, but also more susceptible to errors and noises. Therefore, it is necessary to find an optimal point between these factors, considering ing the characteristics and restrictions of the embedded device. Another challenge is the process training and updating the machine learning model. The training re- want a large amount of labeled data and a suitable environment to perform the necessary calculations. The update requires a way to send the new model to the device or allow it to learn from new data. Given this context, this This work aims to research the main tools that can assist in development of intelligent systems. As a result, an example was presented practical based on the Integrated Development Environment(STM32CubeIDE) platform. In this example, the STM32F429I DISC1STMicroelectronics platform was used to implement the algorithm.

Keywords: Artificial neural networks, microcontroller,embedded systems, learning machine.

Lista de Figuras

Lista de Tabelas

Lista de quadros

Lista de abreviaturas, acrônimos e siglas

AG	Algoritmos Genéticos
API	*Application Programming Interface*
AWS	*Amazon Web Services*
ADC	Analógico-Digital
ANN	*Artificial Neural Network*
API	*Application Programming Interface*
CPU	*Central Processing Unit*
CNN	*Convolutional Neural Network*
DE	Defeito Externo
DI	Defeito Interno
DFT	*Discrete Fourier Transform*
DWT	*Discrete Wavelet Transform*
DSP	*Digital Signal Processor*
DAC	Digital-Analógico
EP	Eletrônica de Potência
END	Ensaios Não Destrutivos
ELM	*Extreme Learning Machine*
EP	Produto das Eficiências
FN	*False Negative*
FP	*False Positive*
GELM	*Genetic Extreme Learning Machine*
GPU	*Graphics Processing Unit*
IC	Inteligência Computacional
IoT	Internet das Coisas
ML	*Machine Learning*
MLP	*Multilayer Perceptron*
PCA	*Principal Component Analysis*
PDF	*Probability Density Function*
PE	Produto das Eficiências

PEC	*Pulsed Eddy Current*
RAM	*Random Access Memory*
RNA	Rede Neural Artificial
RTOS	*Real Time Operating System*
SVM	*Support Vector Machine*
SIMD	*Single Instruction Multiple Data*
SLFN	*Singlehidden Layer Feedforward Neural Networks*
SD	Sem Defeito
SE	Sistema Embarcado
TM	*Thematic Mapper*
TN	*True Negative*
TP	*True Positive*

Lista de Símbolos

$\in$	Pertence
Hz	Hertz
A	Ampere
V	Volt
W	Watt
mW	Milliwatt
MIPS	Millions of Instructions Per Second
φ	Função de ativação
D	Classificações numéricas
q	Quantidade de características
Ω	Espaço de classes
k	Neurônios
x_m	Sinais de entrada
w_{km}	Pesos sinápticos
u_k	Saída do combinador linear
v	campo local induzido
w_i	Pesos
b_i	Bias
β_i	Pesos da camada de saída‘
$g(x)$	Função de ativação na camada oculta
$H^\dagger$	Matriz de saída
NC_{CPU}	Número de ciclos da CPU do microcontrolador
t_{ex}	Tempo de execução

F_{op}	Frequência da CPU do microcontrolador
NNO	Número de Neurônios Ocultos
NC	Número de Características de Entrada
FP_{SD}	Falso Positivo da Classe Sem Defeito
FN_{SD}	Falso Negativo da Classe Sem Defeito

Sumário

Capítulo 1

INTRODUÇÃO

1.1 Considerações Iniciais

A Inteligência Computacional (IC) é um campo interdisciplinar que engloba técnicas computacionais inspiradas em processos naturais, como redes neurais, Algoritmos Genéticos (AG), lógica *fuzzy*, sistemas imunológicos artificiais, entre outros. A IC visa resolver problemas complexos que não podem ser abordados por métodos tradicionais, como otimização, classificação, reconhecimento de padrões, aprendizado, etc. Esses problemas exigem soluções adaptativas, flexíveis e robustas, que possam lidar com incertezas, ruídos e dinamismo dos dados. A IC oferece ferramentas para modelar e simular fenômenos naturais e artificiais, bem como para projetar e implementar sistemas inteligentes.

As Rede Neural Artificial (RNA) são uma técnica da IC que se inspira no funcionamento do cérebro humano, composto por bilhões de neurônios interconectados. As redes neurais são formadas por unidades de processamento simples chamadas de neurônios artificiais, que recebem entradas, aplicam uma função de ativação e geram saídas. Essas saídas podem ser conectadas a outras entradas de outros neurônios, formando camadas e estruturas complexas. Elas podem aprender a partir de dados, ajustando os pesos das conexões entre os neurônios e geralmente é usada para realizar tarefas como classificação, regressão e agrupamento.

A implementação de uma RNA em plataformas dedicadas como microcontroladores está se expandindo e alcançando diversas áreas (CHEN et al., 2020). RNAs podem ser utilizadas para analisar dados e reconhecer padrões, incorporando inteligência em equipamentos e produtos com a capacidade de reagir ao seu entorno sem que o usuário tenha que fornecer entrada ativamente (REZENDE, 2003). A principal limitação associada a

implementação de sistemas inteligentes em plataformas de *hardware* [1] é a complexidade das RNAs, que exigem grandes capacidades de memória e alto poder de processamento do dispositivo eletrônico utilizado (LOPEZ-MONTIEL et al., 2021).

Sistemas eletrônicos microprocessados vêm sendo propostos para desenvolvimento de equipamentos em diversas áreas: automação industrial, controle, instrumentação, eletrônica de consumo e várias outras (SILVA et al., 2021b). O desenvolvimento de um Sistema Embarcado (SE) caracteriza-se por apresentar algumas restrições que não são encontradas nos sistemas computacionais de propósito geral. Dentre elas pode-se citar (ZURITA, 2011):

1. Processador, as limitações quanto a capacidade de processamento;
2. Memória, o espaço reduzido de memória utilizada para armazenar o programa executável e manipular dados;
3. Consumo de Energia:, a necessidade de redução da potência (mW/MIPS) consumida durante operação;
4. Tempo de vida (*lifetime*), as exigências quanto ao tempo esperado para que o sistema esteja em uso e suas possibilidades de expansão;
5. Confiabilidade, o elevado grau de confiabilidade exigido pelas aplicação embarcadas.

Neste contexto, no presente trabalho foi desenvolvido um estudo de metodologias de implementação de sistemas inteligentes embarcado. Além de apresentar um exemplo de implementação utilizando a plataforma de hardware STM32F4 *discovery*, que dispõe de recursos dos microcontroladores com processadores ARM de alto desempenho, e, biblioteca otimizada para implementação de redes neurais, maximizando o desempenho e minimizando o consumo de memória em núcleos de processadores *Cortex-M*.

1.2 Objetivo Geral

Analisar as vantagens e desvantagens das metodologias, bem como os requisitos e desafios para o desenvolvimento de aplicações de *Machine Learning* (ML) embarcado. A motivação para este estudo surge da crescente demanda por soluções inteligentes que possam ser executadas em dispositivos com recursos limitados, como sensores, câmeras, drones, robôs, entre outros.

[1] *hardware*: É um conjunto de componentes físicos que constitui um determinado sistema.

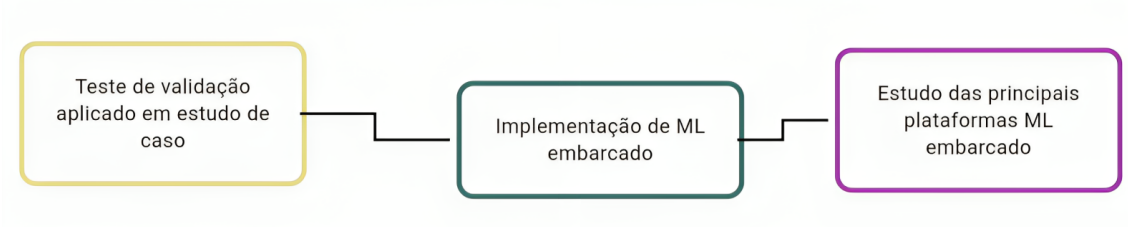
1.3 Objetivos Específicos

1. Levantamento das principais plataformas utilizadas;
2. Avaliação do desempenho das metodologias de implementação de redes neurais combinadas com as plataforma de *hardware* utilizada;
3. Implementação e avaliação de um sistema inteligente embarcado.

1.4 Metodologia

Neste trabalho foi realizado um estudo para implementação de ML embarcada para classificação de dados. A proposta da metodologia utilizada esta ilustrada na Figura 1. Pode-se evidenciar como principais interesses deste projeto:

Figura 1 – Fluxograma da metodologia proposta



Serão apresentadas as principais plataformas disponíveis para implementação de ML embarcada, tais como *TensorFlow Lite*, *PyTorch Mobile*, *Edge Impulse*, Arduino AI, OpenMV, *AWS IoT Greengrass*, TinyML, STM32CubeIDE e CMSIS-NN. Cada uma dessas plataformas será analisada em termos de suas características, vantagens e desvantagens, bem como suas possibilidades de uso em diferentes cenários e domínios. Assim, fornecer uma visão geral e comparativa das plataformas de ML embarcada, destacando seus pontos fortes e fracos, e contribuir para a escolha da plataforma mais adequada para cada projeto.

1.5 Organização da monografia

Capítulo 2 – Apresenta-se as principais referencias bibliográficas utilizadas para o desenvolvimento desta proposta.

Capítulo 3 – Apresenta-se a fundamentação teórica com alguns conceitos sobre Redes Neurais Artificiais, tipos de RNAs e suas arquiteturas. Além disso, apresenta-se também sobre sistemas embarcados e metodologia de implementação embarcada.

Capítulo 4 – Neste capítulo mostra-se o estudo das principais plataformas. Além disso, a é apresentada a metodologia adotada para realização do procedimento experimental e de tratamento das informações também são apresentados.

Capítulo 5 – Aborda-se os resultados sobre o sistema de classificação escolhido para implementação.

Capítulo 6 – São apresentados as considerações finais deste trabalho.

Capítulo 2

REFERENCIAL BIBLIOGRÁFICO

2.1 Revisão Bibliográfica

No Quadro 1 é apresentado as principais bibliografias utilizadas para o desenvolvimento deste trabalho. Destaca-se o trabalho desenvolvido por (SILVA et al., 2021a), o qual apresenta uma implementação em *hardware* da RNA do tipo ELM para classificação de defeitos usando a técnica de ultrassom.

Quadro 1 – Principais referências bibliográficas utilizadas

Autor	**Tipo**	**Objetivo**	**Resultados**
Louzada, Siravenha e Pelaes (2015)	Artigo	Desenvolver uma arquitetura de *Feedforward Neural Network*, utilizando o algoritmo ELM, que possua um conjunto de parâmetros capazes de identificar as características de desmatamento ao longo dos anos e as diferentes feições de uso e cobertura do solo das imagens de sensoriamento remoto adquiridas do satélite *Landsat-5 Thematic Mapper* (TM), através de um mapa de classificação de uso do solo referente ao município de Novo Progresso no período de 1984 à 2011.	Este trabalho mostra a eficiência do Algoritmo ELM aplicado a uma *Singlehidden Layer Feedforward Neural Networks* (SLFN), com o objetivo de identificar as características de desmatamento ao longo dos anos e as diferentes feições de uso e cobertura do solo de imagens do sensor *Landsat-5* TM, referentes a região de Novo Progresso, através de um mapa de classificação de uso do solo gerado pela rede neural.

Continua na próxima página

Guimarães e Fernandes (2020)	Artigo	Mostrar uma estratégia de implementação de uma rede neural do tipo MLP, em um microcontrolador (plataforma de baixo custo e baixo consumo de energia). Para isso, um MLP baseado em matriz modular com o processo de classificação completo foi implementado assim como o treinamento de retropropagação no microcontrolador.	Os resultados das duas análises mostraram que o número de hiperparâmetros apresenta uma relação diretamente proporcional com o tempo de inferência. Além disso, o tamanho do código e a ocupação da memória crescem proporcionalmente ao número de neurônios artificiais, camadas e entradas. Isso dá ao projetista de RNA a chance de compreender melhor os requisitos de memória para um modelo e também quais plataformas de hardware podem se adequar ao aplicativo de MLPs desejado. Por fim, os resultados que apresentam que a latência de inferência e o tamanho do código alcançados com esta proposta de MLPs são compatíveis com os trabalhos de SOTA e que embutir MLPs em μCs [1] é uma opção viável, utilizando dispositivos de baixo custo e baixo consumo de energia.

Continua na próxima página

[1] μCs São classificados como plataformas de *hardware* programáveis que permitem a utilização de SE para aplicações especificas.

TCACENCO (2021)	Artigo	Projetar um sistema para processamento de redes neurais do tipo *feedforward* em ponto fixo (inferência) em que, por meio de um microcontrolador com comunicação com um *host* [2], o usuário irá especificar os hiperparâmetros da rede e enviar os valores de pesos e bias da rede treinada e quantizada.	Analisando os gráficos das funções custos, foi possível notar que, quanto maior o número de camadas escondidas, mais eficiente o algoritmo tende a ser para otimizar o custo em função do número de neurônios em cada camada. Por fim, as redes com 4 e 5 camadas escondidas foram selecionadas, pois resultaram em uma menor função custo e menores alocações de memória no fim da otimização.

Continua na próxima página

[2] *Host* é estrutura para processamento e transmissão de dados.

Silva et al. (2021a)	Artigo	Investigar a viabilidade de embutir máquinas de aprendizado extremo em microcontroladores, a fim de executar treinamento in situ para uma avaliação não destrutiva por ultrassom.	Quatro funções de densidade de probabilidade usadas para gerar pesos aleatórios foram avaliadas, e cada uma tinha a precisão do classificador, o desempenho diminuiu à medida que as variações dos *Probability Density Function* (PDF) aumentaram. Não havia valor óbvio para a constante de regularização da rede C que levou a um melhor desempenho de precisão em qualquer função avaliada. A discretização de pesos aleatórios por codificação de bits fornecida resultados semelhantes aos obtidos usando pesos gerados continuamente ao acaso. Por fim, a implementação no microcontrolador proporcionou desempenhos mais próximos dos casos simulados e dos resultados em trabalhos anteriores na literatura.

Continua na próxima página

Silva et al. (2021b)	Artigo	Propor um sistema eletrônico inteligente para gerar, adquirir e processar sinais de *Pulsed Eddy Current* (PEC) para detecção de corrosão em tubos de aço carbono isolados termicamente com revestimento compósito.	O classificador neural foi avaliado a partir de a matriz de confusão e os Produto das Eficiências (EP). Resultados para estas aplicações indicaram alta eficiência de detecção das classes envolvidas no experimento. O melhor desempenho de classificação foi observado quando foi utilizado o módulo de seleção de características O processamento cadeia DFT + DWT + PCA + MLP apresentou a melhor classificação resultado, mostrando a menor taxa de falsos negativos e erro máximo de 1%. Nenhum defeito e classes de defeitos externos foram identificados sem erro. Outro requisito avaliado foi a média de custo computacional da cadeia de processamento de dados. Os resultados indicam que o uso de DFT + DWT + PCA + MLP apresentou processamento mais rápido.

Continua na próxima página

Silva (2022)	Artigo	Implementação de um sistema de classificação automático, embarcado utilizando uma plataforma de hardware, a qual considera etapas de geração, aquisição e processamento dos sinais de inspeção por corrente parasita pulsada PEC.	Os resultados indicaram alta eficiência de detecção das classes envolvidas no experimento. O melhor desempenho de classificação foi observado quando o módulo de seleção de atributos foi usado. A cadeia de processamento DFT+ DWT+ PCA + MLP apresentou o melhor resultado de classificação, apresentando a menor taxa de falsos negativos, com reduzidas características de entrada, menor número de neurônios ocultos e erro máximo do discriminador de 1%. Enquanto as classes sem defeito e com defeito externo foram identificadas sem erro.

Fim do Quadro

Levando em consideração os trabalhos apresentados pode-se verificar a ampla aplicabilidade de sistemas inteligentes embacados, e, sua crescente utilização nos diversos setores produtivos. Neste contexto, destaca-se a importância de ferramentas dedicadas a implementações de redes neurais em *hardware*.

Capítulo 3

FUNDAMENTAÇÃO TEÓRICA

3.1 Sistemas de Classificação

O ML é uma subárea da IC que se dedica ao estudo e desenvolvimento de algoritmos e sistemas capazes de aprender a partir de dados e experiências, sem necessidade de programação explícita. O ML permite que os sistemas se ajustem automaticamente aos dados e ao contexto, melhorando seu desempenho ao longo do tempo. O ML tem aplicações em diversas áreas, como visão computacional, processamento de linguagem natural, bioinformática, robótica, etc. Alguns exemplos de tarefas realizadas pelo ML são: reconhecimento facial, tradução automática, diagnóstico médico, detecção de fraudes, ou até mesmo recomendação de produtos.

As RNA são cada vez mais postas em problemas de modelagem computacional ou quando o modelo matemático é muito complexo (GHONI et al., 2014). As RNAs são amplamente utilizadas para identificar padrões. Em alguns casos, é usado para resolver problemas de classificação (BOUTABA et al., 2018). No entanto, o desafio de projetar esses sistemas é determinar modelos adequados com base em um conjunto finito de observações de mapeamento de entrada-saída (DAS; BEHERA, 2017).

Normalmente, os sistemas de classificação incluem a atribuição de elementos em um conjunto limitado de classes ou categorias. As redes neurais podem lidar com muitos tipos de situações. As classificações podem ser numéricas como (velocidade, comprimento e temperatura) ou categóricas como (profissão, tipo sanguíneo e nacionalidade). Para as classificações numéricas pode ser definida como:

$$D : \mathbb{R}^q \rightarrow \Omega \tag{1}$$

onde q é quantidade de características do objeto, $\Omega = \omega_1, ..., \omega_c$ o espaço de classes do problema, com c possíveis classes.

Os sistemas de classificação ocuparam uma posição importante em muitos campos. Em Ensaios Não Destrutivos (END), podemos nos concentrar no artigo (TIAN; XU, 2017) que combina algoritmos ELM e *Genetic Extreme Learning Machine* (GELM) para identificar defeitos superficiais em chapas de aço laminadas. O artigo (LALITHAKUMARI; SHEELARANI; VENKATRAMAN, 2012) usa um classificador baseado em RNA para detectar defeitos na soldagem de aço inoxidável em um sistema automatizado, permitindo a identificação de quatro tipos de defeitos. O artigo (ZHANG; WEN; CHEN, 2019) apresenta a detecção de defeitos online baseada na *Artificial Neural Network* (ANN) *Convolutional Neural Network* (CNN) para ligas de alumínio em soldagem a arco robótica usando RNA.

3.1.1 Redes Neurais Artificiais

As RNAs são sistemas computacionais inspirados em redes de neurônios biológicos no encéfalo animal (CHEN et al., 2019). Uma RNA é um sistema composto por unidades conhecidas como neurônios, elementos de processamento interconectados que trabalham em paralelo para obter um bom desempenho em determinada tarefa, é capaz de classificar e fazer previsões sobre um conjunto de dados de entrada, adaptando o processamento de entrada para criar o melhor resultado possível (LOUZADA; SIRAVENHA; PELAES, 2015).

A rede é implementada usando componentes eletrônicos ou simulada em um ambiente computacional programaticamente. Semelhante ao cérebro, o conhecimento é adquirido pela rede de seu ambiente, por meio de um processo de aprendizado e graus de conexão entre os neurônios, chamados de pesos sinápticos (HAYKIN, 2001). A principal área de atividade é a previsão e classificação de padrões que é a área que se refere ao estudo deste trabalho.

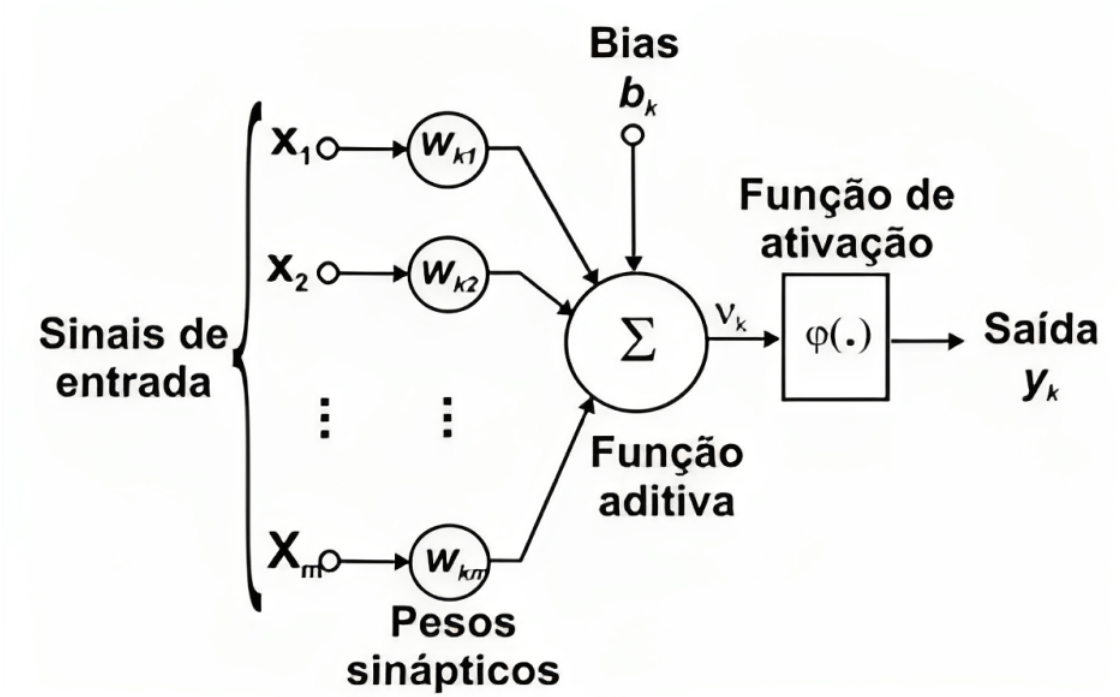
Podemos identificar três componentes básicos de uma RNA na Figura 2 de um modelo de neurônio:

- Um conjunto de sinapses ou associações, definido por seu próprio peso. O sinal na entrada x_j dá sinapse j vinculado ao neurônio k é multiplicado pelo peso sináptico w_{kj}. Enquanto isso, o primeiro índice está relacionado ao neurônio em questão, e o segundo índice relacionado à entrada da sinapse à qual o peso se refere.

- Um somador para adicionar o sinal de entrada, ponderado pelas respectivas sinapses dos neurônios, as operações descritas constituem um combinador linear.

- Uma função de ativação que limita a amplitude da saída do neurônio. Também chamada de função limite, restringi a faixa permitida da amplitude do sinal de saída a um valor finito. A faixa normalizada de saídas dos neurônios é escrita como a faixa de unidade fechada $[0, 1]$.

Figura 2 – Modelo não-linear de um neurônio



Fonte: (HAYKIN, 2001)

O modelo neural também inclui um **bias** imposto externamente, denotado por b_k. O **bias** b_k tem o efeito de aumentar ou diminuir a entrada líquida da função de ativação, dependendo de ser positiva ou negativa. Matematicamente, podemos descrever o neurônio k escrevendo o seguinte par de Equações 2 e 3:

$$u_k = \sum_{j=1}^{m} w_{kj} x_j \tag{2}$$

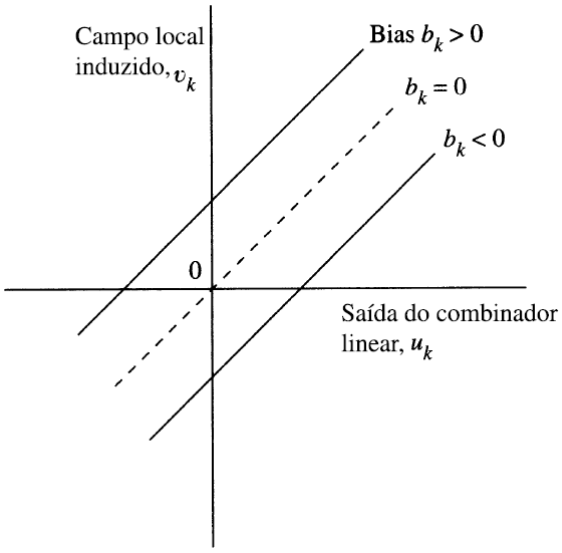
$$y_k = \varphi(u_k + b_k) \tag{3}$$

em que $x_1, x_2, ..., x_m$, são os sinais de entrada; $w_{k1}, w_{k2}, ..., w_{km}$ são os pesos sinápticos k neurônios; u_k é a saída do combinador linear induzida pelo sinal de entrada; b_k é o *bias*; $\varphi(.)$ é a função de ativação e y_k é o sinal de saída do neurônio. O uso de uma polarização b_k tem o efeito de aplicar uma transformação afim à saída u_k do combinador linear, conforme mostrado na Equação 4 e pela Figura 3.

$$v_k = u_k + b_k \tag{4}$$

O *bias* b_k é um parâmetro externo do neurônio artificial k. Dependendo do bias b_k for negativo ou positivo, a relação do campo local induzido ou potencial de ativação v_k do neurônio k e a saída do combinador linear u_k é modificada para a Figura 4 . Com o resultado dessa transformação afim o gráfico v_k em função de u_k não passa mais pela origem.

Figura 3 – Transformação afim produzida pela presença de um bias.



Fonte: (HAYKIN, 2001)

A função de ativação $\varphi(v)$, define a saída de um neurônio em termos do campo local induzido v. São definidos três tipos básicos de funções de ativação:

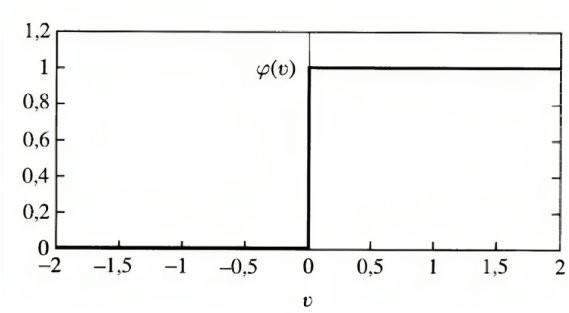
1. Função de Limiar normalmente referida como função de *Heaviside*. Para esse tipo de função de ativação temos as características descritas pela Função 5 e pela Figura 4:

$$\varphi(v) = \begin{cases} 1 & \text{se } v \geq 0 \\ 0 & \text{se } v \leq 0 \end{cases} \tag{5}$$

2. Função linear por partes é vista como a aproximação de um amplificador não-linear. A função linear por partes se reduz à função de limiar, se o fator de amplificação da região linear é feito infinitamente grande, essa característica pode ser vista pela Figura 5.

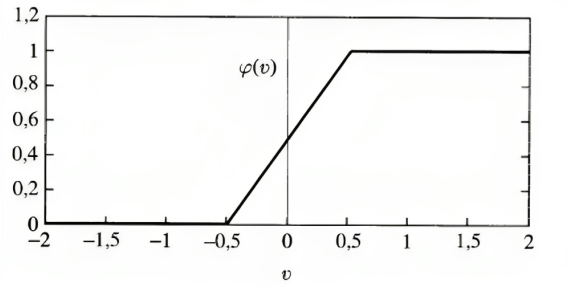
3. Função Sigmóide é de longe a forma mais comum de função de ativação utilizada na construção de RNA. Ela é definida como uma função estritamente crescente que exibe um balanceamento adequado entre comportamento linear e não-linear, dada pela Figura 6 (HAYKIN, 2009).

Figura 4 – Função Limiar.



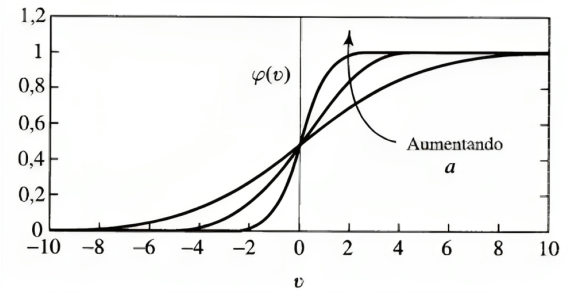
Fonte: (HAYKIN, 2001)

Figura 5 – Função Linear por Partes.



Fonte: (HAYKIN, 2001)

Figura 6 – Função Sigmóide.



Fonte: (HAYKIN, 2001)

3.1.2 Inteligência Computacional e *Machine Learning*

IC e ML são dois campos interdisciplinares que envolvem o estudo e a aplicação de técnicas computacionais inspiradas em sistemas naturais ou artificiais para resolver problemas complexos. Essas técnicas incluem, por exemplo, RNA, AG, lógica *fuzzy*, sistemas multiagentes, entre outras (RUSSELL STUART J E NORVIG, 2016); (MITCHELL, 2019); (HAYKIN, 2009). O tema escolhido para este trabalho é a aplicação dessas técnicas na análise de dados e na geração de conhecimento.

A análise de dados e a geração de conhecimento são processos essenciais para a compreensão e a solução de problemas complexos em diversos domínios. Esses processos

envolvem a coleta, o processamento, a transformação, a visualização e a interpretação de grandes volumes de dados, que podem apresentar diferentes características, como estrutura, formato, qualidade, proveniência e contexto. O objetivo é extrair informações úteis e relevantes dos dados, que possam ser usadas para criar novas hipóteses, teorias ou modelos que expliquem, prevejam ou otimizem fenômenos de interesse. Essas informações e conhecimentos podem ser aplicados para a melhoria da ciência, da tecnologia, da indústria, da educação, da saúde, entre outras áreas (DHAR, 2013); (PROVOST; FAWCETT, 2013); (HASTIE; TIBSHIRANI; FRIEDMAN, 2017).

Para realizar a análise de dados e a geração de conhecimento de forma eficiente e eficaz, é necessário o uso de técnicas avançadas de Inteligência Computacional e ML. Essas técnicas são capazes de lidar com dados complexos, heterogêneos, ruidosos, incompletos ou incertos, que desafiam os métodos tradicionais de análise. Além disso, essas técnicas são capazes de aprender com os dados e se adaptar a mudanças, sem a necessidade de programação explícita ou supervisão humana. Essas técnicas também são capazes de descobrir padrões, relações ou regularidades não triviais ou não esperadas nos dados, que podem revelar novos *insights* ou conhecimentos. E ainda, essas técnicas são capazes de gerar soluções criativas, inovadoras ou surpreendentes para os problemas, que podem superar as expectativas ou as limitações humanas (LECUN; BENGIO; HINTON, 2015); (GOODFELLOW; BENGIO; COURVILLE, 2016); (NG, 2017); (BISHOP, 2006). Essas vantagens podem contribuir para o avanço do conhecimento humano em diversas áreas e para o desenvolvimento de aplicações inteligentes que possam auxiliar na tomada de decisão, na automação de tarefas ou na melhoria da qualidade de vida das pessoas.

3.1.3 Problemas Resolvidos Usando Inteligência Computacional

Baseando-se em modelos que imitam a natureza, os sistemas de regras e a representação da incerteza, ou na cognição humana, a IC é um ramo da IA que busca soluções inexatas ou imprecisas para problemas complexos. Os problemas que podem ser solucionados com IC são de três tipos principais: problemas de classificação, problemas de regressão e problemas de *clustering* [1]. Problemas de classificação consistem em atribuir um rótulo ou uma categoria a uma instância, a partir de um conjunto de atributos ou características. Alguns exemplos são: classificar se um *e-mail* é *spam* ou não, classificar se uma imagem contém um cachorro ou um gato, classificar se um paciente tem uma doença ou não (RUSSELL; NORVIG, 2016).

Uma forma de resolver problemas de classificação é aplicar técnicas de IC como árvores de decisão, redes neurais artificiais, máquinas de vetores de suporte. Problemas

[1] Problemas de *clustering* envolvem a divisão de um conjunto de dados em grupos homogêneos, de acordo com algum critério de similaridade

de regressão consistem em estimar um valor numérico para uma instância, baseado em um conjunto de características ou atributos. Por exemplo, estimar o valor de uma casa estimar o desempenho de um aluno, estimar o consumo de energia de um equipamento (BEZDEK; EHRLICH; FULL, 1984).

Para resolver problemas de regressão, a IC utiliza técnicas como regressão logística, redes neurais artificiais, regressão linear. Problemas de *clustering* consistem em agrupar instâncias parecidas em grupos ou *clusters*, sem ter uma classificação prévia para cada grupo. Por exemplo, agrupar genes com base em suas funções, agrupar documentos com base em seus assuntos, agrupar clientes com base em seus padrões de consumo (ENGELBRECHT, 2007).

A IC também pode aplicar técnicas como *k-means*, algoritmos genéticos, inteligência de enxames para resolver problemas de agrupamento (problemas de *clustering*). O agrupamento é uma técnica versátil e eficaz, que permite identificar grupos e estruturas em conjuntos de dados. Essa técnica consiste em separar os dados em subconjuntos distintos e homogêneos, denominados *clusters*, de modo que os dados dentro de um mesmo *cluster* sejam mais parecidos entre si do que os dados de *clusters* diferentes. Assim, pode ser usado para vários propósitos, como análise exploratória de dados, redução da dimensão dos dados, segmentação de mercado, detecção de anomalias e recomendação personalizada (NEGNEVITSKY, 2011).

3.1.4 Principais Técnicas Usadas Em Inteligência Computacional

Algumas das principais técnicas usadas em inteligência computacional são: redes neurais artificiais, lógica *fuzzy*, algoritmos genéticos, sistemas multiagentes, computação evolutiva, entre outras. Essas técnicas podem ser combinadas ou integradas para formar sistemas híbridos que exploram as vantagens de cada uma delas. Outras técnicas são:

Support Vector Machine (SVM) é uma técnica de aprendizado de máquina que utiliza um hiperplano para separar os dados em classes, buscando maximizar a margem entre elas. É útil para problemas de classificação, regressão e detecção de *outliers*. Um hiperplano é uma superfície que divide um espaço em duas partes. Em um espaço bidimensional, um hiperplano é uma reta. Em um espaço tridimensional, um hiperplano é um plano. Em um espaço de dimensão n, um hiperplano é definido por uma equação linear de n variáveis. Um hiperplano pode ser usado para separar dados que pertencem a diferentes classes, de forma que os dados de uma mesma classe fiquem do mesmo lado do hiperplano.

Uma SVM é um método de aprendizado de máquina que busca encontrar o melhor

hiperplano que separa os dados em classes, maximizando a distância entre o hiperplano e os pontos mais próximos de cada classe. Esse método é aplicado em problemas de classificação, regressão e detecção de *outliers*, pois permite encontrar uma fronteira de decisão robusta e precisa.

AG são técnicas de otimização inspiradas na evolução natural, que utilizam operadores como seleção, cruzamento e mutação para gerar novas soluções a partir de uma população inicial. São úteis para problemas que envolvem espaços de busca complexos e não-lineares.

RNA são estruturas computacionais que simulam o funcionamento dos neurônios biológicos, conectados por sinapses com pesos ajustáveis. São capazes de aprender padrões a partir de dados empíricos, utilizando algoritmos de treinamento como o *backpropagation*. *Backpropagation* é um algoritmo que calcula o erro entre a saída desejada e a saída obtida pela rede neural, e propaga esse erro de forma inversa pelas camadas, ajustando os pesos das sinapses de acordo com uma regra de aprendizado. São úteis para problemas que envolvem reconhecimento de padrões, aproximação de funções e modelagem de sistemas dinâmicos (NEGNEVITSKY, 2011).

Deep Learning é um subcampo do ML que utiliza redes neurais artificiais com múltiplas camadas ocultas, capazes de extrair características abstratas dos dados de forma hierárquica e automática. Enquanto isso, o ML é um campo mais amplo, que engloba outras técnicas além das redes neurais, como os SVMs, os AG e os métodos estatísticos. Nesse sentido, o *deep learning* é mais indicado para problemas que envolvem grandes volumes de dados e alta complexidade, como visão computacional, processamento de linguagem natural e computação cognitiva (**??**).

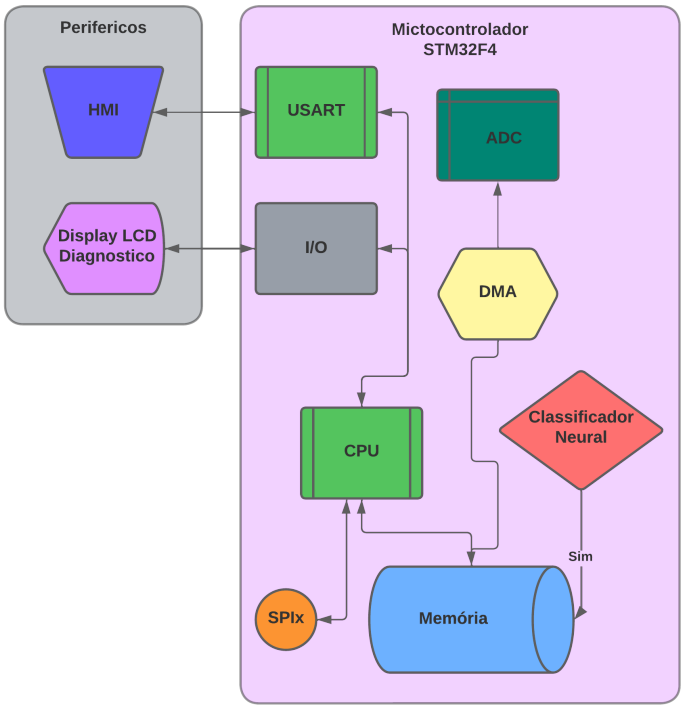
3.2 Sistemas Embarcados

SEs são projetados para realizar uma função ou uma gama de funções e não para serem programados pelo usuário final, como os computadores pessoais. O usuário, pode alterar ou configurar a maneira como o sistema se comporta, porém não pode alterar a função que este realiza. Normalmente interagem com o ambiente em que se encontram, coletando dados de sensores e modificando o ambiente utilizando atuadores. Muitos destes sistemas realizam funções críticas para isso deve ser garantido a estabilidade, recuperação, disponibilidade e a segurança (SANTOS, 2005).

3.2.1 Microcontroladores

Microcontroladores são processadores de *software* [2] que integram muitas funções em um único microchip. Os microcontroladores são projetados especificamente para SEs. Portanto, muitas vezes há um conjunto de formações mais adequado para esse fim, como formações para processamento de bits ou acesso a determinados pinos do processador para facilitar a conexão com dispositivos externos. Na Figura 7 pode ser vista uma configuração genérica de um microcontrolador, na sua estrutura podem incluir uma variedade de dispositivos, como conversores Analógico-Digital (ADC) e Digital-Analógico (DAC), temporizadores, contadores, interfaces seriais, memória de instrução e/ou dados, contadores de interrupção, geradores de relógio, etc. Logo, é comum que não seja desenvolvido um, mas sim uma família de microcontroladores, cada um com um conjunto diferente de recursos, taxa de *clock*, consumo de energia, faixa de temperatura suportada, embalagem e preço suportado por essas instalações. Nesse sentido, o projetista pode escolher o modelo que melhor se adapta às suas exigências tecnológicos e de custo. Além disso, quanto maior a família de microcontroladores, maior a probabilidade de encontrar um processador na mesma família com futuras alterações de design, estendendo assim sua vida útil (BARROS; CAVALCANTE, 2010). Na Figura 8 pode ser visto uma placa de desenvolvimento, em que, o microcontrolador $STM32F429$ é o seu principal atrativo.

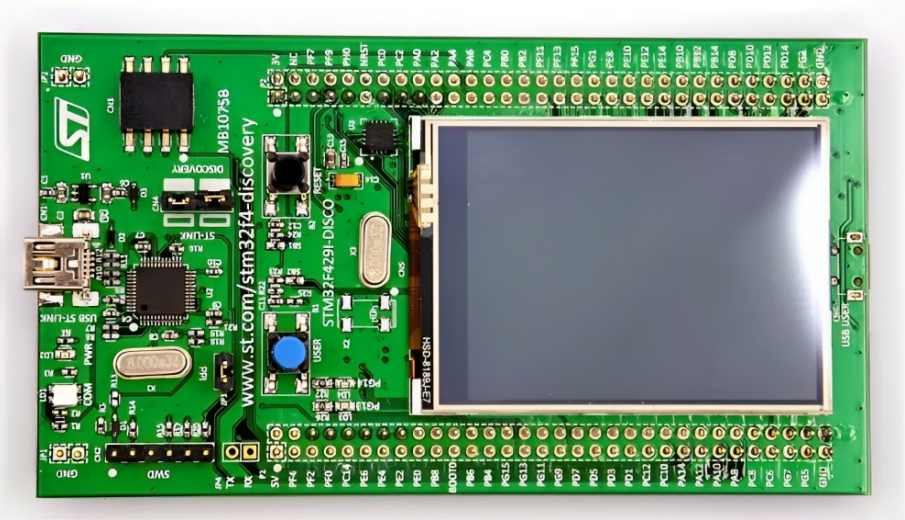
Figura 7 – Diagrama de Bloco Genérico de um Microcontrolador.



Fonte: Produzido pelo autor

[2] *Software é um conjunto de instruções que permitem ao usuário controlar um aparelho eletrônico*

Figura 8 – Microcontrolador STM32F4.



3.2.2 Metodologia de Projeto

Ao projetar um sistema embarcado existem diferentes abordagens. O processo mais completo reúne as seguintes etapas: Especificação, Implementação, Validação e teste. O processo básico pode incluir, como etapa principal, fases de análise, projeto. O estágio de desenvolvimento consiste em um conjunto de atividades que geralmente são executadas por meio de uma abordagem iterativa. Na fase de análise as metas do sistema devem ser delineadas e documentadas. Enquanto a fase de implementação é destinada para construção do sistema e teste de desempenho. As primeiras informações coletadas sobre objetivos e funções está incompleto, porém, permite aos desenvolvedores uma visão inicial da arquitetura do sistema.

Durante o processo de concepção do projeto são necessárias as especificações do sistema (detalhes de recursos de *hardware* e o *software* a ser desenvolvido). Nesta fase, é necessário analisar recursos disponíveis e os limites do projeto (JANAKIRAMAN et al., 2018). Algumas restrições devem ser consideradas para evitar problemas de operação do produto. Fatores como geometria, peso, condições ambientais, restrições térmicas, robustez e custo final precisam ser especificados para o sucesso do projeto. Os algoritmos que processam os eventos é avaliando pelas estimativas a quantidade de memória necessária para armazenamento todos os procedimentos de dados relevantes. Desta forma, pode-se definir o espaço de armazenamento para usar em formação (ESCOTTÁ; BECCARO, 2021). Projetos usando tecnologia embarcada são organizados em tarefas, portanto, recomenda-se começar com a definição de periféricos. Durante a fase de desenvolvimento, realiza-se a protótipos de circuitos, além de programas meticulosamente feito. Antes de montar, recomenda-se o uso de ferramentas computacionais para apoiar a análise de circuitos implementados. Por fim, a etapa de verificação do sistema por meio

de simulação e teste, os protótipos utilizados são o mais próximo dos produtos projetados. Neste caso, algumas especificações do projeto devem ser consideradas:

1. Custo do projeto, uma vez projetado, qualquer outra unidade pode ser construído sem custos adicionais;
2. Desempenho, o tempo de execução do sistema;
3. Potência, a quantidade de eletricidade consumida;
4. Flexibilidade, a capacidade de alterar a função dos sistemas sem custos elevados; harmonia,
5. Tempo de produção do protótipo, o tempo necessário para o desenvolvimento do projeto.

A topologia usada para o projeto de RNA tem uma camada de entrada, uma camada intermediária e uma camada de saída. Os resultados apresentados via matriz de confusão referem-se aos conjuntos de dados associados à fase de teste da ANN.

Capítulo 4

METODOLOGIA

4.1 Plataformas de *Hardware e Software*

A importância de conhecer as configurações do computador para escolher qual plataforma de hardware e software para a implementação de ML Embarcado é fundamental para garantir o sucesso do projeto. Nesse sentido, para implementar o ML Embarcado, é preciso considerar as características do computador que será usado para treinar e testar os modelos, bem como a plataforma que será usada para executá-los no dispositivo final. Algumas das configurações que devem ser levadas em conta são:

1. A capacidade de processamento: o computador deve ter um processador potente e uma memória *Random Access Memory* (RAM) suficiente para lidar com grandes volumes de dados e realizar operações matemáticas complexas. Além disso, é recomendável que o computador tenha uma placa de vídeo *Graphics Processing Unit* (GPU) dedicada, que pode acelerar o treinamento dos modelos usando técnicas como o *Deep Learning*;

2. O sistema operacional: o computador deve ter um sistema operacional compatível com as ferramentas e bibliotecas que serão usadas para desenvolver os algoritmos. Por exemplo, se for usar o TensorFlow, uma das principais plataformas de ML, é preciso ter um sistema operacional baseado em *Linux*, *Windows* ou *Mac OS*. Além disso, é preciso verificar se o sistema operacional suporta a comunicação com o dispositivo embarcado, por meio de portas USB, Bluetooth, Wi-Fi, etc.

Para este projeto foi utilizado um computador pessoal com sistema operacional

Windows de 64*bits*, cuja as especificações são: processador *Intel(R) core(TM)* i3 8130U, *Central Processing Unit* (CPU) de $2,20GHz$, $2,21GHz$ e RAM instalada de $16GB$.

Foi realizado um estudo comparativo entre diferentes plataformas de desenvolvimento para a implementação de ML Embarcado em um projeto de pesquisa. O objetivo era identificar qual plataforma oferecia o melhor desempenho, custo-benefício, facilidade de uso e compatibilidade com as ferramentas e bibliotecas utilizadas. Após uma análise criteriosa, foi escolhida a plataforma STM32CubeIDE, que é um ambiente integrado de desenvolvimento baseado no Eclipse e que suporta os microcontroladores STM32 da STMicroelectronics. Essa plataforma apresentou vantagens como uma interface gráfica intuitiva, uma ampla gama de recursos e funcionalidades, uma boa documentação e suporte técnico, e uma integração com o *TensorFlow Lite for Microcontrollers*, que é uma biblioteca de ML otimizada para dispositivos de baixo consumo e recursos limitados.

A plataforma oferece uma série de bibliotecas para facilitar a implementação de ML Embarcado. Algumas das bibliotecas disponíveis são:

- STM32Cube.AI: uma ferramenta que converte modelos de redes neurais treinados em código C otimizado para rodar no STM32;
- X-CUBE-AI: um pacote de software que contém exemplos de projetos e *drivers* para executar modelos de ML Embarcado no STM32;
- CMSIS-NN: uma coleção de kernels otimizados para operações matemáticas comuns em redes neurais, como convoluções, pooling e ativações;
- *TensorFlow Lite Micro*: uma versão reduzida do *TensorFlow Lite* que permite executar modelos de *Machine Learning* Embarcado em microcontroladores com pouca memória e potência.

Para a validação de *Hardware* e *Software* foi realizado um estudo de caso para avaliar o comportamento e a performasse do Hardware, para isso foi realizado um estudo preliminar dos principais parâmetros a serem utilizados para o projeto da **ANN‘!** (**ANN‘!**) utilizando como base os parâmetros retirados da tese de (SILVA, 2022), onde os resultados experimentais foram apresentados em duas etapas: Avaliação dos sistema de classificação projetado, e, o tempo de execução dos módulos implementado para o tratamento dos sinais mais o classificador.

4.2 Principais Plataformas de implementação de *Machine Learning* Embarcado

Para implementar o ML embarcado, é preciso escolher uma plataforma adequada, que leve em conta as características do dispositivo, como o poder de processamento, a memória, a energia, a conectividade e os sensores disponíveis. Existem diversas plataformas no mercado que oferecem soluções para o desenvolvimento de aplicações de ML embarcado, cada uma com suas vantagens e desvantagens. Neste texto, vamos apresentar as principais plataformas e suas características (SILVA; SANTOS, 2020).

TensorFlow Lite, uma plataforma de código aberto desenvolvida para implementar redes neurais do tipo ML em dispositivos com recursos limitados (BROWNLEE, 2016). Essa plataforma tem seus prós e contras que devem ser analisados antes de optar por essa solução para um projeto. Alguns dos benefícios dessa plataforma é que ela possibilita rodar modelos de ML em dispositivos que não dependem de conexão com a internet ou que demandam baixo consumo de energia, como sensores, dispositivos *wearables* e dispositivos IoT (MARTINS; FERREIRA, 2019).

Wearables são aparelhos eletrônicos que podem ser vestidos no corpo, como *smartwatches*, óculos de realidade aumentada ou *headphones* sem fio. Eles podem capturar dados sobre o usuário, como batimentos cardíacos, localização ou atividade física, e analisá-los usando modelos de ML. Esses modelos podem oferecer informações úteis ao usuário, como orientações de saúde, trajetos de navegação ou tradução de idiomas. Para rodar esses modelos, os *wearables* precisam ter baixo consumo de energia e operar sem depender de uma conexão com a internet. Por isso, eles usam técnicas de ML que possibilitam treinar e inferir os dados localmente, sem enviar para a nuvem. Essas técnicas são denominadas de ML em dispositivos edge, e são uma tendência crescente na área de inteligência artificial (SOUZA; GOMES, 2021).

A Internet das Coisas (IoT) é um conceito que se refere aos dispositivos que podem se conectar à internet ou a outros dispositivos de forma independente, sem a necessidade de intervenção humana. Eles podem gerar, analisar e enviar dados por meio de sensores e atuadores. Alguns dispositivos IoT que podemos citar são as câmeras de vigilância, os termostatos inteligentes, as lâmpadas inteligentes e os relógios inteligentes. Esses dispositivos têm várias aplicações em diferentes áreas, como indústria, agricultura, saúde, transporte, energia e meio ambiente. Eles podem aumentar a eficiência, a produtividade, a qualidade, a segurança e a sustentabilidade dos processos e serviços (DONG; WANG; WANG, 2021). Por exemplo, os dispositivos IoT podem acompanhar as condições meteorológicas e do solo na agricultura, reduzir o consumo de energia em edifícios inteligentes, identificar anomalias e falhas em máquinas industriais, localizar e controlar veículos autô-

nomos, e ajudar no diagnóstico e tratamento de doenças. Para que os dispositivos IoT possam realizar essas aplicações, eles precisam ser capazes de rodar modelos de ML que demandam alto poder computacional e grande quantidade de dados (SILVA, 2023a).

Entre as principais ventagens do *TensorFlow Lite* se destacam:

- Possui uma biblioteca de modelos pré-treinados e otimizados para dispositivos de baixa potência, que podem ser facilmente integrados ao código do microcontrolador;
- Suporta diversas arquiteturas de microcontroladores, como ARM Cortex-M, ESP32, Arduino e RISC-V;
- Fornece ferramentas para converter modelos de *TensorFlow* padrão para o formato *TensorFlow Lite*, reduzindo o tamanho e a complexidade dos modelos;
- Possibilidade de uso de técnicas de quantização, poda e compressão para melhorar o desempenho e a eficiência dos modelos;

Entretanto também apresenta algumas limitações, sendo elas:

- Requer um conhecimento prévio de ML e *TensorFlow* para desenvolver e treinar os modelos;
- Exigir um ajuste fino dos parâmetros e das operações dos modelos para garantir a qualidade e a precisão dos resultados;
- Apresenta limitações de memória e processamento em alguns dispositivos, impedindo o uso de modelos mais complexos ou com maior número de parâmetros;
- Também pode ter uma compatibilidade limitada com algumas bibliotecas e *frameworks* de ML, como *PyTorch*, *Keras* e *Scikit-learn*;
- Apresenta uma comunidade menos abrangentes do que outras plataformas de ML, dificultando o aprendizado e a resolução de problemas.

PyTorch Mobile é uma versão móvel do *PyTorch*, uma biblioteca de ML muito popular. O *PyTorch Mobile* permite transformar modelos treinados em *PyTorch* para um formato adequado para dispositivos móveis, como celulares e *tablets*. Além disso, ele suporta Android e iOS e possibilita a execução de modelos tanto na CPU quanto na GPU. Ele também conta com ferramentas para auxiliar o desenvolvimento e a avaliação de aplicações de ML embarcada, como o *PyTorch Mobile Interpreter*, o *PyTorch Mobile Quantization* e o *PyTorch Mobile Profiler*. *PyTorch Mobile* ainda é compatível com diversas plataformas, como Android, iOS, Linux e Windows, e permite a integração com

outras bibliotecas e *frameworks* [1], como *TensorFlow Lite*, Core ML e ONNX. Outra vantagem é a facilidade de atualização dos modelos em tempo real, sem precisar recompilar ou reinstalar o aplicativo (SILVA, 2021).

Uma forma de usar o *PyTorch* em dispositivos móveis, como Android e iOS, é através do *PyTorch Mobile Interpreter*. Esse é um interpretador *PyTorch* otimizado e portátil que oferece uma forma rápida e fácil de carregar, rodar e liberar modelos *PyTorch*, usando APIs padronizadas e simples. Além disso, ele permite o uso de operações personalizadas, que podem ser criadas e registradas para ampliar as funcionalidades do interpretador (SANTOS, 2023).

PyTorch Mobile Quantization, um método que permite diminuir o tamanho e o uso de memória dos modelos, com pouca perda de qualidade. Ele funciona transformando os pesos e as ativações dos modelos de números reais para números inteiros, com diferentes graus de precisão, como 8 *bits* ou 16 *bits*. Além disso, ele possui diferentes modos de quantização, como estático, dinâmico ou consciente, que podem ser usados em diferentes fases do ciclo de vida do modelo, como treinamento, inferência ou exportação (SILVA, 2023b).

O *PyTorch Mobile Profiler* é uma ferramenta que possibilita examinar o desempenho dos modelos em dispositivos móveis, detectando pontos críticos e possibilidades de aprimoramento. Ele mostra e analisa métricas como tempo de execução, uso de memória, consumo de energia e latência de rede, para cada operação do modelo. Além de possibilitar comparar diferentes versões ou configurações do modelo, como com ou sem quantização, para verificar o efeito das otimizações no desempenho (SILVA, 2023b).

Entre as principais limitações da plataforma *PyTorch Mobile* estão:

- É uma plataforma que ainda está em desenvolvimento e pode apresentar *bugs*, limitações e incompatibilidades com alguns modelos ou funcionalidades do *PyTorch original*;
- Assim como algumas das plataformas apresentadas a *PyTorch Mobile* também requer um conhecimento prévio de *PyTorch* e de programação para dispositivos móveis, o que pode ser um desafio para iniciantes ou usuários menos experientes. Nesse sentido, pode não ser a melhor opção para alguns casos de uso, como modelos que exigem muitos recursos computacionais, precisão ou segurança, ou que dependem de serviços externos ou conexão com a internet.

Edge Impulse é uma plataforma online que facilita a criação, o treinamento e a implantação de modelos de ML embarcada. A plataforma permite trabalhar com vários

[1] *Frameworks* são conjuntos de componentes, ferramentas e bibliotecas que facilitam o desenvolvimento de software. Eles fornecem uma estrutura básica para organizar o código

tipos de dados, como áudio, imagem, vídeo, e com vários tipos de modelos, como redes neurais convolucionais, redes neurais recorrentes e árvores de decisão. O *Edge Impulse* também é compatível com vários dispositivos de baixo consumo, como Arduino, ESP32, *Raspberry* Pi e STM32. O *Edge Impulse* tem uma interface gráfica simples que possibilita realizar todo o processo de ML embarcada sem precisar de programação (SILVA, 2023b).

Entre as principais vantagens da plataforma, podemos citar:

- A facilidade de uso, pois a plataforma oferece uma interface gráfica intuitiva e um conjunto de bibliotecas e exemplos que facilitam o processo de criação, treinamento e teste das redes neurais;
- A portabilidade, pois a plataforma suporta diversos tipos de microcontroladores, como Arduino, STM32, ESP32, entre outros, e permite a integração com diferentes sensores e atuadores;
- A eficiência, pois a plataforma utiliza técnicas de otimização e compressão das redes neurais para reduzir o consumo de memória e energia dos microcontroladores.

Por outro lado, também apresenta algumas limitações, como:

- A dependência da internet, pois a plataforma requer uma conexão com a nuvem para realizar o treinamento e a atualização das redes neurais, o que pode limitar o uso em locais remotos ou com baixa conectividade;
- A limitação de recursos, pois os microcontroladores possuem restrições de capacidade de processamento, armazenamento e comunicação, o que pode impedir a execução de redes neurais mais complexas ou que demandem grande volume de dados;
- A segurança, pois a plataforma envolve o envio de dados sensíveis para a nuvem, o que pode expor os microcontroladores a riscos de invasão ou vazamento de informações.

Já Uma forma de desenvolver aplicações de ML embarcada com placas Arduino é usar a plataforma Arduino AI. Essa plataforma possibilita trabalhar com diferentes tipos de dados, como áudio, imagem e sensores, e com diferentes tipos de modelos, como redes neurais artificiais e máquinas de vetores de suporte. A plataforma também permite treinar os modelos na nuvem ou no próprio dispositivo e implantá-los em placas Arduino compatíveis com o Arduino Nano 33 BLE Sense ou o Arduino Portenta H7 (ARDUINO, 2021).

Essa plataforma tem algumas vantagens e desvantagens que devem ser consideradas antes de escolher usá-la em um projeto.

As principais vantagens da plataforma Arduino AI são:

- Facilidade de uso, a plataforma oferece uma interface gráfica que permite criar, treinar e testar redes neurais de forma simples e intuitiva, sem a necessidade de programar em linguagens de baixo nível ou conhecer os detalhes matemáticos dos algoritmos;
- Portabilidade, a plataforma permite exportar as redes neurais treinadas para diferentes tipos de microcontroladores, como Arduino Nano 33 BLE Sense, Arduino Nano 33 IoT, Arduino MKR WiFi 1010, entre outros, que podem ser integrados em diversos projetos de IoT, robótica, automação, etc;
- Eficiência, a plataforma utiliza técnicas de otimização e compressão para reduzir o tamanho e o consumo de energia das redes neurais, tornando-as adequadas para rodar em dispositivos com recursos limitados.

As principais desvantagens da plataforma Arduino AI são:

- Limitação de recursos, a plataforma ainda está em desenvolvimento e não oferece suporte a todos os tipos de redes neurais, como as convolucionais ou as recorrentes, que são mais complexas e poderosas para lidar com dados visuais ou sequenciais. Além disso, a plataforma depende da disponibilidade e da compatibilidade dos microcontroladores, que podem ter restrições de memória, processamento ou comunicação;
- Dependência de internet, a plataforma requer uma conexão à internet para acessar o serviço de nuvem onde as redes neurais são criadas e treinadas. Isso pode ser um problema em locais remotos ou com baixa qualidade de sinal. Além disso, a plataforma pode ter problemas de segurança ou privacidade ao enviar os dados para o servidor externo;
- Custo, a plataforma é gratuita para uso pessoal ou educacional, mas requer uma assinatura paga para uso comercial ou profissional. O custo pode variar de acordo com o número de projetos, o tamanho das redes neurais e o tempo de uso do serviço.

OpenMV, é uma plataforma que permite criar aplicações de visão computacional embarcada usando câmeras *OpenMV* [2] . Além disso, ela oferece suporte a diversos tipos

[2] Câmeras *OpenMV* são dispositivos que permitem usar visão computacional de forma fácil e prática. Elas combinam o controle de pinos de I/O, com algoritmos de visão computacional de alto nível, para rastrear cores, detectar rostos, reconhecer objetos

de dados, como imagem, vídeo e infravermelho, e a diversos tipos de modelos, como redes neurais convolucionais e algoritmos clássicos de visão computacional. Outro ponto importante da plataforma é a possibilidade de treinar os modelos na nuvem ou no próprio dispositivo e implantá-los em câmeras *OpenMV* compatíveis com o *OpenMV* H7 Plus ou o *OpenMV* H7 RISC-V. Ela consiste em uma placa com um sensor de imagem, uma interface USB e um software de programação baseado em *Python* (OLIVEIRA, 2021).

Algumas das vantagens da plataforma *OpenMV* são:

- Fácil uso e configuração, permitindo que usuários iniciantes possam criar aplicações de visão computacional com poucas linhas de código;
- Compatível com diversos modelos de redes neurais, como *TensorFlow Lite*, *Edge Impulse* e *Keras*, podendo importar e executar modelos treinados em outras plataformas;
- Possui uma biblioteca de funções para processamento de imagens, detecção de objetos, reconhecimento facial, rastreamento de cores, entre outras;
- Baixo custo e um baixo consumo de energia, sendo ideal para projetos embarcados e portáteis.

Algumas das desvantagens da plataforma *OpenMV* são:

- Ela tem uma capacidade limitada de processamento e memória, o que restringe o tamanho e a complexidade dos modelos de redes neurais que podem ser executados;
- Ela depende de uma conexão USB para a programação e o depuração do código, o que pode ser inconveniente em alguns cenários;
- Ela tem uma documentação incompleta e uma comunidade pequena, o que dificulta o acesso a informações e suporte técnico.

Com a plataforma *Amazon Web Services* (AWS) IoT *Greengrass*, é possivel criar soluções de *machine learning* embarcado que utilizam os dispositivos conectados à nuvem da AWS. A plataforma suporta vários formatos de dados, como áudio. Além disso, também, permite que as aplicações de IoT sejam executadas na borda da rede, usando os recursos de processamento e armazenamento dos dispositivos locais. Assim, você pode obter resultados em tempo real sem depender da conexão com a nuvem (OLIVEIRA, 2020).

Essa abordagem tem algumas vantagens e desvantagens, que serão apresentadas a seguir.

As principais vantagens são:

- Redução da latência: as inferências são realizadas localmente, sem a necessidade de enviar os dados para a nuvem e esperar pela resposta. Isso é especialmente importante para aplicações críticas ou sensíveis ao tempo, como controle de processos industriais, monitoramento de saúde ou segurança;

- Economia de banda, os dados são processados na borda, evitando o consumo de largura de banda e custos associados à transmissão para a nuvem. Isso é relevante para aplicações que geram grandes volumes de dados, como visão computacional, reconhecimento de voz ou análise de sensores;

- Privacidade e segurança, os dados são mantidos nos dispositivos locais, reduzindo os riscos de exposição ou vazamento de informações sensíveis ou pessoais. Além disso, a plataforma AWS *IoT Greengrass* oferece recursos de criptografia, autenticação e autorização para proteger as comunicações entre os dispositivos e a nuvem.

As principais desvantagens são:

- Limitação de recursos, os microcontroladores têm capacidade limitada de processamento, memória e energia, o que pode restringir o desempenho e a complexidade das redes neurais. Por isso, é necessário otimizar os modelos de ML para adequá-los aos requisitos dos dispositivos, o que pode exigir mais tempo e conhecimento técnico;

- Dependência da nuvem, apesar de poderem funcionar de forma autônoma, os dispositivos ainda precisam se conectar periodicamente à nuvem para receber atualizações, sincronizar dados ou acessar serviços complementares. Isso pode gerar vulnerabilidades ou indisponibilidades em caso de falhas na rede ou na infraestrutura da nuvem;

- Dificuldade de gerenciamento, a implementação de redes neurais na borda implica em lidar com uma grande quantidade e diversidade de dispositivos, que podem estar distribuídos em diferentes locais ou ambientes. Isso pode aumentar a complexidade e o custo de operação e manutenção dos sistemas de IoT.

TinyML é uma plataforma que habilita a execução de modelos de aprendizado de máquina em dispositivos de baixa potência e baixo custo, como microcontroladores, sensores e *wearables*. Essa tecnologia possibilita o desenvolvimento de aplicações inteligentes que podem interagir com o ambiente, processar dados em tempo real e funcionar de forma autônoma, sem precisar de conexões com a nuvem ou servidores. Alguns exemplos de aplicações de *TinyML* são: reconhecimento de voz, detecção de gestos, classificação de imagens, monitoramento de saúde, rastreamento de objetos, entre outras.

A plataforma também combina técnicas de otimização de modelos, como compressão, quantização e poda, com ferramentas de desenvolvimento e implantação específicas para dispositivos embarcados. Além disso a plataforma utiliza *frameworks* populares de aprendizado de máquina, como *TensorFlow Lite* e *PyTorch*, para facilitar a integração com outras plataformas e ecossistemas. *TinyML* é uma plataforma que representa um progresso na área de inteligência artificial, pois permite levar a inteligência para a borda da rede, reduzindo a latência, o consumo de energia e os custos operacionais das aplicações (SITARA et al., 2020).

Algumas das vantagens da plataforma *TinyML* são:

- Possibilita a criação de aplicações inteligentes que podem funcionar de forma autônoma, sem depender de conexão com a internet ou com servidores em nuvem;
- Reduz os custos e os riscos associados à transmissão e ao armazenamento de dados sensíveis, como informações pessoais, biométricas ou de saúde;
- Aumenta a eficiência e a sustentabilidade dos sistemas embarcados, pois reduz o consumo de energia e os recursos necessários para o processamento de dados;
- Amplia o potencial de inovação e de solução de problemas em diversas áreas, como saúde, agricultura, indústria, educação e entretenimento.

Algumas das desvantagens da plataforma são:

- Exige um alto nível de conhecimento técnico e de programação para o desenvolvimento e a otimização das redes neurais, bem como para a integração com os microcontroladores;
- Apresenta limitações de desempenho e de precisão em relação às redes neurais convencionais, pois depende da capacidade de processamento e de memória dos microcontroladores;
- Requer um cuidado especial com a segurança e a privacidade dos dados, pois os microcontroladores podem ser vulneráveis a ataques cibernéticos ou a interferências físicas;
- Implica em desafios éticos e sociais relacionados ao uso e ao impacto das aplicações inteligentes na sociedade e no meio ambiente.

Além dessas ferramentas, uma importante aliada para implementação de RNA embarcadas são as bibliotecas otimizadas, que como o próprio nome sugere tem como

objetivo otimizar o programa reduzindo o tempo de desenvolvimento, nesse sentido, o programador não precisa implementar todas as operações matemáticas envolvidas nas redes neurais artificiais do zero. Além disso, também Aumenta o desempenho, devido as bibliotecas otimizadas aproveitarem as características específicas do hardware do sistema embarcado, como arquitetura, instruções e periféricos, para acelerar o processamento e reduzir o consumo de energia.

Uma dessas bibliotecas é a CMSIS-NN, onde se trata de é uma coleção de bibliotecas de software otimizadas para a execução de redes neurais do tipo ML em microcontroladores baseados na arquitetura ARM Cortex-M. Essa plataforma oferece algumas vantagens e desvantagens para os desenvolvedores de aplicações de inteligência artificial embarcada. Algumas das principais vantagens são:

- Permite o uso de ferramentas de alto nível, como o *TensorFlow Lite for Microcontrollers*, para treinar e converter modelos de redes neurais para o formato adequado para os microcontroladores;
- Aproveita as instruções *Single Instruction Multiple Data* (SIMD) e *Digital Signal Processor* (DSP) dos microcontroladores ARM Cortex-M, que aceleram o processamento dos dados das redes neurais e reduzem o consumo de energia;
- Oferece uma *Application Programming Interface* (API) simples e consistente, que facilita a integração dos modelos de redes neurais com o código da aplicação e o *Real Time Operating System* (RTOS);
- Suporta uma variedade de operações de redes neurais, como convoluções, *pooling*, ativações, *softmax*, *fully connected*, entre outras, que permitem a implementação de diferentes arquiteturas e funcionalidades.

Algumas das principais desvantagens são:

- Requer que os modelos de redes neurais sejam quantizados, ou seja, que os pesos e os dados sejam representados por números inteiros de 8 bits, em vez de números de ponto flutuante. Isso pode causar uma perda de precisão e acurácia dos modelos, dependendo do tipo e da complexidade da rede neural;
- É limitada aos microcontroladores ARM Cortex-M, que têm recursos limitados de memória e processamento. Isso pode restringir o tamanho e a complexidade dos modelos de redes neurais que podem ser executados nesses dispositivos;
- Dependente da versão do compilador *GNU Compiler Collection* usado para gerar o código binário dos microcontroladores. Algumas versões do GCC podem gerar

código mais eficiente do que outras, afetando o desempenho e o consumo de energia das aplicações.

4.3 Plataforma de Implementação

Ao fornecer um conjunto de condições exigidas, como: poder de processamento, capacidade de memória, *kernels* de tempo real e bibliotecas otimizadas para implementação de sistemas de classificação (CMSIS - NN) (NOVAC et al., 2021) e técnicas de processamento de sinal (CMSIS - DSP). O desempenho do microcontrolador STM32F4 é compatível com os requisitos do sistema implementado, mostrando custo-benefício interessante para a aplicação (SAKR et al., 2020). Outro aspecto importante desta aplicação diz respeito à eficiência de discriminação das categorias envolvidas neste experimento. Portanto, para o processamento da informação, um método para processar sinais PEC com base em um módulo de pré-processamento para melhorar o desempenho do discriminador neural projetado.

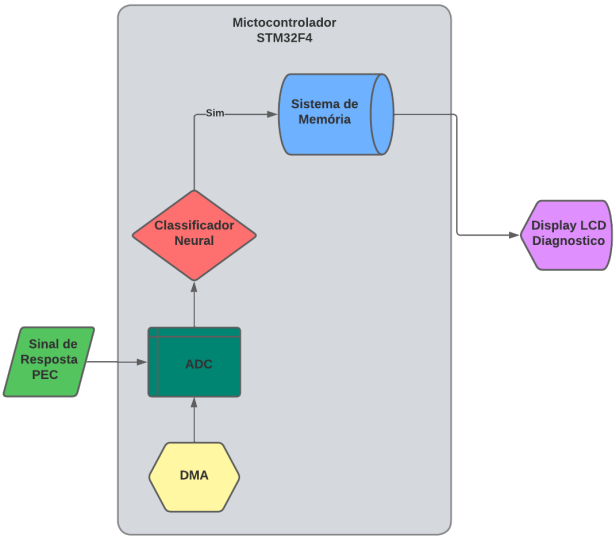
Para este trabalho serão utilizadas bibliotecas otimizadas para implementação de técnicas de processamento de sinais e redes neurais, CMSIS–DSP e CMSIS–NN, onde a biblioteca CMSIS-NN fornece núcleos eficientes projetados para aplicações envolvendo sistemas inteligentes, maximizando o desempenho e reduzindo o consumo de memória nos processadores Arm Cortex-M.

Enquanto a biblioteca CMSIS-DSP inclui recursos como conjunto de funções genéricas de processamento de sinal dedicadas ao Cortex-M, possui mais de 60 funções, além disso as implementações são otimizadas para o conjunto de instruções SIMD que está disponível para Cortex-M4/M7/M33/M35P.

4.4 Montagem Experimental

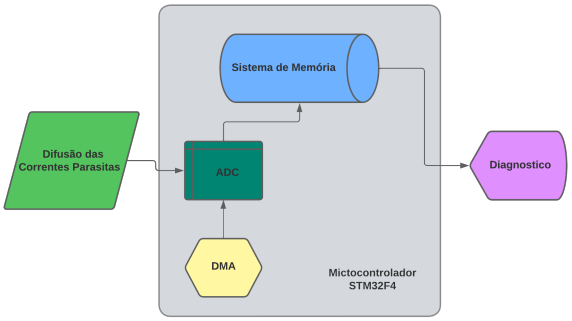
A Figura 9 apresenta o diagrama geral de implementação do sistema de classificação. Para avaliação do classificador projetado deve-se utilizar um conjunto de dados adquirido através da montagem experimental mostrada na Figura 10, em que as informações de aquisição dos sinais estão apresentadas na Tabela 2. Trata-se de um experimento utilizando a técnica PEC aplicada a um tubo de aço carbono com revestimento de material compósito (utilizado na indústria petroquímica), para identificar três regiões: SD, Defeito Interno (DI) e Defeito Externo (DE). Na Figura 11 pode ser vista a seção transversal do tubo de aço carbono revestimento de material compósito enquanto na Tabela 3 são resumidas as características dos defeitos.

Figura 9 – Diagrama do processamento digital



Fonte: Produzido pelo autor

Figura 10 – Ilustração da montagem experimental realizada



Fonte: Produzido pelo autor

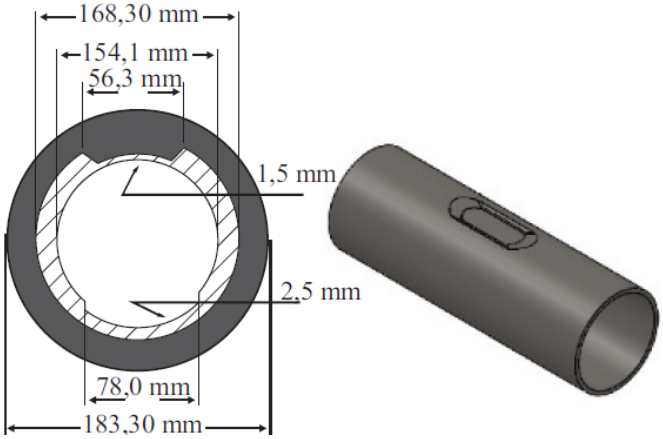
Tabela 2 – Informações sobre aquisição dos sinais PEC

Resolução do ADC	12 bits
Intervalo de amostragem	$5 * 10^{-6} s$
Frequência de Amostragem	200 kHz
Amostras no tempo	200
Exemplos da Classe SD	100
Exemplos da Classe DI	100
Exemplos da Classe DE	100

Tabela 3 – Dimensões dos defeitos em mm (onde DI e DE representam Defeito Interno e Defeito Externo, respectivamente).

Defeito	Comprimento (mm)	Largura (mm)	Espessura (mm)
DI	96,0	78,0	2,5
DE	86,8	56,2	1,5

Figura 11 – Ilustração com as dimensões do corpo de prova da seção transversal e a vista 3D sem o isolamento mostrando o defeito externo.



Fonte: (SILVA, 2022)

4.5 Metodologia Para Avaliação dos Resultados Obtidos

4.5.1 Medidas de Desempenho para Classificadores

Nos sistemas de classificação uma das etapas fundamentais é a fase de avaliação que garante a qualidade da RNA implementada. Nesse sentido, pode-se utilizar diversas formas de avaliar seu desempenho. Entre elas estão a matriz de confusão e o Produto das Eficiências (PE). Uma maneira de medir o desempenho do RNA é calculando uma "matriz de confusão". A Tabela 4 mostra um exemplo dessa matriz onde mostra a da matriz, onde, as classificações corretas são registradas nas células da diagonal principal *True Positive* (TP). As demais classificações expressas nas demais linhas, indicadas pela soma dos elementos de todas as linhas que estão fora da diagonal principal estão expressas as incorretas corresponde aos *False Positive* (FP) da $Classe_i$. Este valor indica o número de vezes que o modelo previu a da $Classe_i$, mas foi sobre a da $Classe_j$. Os Falsos Negativos *False Negative* (FN) coletados da adição de cada coluna j que não está alinhada na diagonal principal indicam a presença de objetos $Classe_i$. Estes indicam que os objetos são classificados como pertencentes a $classe_j$. Além disso, os verdadeiros negativos *True Negative* (TN) resultam da eliminação da linha e coluna $Classe_i$ ao somar os elementos restantes.

A Equação 7 ajuda a calcular a média geométrica das eficiências PE de cada classe. Este valor pode ser encontrado na matriz de confusão; a matriz mostra as probabilidades de detecção de cada classe na diagonal principal, bem como a classificação incorreta nos elementos fora da diagonal. Ao calcular o PE para uma classe, 1 resulta em 100% de

Tabela 4 – Matriz de confusão

		Classe Real			
		$Classe_1$	$Classe_2$	$Classe_3$	**FP**
Classe Predita	$Classe_1$	$n_{1,1}$	$n_{1,2}$	$n_{1,3}$	$(n_{1,2} + n_{1,3})$
	$Classe_2$	$n_{2,1}$	$n_{2,2}$	$n_{2,3}$	$(n_{2,1} + n_{2,3})$
	$Classe_3$	$n_{3,1}$	$n_{3,2}$	$n_{3,3}$	$(n_{3,1} + n_{3,2})$
	FN	$(n_{2,1} + n_{3,1})$	$(n_{1,2} + n_{3,2})$	$(n_{1,3} + n_{2,3})$	

precisão para todas as classes. Qualquer elemento da diagonal principal com valor baixo causa redução de PE. Se algum elemento tiver valor nulo, o produto das eficiências também é zerado independentemente de outros valores.

$$PE = (E_{f1} \times E_{fi} \times ... \times E_{fd})^{\frac{1}{q}} \tag{6}$$

onde Efi é a eficiência de discriminação da classe i (encontrado na diagonal principal da matriz de confusão) e q é o número total de classes.

4.5.2 Validação do Sistema Eletrônico

Para o sistema desenvolvido é possível realizar a estimativa dos tempos de execução de processamento (classificador). O conjunto de dados é carregado no *hardware* e os tempos de processamento são registrados. O tempo de execução t_{ex} é obtido calculando o número de ciclos de CPU (NC_{CPU}) assumindo 1 ciclo de CPU $\sim \frac{1}{F_{op}}$ s, onde F_{op} é a frequência da CPU do microcontrolador (180 MHz). Nesse sentido, podemos calcular o tempo de execução pela Equação 8.

$$t_{ex} = \frac{NC_{CPU}}{F_{op}} \tag{7}$$

Capítulo 5

RESULTADOS

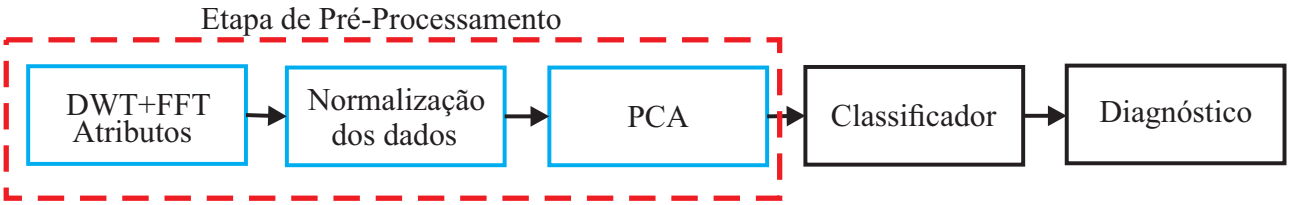
5.1 Considerações Iniciais

O estudo de caso foi realizado utilizando uma das plataformas de implementação pesquisadas. Os testes foram aplicados para um problema de classificação de 3 classes distintas. Neste caso, buscou-se identificar defeitos provocados por corrosão, usando a base de dados disponibilizada na pesquisa de (SILVA et al., 2021b). Logo, os resultados encontrados são apresentados em 2 avaliações: capacidade de discriminação das classes envolvidas no experimento e tempo de execução dos algoritmos implementados.

5.2 Metodologia de Implementação

Para implementação neste trabalho foi considerado a melhor configuração obtida na tese de (SILVA et al., 2021b), conforme Figura 12. Utilizando uma sonda com diâmetro externo de 20 mm, diâmetro interno de 10 mm, altura de 15 mm, diâmetro do fio de 0, 27 mm e 500 voltas, foram capturados sinais de inspeção por correntes parasitas pulsadas de três regiões, sendo duas com defeitos provocados por corrosão DI e DE e SD. A partir das informações coletadas, 70% foram usadas para treinamento, 30% para testes. Os treinamentos, testes e projeto do classificador implementado foram executados em um ambiente *Python*, Windows de 64 bits. Após a etapa de treinamento e testes foi possível gerar arquivos que foram utilizados na implementação em *hardware*. A biblioteca *X-Cube-AI* foi utilizada para importar e converter a RNA pré-treinada em código *Python* para C. Finalmente, o sistema de classificação pode ser implementado no *hardware*. A metodologia de implementação é mostrada na Figura 13.

Figura 12 – Sistema de processamento dos sinais de inspeção.



Fonte: Autoria própria

1. A IDE foi instalada no computador, baixando o instalador do site oficial da ST-Microelectronics e seguindo as instruções de instalação. Um microcontrolador da família ARM M4, no caso um STM32F429, foi conectado ao computador por meio de um cabo USB;

2. Um novo projeto foi criado na IDE selecionando o mesmo modelo do microcontrolador que estava sendo usado. Em seguida, o *clock* do sistema foi configurado, determinando a frequência de operação do microcontrolador e dos periféricos. Essa plataforma era uma das mais intuitivas para essas configurações. Mas se fosse necessário, o assistente gráfico STM32CubeMX poderia ser usado para ajustar os valores dos osciladores, divisores e multiplicadores conforme a necessidade e o microcontrolador. Além disso, também era possível habilitar ou desabilitar os modos de economia de energia, como o *sleep* e o *stop*;

3. Os parâmetros do microcontrolador foram configurados, como os pinos de entrada e saída, as interrupções, os *timers*, os conversores analógico-digitais, os módulos de comunicação e outros recursos. Para facilitar, o assistente STM32CubeMX foi usado novamente para fazer isso de forma intuitiva, arrastando e soltando os componentes na tela e definindo as suas propriedades. Como uma RNA foi testada nesse trabalho, alguns parâmetros dela também foram configurados, como o número de camadas, neurônios, pesos, funções de ativação e algoritmos de aprendizado;

4. Foi implementado a RNA no microcontrolador STM32F429, usando a biblioteca CMSIS-NN, que é uma coleção de funções otimizadas para redes neurais em microcontroladores ARM Cortex-M. Foram usadas as funções da biblioteca para inicializar, treinar e executar a sua RNA, passando os dados de entrada e recebendo os dados de saída. Também foram usadas as funções de depuração para verificar o funcionamento da RNA e corrigir possíveis erros.

Importante observar que neste trabalho apenas foi desenvolvido a metodologia de implementação da rede neural em hardware, etapas de pré-processamentos foram realizadas pela pesquisa (SILVA et al., 2021b).

5.2.1 Sistema de Classificação

Para os testes do sistema de classificação foi utilizado os dados pré-treinados em uma plataforma *Python*, conforme já mencionado. Assim, após essa etapa foi implementado o sistema de classificação no hardware escolhido, de acordo com a metodologia anteriormente apresentada. O número de neurônios na camada de saída da rede neural projetada está relacionado ao número de classes envolvidas, portanto, neste trabalho, são utilizados três neurônios. A saída de destino do classificador treinado é definida como $y_1 = [1, -1, -1]^T$, $y_2 = [-1, 1, -1]^T$ $y_3 = [-1, -1, 1]^T$, correspondendo às classes SD, DI e DE respectivamente. Neste caso, para uma etapa de operação do classificador, o neurônio com o maior valor de saída é associado à classe predita.

A Tabela 5 resume os resultados encontrados durante a fase de treinamento. Nesta configuração, utilizando a arquitetura MLP, observou-se que houve o melhor resultado alcançado foi com $PE(\%) = 99,7 \pm 0,3$ com identificação de todas as assinaturas sem defeito e com defeito externo. Nesta configuração não houve taxas de falso negativo e falso positivo para a classe sem defeito, característica importante para aplicação neste projeto.

Tabela 5 – Matriz de confusão (em %) para as classes sem defeito (SD), com defeito interno (DI) e defeito externo (DE), considerando classificador MLP e ELM alimentado após combinação dos dados.

DFT+DWT+PCA+MLP→ PE(%) = 99,7±0,3		**Classe Real**		
		SD	**DI**	**DE**
Classe detectada pelo classificador	**SD**	100	0	0
	DI	0	99	0
	DE	0	1	100

A Figura 14 apresenta o resumo do produto das eficiências e erros de classificação associados às taxas de falso positivo e falso negativo da classe SD utilizando o classificador MLP.

Neste contexto, buscou-se avaliar os resultados obtidos durante a fase de treinamentos e testes. A cada inspeção realizada foram identificados, previamente, a posição da sonda, conforme apresentado na Figura 15. Foram capturadas 60 assinaturas medidas de cada região inspecionada, em que, para classe DI foram capturados duas amostras em uma mesma posição, enquanto para as outras classes foram tomadas quatro amostras. Os sinais coletados foram submetidos ao sistema de classificação após implementação em hardware, em que foram avaliados:

- Capacidade do sistema implementado em identificar as regiões envolvidas no experimento, além de avaliar os locais de fronteiras entre as classes envolvidas;
- Localização dos erros e acertos dos classificadores projetados.

A Figura 16 apresenta a visão do perímetro do tubo inspecionado, assim como as indicações das predições realizadas pelo classificador. Os testes mostraram que na região de defeito externo ocorreu erro de classificação, em que, identificou-se defeito interno ao invés de externo (conforme saída alvo $y_i = [-1 \;\; 0,98 \;\; 0,89]$), apresentando um erro de 5% na identificação da classe envolvida. Na região de defeito interno ocorreram dois erros de classificação, as amostras testadas foram identificada como sendo pertencente a classe DE (conforme saída alvo $y_i = [-1 \;\; 0,98 \;\; 1]$ e $y_i = [-1 \;\; 0,40 \;\; 1]$).

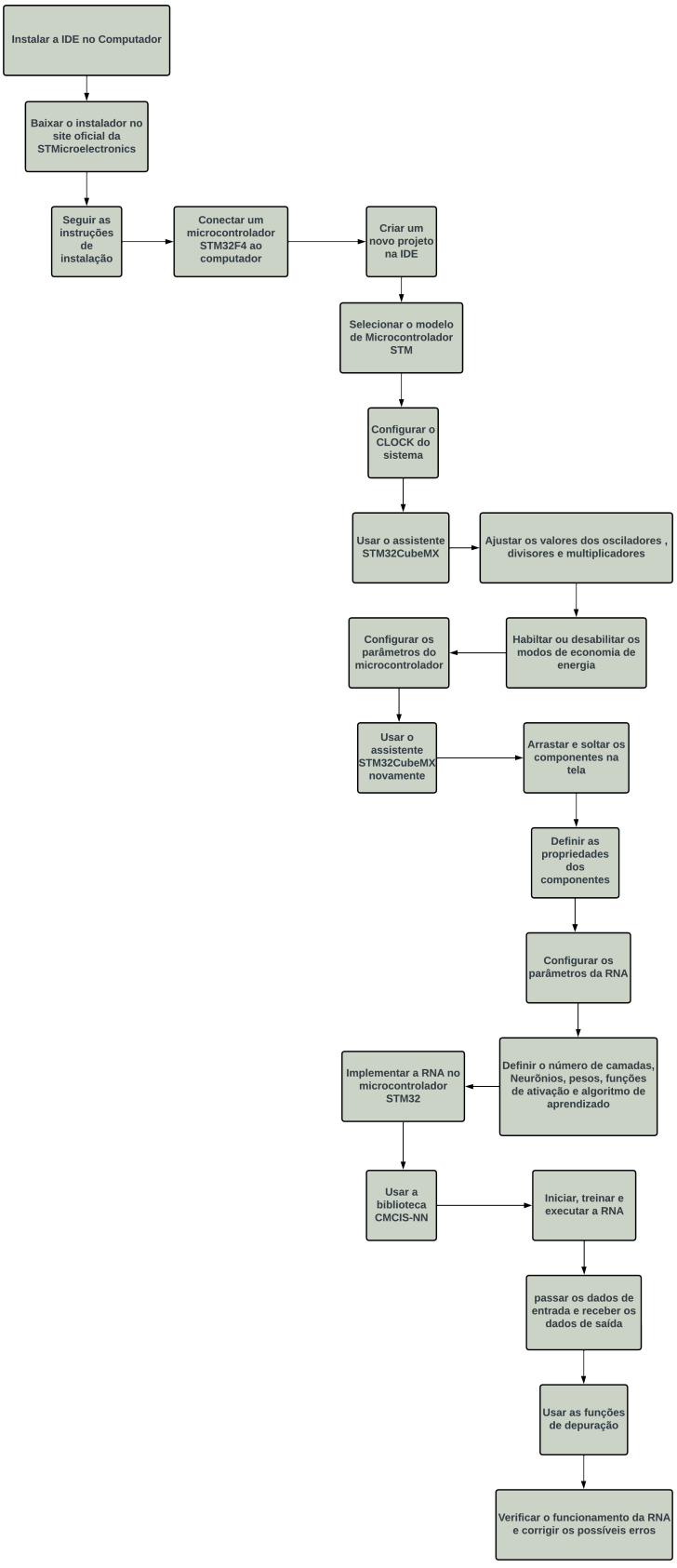
5.3 Testes de Desempenho do *Hardware*

Na Tabela 6 está presente os resultados encontrados do tempo de execução dos algoritmos utilizados. A Figura 17 apresenta a contribuição de cada etapa de processamento quando considerado a execução dos algoritmos no hardware utilizado. Pode-se notar que utilizando a biblioteca otimizada CMSIS-DSP e CMSIS-NN, obteve-se tempos bastante satisfatório.

Tabela 6 – Tempo médio de processamento referente a operação dos classificadores neurais (em milissegundos).

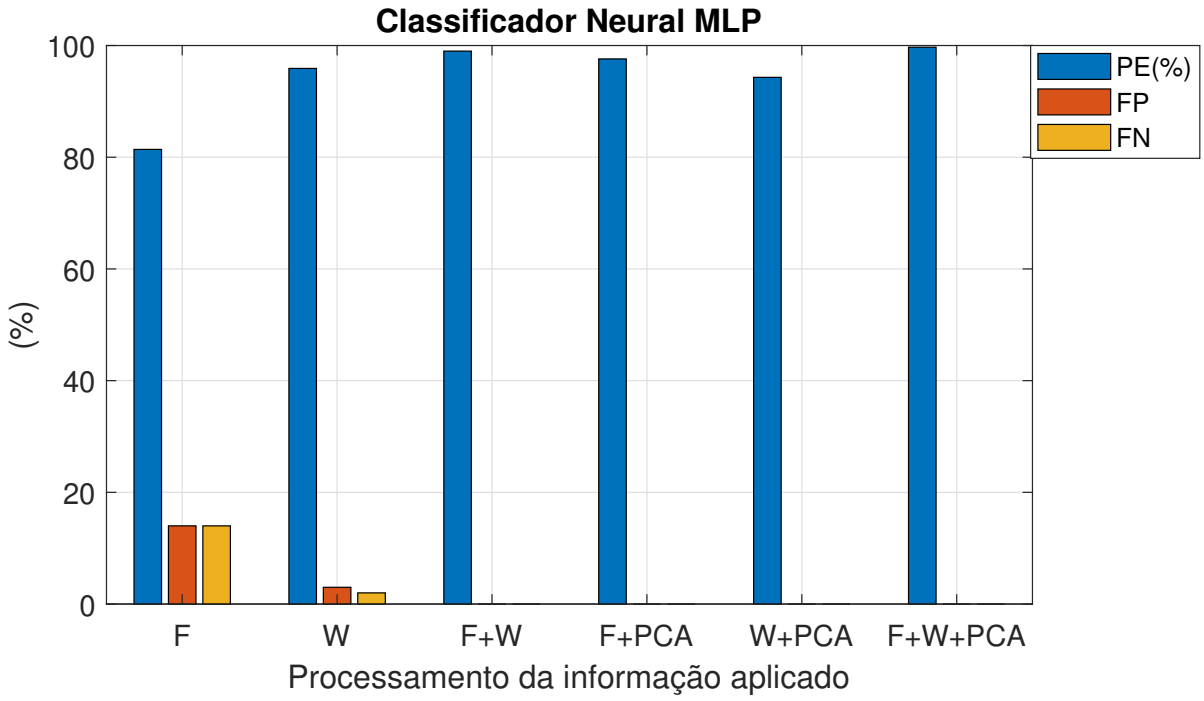
Processamento do Sinal	tempo (ms)
MLP	
FFT! (**FFT!**)+DWT+PCA	32,10± 0,18
FFT!+DWT+PCA (CMSIS)	2,38± 0,01

Figura 13 – Metodologia utilizada.



Fonte: Autoria própria

Figura 14 – Resumo do produto das eficiências e erros de classificação associados às taxas de falso positivo e falso negativo da classe SD utilizando o classificador MLP.



Fonte: Autoria própria

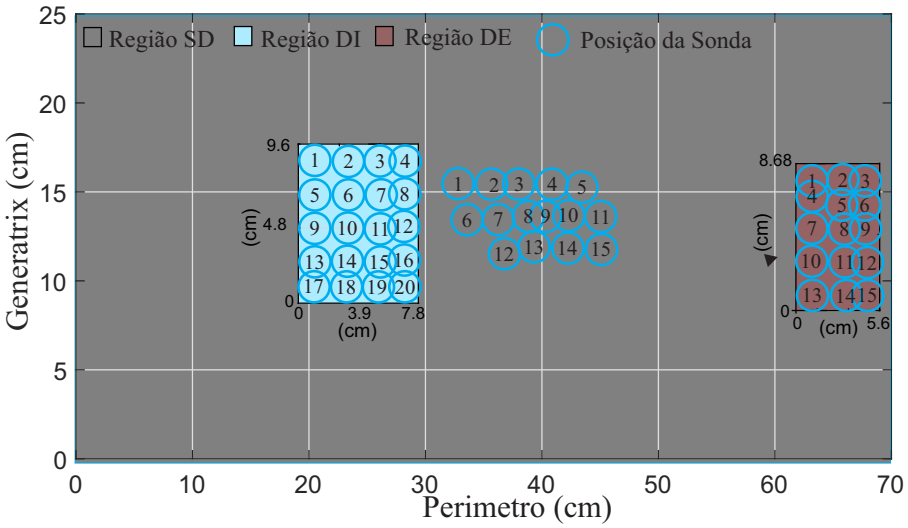


Figura 15 – Visão do perímetro do tubo para indicar as posições da sonda.

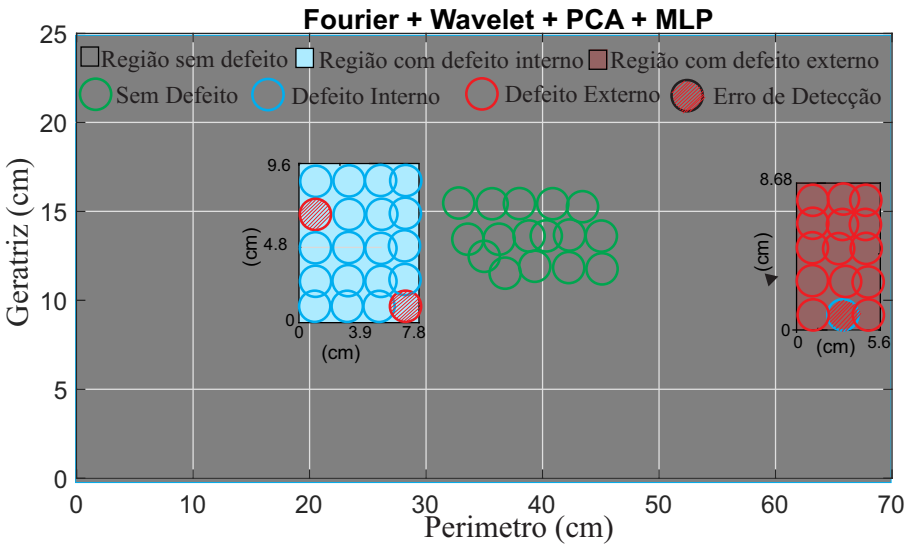


Figura 16 – Visão do perímetro do tubo para identificar as regiões usando a arquitetura MLP.

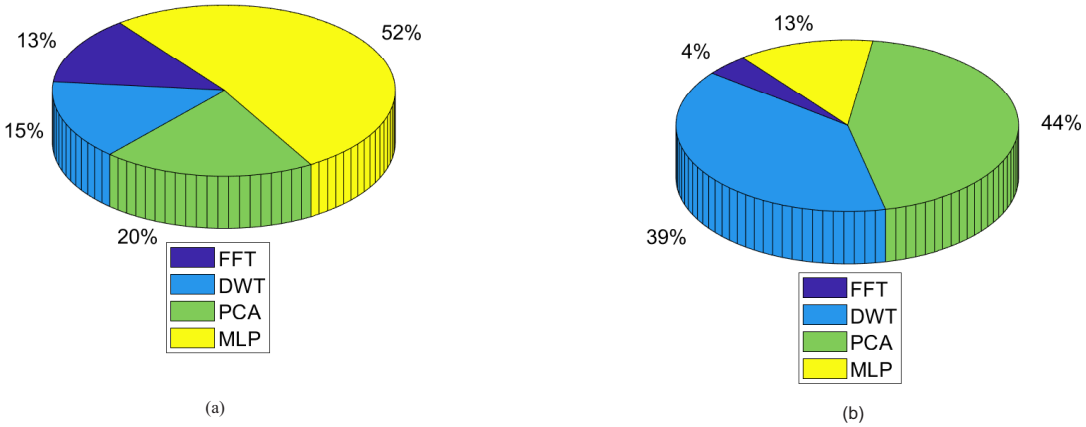


Figura 17 – Contribuição de cada bloco para o tempo total de execução da cadeia de processamento sem usar a CMSIS (a) considerando o processamento dos módulos DFT + DWT + PCA + MLP (b) considerando o processamento dos módulos DFT + DWT + PCA + MLP.

Capítulo 6

CONSIDERAÇÕES FINAIS

Neste trabalho, foi realizada uma análise das metodologias para implementação de algoritmos de ML embarcado, com foco nas redes neurais artificiais. O objetivo geral foi analisar as vantagens e desvantagens das metodologias, bem como os requisitos e desafios para o desenvolvimento de aplicações de ML embarcado. Para isso, foram realizados os levantamentos das principais plataformas utilizadas em seguida a Avaliação do desempenho das metodologias de implementação de redes neurais combinadas com as plataforma de *hardware* utilizada e por fim a Implementação e avaliação de um sistema inteligente embarcado.

A partir do levantamento bibliográfico, foram identificadas as principais plataformas utilizadas para o desenvolvimento de algoritmos de *Machine Learning* embarcado, tais como *TensorFlow Lite for Microcontrollers*, STM32Cube.AI, CMSIS-NN, entre outras. Cada plataforma apresenta suas características, vantagens e limitações, que devem ser consideradas na escolha da melhor opção para cada projeto.

Em seguida, foi realizada uma avaliação do desempenho das metodologias de implementação de redes neurais combinadas com as plataformas de hardwares utilizadas. Foram comparados os resultados obtidos em termos de acurácia, tempo de execução, consumo de memória e energia. Os resultados mostraram que há um *trade-off* entre essas métricas, e que a escolha da melhor metodologia depende dos requisitos e restrições de cada aplicação. A plataforma escolhida para a implementação foi a STM32CubeIDE, que se mostrou vantajosa pela facilidade de uso e pela especialização em microcontroladores da família ARM Cortex-M. A biblioteca XCube-AI foi utilizada para importar e converter a RNA pré-treinada em código *Python* para C.

Por fim, foi implementado e avaliado um sistema inteligente embarcado para a

classificação de defeitos em tubos metálicos usando correntes parasitas pulsadas. O estudo de caso foi baseado na melhor configuração obtida na tese de (SILVA et al., 2021b), que utilizou uma sonda com diâmetro externo de $20mm$, diâmetro interno de $10mm$, altura de $15mm$, diâmetro do fio de $0,27mm$ e $500voltas$. Foram capturados sinais de inspeção por correntes parasitas pulsadas de três regiões, sendo duas com defeitos provocados por corrosão DI e DE e SD. Após a etapa de treinamento e testes foi possível gerar arquivos que foram utilizados na implementação em *hardware*. O sistema de classificação foi implementado no *hardware* STM32F429 e apresentou uma acurácia satisfatória na detecção dos defeitos.

Como contribuições deste trabalho, pode-se destacar:

- A realização de uma revisão sistemática sobre as metodologias para implementação de algoritmos de *Machine Learning* embarcado;
- A proposta de uma metodologia para avaliar o desempenho das plataformas e algoritmos de *Machine Learning* embarcado;
- A avaliação de um sistema inteligente embarcado para a classificação de defeitos em tubos metálicos usando correntes parasitas pulsadas.

Como sugestões para trabalhos futuros, pode-se citar:

- A realização de mais estudos de caso com diferentes aplicações e cenários;
- A comparação entre diferentes plataformas e algoritmos de ML embarcado;
- A otimização dos parâmetros e arquiteturas das redes neurais para melhorar o desempenho do sistema inteligente embarcado.

Referências Bibliográficas

ARDUINO. Arduino ai: ***a platform for embedded machine learning applications***. 2021. Acessado em 04/12/2023.

BARROS, E.; CAVALCANTE, S. **Introdução aos sistemas embarcados**. *Artigo apresentado na Universidade Federal de Pernambuco-UFPE*, p. 36, 2010.

BEZDEK, J. C.; EHRLICH, R.; FULL, W. Fcm: ***the fuzzy c-means clustering algorithm***. *Computers & Geosciences*, Elsevier, v. 10, n. 2-3, p. 191–203, 1984.

BISHOP, C. M. ***pattern recognition and machine learning***. springer, 2006.

BOUTABA, R. et al. ***A comprehensive survey on machine learning for networking: evolution, applications and research opportunities***. *Journal of Internet Services and Applications*, Springer, v. 9, n. 1, 2018. DOI: 10.1186/s13174-018-0087-2.

BROWNLEE, J. ***deep learning with python: develop deep learning models on theano and tensorflow using keras***. Machine Learning Mastery, 2016.

CHEN, Y. et al. ***deep learning on mobile and embedded devices: state-of-the-art, challenges, and future directions***. *ACM Computing Surveys (CSUR)*, ACM New York, NY, USA, v. 53, n. 4, p. 1–37, 2020.

CHEN, Y.-Y. et al. ***Design and implementation of cloud analytics-assisted smart power meters considering advanced artificial intelligence as edge analytics in demand-side management for smart homes***. *Sensors*, MDPI, v. 19, n. 9, p. 2047, 2019.

DAS, K.; BEHERA, R. N. ***A survey on machine learning: concept, algorithms and applications***. *International Journal of Innovative Research in Computer and Communication Engineering*, v. 5, n. 2, p. 1301–1309, 2017. DOI: 10.15680/IJIRCCE.2017.0502001.

DHAR, V. ***data science and prediction***. *Communications of the ACM*, ACM New York, NY, USA, v. 56, n. 12, p. 64–73, 2013.

DONG, B.; WANG, Z. L.; WANG, S. ***technology evolution from self-powered sensors to aiot enabled smart homes***. *Nano Energy*, Elsevier, v. 79, p. 105414, 2021.

ENGELBRECHT, A. P. ***computational intelligence: an introduction***. John Wiley & Sons, 2007.

ESCOTTÁ, A.; BECCARO, W. **Controle Automático de Volume em Tempo Real Utilizando Inferência Fuzzy em um Sistema Embarcado**. *Trends in Computational and Applied Mathematics*, SciELO Brasil, v. 22, p. 41–60, 2021.

GHONI, R. et al. ***Defect characterization based on eddy current technique: Technical review***. *Advances in Mechanical Engineering*, SAGE Publications Sage UK: London, England, v. 6, p. 182496, 2014. DOI: 10.1155/2014/182496.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. ***deep learning***. MIT press, 2016.

GUIMARÃES, C. J. B. V.; FERNANDES, M. A. C. ***Real-time Neural Networks Implementation Proposal for Microcontrollers***. *Electronics*, MDPI AG, v. 9, n. 10, p. 1597, sep 2020. DOI: 10.3390/electronics9101597.

HASTIE, T.; TIBSHIRANI, R.; FRIEDMAN, J. ***springer series in statistics: the elements of statistical learning: data mining, inference, and prediction***. Springer, 2017.

HAYKIN, S. **Redes neurais: princípios e prática**. *Bookman Editora*, 2001.

HAYKIN, S. **Redes neurais e máquinas de aprendizado, 3/E**. *Pearson Education India*, 2009.

JANAKIRAMAN, S. et al. ***Lightweight chaotic image encryption algorithm for real-time embedded system: Implementation and analysis on 32-bit microcontroller***. *Microprocessors and Microsystems*, v. 56, p. 1–12, 2018. ISSN 0141-9331. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0141933117302478>.

LALITHAKUMARI, S.; SHEELARANI, B.; VENKATRAMAN, B. ***Artificial neural network based defect detection of welds in tofd technique***. *International Journal of Computer Applications*, Foundation of Computer Science, v. 41, n. 20, 2012. DOI: 10.5120/5808-8069.

LECUN, Y.; BENGIO, Y.; HINTON, G. ***deep learning***. *Nature*, v. 521, n. 7553, p. 436–444, 2015.

LOPEZ-MONTIEL, M. et al. ***evaluation method of deep learning-based embedded systems for traffic sign detection***. *IEEE Access*, IEEE, v. 9, p. 101217–101238, 2021.

LOUZADA, H. A.; SIRAVENHA, A. C. Q.; PELAES, E. G. **Utilização de Redes Neurais do tipo *Extreme Learning Machine* na classificação da cobertura de solo do Município de Novo Progresso-PA**. 2015.

MARTINS, R.; FERREIRA, A. ***tensorflow lite*: uma plataforma para implementação de redes neurais em dispositivos com recursos limitados**. *Revista de Informática Teórica e Aplicada*, v. 26, n. 3, p. 23–38, 2019.

MITCHELL, M. ***complexity: a guided tour***. Oxford University Press, 2019.

NEGNEVITSKY, M. ***artificial intelligence: a guide to intelligent systems***. Pearson Education, 2011.

NG, A. ***machine learning yearning: technical strategy for ai engineers, in the era of deep learning***. 2017.

NOVAC, P.-E. et al. ***Quantization and deployment of deep neural networks on microcontrollers***. *Sensors*, Multidisciplinary Digital Publishing Institute, v. 21, n. 9, p. 2984, 2021.

OLIVEIRA, L. **OpenMV: Uma plataforma para visão computacional embarcada**. *Revista Brasileira de Computação Aplicada*, Universidade de Passo Fundo, v. 13, n. 3, p. 1–10, 2021.

OLIVEIRA, R. ***amazon web services (aws) iot greengrass*: uma plataforma para *machine learning* embarcada**. *Revista de Tecnologia da Informação*, v. 10, n. 2, p. 15–25, 2020.

PROVOST, F.; FAWCETT, T. ***data science for business: what you need to know about data mining and data-analytic thinking***. O'Reilly Media, Inc., 2013.

REZENDE, S. O. **Sistemas inteligentes: fundamentos e aplicações**. *Editora Manole Ltda*, 2003.

RUSSELL, S. J.; NORVIG, P. ***inteligência artificial***. Elsevier Brasil, 2016.

RUSSELL STUART J E NORVIG, P. ***Inteligência Artificial: uma abordagem moderna. Malásia***. *Pearson Education Limited Londres, Reino Unido*, 2016.

SAKR, F. et al. ***Machine learning on mainstream microcontrollers***. *Sensors*, MDPI, v. 20, n. 9, p. 2638, 2020.

SANTOS, D. M. **Projeto de sistemas embarcados: Um estudo de caso baseado em microcontrolador e seguindo AOSD**. 2005.

SANTOS, J. ***pytorch mobile interpreter*: uma ferramenta para execução de modelos pytorch em dispositivos móveis**. *Revista Brasileira de Computação Móvel*, v. 12, n. 1, p. 23–35, 2023.

SILVA, J. ***pytorch mobile*: uma biblioteca de aprendizado de máquina para dispositivos móveis**. *Revista Brasileira de Computação Móvel*, v. 15, n. 2, p. 23–35, 2021.

SILVA, J. **Dispositivos IoT e modelos de ML: desafios e oportunidades**. *Revista Brasileira de Computação*, v. 23, n. 1, p. 1–10, 2023.

SILVA, J. ***pytorch mobile quantization*: uma técnica para reduzir o tamanho e o consumo de memória dos modelos de aprendizado profundo**. *Revista Brasileira de Inteligência Artificial*, v. 1, n. 1, p. 10–15, 2023.

SILVA, J.; SANTOS, M. **Plataformas para o desenvolvimento de aplicações de *machine learning* embarcado**. *Revista Brasileira de Computação Embarcada*, v. 1, n. 2, p. 15–25, 2020.

SILVA, L. C. et al. ***Embedded decision support system for ultrasound nondestructive evaluation based on extreme learning machines***. *Computers Electrical Engineering*, v. 90, p. 106891, 2021. ISSN 0045-7906. DOI: 10.1016/j.compeleceng.2020.106891.

SILVA, M. M. et al. ***Intelligent embedded system for decision support in pulsed eddy current corrosion detection using Extreme Learning Machine***. *Measurement*, v. 185, p. 110069, 2021. ISSN 0263-2241. DOI: 10.1016/j.measurement.2021.110069.

SILVA, M. M. J. **Sistema Embarcado Inteligente para Auxílio ao Diagnóstico em Inspeções por Correntes Parasitas Pulsadas**. 178 f. Tese (Doutorado em Engenharia Elétrica) — Universidade Federal da Bahia, Salvador - BA, 2022.

SITARA, P. et al. ***tinyml: machine learning with tensorflow lite on arduino and ultra-low-power microcontrollers***. *O'Reilly Media*, 2020.

SOUZA, J.; GOMES, M. ***Wearables*: dispositivos inteligentes que usam ML em dispositivos edge**. *Revista Brasileira de Inteligência Artificial*, v. 4, n. 2, p. 37–45, 2021.

TCACENCO, F. d. S. A. **Otimização de rede neural em ARM32**. 77 f. Monografia (Bacharelado em Engenharia Física) — UFRGS, Porto Alegre, RS, 2021.

TIAN, S.; XU, K. ***An algorithm for surface defect identification of steel plates based on genetic algorithm and extreme learning machine***. *Metals*, MDPI, v. 7, n. 8, p. 311, 2017. DOI: 10.3390/met7080311.

ZHANG, Z.; WEN, G.; CHEN, S. ***Weld image deep learning-based on-line defects detection using convolutional neural networks for Al alloy in robotic arc welding***. *Journal of Manufacturing Processes*, Elsevier, v. 45, p. 208–216, 2019. DOI: 110.1016/j.jmapro.2019.06.023.

ZURITA, M. E. **Projeto de sistemas embarcados**. Universidade Federal do Piauí, Curso de Engenharia Elétrica, Campus Universitário Ministro Petrônio Portela, 2011.