# INTELIGÊNCIA ARTIFICIAL EMBARCADA: EXPERIMENTOS COM UTILIZAÇÃO DE KNN EM FPGAs

**Luís Otávio Lopes Amorim**
Engenharia Eletrônica
Instituto Federal de São Paulo – IFSP, São Paulo, SP, Brasil.
amorim.luis@aluno.ifsp.edu.br

**Gustavo Senzaki Lucente**
Engenharia Eletrônica
Instituto Federal de São Paulo – IFSP, São Paulo, SP, Brasil.
gustavo.senzaki@aluno.ifsp.edu.br

**Ricardo Pires**
Departamento de Elétrica
Instituto Federal de São Paulo – IFSP, São Paulo, SP, Brasil.
ricardo_pires@ifsp.edu.br

**Resumo**

Este artigo explora a viabilidade de implementar o algoritmo K-Vizinhos Mais Próximos (KNN) em circuitos integrados do tipo FPGA. A metodologia adotada compreendeu o desenvolvimento inicial do algoritmo em linguagem Python e C, seguido pela implementação em linguagem VHDL para hardware FPGA. Foram realizados experimentos utilizando os conjuntos de dados Iris, Wine e Digitos, com variações nos parâmetros para avaliar o desempenho e a precisão. A implementação do KNN em FPGA, embora tenha demonstrado uma acurácia ligeiramente inferior à das versões em software, confirmou a viabilidade e a eficiência desta abordagem. A diferença na acurácia é atribuída à utilização de números de ponto fixo, que oferecem precisão limitada em comparação aos números de ponto flutuante usados em software. A análise do uso dos recursos dos sistemas sintetizados mostrou que a implementação em FPGA é viável e eficiente, sugerindo que dispositivos de hardware programável são uma opção promissora para o uso de algoritmos de aprendizado de máquina. Esses resultados indicam um caminho claro para futuras otimizações e a potencial implementação de algoritmos de aprendizado de máquina mais complexos em FPGAs, expandindo o escopo de aplicações avançadas.

**Palavras-chave:** Aprendizagem de Máquina; Inteligência Artificial; Sistemas Embarcados; FPGA

# EMBEDDED ARTIFICIAL INTELLIGENCE: EXPERIMENTATION WITH KNN IN FPGAs

**Abstract**

This article explores the feasibility of implementing the K-Nearest Neighbors (KNN) algorithm on FPGA type integrated circuits. The methodology involved the initial development of the algorithm in Python and C languages, followed by implementation in VHDL language for FPGA hardware. Experiments were conducted using the Iris, Wine, and Digits datasets, with variations in parameters to assess performance and accuracy. The FPGA implementation of KNN, although showing slightly lower accuracy compared to the software versions, confirmed the feasibility and efficiency of this approach. The accuracy difference is attributed to the use of fixed-point numbers, which offer limited precision compared to floating-point numbers used in software. The analysis of resource utilization of the synthesized systems demonstrated that the FPGA implementation is viable and efficient, suggesting that programmable hardware devices are a promising option for machine learning algorithms. These results indicate a clear path for future optimizations and the potential implementation of more complex machine learning algorithms on FPGAs, expanding the scope of advanced applications.

**Keywords:** Machine Learning; Artificial Intelligence; Embedded Systems; FPGA

## 1. INTRODUÇÃO

A automação industrial é um fenômeno que perpassa os séculos, desde os primórdios da Primeira Revolução Industrial até os avanços contemporâneos da Indústria 4.0. Iniciada com a introdução de máquinas na produção no século XVIII, esse processo tem sido marcado por uma busca constante por eficiência e inovação. Com a evolução tecnológica, testemunha-se uma sucessão de desenvolvimentos tecnológicos que transformaram radicalmente os processos de produção (Sakurai; Zuchi, 2018). Na era da indústria 4.0, ou 4ª Revolução Industrial, a automação é impulsionada pela integração de tecnologias como inteligência artificial, *big data* e Internet das Coisas (IoT) (Schwab, 2018), prometendo não apenas aumentar a eficiência, mas também revolucionar a forma como as fábricas operam e se relacionam com o mundo ao seu redor (Ghobakhloo, 2020).

Dispositivos programáveis desempenham um papel fundamental na automação, permitindo a criação de sistemas flexíveis e adaptáveis às necessidades específicas de produção. Os controladores lógicos programáveis (CLPs), por exemplo, são amplamente empregados para controlar e coordenar processos industriais complexos. Esses dispositivos combinam hardware e software para executar tarefas de controle, como monitoramento de

sensores, acionamento de atuadores e tomada de decisões em tempo real. Além dos CLPs, há também outros dispositivos inteligentes, como computadores industriais e sistemas de supervisão e controle (SCADA), que desempenham papéis essenciais na automação. A capacidade de programação desses dispositivos permite a personalização e otimização contínua dos processos industriais, contribuindo para aumentar a eficiência, reduzir custos e melhorar a qualidade dos produtos (Frey; Litz, 2002; Alphonsus; Abdullah, 2016).

Ainda assim, o elevado custo de dispositivos de automação industrial torna-se muitas vezes um fator impeditivo na adoção de tais práticas. Nesse contexto, entra a busca por outras soluções como opções de código aberto ou até mesmo utilizando outros tipos de equipamentos, como dispositivos lógicos configuráveis e microcontroladores (Alves *et al.*, 2014; Andina *et al.* 2015; Chen *et al.*, 2018).

A inteligência artificial (IA) desempenha um papel importante na automação industrial, proporcionando avanços significativos em eficiência, precisão e tomada de decisões. Por meio da IA, os sistemas de automação podem aprender com dados históricos e em tempo real, identificando padrões, otimizando processos e antecipando falhas antes mesmo de ocorrerem. Exemplos de aplicação da IA na indústria incluem a manutenção preditiva, que utiliza algoritmos para prever falhas em equipamentos, reduzindo custos com paradas não programadas (Lee *et al.*, 2019; Dalzochio *et al.*, 2020; Zonta *et al.*, 2020); o controle de estoque, que utiliza análise de dados para otimizar níveis de inventário e reduzir desperdícios (Giannoccaro; Pontrandolfo, 2002; Bertsimas *et al.*, 2016; Pournader *et al.*, 2021); e o *design* generativo, que utiliza algoritmos para criar *designs* otimizados e eficientes. A robótica é outra área onde a IA é essencial, permitindo o desenvolvimento de robôs autônomos capazes de realizar tarefas complexas com precisão e eficiência (Murphy, 2019; Yueh; Chiang, 2020). Em suma, a IA está redefinindo a automação industrial, capacitando as empresas a alcançarem níveis inéditos de produtividade e competitividade.

O ramo industrial não é o único a se beneficiar de tal ferramenta. Nos dias atuais, esses algoritmos estão presentes no dia a dia de sistemas médicos no auxílio de diagnósticos e prevenção de doenças (Hamet; Tremblay, 2017), automação residencial (Cardozo, 2021), cibersegurança (Li, 2018), entre outros.

O campo do Aprendizado de Máquina (ML) é um ramo dentro da área de IA que está em grande evolução (Shavlik; Dietterich, 1990). De acordo com Mohri *et al.* (2018) o aprendizado de máquina é o uso de algoritmos computacionais que, a partir de um conjunto

de exemplos conhecidos, criam soluções gerais para um problema, buscando, assim, realizar previsões. Por ser uma definição ampla, trata-se de uma área com diversos objetivos, como regressão, classificação e identificação de agrupamentos, métodos como aprendizagem por reforço, aprendizado supervisionado e não supervisionado, e algoritmos como identificação de vizinhos mais próximos, algoritmos baseados em árvores, aprendizagem profunda, métodos estatísticos, entre outros (Mohri *et al*., 2018; Sarker, 2021).

Os sistemas embarcados também desempenham um papel crucial nessa revolução, especialmente com o surgimento do TinyML, uma abordagem de ML voltada para dispositivos com recursos limitados (Ray, 2022). Os sistemas embarcados habilitados para TinyML permitem a execução de modelos de ML diretamente em microcontroladores e microprocessadores de baixo consumo de energia, com processamento realizado localmente, possibilitando a implementação de recursos inteligentes em dispositivos IoT e equipamentos industriais (Dutta; Bharali, 2021).

Além da possibilidade de utilizar microcontroladores e microprocessadores para embarcar sistemas que utilizam IA, é possível fazer isso pelo uso de circuitos integrados configuráveis chamados FPGAs. Portanto, além de colocar o processamento pesado em dispositivos mais baratos (Parnell; Brayner, 2004), também utiliza-se do grande poder de computação paralela (Asano *et al.*, 2009) disponível neste tipo de circuito digital configurável.

A combinação de ML embarcado e sistemas industriais inteligentes promete revolucionar a maneira como as empresas operam, aumentando a eficiência, reduzindo custos e melhorando a qualidade dos produtos (O'donovan *et al.* 2018). Com o avanço contínuo da tecnologia, espera-se que mais aplicações inovadoras surjam, impulsionando ainda mais a transformação digital na indústria além de tornar todo o desenvolvimento mais acessível.

O objetivo deste trabalho foi a realização de experimentos nos quais o algoritmo de ML KNN foi implementado em um circuito integrado do tipo FPGA, para se avaliar o grau de facilidade ou de dificuldade nesse processo, a acurácia nos resultados obtidos e outros aspectos do problema que pudessem surgir na implementação prática.

## 2. TRABALHOS RELACIONADOS

A aplicação de modelos de ML em FPGAs é uma área de estudo relevante desde o fim do século passado. Cox e Blanz (1992) implementaram uma rede neural artificial com o


Ciência em Evidência, Revista Multidisciplinar, ISSN 2763-5457, v.5, FC, e024005, 2025, 20 pág.

objetivo de classificação, com foco principal em segmentação de imagens. Já Bade e Hutchings (1994) aprimoraram esta ideia, possibilitando o uso de redes maiores ao utilizar processos estocásticos no cálculo.

Mais recentemente Kim *et al.* (2009) utilizaram do massivo poder de computação paralela da FPGA para acelerar a etapa de treinamento de máquinas restritas de Boltzmann, reduzindo em até 30 vezes o tempo de processamento. Concomitantemente Rice *et al.* (2009) realizaram um trabalho parecido, porém focando na inferência de redes neurais pulsadas para reconhecimento de caracteres, acelerando esta etapa em até 8 vezes.

Em uma comparação realizada para o processamento de imagens entre execução em CPU, GPU e FPGA feita por Asano *et al.* (2009), foi verificado que o desempenho de um único filtro foi muito menor na FPGA do que nas outras plataformas. Porém, na aplicação de múltiplos filtros, a FPGA consegue superar as outras opções. Além disso, realizaram um teste também utilizando o algoritmo de agrupamento K-médias (Roy et al., 2009) alcançando resultados similares.

No mundo do aprendizado profundo, Farabet *et al.* (2009) criaram o CNP, um processador baseado em uma FPGA realizando inferência neste dispositivo com o objetivo de trabalhar com visão computacional. Assim, foi possível realizar o processamento de imagens de 512 x 384 pixels em escala de cinza a cada 100 ms. Além disso, apresentaram uma aplicação de reconhecimento facial em tempo real utilizando tal processador.

Na última década, o tópico foi bem explorado em diversos trabalhos como o de Zamorano *et al.* (2015), que focou na etapa de treino de redes neurais, criando blocos que permitem o cálculo do algoritmo de retropropagação de erros conseguindo uma execução cerca de 100 vezes mais rápida do que a versão em computadores pessoais e microcontroladores.

Houve também grande avanço no estudo de visão computacional embarcada, tanto com a criação de um *framework* para redes neurais convolucionais em FPGA por Venieris e Bouganis (2016), quanto com a execução da segunda versão da rede YOLOv2, com objetivo de detecção de objetos, em circuitos configuráveis (Nakahara *et al.*, 2018).

Após adoção em diversos campos da arquitetura de transformadores (Vaswani *et al.*, 2018), muitos trabalhos buscaram utilizar tal técnica em hardware como uma implementação focada em eficiência energética (Li *et al.*, 2020), a criação de um modelo embarcado de

tradução bidirecional (Li *et al.*, 2019), o uso de transformadores visuais (Okube *et al.*, 2023), entre outros.

Além disso, há diversos outros trabalhos com execução de algoritmos distintos nessa plataforma, como algoritmos Fuzzy (Aldair; Wang, 2010), árvores de decisão e conjuntos de árvores (Saqib *et al.*, 2013; Kulaga; Gorgon, 2015; Owaida; Alonso, 2018) e máquinas de vetores de suporte (SVM) (Groléat *et al.*, 2014; Afifi et al. 2020). Chen *et al.* (2022) também fizeram uma experimentação completa com diversos tipos de algoritmos em diversos circuitos de FPGA de diferentes fabricantes.

## 3. FUNDAMENTAÇÃO TEÓRICA

### 3.1. FPGA

Um FPGA (Field-Programmable Gate Array) é um tipo de dispositivo eletrônico que, similar a um MPGA (Masked Programmable Gate Array), consiste em uma matriz de elementos não comprometidos que podem ser interconectados de maneira geral. Ao contrário de dispositivos com interconexões fixas, como os PAL (*Programmable Array Logic*), as interconexões dentro de um FPGA são reconfiguráveis pelo usuário (Trimberger, 2012).

#### 3.1.1. Composição do sistema

Os blocos lógicos são componentes fundamentais de um FPGA e sua arquitetura pode variar. Alguns blocos são tão simples quanto portas NAND de duas entradas, enquanto outros são mais complexos, como multiplexadores ou tabelas de consulta (*lookup tables*). Em alguns casos, um bloco lógico pode corresponder a uma estrutura inteira similar a um PAL. A maioria dos blocos lógicos também inclui flip-flops para auxiliar na implementação de circuitos sequenciais. A arquitetura dos blocos lógicos impacta tanto a área total do chip quanto a velocidade dos circuitos implementados (Rose *et al.*, 1990).

A arquitetura de interconexão de um FPGA consiste em segmentos de fios e chaves programáveis. Essas chaves podem ser construídas usando várias tecnologias, como transistores de passagem controlados por células de RAM estática, anti-fusíveis, transistores, EPROM e EEPROM. A maneira como a interconexão é projetada pode variar, oferecendo diferentes quantidades de conectividade simples ou rotas mais complexas. A arquitetura de interconexão influencia a área do circuito e seu desempenho (Rose *et al.*, 1993).

Para implementar circuitos em um FPGA, a lógica é dividida em blocos lógicos individuais, que são então interconectados conforme necessário por meio de chaves

programáveis. A versatilidade do FPGA permite a implementação de uma grande variedade de circuitos digitais, adaptando-se a diferentes necessidades e projetos (Brown S. *et al.* 1992; Costa *et. al.*, 2011).

### 3.1.2. Linguagens de configuração de hardware

Um FPGA não é um dispositivo do tipo simplesmente programável, mas sim configurável. A diferença é que o primeiro é programado com um código fonte e o executa, já a configuração é o processo de organizar e interconectar elementos de hardware entre si, algo impossível em dispositivos programáveis no geral (FPGAKEY, 2024).

Destarte, é necessário utilizar algum tipo de linguagem de configuração para realizar essa modificação da estrutura interna do dispositivo. Atualmente há duas linguagens principais desse tipo no mercado, VHDL (IEEE, 2019) e Verilog (IEEE, 2008), ambas padronizadas pelo Instituto dos Engenheiros Elétricos e Eletrônicos (IEEE).

Uma forma comum de criar configurações utilizando tais linguagens é a partir da criação de entidades (ou módulos). Desta forma, ao invés de se criar uma única configuração grande e complexa, quebra-se esta tarefa em diversas menores (D'Amore, 2012).

Um exemplo disso é a criação de um módulo que realiza o cálculo da função afim, descrita pela equação (1) dado um valor de entrada X. Para isso, pode-se criar um módulo que realiza a adição de dois valores, outro que realiza a multiplicação de dois valores e utilizá-los no módulo de função afim. O interessante é que, além da reusabilidade dos módulos utilizados, é possível criar diversos exemplares dessa função, com parâmetros $a$ e $b$ variados.

$$Y = aX + b. \tag{1}$$

### 3.1.3. Aritmética de ponto fixo

A aritmética de ponto fixo é essencial em sistemas embarcados, especialmente em microcontroladores de baixo custo sem unidade de ponto flutuante (FPU). Ela permite representar quantidades fracionárias usando variáveis inteiras, o que é crucial para aplicações como interfaces homem-máquina e controle de malhas. A notação $Q_{m.n}$ é comumente utilizada, onde 'm' e 'n' indicam os bits para a parte inteira e fracionária, respectivamente. Por exemplo, o formato $Q_{1.14}$ utiliza 16 bits e representa valores entre -2 e 1,999938965 com uma resolução de 0,000061035 (Kormanyos, 2021). Formatos sem sinal também são possíveis,

denotados como $UQ_{m.n}$. A conversão entre ponto fixo e ponto flutuante é direta, mas deve-se escolher um formato adequado para representar os valores desejados.

As operações aritméticas em ponto fixo, como adição, subtração, multiplicação e divisão, devem ser realizadas com cuidado, para manter a precisão e evitar o estouro da capacidade de representação. A adição e subtração são simples quando os valores estão no mesmo formato. Já a multiplicação de dois valores $Q_{15}$, por exemplo, resulta em um formato $Q_{30}$, que precisa ser ajustado para $Q_{15}$ usando um deslocamento. Implementações práticas dessas operações utilizam técnicas de arredondamento e saturação para garantir precisão e evitar erros de estouro, essenciais para o bom funcionamento em microcontroladores de 16 bits (Yates, 2007).

### 3.2. Algoritmos de aprendizagem de máquina

Aprendizagem de máquina é o ramo da inteligência artificial que utiliza grandes quantidades de dados para que algoritmos aprendam a identificar padrões e criar generalizações a partir disso (Mohri *et al.*, 2018).

Dentro do domínio da ML, há ainda subdivisões. Uma delas é entre algoritmos baseados em exemplos e algoritmos baseados em modelos. A diferença entre eles é que o primeiro grupo realiza o aprendizado ao memorizar os dados utilizados na etapa de treino. Já o segundo busca encontrar alguma relação matemática entre os dados de entrada e saída, criando assim um modelo matemático para o problema a ser resolvido (Diressens; Džeroski, 2005).

Assim, os algoritmos baseados em exemplos têm o objetivo de salvar em sua memória todo o conjunto de dados utilizado para o treinamento e, na etapa de inferência, eles comparam os dados a serem analisados com o conjunto de treino para realizar a sua predição. Isso torna esse tipo de modelo muito interessante neste estudo, já que o objetivo aqui foi o de implementar em FPGA apenas a etapa de inferência, não de treinamento de tais algoritmos.

#### 3.2.1. K Vizinhos mais próximos

O algoritmo a ser usado neste estudo é o K-vizinhos mais próximos (KNN) (Fix e Hodges, 1951). Como já mencionado, trata-se de um algoritmo baseado em exemplos, que classifica uma nova amostra de acordo com a distância desta amostra até as amostras do conjunto utilizado no treino.


Ciência em Evidência, Revista Multidisciplinar, ISSN 2763-5457, v.5, FC, e024005, 2025, 20 pág.

A execução da etapa de classificação do algoritmo é dividida em três etapas. Na primeira etapa, devem ser calculadas as distâncias da amostra a ser classificada até todas as amostras no conjunto de treino. A distância utilizada nem sempre é a euclidiana, muitas vezes é utilizada alguma distância de Minkowski ou até outros tipos de distância, a depender do conjunto de dados (Chomboom *et al.*, 2015).

A segunda etapa do algoritmo é a execução de um algoritmo de ordenação. Isso deve ser feito para que as amostras de treino com menor distância até a amostra classificada sejam identificadas. Por fim, a última etapa do processamento é a verificação da moda das classes das $k$ amostras de treino com menor distância até a amostra a ser classificada, sendo $k$ um hiperparâmetro do algoritmo.

## 4. METODOLOGIA

O desenvolvimento do projeto foi dividido em duas etapas: compreensão do algoritmo e a de desenvolvimento, simulações e síntese.

### 4.1. Compreensão do algoritmo

O objetivo da primeira etapa do desenvolvimento foi o de criar maior intimidade com os algoritmos trabalhados. Para isso foi realizado um estudo do funcionamento interno do algoritmo e sua construção nas linguagens de programação Python e C, visando a sair do maior nível de abstração e familiaridade para o menor.

O desenvolvimento da solução em Python teve o objetivo de dominar o funcionamento do passo a passo do algoritmo. Assim, dividiu-se o KNN em cinco etapas executadas nesta ordem: normalização, distâncias, argumento do mínimo e, por fim, a etapa de moda.

Já o desenvolvimento em C objetivou trazer o entendimento das estruturas de dados e operações que deveriam ser realizadas em uma linguagem de mais baixo nível, diminuindo assim a barreira de transição entre a linguagem de programação e a de descrição para FPGA.

#### 4.1.1. Normalização

A etapa de normalização consistiu em uma implementação da função MinMaxScaler da biblioteca de código aberto em Python scikit-learn (Pedregosa, 2012). Assim, a Equação (1) é utilizada para normalizar cada elemento do vetor de entrada entre 0 e 1, utilizando como $X_{max}$ e $X_{min}$ os valores máximos e mínimos para aquela variável no conjunto de treino.

$$X_{std} = \frac{X - X_{min}}{X_{max} - X_{min}}. \quad (2)$$

A normalização torna-se muito importante, pois qualquer métrica de distância é sensível à ordem de grandeza das variáveis utilizadas. Assim, o ideal é manter todos os elementos dos vetores de entrada com a mesma ordem de grandeza.

### 4.1.2. Distâncias

Trata-se da etapa do cálculo das distâncias do vetor de entrada a se classificar até cada um dos vetores no conjunto de treino. A equação utilizada para o cálculo da distância depende da métrica utilizada. Nos experimentos realizados neste trabalho, foram usadas, principalmente, as distâncias euclidiana e a de Manhattan (Malkauthekar, 2013).

Assim, tem-se uma função que recebe um vetor genérico de $m$ componentes e uma matriz $n \times m$, sendo $n$ o tamanho do conjunto de treino e $m$ o número de componentes por vetor de entrada (número de variáveis de cada amostra a ser classificada) e o resultado é um outro vetor de $n$ elementos, as $n$ distâncias calculadas.

### 4.1.3. Argumento do mínimo

Argumento do mínimo é uma função que percorre uma dada lista de números e encontra a posição do menor elemento desta. Neste projeto, foi necessário fazer uma modificação no funcionamento padrão deste algoritmo para encontrar os $k$ menores elementos, sendo $k$ o parâmetro que nomeia o algoritmo. Assim o resultado desta etapa é uma outra lista com $k$ elementos sendo esses os índices das amostras de treino mais próximas da amostra sendo classificada.

Para isso, a abordagem utilizada foi o algoritmo de inserção por ordenação. Aqui a complexidade temporal dele pode ser reescrita de $O(n^2)$ para $O(kn)$, já que a lista na qual os elementos serão inseridos possui tamanho máximo $k$. Isso aliado ao fato de que o valor de $k$ utilizado geralmente é baixo e torna esse algoritmo viável.

Entre esta etapa e a próxima é realizado um procedimento: as classes correspondentes às amostras de treino desses índices, ou seja, as classes das amostras de treino com a menor distância até a amostra sendo classificada, são buscadas.

### 4.1.4. Moda

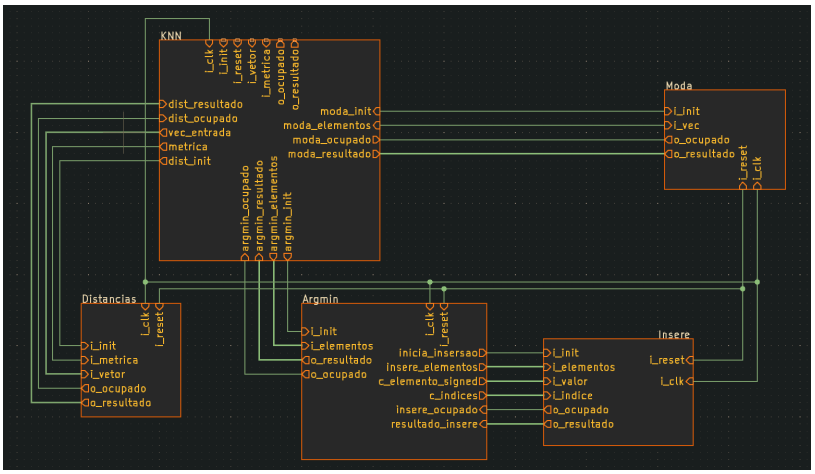
A última etapa é a mais simples de todas, realizando apenas o cálculo da moda da lista de classes com o intuito de encontrar a classe predominante. Para isso, é feito um laço de

iteração pela lista de classes, contando a quantidade de aparições de cada uma. Por fim, a classe predominante é retornada, finalizando a execução do KNN.

### 4.2. Desenvolvimento, simulações e síntese

A segunda etapa do desenvolvimento foi a implementação do algoritmo, nesta vez em linguagem de descrição de hardware. Neste momento foi necessário criar em VHDL um módulo para cada uma das etapas descritas na metodologia, com o adendo de um módulo extra utilizado no processamento da ordenação por inserção. A figura 1 mostra um diagrama de blocos com as interconexões de todos os módulos.

Figura 1 - Diagrama de blocos do projeto finalizado.




Fonte: Elaborado pelo(s) autor(es).

Ao final desta etapa, foi construída não só a descrição dos módulos, mas também uma infraestrutura para compilação que permitia a criação de hardware variando diversos parâmetros, como a quantidade de bits na parte inteira e fracionária dos números de ponto fixo, o conjunto de dados e a forma como sua separação em treino e teste é realizada, permitindo, assim, realizar os experimentos futuramente.

Já na etapa de síntese foi utilizado o software gratuito Quartus Prime Lite edition (Intel, 2024) para poder enviar a configuração para placas de FPGA da Altera, mais especificamente o kit de desenvolvimento DE10 LITE da Terasic (Intel, 2024). Um ponto importante é que devido às dificuldades na etapa de síntese do sistema.

## 5. RESULTADOS

Foram realizados experimentos com dois conjuntos de dados distintos variando tanto o parâmetro K quanto o número de bits na parte inteira e na parte fracionária. Um ponto limitante importante para esses experimentos é a quantidade de portas de comunicação da

FPGA. Na arquitetura desenvolvida, os dados a serem classificados são enviados de forma paralela ao sistema. Isso faz com que tanto o número de bits quanto o número de características tenham um limite superior definido pela quantidade de portas, de acordo com a inequação (3). Considerando $Inteira$ como a quantidade de bits na parte inteira, $Fracionária$ a quantidade de bits na parte fracionária, $Características$ o número de características (componentes por vetor) e $Classes$ o número de classes presentes no conjunto de dados,

$$(Inteira + Fracionária + 1) \times Características + Classes \leq 36. \quad (3)$$

Os conjuntos de dados utilizados foram o Iris (Fisher, 1936), Wine (Forina, 1992) e Digits (Alpaydin, 1998). Os três foram altamente explorados e são muito utilizados para validação de metodologias novas e experimentações. O Quadro 1 resume os parâmetros que trouxeram melhor acurácia do sistema desenvolvido na FPGA e uma comparação com acurácia em software.

Quadro 1 - Melhores resultados para cada conjunto de dados.

| Conjunto de dados | K | Arquitetura de ponto fixo | Distância | Acurácia (sistema) | Acurácia (software) | Número de classes |
|---|---|---|---|---|---|---|
| Iris | 3 | Q5.4 | Euclidiana | 95,56% | 98,00% | 3 |
| Wine | 5 | Q13.3 | Euclidiana | 90,74% | 94,91% | 3 |
| Digits | 5 | Q6.0 | Euclidiana | 98.89% | 99,15% | 10 |

Fonte: Elaborado pelo(s) autor(es).

O conjunto Wine foi o que apresentou a maior queda na acurácia. Aqui, a ausência do normalizador foi o maior agravante. Esse conjunto possui componentes de vetores com ordens de grandeza muito distintas, tornando o uso da normalização mais necessário e inclusive esse é o motivo da necessidade de uso de mais bits para a parte inteira. Porém, ainda assim, uma acurácia de 90% é excelente.

Dados os parâmetros definidos no Quadro 1, o Quadro 2 descreve o uso de recursos pelos sistemas sintetizados. De acordo com as capacidades da FPGA utilizada, tanto o conjunto de dados Iris quanto o Wine são possíveis de serem executados na placa. Porém, o

Digits não, devido à sua maior complexidade, já que esse possui o maior número de elementos no conjunto de treino e o maior número de variáveis por vetor.

Quadro 2 - Uso de recursos em cada conjunto de dados

| Conjunto de dados | Blocos Lógicos Disponíveis | Blocos Lógicos Utilizados | Percentual de uso de CHIP | Pinos de clock |
|---|---|---|---|---|
| Iris | 49760 | 40772 | 81,93% | 1 |
| Wine | 49760 | 39660 | 79,70% | 1 |
| Digits | 49760 | 408141 | 820% | 2 |

Fonte: Elaborado pelo(s) autor(es).

O Quadro 3 é uma comparação do tempo de execução do algoritmo entre os sistemas sintetizados e a execução em software. O tempo calculado na execução em FPGA é o tempo completo do processamento, desde o sinal de iniciar ser enviado até que o sinal de finalização seja ativado, a frequência de clock utilizada para esse teste foi de 50MHz. Já em software o tempo considerado é o tempo de execução da função que define o algoritmo de K-NN utilizando o processador Intel Core i5-10300H com frequência de clock base de 25GHz, 24Gb de memória RAM no sistema operacional Windows 10 Pro 22H2.

Quadro 3 - Comparação do tempo de execução

| Conjunto de dados | Tempo médio de Execução FPGA | Tempo médio de Execução Software | Fator de velocidade |
|---|---|---|---|
| Iris | 0,61ms/inferência | 0,60 ms/inferência | 0,98 |
| Wine | 1,66 ms/inferência | 1,37 ms/inferência | 0,82 |
| Digits | 66,5 ms/inferência | 5,13 ms/inferência | 0,07 |

Fonte: Elaborado pelo(s) autor(es).

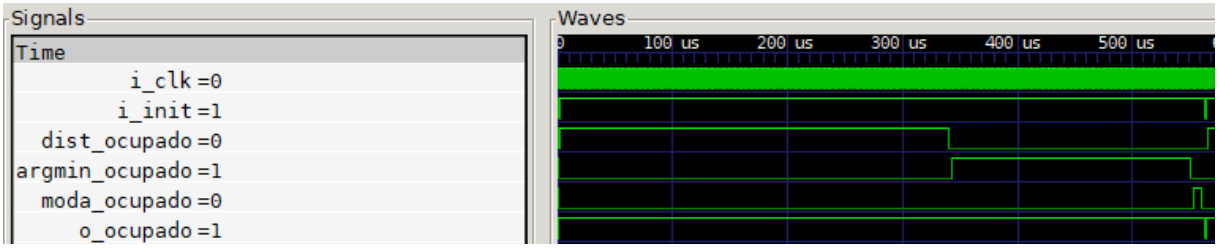
A coluna fator de velocidade é o resultado da divisão do tempo de execução em software pelo tempo de execução na FPGA. Isso mostra que, no geral, a execução em software é mais rápida muito provavelmente devido às otimizações que bibliotecas de código aberto já implementam no algoritmo, otimizações essas que não foram colocadas na FPGA.

Em uma análise mais detalhada, é possível verificar que o maior gargalo da execução é na etapa de cálculo das distâncias. Da forma como o algoritmo foi implementado, as distâncias são calculadas uma após a outra. Porém, é possível paralelizar tal tarefa, acelerando

ainda mais a execução. Esse impacto se torna ainda maior quando forem maiores a quantidade de amostras no conjunto de treino e a quantidade de variáveis por vetor, pois o primeiro aumenta a quantidade de multiplicações a serem realizadas e o segundo aumenta a quantidade de cálculos realizados por multiplicação.

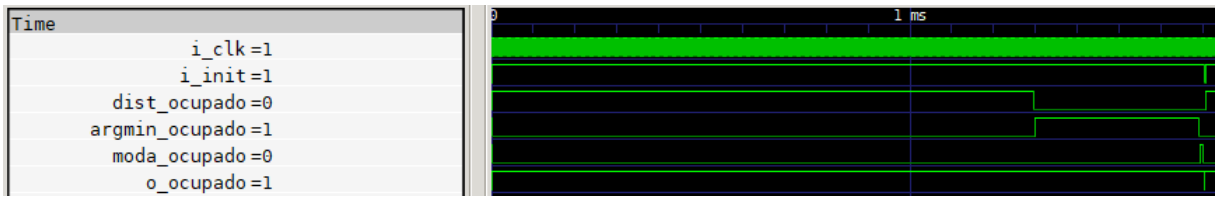
As figuras 1, 2 e 3 mostram formatos de onda da execução do algoritmos para cada um dos datasets (iris, wine e digits respectivamente). Nelas é visível o tempo de execução de cada etapa do algoritmo, contendo, além do sinal de clock, cinco sinais: dist_ocupado (módulo de distâncias), argmin_ocupado (módulo de argumento do mínimo), moda_ocupado (módulo de moda) e o_ocupado (algoritmo como um todo) e i_init (sinal de inicialização do algoritmo). Esses sinais indicam qual é o processamento sendo realizado. Por exemplo, quando dist_ocupado está em nível lógico alto, significa que, atualmente, a etapa de distâncias está sendo calculada.

Figura 1 - Formatos de onda - tempo de execução de cada etapa no conjunto de dados Iris.



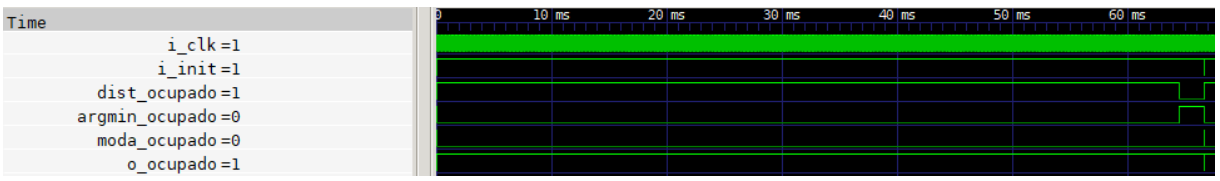

Fonte: Elaborado pelo(s) autor(es).

Figura 2 - Formatos de onda - tempo de execução de cada etapa no conjunto de dados Wine.




Fonte: Elaborado pelo(s) autor(es).

Figura 3 - Formatos de onda - tempo de execução de cada etapa no conjunto de dados Digits.




Fonte: Elaborado pelo(s) autor(es).

Observando os dados da Figura 1, é notável que mais da metade do tempo de processamento é gasto calculando as distâncias, o segundo maior responsável pela demora é o módulo de argmin, sendo que a moda é calculada quase que instantaneamente. Na Figura 2 e

principalmente na 3, isso fica ainda mais visível. Para o conjunto de dados Digits com 1257 amostras de treino, 64,0 ms dos 66,5 ms de inferência foram utilizados calculando as distâncias, ou seja, 96% do tempo.

## 6. CONCLUSÃO

A metodologia aplicada demonstrou a viabilidade de implementar o algoritmo KNN em FPGAs, integrando a compreensão teórica e prática do algoritmo. A abordagem incluiu o desenvolvimento inicial em linguagem Python e C, seguido pela implementação em VHDL, destacando assim a importância de entender tanto as operações em alto nível quanto as limitações de hardware.

Os experimentos realizados com diferentes conjuntos de dados (Iris, Wine e Digits) e variações nos parâmetros mostraram que, embora a acurácia da implementação em FPGA fosse inferior à do software, os resultados ainda foram promissores. A queda na acurácia pode ser atribuída à limitação de precisão dos números de ponto fixo utilizados.

A análise do uso dos recursos dos sistemas sintetizados confirmou a viabilidade e eficiência do uso de FPGAs para algoritmos de aprendizado de máquina. Esses resultados abrem caminho para futuras otimizações e implementação de algoritmos mais complexos em dispositivos de hardware programável.

Como próximos passos, propõem-se: a adição da normalização dos dados, incluindo a etapa de divisão que foi o grande impeditivo dessa adição no projeto, implementação de outros tipos de modelos como redes neurais e regressão logística, a utilização de comunicação serial para recebimento dos dados, permitindo utilizar representações de ponto fixo com mais bits e, por fim, a paralelização do módulo de distâncias, para que mais de uma distância seja calculada ao mesmo tempo, diminuindo, assim, o tempo total de inferência consideravelmente.

**Referências**

AFIFI, S.; GHOLAMHOSSEINI, H.; SINHA, R. FPGA implementations of SVM classifiers: A review. **SN computer science**, v. 1, n. 3, 2020.

ALDAIR, A. A.; WANG, W. FPGA based adaptive neuro fuzzy inference controller for full vehicle nonlinear active suspension systems. **International Journal of Artificial Intelligence & Applications**, v. 1, n. 4, p. 1–15, 2010.

ALPHONSUS, E. R.; ABDULLAH, M. O. **A review on the applications of programmable logic controllers (PLCs)**. Renewable and Sustainable Energy Reviews, v. 60, p. 1185–1205, 2016.

ALVES, T. *et. al*. **OpenPLC: An open source alternative to automation**. IEEE Global Humanitarian Technology Conference (GHTC 2014). **Anais**...IEEE, 2014.

ANDINA, J. J.; VALDES-PENA, M. D.; MOURE, M. J. **Advanced features and industrial applications of FPGAs—A review**. IEEE transactions on industrial informatics, v. 11, n. 4, p. 853–864, 2015.

ASANO, S.; MARUYAMA, T.; YAMAGUCHI, Y. **Performance comparison of FPGA, GPU and CPU in image processing**. 2009 International Conference on Field Programmable Logic and Applications. **Anais**...IEEE, 2009.

BADE, S. L.; HUTCHINGS, B. L. **FPGA-based stochastic neural networks-implementation**. Proceedings of IEEE Workshop on FPGA's for Custom Computing Machines. Anais...IEEE Comput. Soc. Press, 2002.

BERTSIMAS, D.; KALLUS, N.; HUSSAIN, A. Inventory management in the era of big data. **Production and operations management**, v. 25, n. 12, p. 2006–2009, 2016.

BROWN S. *et. al*. **Field-programmable gate arrays**. 1992. ed. Nova Iorque, NY, USA: Springer, 1992.

CARDOZO, G. **Inteligência artificial aplicada a sistemas supervisórios de automação residencial com internet das coisas**. São Paulo - SP Brasil: Universidade Presbiteriana Mackenzie, 2021.

CHEN, C.; LIN, M..; LIU, C.. **Edge computing gateway of the industrial internet of things using multiple collaborative microcontrollers**. IEEE network, v. 32, n. 1, p. 24–32, 2018.

CHEN, R. *et. al*. MLoF: Machine Learning accelerators for the low-cost FPGA platforms. **Applied sciences (Basel, Switzerland)**, v. 12, n. 1, p. 89, 2021.

CHOMBOOM, K. *et. al*. **An empirical study of distance metrics for k-nearest neighbor algorithm**. The Proceedings of the 2nd International Conference on Industrial Application Engineering 2015. **Anais**...The Institute of Industrial Applications Engineers, 2015.

COX, C. E.; BLANZ, W. E. **GANGLION-a fast field-programmable gate array implementation of a connectionist classifier**. IEEE journal of solid-state circuits, v. 27, n. 3, p. 288–299, 1992.]

DA COSTA, C.; MESQUITA, L.; PINHEIRO, E. **Elemento De Logica Programavel Com VHDL E DSP. Teoria & Prática**. [s.l.] Érica, 1 janeiro 2011.

DALZOCHIO, J. *et. al.* **Machine learning and reasoning for predictive maintenance in Industry 4.0: Current status and challenges**. Computers in industry, v. 123, n. 103298, p. 103298, 2020.

D'AMORE, R. **VHDL - DESCRIÇÃO E SÍNTESE DE CIRCUITOS DIGITAIS**. 2. ed. [s.l.] Ltc, 2012.

DRIESSENS, K.; DŽEROSKI, S. **Combining model-based and instance-based learning for first order regression.** Proceedings of the 22nd international conference on Machine learning - ICML '05. Anais...New York, New York, USA: ACM Press, 2005.

DUTTA, D. L.; BHARALI, S. TinyML meets IoT: A comprehensive survey. **Internet of Things**, v. 16, n. 100461, p. 100461, 2021.

E. ALPAYDIN, C. K. **Optical recognition of handwritten digits**. UCI Machine Learning Repository, , 1998.

FISHER, R. A. **Iris**. UCI Machine Learning Repository, , 1936.

FIX, E.; HODGES, J. L. **Discriminatory analysis. Nonparametric discrimination: Consistency properties**. Revue internationale de statistique [International statistical review], v. 57, n. 3, p. 238, 1989.

FORINA, S. A. M. **Wine**. UCI Machine Learning Repository, , 1992. Disponível em: <http://dx.doi.org/10.24432/C5PC7J>

FREY, G.; LITZ, L. **Formal methods in PLC programming**. SMC 2000 Conference Proceedings. 2000 IEEE International Conference on Systems, Man and Cybernetics. "Cybernetics Evolving to Systems, Humans, Organizations, and their Complex Interactions" (Cat. No.00CH37166). **Anais**...IEEE, 2002.

FPGAKEY. **How Does Configurable Logic Work?** Disponível em: <https://www.fpgakey.com/tutorial/section67>. Acesso em: 27 maio. 2024.

GHOBAKHLOO, M. **Industry 4.0, digitization, and opportunities for sustainability**. Journal of cleaner production, v. 252, n. 119869, p. 119869, 2020.

GIANNOCCARO, I.; PONTRANDOLFO, P. Inventory management in supply chains: a reinforcement learning approach. **International journal of production economics**, v. 78, n. 2, p. 153–161, 2002.

GROLEAT, T.; ARZEL, M.; VATON, S. Stretching the edges of SVM traffic classification with FPGA acceleration. **IEEE transactions on network and service management**, v. 11, n. 3, p. 278–291, 2014.

HAMET, P.; TREMBLAY, J. **Artificial intelligence in medicine**. **Metabolism: clinical and experimental**, v. 69, p. S36–S40, 2017.

KIM, S. K. *et. al.* **A highly scalable Restricted Boltzmann Machine FPGA implementation**. 2009 International Conference on Field Programmable Logic and Applications. **Anais**...IEEE, 2009.

KORMANYOS, C. Fixed-Point Mathematics. Em: **Real-time C++: Efficient object-oriented and template microcontroller programming**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2021. p. 329–351.

KULAGA, R., GORGÓN, M. FPGA implementation of decision trees and tree ensembles for character recognition in vivado hls. **Image Processing & Communications**, v. 19, n. 2–3, p. 71–82, 2014.

**IEEE Standard for Verilog Hardware Description Language**. Piscataway, NJ, USA: IEEE, 7 abr. 2008.

**IEEE Standard for VHDL Language Reference Manual**. Piscataway, NJ, USA: IEEE, 23 dez. 2019.

INTEL. **DE10 Lite User Manual.** Disponível em: <https://ftp.intel.com/Public/Pub/fpgaup/pub/Intel_Material/Boards/DE10-Lite/DE10_Lite_User_Manual.pdf>. Acesso em: 01 ago. 2024

INTEL. **Intel Quartus Prime Standard Edition User Guides**. Disponível em:<https://www.intel.com/programmable/technical-pdfs/qps-ugs.pdf>. Acesso em: 01 ago. 2024.

LEE, W. J. *et. al.* **Predictive maintenance of machine tool systems using artificial intelligence techniques applied to machine condition data**. Procedia CIRP, v. 80, p. 506–511, 2019.

LI, B. *et. al.* **FTRANS: Energy-efficient acceleration of transformers using FPGA**. Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design. **Anais**...New York, NY, USA: ACM, 2020.

LI, J.-H. **Cyber security meets artificial intelligence: a survey**. **Frontiers of Information Technology & Electronic Engineering**, v. 19, n. 12, p. 1462–1474, 2018.

LI, Q. *et. al.* **Implementing neural machine translation with bi-directional GRU and attention mechanism on FPGAs using HLS**. Proceedings of the 24th Asia and South Pacific Design Automation Conference. **Anais**...New York, NY, USA: ACM, 2019.

MALKAUTHEKAR, M. D. (2013). **Analysis of euclidean distance and manhattan distance measure in face recognition**. Third International Conference on Computational Intelligence and Information Technology (CIIT 2013).

MOHRI, M.; ROSTAMIZADEH, A.; TALWALKAR, A. **Foundations of machine learning**. 2. ed. Londres, England: MIT Press, 2018.

MURPHY, R. R. **Introduction to AI Robotics**. 2. ed. Londres, England: MIT Press, 2019.

NAKAHARA, H. *et. al.* **A lightweight YOLOv2: A binarized CNN with A parallel support vector regression for an FPGA**. Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. **Anais**...New York, NY, USA: ACM, 2018.

O'DONOVAN, P. *et. al.* **A fog computing industrial cyber-physical system for embedded low-latency machine learning Industry 4.0 applications**. Manufacturing letters, v. 15, p. 139–142, 2018.

OKUBO, I. *et. al.* **A lightweight transformer model using neural ODE for FPGAs**. 2023 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW). **Anais**. IEEE, 2023.

OWAIDA, M.; ALONSO, G. **Application partitioning on FPGA clusters: Inference over decision tree ensembles**. 2018 28th International Conference on Field Programmable Logic and Applications (FPL). **Anais**...IEEE, 2018.

PARNELL, B. K.; BRYNER, R. **Comparing and contrasting FPGA and microprocessor system design and development**. Disponível em: <https://application-notes.digchip.com/077/77-43447.pdf>. Acesso em: 30 abr. 2024.

PEDREGOSA, F. *et. al.* **Scikit-learn: Machine Learning in Python**. 2012. Disponível em: <http://arxiv.org/abs/1201.0490>.

POURNADER, M. *et. al.* Artificial intelligence applications in supply chain management. **International journal of production economics**, v. 241, n. 108250, p. 108250, 2021.

RAY, P. P. A review on TinyML: State-of-the-art and prospects. **Journal of King Saud University - Computer and Information Sciences**, v. 34, n. 4, p. 1595–1623, 2022.

RICE, K. L. *et. al.* **FPGA implementation of Izhikevich spiking neural networks for character recognition**. 2009 International Conference on Reconfigurable Computing and FPGAs. **Anais**...IEEE, 2009.

ROSE, J. *et. al.* Architecture of field-programmable gate arrays: the effect of logic block functionality on area efficiency. **IEEE journal of solid-state circuits**, v. 25, n. 5, p. 1217–1225, 1990.

ROSE, J.; EL GAMAL, A.; Sangiovanni-Vincentelli, A. Architecture of field-programmable gate arrays. **Proceedings of the IEEE. Institute of Electrical and Electronics Engineers**, v. 81, n. 7, p. 1013–1029, 1993.

ROY, H. L. L.; Lecam, L.; Neyman, J. Proceedings of the fifth Berkeley symposium on mathematical statistics and probability; Vol. IV. **Revue de l'Institut international de statistique**, v. 37, n. 2, p. 230, 1969.

ROY, H. L. L.; LECAM, L.; NEYMAN, J. Proceedings of the fifth Berkeley symposium on mathematical statistics and probability; Vol. IV. **Revue de l'Institut international de statistique**, v. 37, n. 2, p. 230, 1969.

SAKURAI, R.; ZUCHI, J. D. **AS REVOLUÇÕES INDUSTRIAIS ATÉ A INDÚSTRIA 4.0**. Revista Interface Tecnológica, v. 15, n. 2, p. 480–491, 30 dez. 2018.

SAQIB, F. *et. al*. Pipelined Decision Tree Classification Accelerator Implementation in FPGA (DT-CAIF). **IEEE transactions on computers. Institute of Electrical and Electronics Engineers**, v. 64, n. 1, p. 280–285, 2015.

SARKER, I. H. Machine learning: Algorithms, real-world applications and research directions. **SN computer science**, v. 2, n. 3, 2021.

SCHWAB, K. **A Quarta Revolução Industrial**. São Paulo - SP: Edipro, 2018.

SHAVLIK, J.; DIETTERICH, T. (eds.). **Readings in Machine Learning**. [s.l.] Morgan Kaufmann, 1990.

TRIMBERGER, S. M. (Ed.). **Field-programmable gate array technology**. Nova Iorque, NY, USA: Springer, 2012.

VASWANI, A. *et. al.* **Attention is all you need**. 2017. Disponível em: <http://arxiv.org/abs/1706.03762>. Acesso em: 9 maio. 2024.

VENIERIS, S. I.; BOUGANIS, C.-S. **FpgaConvNet: A framework for mapping convolutional neural networks on FPGAs**. 2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM). **Anais**...IEEE, 2016.

YATES, R. **Fixed-Point Arithmetic: An Introduction**. 23 ago. 2007.

YUEH, H.-P.; CHIANG, F.-K. AI and robotics in reshaping the dynamics of learning. **British journal of educational technology: journal of the Council for Educational Technology**, v. 51, n. 5, p. 1804–1807, 2020.

ZAMORANO, F. O. *et. al.* Efficient implementation of the backpropagation algorithm in FPGAs and microcontrollers. **IEEE transactions on neural networks and learning systems**, v. 27, n. 9, p. 1840–1850, 2016.

ZONTA, T. *et. al.* **Predictive maintenance in the Industry 4.0: A systematic literature review**. Computers & industrial engineering, v. 150, n. 106889, p. 106889, 2020.

