Universidade Federal do Rio Grande do Norte
Centro de Ciências Exatas e da Terra
Departamento de Informática e Matemática Aplicada
Programa de Pós-Graduação em Sistemas e Computação
Mestrado Acadêmico em Sistemas e Computação

HARS1DE: Arquitetura de hardware para processamento de CNNs-1D na borda

Mailson Rodrigues de Medeiros Guimarães

Natal-RN
Janeiro 2025

Mailson Rodrigues de Medeiros Guimarães

HARS1DE: Arquitetura de hardware para processamento de CNNs-1D na borda

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Sistemas e Computação do Departamento de Informática e Matemática Aplicada da Universidade Federal do Rio Grande do Norte como requisito parcial para a obtenção do grau de Mestre em Sistemas e Computação.

Linha de pesquisa:
Sistemas Integrados

Orientador

Prof. Dr. Márcio Eduardo Kreutz

PPgSC – Programa de Pós-Graduação em Sistemas e Computação
DIMAp – Departamento de Informática e Matemática Aplicada
CCET – Centro de Ciências Exatas e da Terra
UFRN – Universidade Federal do Rio Grande do Norte

Natal-RN

Janeiro 2025

Universidade Federal do Rio Grande do Norte - UFRN
Sistema de Bibliotecas - SISBI
Catalogação de Publicação na Fonte. UFRN - Biblioteca Setorial Prof. Ronaldo Xavier de Arruda - CCET

Guimaraes, Mailson Rodrigues de Medeiros.
HARS1DE: arquitetura de hardware para processamento de CNNs-1D na borda / Mailson Rodrigues de Medeiros Guimaraes. - 2025.
105 f.: il.

Dissertação (mestrado) - Universidade Federal do Rio Grande do Norte, Centro de Ciências Exatas e da Terra, Programa de Pós-Graduação em Sistemas e Computação. Natal, RN, 2025.
Orientação: Prof. Dr. Márcio Eduardo Kreutz.

1. Computação na borda - Dissertação. 2. Arquitetura de hardware - Dissertação. 3. Aprendizado de máquina - Dissertação. 4. Redes neurais convolucionais - Dissertação. 5. Sensoriamento remoto - Dissertação. I. Kreutz, Márcio Eduardo. II. Título.

RN/UF/CCET CDU 004(043.3)

Elaborado por Joseneide Ferreira Dantas - CRB-15/324

MAILSON RODRIGUES DE MEDEIROS GUIMARAES

"*HARS1DE: Arquitetura de hardware para processamento de CNNs-1D na borda*"

Esta Dissertação foi julgada adequada para a obtenção do título de Mestre em Sistemas e Computação e aprovada em sua forma final pelo Programa de Pós-Graduação em Sistemas e Computação do Departamento de Informática e Matemática Aplicada da Universidade Federal do Rio Grande do Norte.

gov.br Documento assinado digitalmente
EVERTON RANIELLY DE SOUSA CAVALCANTE
Data: 10/06/2025 15:42:45-0300
Verifique em https://validar.iti.gov.br

Prof. Dr. EVERTON RANIELLY DE SOUSA CAVALCANTE
Coordenador do PPgSC

Banca Examinadora:

gov.br Documento assinado digitalmente
CESAR ALBENES ZEFERINO
Data: 10/06/2025 11:35:34-0300
Verifique em https://validar.iti.gov.br

Examinador(a) Externo(a): **Dr. CESAR ALBENES ZEFERINO**

gov.br Documento assinado digitalmente
MARCELO AUGUSTO COSTA FERNANDES
Data: 10/06/2025 11:49:29-0300
Verifique em https://validar.iti.gov.br

Examinador(a) Externo(a): **Dr. MARCELO AUGUSTO COSTA FERNANDES**

gov.br Documento assinado digitalmente
MONICA MAGALHAES PEREIRA
Data: 10/06/2025 11:12:26-0300
Verifique em https://validar.iti.gov.br

Examinador(a) Interno(a): **Dr.ª MONICA MAGALHAES PEREIRA**

gov.br Documento assinado digitalmente
MARCIO EDUARDO KREUTZ
Data: 09/06/2025 13:56:37-0300
Verifique em https://validar.iti.gov.br

Presidente: **Dr. MARCIO EDUARDO KREUTZ**

gov.br Documento assinado digitalmente
MAILSON RODRIGUES DE MEDEIROS GUIMARAE
Data: 09/06/2025 13:45:52-0300
Verifique em https://validar.iti.gov.br

Discente: **MAILSON RODRIGUES DE MEDEIROS GUIMARAES**

Natal, 31 de janeiro de 2025.

Aos meus pais, José Marcone e Maria Lúcia, e à minha tia, Maria José, por toda a disciplina, educação de base e todo o suporte que me deram desde sempre.

Agradecimentos

Agradeço primeiramente a meus pais, José Marcone e Maria Lúcia, por toda a educação, disciplina e apoio que me deram ao longo da minha vida, abrindo mão de coisas em prol do meu bem-estar. Também agradeço à minha tia, Maria José, por ter me alfabetizado desde cedo, sendo responsável pelo que sou hoje em dia.

À UFRN, por toda a infraestrutura, serviços e recursos que possibilitaram meu desenvolvimento acadêmico, profissional e pessoal. A Márcio Eduardo Kreutz, por toda a orientação ao longo desses anos de mestrado. Aos demais professores de excelência, por todo o conhecimento passado, citando nominalmente: Monica Magalhães, Silvia Maria e Anne Magaly. Ao CNPq pela oportunidade de bolsa de estudos para o desenvolvimento deste trabalho.

À Débora Everly, meu amor, por todo o suporte ao longo de todos esses anos, por estar ao meu lado nos momentos bons e ruins e por ter me aturado durante a realização deste trabalho.

Aos meus amigos, sejam os mais antigos ou os que fiz nessa jornada, por estarem sempre ao meu lado, torcendo por mim e me proporcionando momentos de felicidade.

"Quero assistir ao sol nascer
Ver as águas dos rios correr
Ouvir os pássaros cantar
Eu quero nascer, quero viver"

Cartola

HARS1DE: Arquitetura de hardware para processamento de CNNs-1D na borda

Autor: Mailson Rodrigues de Medeiros Guimarães
Orientador(a): Prof. Dr. Márcio Eduardo Kreutz

Resumo

Atualmente há uma tendência ao uso do paradigma de *cloud computing*, onde os recursos, armazenamento e processamento de informação são realizados nas chamadas "nuvens", gerenciadas por provedores. Tal paradigma é aproveitado para o uso, por exemplo, de algoritmos de aprendizado de máquina sobre grandes volumes de dados. Em contrapartida, há o paradigma de computação na borda (*edge computing)*, onde essa carga de processamento é transferida para elementos próximos de onde os dados são gerados (na borda da rede). O investimento de empresas de tecnologia sobre esse tipo de computação e suas técnicas tem crescido, pois essa pode promover ganhos, por exemplo, em termos de latência no processamento, consumo de energia e recursos que eventualmente não estejam disponíveis na nuvem. Analogamente à computação na nuvem, também é possível realizar a aplicação de modelos preditivos de aprendizado de máquina na borda, onde arquiteturas de hardware dedicadas à aceleração desses processos podem ser empregadas. Dessarte, esse trabalho tem como objetivo principal a implementação, teste e validação de uma arquitetura de hardware capaz de acelerar a computação da inferência em CNNs-1D, incluindo camadas de *pooling*, ativação e *dense*, onde são analisadas métricas de desempenho, acurácia e uso de recursos de hardware. Para a obtenção dos resultados, foram realizadas duas representações da arquitetura, sendo uma em VHDL e sintetizada para FPGA de forma a se obter resultados de alocação de recursos de hardware e de tempo. A outra representação foi realizada em Python, linguagem de alto nível de abstração, para a obtenção de resultados mais rápidos sobre o comportamento da arquitetura mediante a execução de processos mais longos, como o processamento de uma rede neural inteira. Foram realizados testes em três diferentes variações da arquitetura proposta. Os resultados foram obtidos através da aplicação da arquitetura no domínio de aplicação de

sensoriamento remoto, especificamente na classificação de pixels em imagens hiperespectrais. A rede neural utilizada foi uma versão simplificada de trabalhos anteriores na área para ser portada ao hardware. A arquitetura obtida, além de ser reconfigurável no sentido da FPGA, também tem seu comportamento mutável dependendo do tipo de camada de rede neural a ser processada. Resultados teóricos mostram um desempenho máximo de $14,4GOP/$s para a melhor variação da arquitetura, além de uma aceleração máxima de $4,52\times$ em relação a um processador AMD EPYC 7B12 de $2,25GHz$, de $8,36\times$ em relação à NVIDIA T4 e $3,39\times$ em relação a um AMD Ryzen 7 7800X3D. Tais resultados foram obtidos a partir da classificação de uma das imagens hiperespectrais e a melhor variação da arquitetura terminou com uma ocupação abaixo de 80% para os recursos da FPGA utilizada.

Palavras-chave: Computação na borda, arquitetura de hardware, aprendizado de máquina, redes neurais convolucionais, sensoriamento remoto.

HARS1DE: Reconfigurable and Scalable Hardware Accelerator for CNNs-1D in Edge computing

Author: Mailson Rodrigues de Medeiros Guimarães
Supervisor: Prof. Dr. Márcio Eduardo Kreutz

Abstract

There is a trend toward using the cloud computing paradigm, where resources, storage, and information processing are carried out in so-called "clouds"managed by providers. This paradigm is leveraged, for instance, to apply machine learning algorithms to large volumes of data. Conversely, there is the edge computing paradigm, where this processing load is transferred to elements closer to where the data is generated (at the network edge). Investment by technology companies in this type of computing and its techniques has been growing, as it can offer advantages, such as reduced processing latency, energy consumption, and resource demands that may not always be available in the cloud. Similarly to cloud computing, it is possible to apply predictive machine learning models at the edge, where hardware architectures dedicated to accelerating these processes can be employed. Thus, this work's main objective is to implement, test, and validate a hardware architecture capable of accelerating the computation of 1D-CNNs inference, including pooling, activation, and dense layers, where performance metrics, accuracy, and hardware resource utilization are analyzed. Two representations of the architecture were developed to obtain the results: one in VHDL, synthesized for FPGA to get results regarding hardware resource allocation and timing, and another in Python, a high-level abstraction language, to obtain quicker results on the architecture's behavior during longer processes, such as the computation of an entire neural network. Tests were conducted on three different variations of the proposed architecture.The results were obtained by applying the architecture in remote sensing, specifically for pixel classification in hyperspectral images. The neural network used was a simplified version of previous works to facilitate porting to hardware. In addition to being reconfigurable in the context of FPGAs, the resulting architecture exhibits adaptable behavior depending on the type of neural network layer

being processed. Theoretical results demonstrate a maximum performance of $14.4GOP/s$ for the best architecture variation, as well as a maximum acceleration of 4.52× compared to an AMD EPYC 7B12 processor, 8.36× compared to an NVIDIA T4 and 3.39× to an AMD Ryzen 7 7800X3D. These results were achieved classifying one of the hyperspectral images and the best architecture variation ended with a FPGA resource usage below 80%.

Keywords: Edge computing, Hardware architecture, Machine learning, Convolutional neural networks, remote sensing.

Lista de figuras

Lista de tabelas

Lista de Quadros

Lista de abreviaturas e siglas

ALM	*Adaptative Logic Module*
ANN	*Artificial Neural Network*
ASIC	*Application-Specific Integrated Circuit*
AVIRIS	*Airbone Visible/Infrared Imaging Spectrometer*
BRAM	*Block Random Acess Memory*
BUFG	*Buffered Global Clock*
CE	*Convolutional Element*
CNN	*Convolutional Neura Network*
CPU	*Central Processing Unit*
DL	*Deep Learning*
DNN	*Deep Neural Network*
DQN	*Deep Q Network*
DRAM	*Dynamic Random Access Memory*
DSP	*Digital Signal Processors*
DT	*Decision Tree*
eDRAM	*Embedded DRAM*
FC	*Fully Connected*
FF	*Flip-Flops*
FIFO	*First In, First Out*
FPGA	*Field-Programmable Gate Array*

GPU	*Graphic Processing Unit*
HARS1DE	*Hardware Architecture Reconfigurable and Scalable for CNN-1D processing on the Edge*
HSI	*Hyperspectral Image*
HPC	*High-Performance Computing*
HyspIRI	*Hyperspectral InfraRed Imager*
IA	Inteligência Artificial
ISA	*Instruction Set Architecture*
IoT	*Internet of Things*
LiDAR	*Light Detection and Ranging*
LUT	*Lookup Tables*
MPSoC	*MultiProcessor System on Chip*
ML	*Machine Learning*
MLP	*Multi-Layer Perceptrons*
NLR	*No-Local-Reuse*
OS	*Output-Stationary*
PE	*Processing Element*
PIM	*Processing-in-memory*
RF	*Random Forest*
RF	*Register File*
RGB	*Red, Green and Blue*
RNN	*Recurrent Neural Network*
ROSIS	*Reflective Optics System Imaging Spectrometer*
RS	*Row Stationary*

SAC	*Soft Actor-Critic*
SEEDS	*Superpixels Extracted via Energy-Driven Sampling*
SoC	*System on a Chip*
SVM	*Support Vector Machine*
ULA	Unidade Lógica e Aritmética
VHDL	*Very high speed integrated circuit Hardware Description Language*
WS	*Weight-Stationary*

Lista de símbolos

TWh	Terawatt-hora
y	Saída de um perceptron
w_i	Peso da sinapse de um perceptron
x_i	Entrada do perceptron
b	Bias de um perceptron
O_m	Saída do m-ésimo neurônio de uma camada convolucional
b_m	*Bias* do m-ésimo neurônio de uma camada convolucional
N	Quantidade de filtros
R	Altura do filtro
S	Largura do filtro
I	Matriz de *ifmaps*
U_x	*Stride* no eixo x
U_y	*Stride* no eixo y
W_m	Matriz de pesos da m-ésima camada convolucional
E	Altura da matriz de *ofmaps*
F	Largura da matriz de *ofmaps*
H	Altura da matriz de *ifmaps*
W	Largura da matriz de *ifmaps*
λ	Comprimento de onda
nm	Nanômetro

i	Componente de iluminação
r	Refletância
GOP/s	Bilhões de operações por segundo
MHz	Megahertz
W	Watt
GOP/s/W	Bilhões de operações por segundo por watt
ms/frame	Milissegundos por quadro
mW	Miliwatt
TB	Terabyte
kB	Kilobyte
$\sigma(x)$	Função sigmoide
N_r	Quantidade de retas para aproximar a função sigmoide
P_r	Quantidade de pontos de borda das retas
p_i	Ponto i de borda
MAC_{PE}	Quantidade de MACs por PE
T_{PE}	Tempo de processamento em um PE
f_u	Fator de utilização dos PEs
N_c	Número de ciclos de processamento
$\#PE_{\text{MAC}}$	Quantidade de PEs realizando operações MAC
$\#PE_{\text{act}}$	Quantidade de PEs ativos
T_{mp}	Tempo de processamento do *max-pool*
E_m	Tamanho da janela de *pooling*
U_m	*Stride* da janela de *pooling*
I_{mp}	Tamanho da entrada a ser processada pelo *max-pool*

T_{re}	Tempo de processamento do bloco de reordenação
N_{re}	Quantidade de entradas na reordenação
I_{re}	Tamanho de cada entrada na reordenação
$Des.$	Desempenho em operações por segundo
$\#PE$	Número de PEs
f	Frequência em Hz
GHz	Gigahertz

Sumário

1 Introdução

Com o avanço do conceito de *Internet of Things* (IoT) e o crescimento de dispositivos conectados, a computação na borda surge como uma alternativa para o processamento local de dados, reduzindo latência e a dependência do processamento na nuvem. A quantidade de dispositivos IoT em 2016 atingiu um marco de 30 bilhões em 2016 e estimava-se que esse número dobraria até 2022 (GURUNATH et al., 2018). Segundo Satyanarayanan (2017), houve um crescimento no interesse em pesquisa e no investimento nesse paradigma de computação, com a criação de plataformas, padronizações e iniciativas reunindo diversas empresas para o desenvolvimento de aplicações na área.

A computação na nuvem consiste em um paradigma que oferece aos usuários recursos de computação, armazenamento de dados e rede para movimentação desses dados, já a computação na borda move tais etapas para a borda da rede, em outras palavras, para próximo dos dispositivos que geram os dados a serem processados. Em comparação à computação na nuvem, a na borda consegue possuir uma latência mais baixa, além de poder ser utilizada em aplicações móveis (e.g. dispositivos vestíveis), pode possuir uma arquitetura distribuída, garante mais segurança, possibilita distribuição geográfica, maior escalabilidade e confiabilidade, porém, possui recursos limitados (ASIM et al., 2020). Ademais, a utilização do paradigma de computação na borda também pode contribuir para a redução do consumo de energia nos *data centers* ao redor do mundo (VARGHESE et al., 2016), que consistiu em $416,2$ TWh em 2015 e tem crescido cada vez mais (Tom Bawden, 2016). Além disso, o aumento da quantidade de dispositivos na borda também eleva a quantidade de dados gerados e enviados à nuvem, impactando diretamente nos *data centers* e no tráfego entre esses e os dispositivos, cujo impacto pode ser reduzido se parte das análises e processamentos sobre os dados forem realizados na borda (VARGHESE et al., 2016).

O uso de algoritmos de *Machine Learning* (ML) também tem ganhado cada vez mais força e está fortemente relacionado à computação na nuvem para, por exemplo, executar modelos preditivos sobre grandes volumes de dados fazendo o uso de recursos da nuvem.

Porém, tais modelos também podem fazer o uso do paradigma de computação na borda, mesmo com as limitações de recursos, visto que esses modelos são normalmente custosos computacionalmente para se treinar e aplicar. Hua et al. (2023) enumera três ideias de aplicação de Inteligência Artificial (IA) na borda: dados são pré-processados na borda antes de serem enviados à nuvem, que realiza o treinamento dos modelos e os aplica; os modelos são aplicados na borda e o treinamento na nuvem; e, por fim, distribuir o treinamento e a aplicação entre a nuvem e a borda.

Com o advento do aprendizado profundo, os modelos têm ficado melhores, porém, cada vez mais complexos e maiores. Por exemplo, a rede neural ResNet-50 possui 50 camadas com mais de 25 milhões de pesos, realizando bilhões de operações (ARDAKANI; CONDO; GROSS, 2020). Como uma forma de acelerar a aplicação desses modelos, Struharik et al. (2020) dividem em dois tipos de soluções: aceleradores de hardware baseados em processadores *multicore*, como *Central Processing Units* (CPUs) e *Graphic Processing Units* (GPUs), e os aceleradores dedicados, por exemplo, em *Application-Specific Integrated Circuit* (ASICs) e *Field-Programmable Gate Arrays* (FPGAs). Silvano et al. (2023) também apontam a atual tendência de aceleração de Redes Neurais Artificiais (*Artificial Neural Networks* - ANNs) através de GPUs, bem como de aceleradores dedicados.

Nesse contexto, muitas arquiteturas dedicadas foram propostas para suprir tal demanda computacional, como Chen et al. (2016) que usam *Embedded Dynamic Random Access Memory* (eDRAM) como uma memória *on-chip* para aumentar a capacidade de armazenamento interno do chip e redução do tempo de acesso, Chi et al. (2016) fazem o uso do paradigma de processamento em memória (*Processing-in-memory* - PIM), fazendo alterações no *datapath* de uma memória para possibilitar a computação de uma rede neural e Shafiee et al. (2016) propõem a utilização de computação analógica para realizar as operações de multiplicação e acumulação. Também há as arquiteturas voltadas ao processamento de *Convolutional Neural Networks* (CNNs) na borda, como a ZASCA proposta por Ardakani, Condo e Gross (2020), a CoNNa de Struharik et al. (2020), a Light-OPU de Yu et al. (2020), entre outras.

Dentro do contexto dos trabalhos citados anteriormente, o presente trabalho apresenta uma nova proposta de arquitetura de hardware em FPGA. Essa arquitetura é capaz de realizar a aceleração de CNNs do tipo 1D, atuando no paradigma de computação na borda, com recursos limitados. A arquitetura é reconfigurável para processar os diferentes tipos de camadas contidas em uma CNN e escalável, podendo ter sua capacidade aumentada para processar redes de maior tamanho ou aumentar o grau de paralelismo. É capaz de

produzir ganhos de tempo de execução e baixa degradação da acurácia da classificação sem utilização excessiva dos recursos da FPGA. Para a avaliação dos resultados, um estudo de caso foi realizado aplicando-se diferentes topologias da arquitetura proposta no domínio de aplicações do sensoriamento remoto, especificamente na classificação de *Hyperspectral Images* (HSIs).

1.1 Problema de pesquisa

O sensoriamento remoto e, especificamente as HSIs, são uma área de grande importância na atualidade, podendo fornecer informações importantes através do monitoramento de recursos naturais e degradação do meio ambiente, por exemplo. Além disso, o paradigma de computação na borda pode ser utilizado como uma forma de levar o processamento dos dados de sensoriamento remoto para próximo do sensor que faz a coleta dos dados. Sendo assim, é possível reduzir a latência dos processos, atingir consumos de energia sustentáveis, lidar com o aumento da quantidade de dados gerados, reduzir tráfego de dados e fomentar o uso de técnicas inteligentes de computação (VARGHESE et al., 2016). Os avanços tecnológicos no domínio de hardware e nas técnicas de aprendizado de máquina têm permitido a utilização de IAs em sistemas embarcados para diferentes aplicações, mantendo-se uma relação entre desempenho e consumo energético (MAZZIA et al., 2020; SHADRIN et al., 2019; ZHANG, 2021). Com isso e considerando trabalhos como os de Gyaneshwar e Nidamanuri (2022), Fabelo et al. (2018) e Martins et al. (2019), os quais propõem formas de aceleração em hardware para a classificação de HSIs, o presente trabalho busca responder à seguinte pergunta de pesquisa:

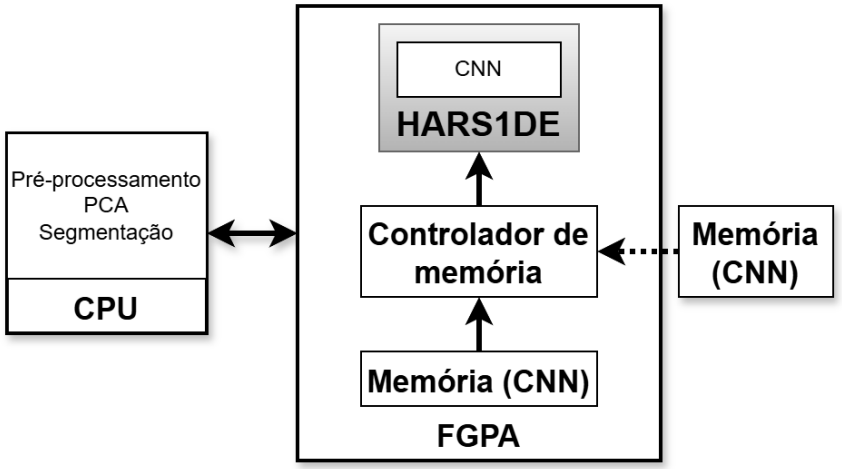
- Uma arquitetura de hardware reconfigurável e escalável em FPGA, voltada para a aceleração de redes neurais convolucionais 1D é capaz de produzir ganhos de desempenho (aceleração) em relação à sua execução em outros tipos de arquiteturas de hardware, com quantização reduzida sem impacto significativo na acurácia resultante e sem utilização acima de 80% de recursos de FPGA?

Para responder tal pergunta, os trabalhos de Viel et al. (2023) e Souza (2023) serão utilizados como base para teste e avaliação da arquitetura, fazendo o uso da rede neural utilizada nos trabalhos com adaptações para a portabilidade na arquitetura de hardware.

1.1.1 Solução Proposta

No presente trabalho, a solução proposta consiste em uma arquitetura contendo um acelerador de hardware (em FPGA). A Figura 1 mostra uma visão de alto nível da arquitetura, contendo o acelerador e componentes externos. A arquitetura consiste na implementação em hardware das operações utilizadas por CNNs-1D, onde os pesos são armazenados externamente em uma memória, podendo essa ser externa ou interna ao próprio chip para redução de latência. Já as entradas da rede são provenientes de um elemento externo, podendo ser, por exemplo, um processador, como representado na figura. Além disso, são representados exemplos de funcionalidades atribuídas aos blocos de processamento.

Figura 1: Vista de alto nível de aplicação da arquitetura proposta.



Fonte: Elaborado pelo autor (2024).

Considerando a solução da Figura 1 acima, são apresentadas as seguintes hipóteses em relação à arquitetura:

- **H0:** A arquitetura de hardware proposta para a aceleração do processamento de CNNs-1D é capaz de realizar a aplicação de um modelo com baixa perda de acurácia, baixa utilização de recursos de hardware e de forma acelerada;
- **H1:** A arquitetura de hardware proposta para a aceleração do processamento de CNNs-1D é capaz de realizar a aplicação de um modelo com baixa perda de acurácia, baixa utilização de recursos de hardware, porém, não é capaz de ser acelerada em comparação a outras plataformas de hardware;
- **H2:** A arquitetura de hardware proposta para a aceleração do processamento de CNNs-1D é capaz de realizar a aplicação de um modelo com baixa perda de acurácia,

de forma acelerada, porém, com alto consumo de recursos de hardware;

- **H3:** A arquitetura de hardware proposta para a aceleração do processamento de CNNs-1D gera perdas significativas de acurácia em relação à realização da classificação completamente em software.

1.1.2 Delimitação de escopo

Neste trabalho, o acelerador a ser implementado será limitado apenas à inferência dos modelos classificadores. O treinamento dos modelos será utilizado apenas como forma de comparação sobre o impacto de alterações nos modelos para portabilidade em hardware. Ademais, o foco de estudo será sobre o processo de classificação, sendo tarefas relacionadas à normalização, redução de dimensionalidade ou qualquer outra etapa de pré-processamento desconsideradas. Também não será abordada a hierarquia de memória diretamente ou barramentos externos para tráfego dos dados, considerando-se o modelo armazenado completamente em uma memória interna da arquitetura, onde serão utilizados pressupostos para a obtenção de uma aproximação teórica do tempo de execução da arquitetura.

1.2 Objetivos

Nas seções abaixo são enumerados e descritos os objetivos geral e específicos do presente trabalho.

1.2.1 Objetivo Geral

Este trabalho tem como objetivo principal acelerar a inferência em redes neurais convolucionais 1D através de uma arquitetura de hardware dedicada.

1.2.2 Objetivos Específicos

1. Identificar os paradigmas envolvidos na otimização da execução de redes neurais em hardware;

2. Implementar a arquitetura em *Very high speed integrated circuit Hardware Description Language* (VHDL) para obtenção de resultados de utilização de recursos e tempo;

3. Obter um modelo comportamental da arquitetura em uma linguagem com alto nível de abstração para a obtenção de resultados rápidos;

4. Avaliar o desempenho da arquitetura proposta em comparação às soluções implementadas apenas em software tomando métricas como: tempo de execução, acurácia e alocação de recursos de hardware.

1.3 Metodologia

O método escolhido para a realização do presente trabalho foi o hipotético-dedutivo, cujas hipóteses para a solução do problema proposto foram expostas na Seção 1.1.1. A aceitação ou rejeição dessas será dada através de experimentações para a obtenção de dados para análise posterior. Quanto aos objetivos, a pesquisa é caracterizada como exploratória e, quanto à abordagem do problema, é de caráter quantitativo.

1.3.1 Procedimentos Metodológicos

As etapas abaixo representam os procedimentos metodológicos utilizados para a realização deste trabalho:

- **Revisão bibliográfica:** Etapa direcionada à investigação e análise na literatura existente acerca da base teórica do trabalho. Os principais temas buscados foram: Aprendizado de máquina, arquiteturas de hardware para aprendizado de máquina e computação na borda.

- **Revisão sistemática da literatura do estado da arte:** Nesta etapa foram identificados e avaliados trabalhos semelhantes sob critérios de inclusão e exclusão para a realização das devidas comparações com essa pesquisa.

- **Avaliação das bases de dados:** As bases de dados que serão utilizadas para a obtenção dos resultados foram apresentadas e analisadas sob critérios relacionados ao tamanho da base e distribuição das classes.

- **Desenvolvimento da arquitetura para CNN:** Nesta etapa, a partir dos trabalhos obtidos através da revisão bibliográfica e sistemática, foi projetada a arquitetura de hardware responsável pela aceleração dos processos envolvidos nas redes neurais convolucionais 1D. Ademais, foram desenvolvidas uma representação da arquitetura

em Python para a obtenção de resultados rápidos e avaliações da arquitetura, assim como uma representação em VHDL para a obtenção de resultados referentes à utilização de recursos, frequência de operação e comportamento dos blocos de hardware.

- **Testes e refinamento:** Etapa dedicada à realização de testes na arquitetura para a correção de erros e acréscimo de funcionalidades.
- **Obtenção de resultados:** Etapa relacionada à obtenção de resultados através de simulações sobre os modelos de hardware realizados.
- **Análise dos resultados e escrita:** Após a obtenção dos resultados, essa etapa consistiu na análise dos resultados para constatar a viabilidade da arquitetura, a aceitação ou rejeite das hipóteses propostas e a finalização da escrita desse trabalho.

1.4 Organização do trabalho

Este trabalho está organizado em sete capítulos, sendo o Capítulo 1 responsável pela contextualização dos temas pesquisados, justificativa do trabalho, apresentação do problema de pesquisa, breve descrição da proposta de solução, objetivos geral e específicos e a metodologia utilizada.

O Capítulo 2 apresenta a fundamentação teórica do trabalho, bem como as fontes utilizadas. O capítulo apresenta os conceitos de: aprendizado de máquina, hardwares para aprendizado de máquina e imagens hiperespectrais.

No Capítulo 3, é apresentada a revisão sistemática realizada para a seleção dos trabalhos considerados como o estado da arte no tema pesquisado.

No Capítulo 4, são apresentadas as principais características da arquitetura de hardware proposta para a solução do problema apresentado.

No Capítulo 5, é apresentado um estudo de caso de aplicação da arquitetura para a classificação de HSIs e os materiais e métodos utilizados, consistindo na descrição das imagens utilizadas e dos processos realizados para a obtenção dos resultados.

O Capítulo 6 apresenta os resultados obtidos a partir dos experimentos realizados, bem como a discussão desses.

Por fim, o Capítulo 7 apresenta as considerações finais do trabalho, principais contribuições e trabalhos futuros.

2 Referencial Teórico

Neste capítulo, são apresentados os principais conceitos e referências para o entendimento do trabalho. Alguns temas são cerceados para se deterem apenas ao que é utilizado no trabalho. Inicialmente, são exploradas as definições acerca do aprendizado de máquina e os algoritmos utilizados nesse trabalho. Em seguida, são abordados os conceitos e paradigmas do projeto de hardware para a otimização de redes neurais. Por fim, são apresentados os conceitos sobre as imagens hiperespectrais.

2.1 Aprendizado de máquina

"Aprendizado de máquina é geralmente entendido como procedimentos automáticos de computação baseados em operações lógicas ou binárias que aprendem uma tarefa a partir de uma série de exemplos"(MICHIE; SPIEGELHALTER; TAYLOR, 1994, tradução nossa). O processo de aprendizado pode ser dividido em duas etapas: treinamento e teste. A primeira consiste na criação do modelo matemático responsável pela execução do procedimento proposto, e a segunda na aferição da qualidade do modelo. A área de aprendizado de máquina é normalmente subdividida entre os diferentes tipos de aprendizado, sendo a divisão mais comum em três tipos: aprendizado supervisionado (*supervised learning*), não-supervisionado (*unsupervised learning*) e aprendizado por reforço (*reinforcement learning*) (SAH, 2020).

O aprendizado supervisionado cria um modelo que mapeia uma determinada entrada para uma saída desejada. O modelo é treinado utilizando-se exemplos previamente mapeados com entradas e saídas definidas (NASTESKI, 2017). Tal método de aprendizado é utilizado em tarefas de classificação e regressão. A primeira diz respeito aos problemas em que se deseja classificar a entrada do modelo dentre múltiplas variáveis (ou classes) categóricas. Alguns exemplos de algoritmos que aprendizado de máquina capazes de realizar a classificação são: Árvore de Decisão (*Decision Tree* - DT), Floresta Aleatória (*Random Forest* - RF), *k-Neares Neighbors* (k-NN) e Máquina de Vetores de Suporte (*Support Vec-*

tor Machine - SVM). Em relação à regressão, o modelo criado é capaz de prever valores contínuos em vez de um número limitado de classes. Alguns dos algoritmos de classificação também podem ser usados para resolver problemas de regressão, como o k-NN e RF, por exemplo. Além disso, existem algoritmos específicos para tal, como o de regressão linear, regressão polinomial e regressão Lasso.

No aprendizado não supervisionado, não se dispõe de exemplos previamente classificados para a criação do modelo. Sendo assim, os algoritmos de forma autônoma devem ser capazes de encontrar padrões nos dados (SAH, 2020). Esse tipo de aprendizado é normalmente utilizado para problemas de agrupamento, onde a partir dos dados disponíveis, são criados grupos que separam os dados utilizando-se de algum parâmetro de semelhança entre esses. Em relação aos algoritmos, pode-se citar o *K-means* e o agrupamento hierárquico.

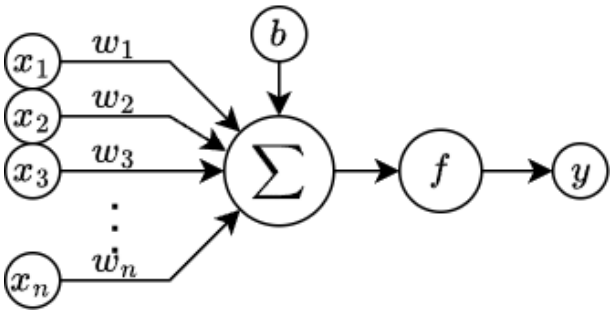
Segundo Kaelbling, Littman e Moore (1996, tradução nossa), "O aprendizado por reforço é o problema enfrentado por um agente que deve aprender o comportamento por meio de interações de tentativa e erro com um ambiente dinâmico". Dependendo do resultado da interação, o agente pode ser recompensado ou penalizado, sendo o objetivo maximizar as recompensas. De forma análoga ao aprendizado não supervisionado, o agente não necessita de conhecimento prévio de classes e deve ser capaz de explorar o ambiente. Dentre os algoritmos desta categoria, têm-se como exemplos: *Q-Learning*, *Deep Q Network* (DQN), *Soft Actor-Critic* (SAC) etc.

Além das três divisórias de formas de aprendizado citadas anteriormente, na literatura é possível encontrar soluções híbridas como o aprendizado semi-supervisionado e o auto-supervisionado, assim como outras abordagens como aprendizado profundo, aprendizado Bayesiano, comitês de classificadores etc. (SAH, 2020). O presente trabalho lida com algoritmos de aprendizado supervisionado e aprendizado profundo, sendo os algoritmos apresentados nas seções a seguir.

2.1.1 Redes Neurais Artificiais

As ANNs são modelos computacionais que podem surgir a partir das diferentes formas de aprendizados citadas anteriormente. As redes neurais convencionais são formadas por unidades básicas de processamento chamadas de *perceptrons*, ou neurônios artificiais, propostas por Rosenblatt (1958), tendo como base o funcionamento de um neurônio biológico. A Figura 2 mostra a representação de um perceptron e como as entradas (x) são processadas para se obter a saída (y).

Figura 2: Representação de um *perceptron*



Fonte: Elaborado pelo autor (2023)

A saída de um neurônio é dada por:

$$y = f\left(\sum_{i=1}^{n}(w_i \times x_i) + b\right) \tag{2.1}$$

onde n é a quantidade de entradas, x_i são as entradas, w_i são os pesos das sinapses (conexões), b o termo de viés (*bias*) e f a função de ativação do neurônio. O termo *bias* permite ajustar o limiar de ativação do neurônio, assim como permite à rede aprender padrões mais complexos. As funções de ativação determinam o sinal total que um determinado neurônio tem em suas entradas, que é representado por um vetor, e produzem um escalar em sua saída (APICELLA et al., 2021). A Tabela 1 mostra alguns exemplos de funções de ativação.

O conjunto de neurônios interconectados forma uma rede neural, onde a Figura 3 mostra a constituição de uma rede e alguns tipos diferentes de redes. Uma rede neural convencional possui uma camada de entrada, onde os dados (ou instâncias) são inseridos para serem processados na rede. A camada escondida contém os neurônios (Figura 2) que irão de fato realizar o processamento, produzindo resultados na camada de saída. As redes podem possuir várias camadas escondidas, como pode ser observado em (b). Quando todos os neurônios de uma camada possuem sinapses com todos os neurônios ou entradas da camada anterior, a rede é dita completamente conectada (*Fully Connected* - FC), como é o caso da Figura 3 em (a). Porém, as redes podem passar pelo processo de poda sob algum critério para a exclusão de sinapses ou neurônios menos relevantes, como é possível observar em (b). O fluxo normal de execução de uma rede neural consiste nos dados serem processados da entrada para a saída (*Feedfoward*), porém, existem redes bidirecionais, onde a saída de um neurônio pode ser transmitida para camadas anteriores da rede influenciando os resultados quando a rede for utilizada novamente, sendo chamadas

Tabela 1: Exemplos de funções de ativação

Nome	**Expressão**	**Intervalo**
Identidade	$id(a) = a$	$(-\infty, +\infty)$
Degrau (Heaviside)	$U(a) = \begin{cases} 0, \text{ se } a < 0 \\ 1, \text{ se } a \geq 0 \end{cases}$	[0,1]
Bipolar	$B(a) = \begin{cases} -1, \text{ se } a < 0 \\ +1, \text{ se } a \geq 0 \end{cases}$	[-1,1]
Sigmoide	$\sigma(a) = \frac{1}{1+e^{-a}}$	[0,1]
Tangente Hiperbólica	$tanh(a)$	[-1,1]
Modular	$abs(a) = \lvert a \rvert$	$[0, +\infty)$
Cosseno	$cos(a)$	[-1,1]
Retificador (ReLU)	$f(a) = \begin{cases} a, \text{ se } a > 0 \\ 0, \text{ se } a \leq 0 \end{cases}$ ou $f(a) = max(0, a)$	$[0, \infty[$
Softmax	$s(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$	[0,1]

Fonte: Adaptado de Apicella et al. (2021)

de Redes Neurais Recorrentes (*Recurrent Neural Networks* - RNNs) (b).

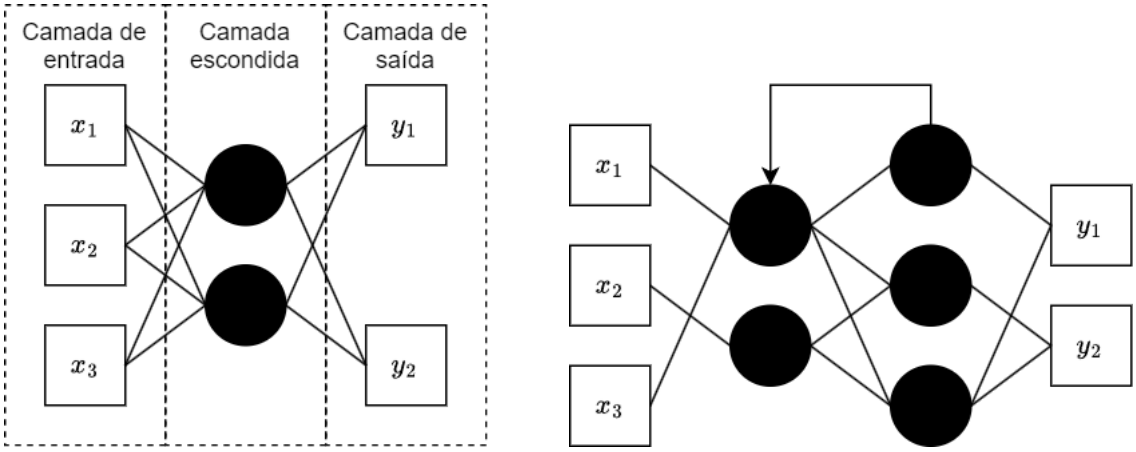
2.1.2 Redes Neurais Convolucionais

O conceito de aprendizado profundo (*Deep Learning* - DL) consiste em métodos eficazes para tarefas de grande complexidade e volume de dados, sendo capazes de extrair características dos dados de entrada progressivamente através de várias camadas de processamento, sendo as topologias mais comuns as Redes Neurais Profundas (*Deep Neural Networks* - DNNs) e Transformers (SILVANO et al., 2023). Dentro das DNNs, tem-se as *Multi-Layer Perceptrons* (MLPs), CNNs e RNNs.

As MLPs são as redes neurais compostas apenas por várias camadas de neurônios, expostas na seção anterior, assim como as RNNs. As CNNs também são compostas por múltiplas camadas com o objetivo de extração e classificação dos dados de entrada, onde esses dados são transformados ao passar pelas camadas, se tornando representações abstratas de suas características, sendo estas características chamadas de *feature maps* (*fmaps*) (CHEN et al., 2017). As redes convolucionais possuem diversos tipos de camadas: convolucionais, *pooling*, normalização, FCs (*dense*), *flatten*, dentre outras (ZHANG et al., 2016).

A Figura 4 mostra o exemplo da CNN LeNet utilizada para a classificação de dígitos em uma imagem de 28×28 pixels. A primeira camada de convolução aplica 6 filtros (*kernels*) para se obter 6 *fmaps* de 28×28. Em seguida, uma camada de *pooling* realiza

Figura 3: Exemplos de redes neurais



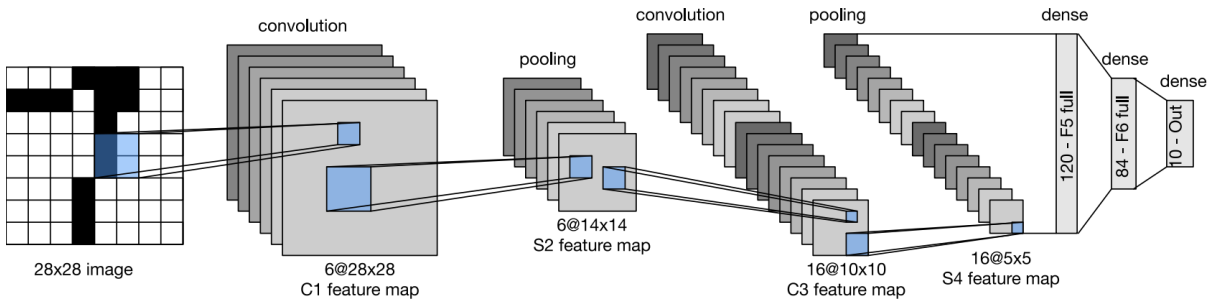
(a) Rede neural completamente conectada e suas camadas

(b) Rede neural recorrente, podada e com múltiplas camadas escondidas

Fonte: Elaborado pelo autor (2023)

a subamostragem das *fmaps*, reduzindo suas dimensões para 14×14. As *fmaps* reduzidas passam por uma nova camada de convolução contendo 16 filtros que combinam as *fmaps* anteriores para se obter características mais abstratas. Novamente, uma camada de *pooling* é utilizada e reduz as *feature maps* para dimensões 5×5.

Figura 4: Fluxo de dados da CNN LeNet



Fonte: Zhang et al. (2023)

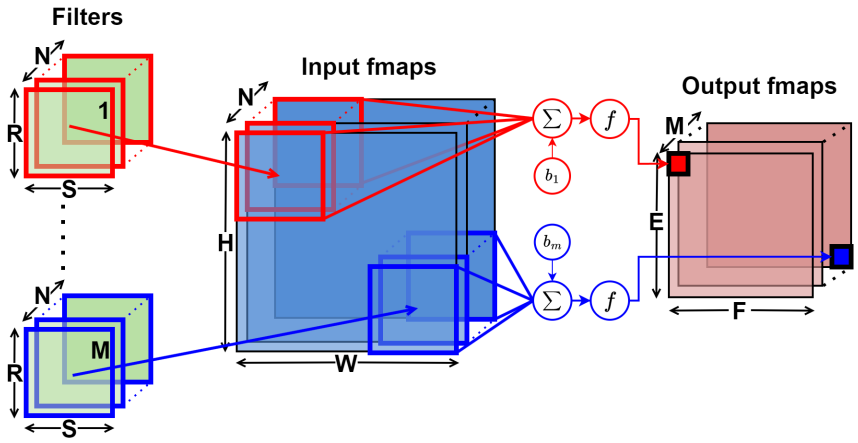
Vale ressaltar que mesmo com as reduções, as *fmaps* finais devem conter as informações relevantes para o processo de classificação. Apesar de não ser exposto na Figura 4, após a última camada de *pooling*, é realizada a operação *flatten*, a qual utiliza as 16 matrizes de dimensão 5×5 e transforma em um vetor com 400 posições $(5 \times 5 \times 16)$, o qual se torna a entrada das últimas camadas (*dense*). Esta parte das camadas é responsável pela classificação, tendo-se uma camada inicial com 120 neurônios e outra com 84, encerrando-se com 10 saídas, correspondentes à classificação dos dígitos de 0 a 9. As

camadas convolucionais e de *pooling* serão detalhadas nas próximas seções.

2.1.2.1 Camadas Convolucionais

As camadas convolucionais (Figura 5) consistem na aplicação de filtros sobre *feature maps* que entram na camada (*input fmaps* ou *ifmaps*) através da operação matemática de convolução, a qual será explicada sob a ótica da convolução 2D, onde os filtros são matrizes bidimensionais cujos valores são chamados de pesos (ou *weights*) (SONG et al., 2017). Na operação de convolução 2D, o filtro é deslizado sobre as *ifmaps*, realizando-se a multiplicação dos valores do filtro pelos da *ifmap* e somando-se o resultado das multiplicações para se gerar uma soma parcial (*psum*) que, ao ser somada com o resultado de outras convoluções, gera um ponto na matriz de saída (*output fmaps* ou *ofmaps*). Opcionalmente, o termo de *bias* também pode aparecer para compor o resultado final, sendo acrescentado às somas de convoluções. O resultado pode então ser passado por uma função de ativação (Tabela 1), analogamente às ANNs, e sua saída compõe a *ofmap*. Após a computação de um ponto de saída, o filtro é deslizado para uma nova posição sobre a *ifmap* e as operações se repetem. É importante ressaltar a existência do parâmetro de passo (*stride*) na operação de convolução o qual diz respeito à quantidade de posições que o filtro irá se deslocar, sendo valores maiores repensáveis por uma redução das dimensões das *ofmaps* em relação às *ifmaps* e aumento da eficiência do processo computacional das operações (ZHANG et al., 2023).

Figura 5: Computação de uma camada convolucional



Fonte: Adaptado de Chen et al. (2017)

Na Figura 5, é apresentado um modelo geral contendo estruturas de filtros, *ifmaps* e *ofmaps* tridimensionais, onde R e S consistem nas dimensões dos filtros, H e W são as dimensões das *ifmaps* e E e F das *ofmaps*. N representa a quantidade de filtros da

camada e a quantidade de *ifmaps*. M consiste na quantidade de *ofmaps* e na quantidade de filtros ($R \times S \times N$) utilizados. Dessa forma, a posição (x, y) da m-ésima matriz de *ofmaps* ($\mathbf{O}_m$) é dada por:

$$\mathbf{O}_m[x][y] = f\left(b_m + \sum_{k=0}^{N-1}\sum_{i=0}^{R-1}\sum_{j=0}^{S-1}\mathbf{I}[k][Ux+i][Uy+j] \times \mathbf{W}_m[k][i][j]\right) \tag{2.2}$$

onde $\mathbf{I}$ corresponde à matriz de *ifmaps*, $\mathbf{W}_m$ a m-ésima matriz de pesos, U ao *stride*, f à função de ativação e b_m o m-ésimo termo do *bias*. Assim como os *perceptrons*, as camadas convolucionais também podem possuir um termo de *bias* Além disso, as dimensões $E \times F$ da saída são dadas por:

$$E = \frac{H - R + U}{U} \tag{2.3}$$

$$F = \frac{W - S + U}{U} \tag{2.4}$$

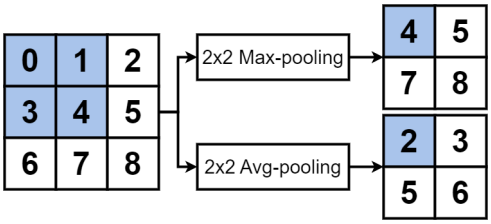
Resumidamente, a saída da camada de uma rede neural convolucional se dá através da soma de N convoluções sobre N entradas. Esse processo se repete M vezes para se obter M saídas da camada, que pode então se tornar a entrada de uma camada seguinte. Um vetor de *bias* de M posições também pode ser incluído, sendo adicionado aos resultados das convoluções.

2.1.2.2 Camadas de *Pooling*

As camadas de *pooling* reduzem a quantidade de *fmaps* (ZHANG et al., 2016), mesclando características semelhantes em uma mesma característica (LECUN; BENGIO; HINTON, 2015). Analogamente às camadas convolucionais, as camadas de *pooling* também funcionam com uma matriz deslizando sobre a *ifmap* da camada, porém, com operações diferentes da convolução. Normalmente, as operações de *pooling* utilizam o valor máximo (*maximum pooling* ou *max-pooling*) ou o valor médio (*average pooling*) (ZHANG et al., 2023). A Figura 6 mostra um exemplo do funcionamento desse tipo de camada para o caso das duas operações descritas anteriormente. Uma matriz 2x2 desliza sobre a entrada 3×3, onde, para o caso do *max-pooling* para cada posição da matriz 2×2, é calculada a operação de máximo, ou $max(0, 1, 3, 4) = 4$ e é preenchida a matriz de saída. Já o *average pooling* calcula a média entre os quatro valores na posição atual da matriz 2×2, ou $avg(0, 1, 3, 4) = 2$. Dessa forma, a matriz 3×3 de entrada foi reduzida para 2×2, con-

tendo ainda informações relevantes sobre as *feature maps*. Vale ressaltar que para valores maiores de *stride*, a matriz de saída pode ter um tamanho ainda menor em relação à de entrada, sendo que as dimensões também podem ser calculadas pelas Equações 2.3 e 2.4.

Figura 6: Exemplo da aplicação da operação de *average* e *maximum pooling*



Fonte: Adaptado de Zhang et al. (2023)

2.1.2.3 Redes Neurais Convolucionais 1D

As redes convolucionais 1D realizam convoluções unidimensionais sobre os dados trabalhados. Em comparação à CNN-2D, a 1D pode possuir complexidade inferior, são mais fáceis de treinar e realizar e são adequadas para aplicações de baixo custo e tempo real (KIRANYAZ et al., 2021). A Equação 2.5 pode ser obtida a partir da Equação 2.2 excluindo-se uma das dimensões (R e H), o que resulta na exclusão de um dos somatórios.

$$\mathbf{O}_m[y] = f\left(\sum_{k=0}^{N-1}\sum_{j=0}^{S-1}\mathbf{I}[k][U_y + j] \times \mathbf{W}_m[k][j] + b_m\right) \tag{2.5}$$

Analogamente, cada saída de uma camada convolucional consiste na soma de múltiplas convoluções 1D acrescidas do termo de *bias* e utilização da função de ativação. Além disso, o *stride* é dado apenas em uma dimensão (ao longo do vetor de *ifmap*).

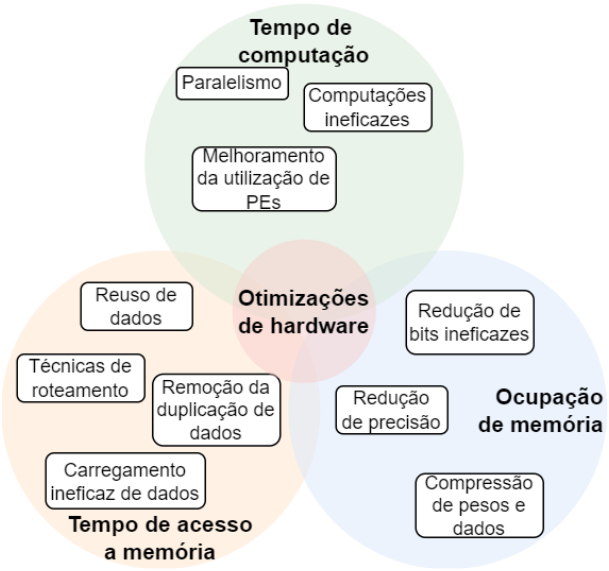
2.2 Hardware para CNNs

Existem paradigmas que são utilizados para otimizar a execução de algoritmos em arquiteturas de hardware, onde as aplicações de ML também tomam proveito desses paradigmas. "Independentemente da tecnologia, um problema comum no projeto de aceleradores é o custo de energia e latência ao acessar a memória *Dynamic Random Access Memory* (DRAM) externa, especialmente considerando a quantidade significativa de dados que as aplicações *High-Performance Computing* (HPC) alvo precisam processar"(SILVANO et al., 2023, tradução nossa). O custo energético para acessar uma memória DRAM pode

ser 200× maior em comparação ao acesso de registradores internos à arquitetura (MOOLCHANDANI; KUMAR; SARANGI, 2021).

Moolchandani, Kumar e Sarangi (2021) divide as otimizações de hardware em três áreas: tempo de computação, tempo de acesso à memória e ocupação da memória (Figura 7).

Figura 7: Diagrama de Venn das possíveis otimizações de hardware



Fonte: Adaptado de Moolchandani, Kumar e Sarangi (2021)

As operações de multiplicação e acumulação de uma rede neural podem ser paralelizadas e, para isso, podem ser usadas arquiteturas temporais ou espaciais (SZE et al., 2017). A primeira utiliza um grande número de Unidades Lógicas e Aritméticas (ULAs) sem memória local e sem comunicação direta entre si, possuindo um controle centralizado, sendo normalmente utilizada em *Central Processing Units* (CPUs) e GPUs. Nas arquiteturas espaciais, cada ULA possui sua própria memória e lógica de controle, sendo chamadas de elementos de processamento (*Processing Elements* - PEs), sendo capazes de trocar dados entre si (GHIMIRE; KIL; KIM, 2022).

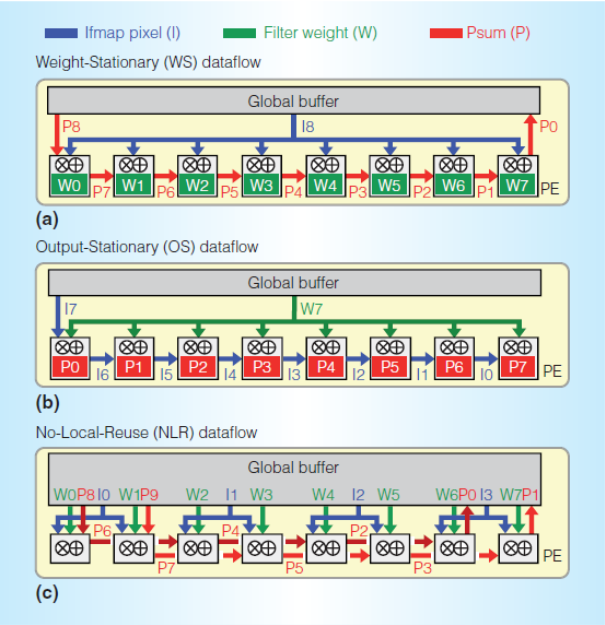
Os modelos de hardware espaciais podem aumentar o reuso de dados. Dentre os tipos de reuso, pode-se citar o reuso convolucional, o de *ifmaps* e o de filtros (CHEN et al., 2017). No primeiro, uma mesma entrada de uma camada convolucional é utilizada com os mesmos pesos de filtros, porém, em combinações diferentes. O reuso de *ifmaps* ocorre quando múltiplos filtros são aplicados a uma mesma entrada, ou seja, os dados da entrada podem ser reutilizados em várias operações com os diferentes filtros. Por fim, o reuso de filtros ocorre quando se tem múltiplas entradas processadas em *batch* (lote) com um único

filtro (SZE et al., 2017).

2.2.1 Fluxos de dados

Os fluxos de dados (*dataflows*) lidam com os padrões de movimentação dos dados dentro da arquitetura e como os recursos são alocados, lidando com *trade-offs* entre desempenho e complexidade da implementação (CHEN; EMER; SZE, 2017). A Figura 8 mostra três tipos de fluxos de dados comumente utilizados, sendo o de peso estacionário (*Weight-Stationary* - WS), saída estacionária (*Output-Stationary* - OS) e sem reuso local (*No-Local-Reuse* - NLR). Em azul, têm-se as *ifmaps*, que são os valores de entrada, em verde os pesos dos filtros e em vermelho as somas parciais das operações de convolução.

Figura 8: Tipos comuns de fluxos de dados



Fonte: Chen, Emer e Sze (2017)

No *dataflow* WS, os pesos dos filtros são fixados e armazenados nas memórias internas dos PEs, havendo movimentação das *ifmaps*, que são carregadas da hierarquia de memória para os PEs. As somas parciais (*Psums*) são então movimentadas entre as PEs. Esse tipo de fluxo de dados maximiza o reuso convolucional e o de filtros (GHIMIRE; KIL; KIM, 2022). Nesse caso, os recursos são mapeados considerando a regra de que todas as operações de multiplicação e acumulação que utilizam o mesmo peso de filtro são alocadas em um mesmo *Processing Element* (PE) (CHEN; EMER; SZE, 2017).

Para o fluxo de dados OS, as somas parciais são mantidas de forma estacionária nos PEs, sendo acumuladas internamente. Os pesos dos filtros são carregados da memória

para os PEs e as *ifmaps* são processadas serialmente, sendo passadas para os demais PEs. A regra de mapeamento dos recursos consiste que as operações de multiplicação e acumulação que geram somas parciais para um mesmo valor de *ofmap* sejam mapeadas em um mesmo PE (CHEN; EMER; SZE, 2017).

No caso em que não há reuso local, não há armazenamento de dados nos PEs, abrindo margem para expandir outras memórias da hierarquia, como por exemplo, o *buffer* global (CHEN; EMER; SZE, 2017).

Além dos *dataflows* citados anteriormente, Chen et al. (2017) propõem o tipo linha estacionária (*Row Stationary* - RS), o qual é capaz de reduzir a movimentação dos dados, reutilizando *ifmaps*, pesos e somas parciais ou *ofmaps*. É o fluxo de dados com menor consumo energético onde a operação de convolução entre uma linha da matriz do filtro e uma linha da matriz *ifmap* é alocada em um mesmo PE (GHIMIRE; KIL; KIM, 2022).

2.3 Imagens Hiperespectrais

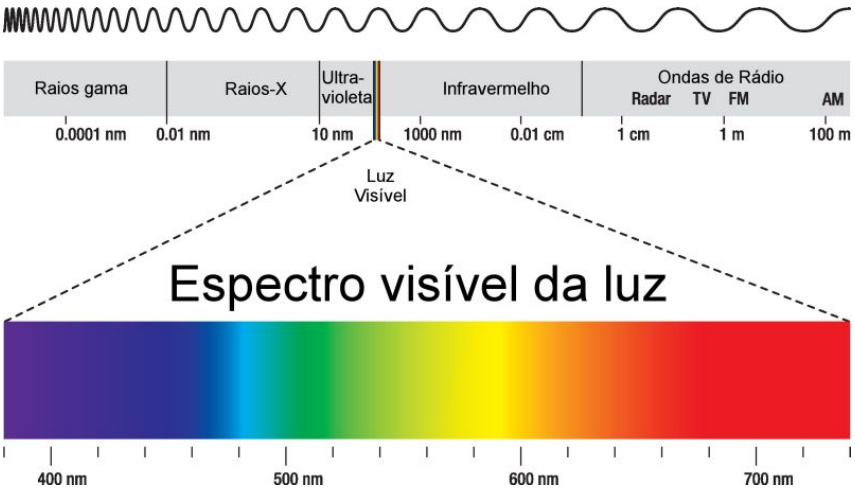
Um feixe de luz pode ser entendido como uma onda eletromagnética, assim como o infravermelho e ultravioleta. Com o passar do tempo, foram descobertas as ondas de rádio, com vasta aplicação. A Figura 9 mostra o espectro eletromagnético (Arco-íris de Maxwell) em função do comprimento de onda (λ). Ondas localizadas à esquerda possuem comprimento de onda menor (frequência maior) e à direita comprimento de onda maior (frequência menor). O espectro visível corresponde ao intervalo de comprimentos de onda ao qual o olho humano é sensível, cujo intervalo é aproximadamente entre 400 e 700 nm (HALLIDAY; RESNICK; WALKER, 2011).

Uma imagem (Figura 10) pode ser definida como uma função $f(x, y)$ que representa a intensidade da imagem nas coordenadas x e y e é resultado do produto entre uma componente de iluminação (i) proveniente da fonte de iluminação e a refletância (r) do objeto (Equação 2.6). A iluminação pode assumir um valor entre zero e infinito, já a refletância é limitada ao intervalo entre 0 e 1, sendo o valor mínimo a representação da total absorção da luz pelo objeto da cena e o máximo a total reflexão da energia emitida pela fonte (GONZALEZ; WOODS, 2002).

$$f(x, y) = i(x, y)r(x, y) \tag{2.6}$$

Quando a função possui um valor finito de possíveis valores (discreta), a imagem é

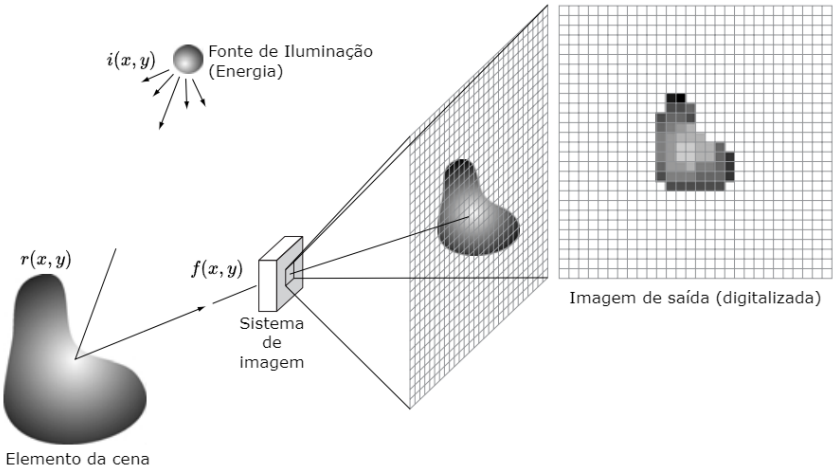
Figura 9: Espectro eletromagnético.



Fonte: Furian (c2023).

dita digital. Os sistemas computacionais normalmente utilizam quantizações de 8 bits, correspondendo a 256 possíveis valores de intensidade (GONZALEZ; WOODS, 2002). Imagens rasterizadas são representadas de forma matricial, onde cada elemento é chamado de pixel (*picture element*). A resolução de uma imagem é dada pela quantidade de pixels que formam as colunas e linhas da matriz representativa da imagem (e.g. 1650×1080) (FOLEY; DAM, 1982).

Figura 10: Aquisição de uma imagem digital

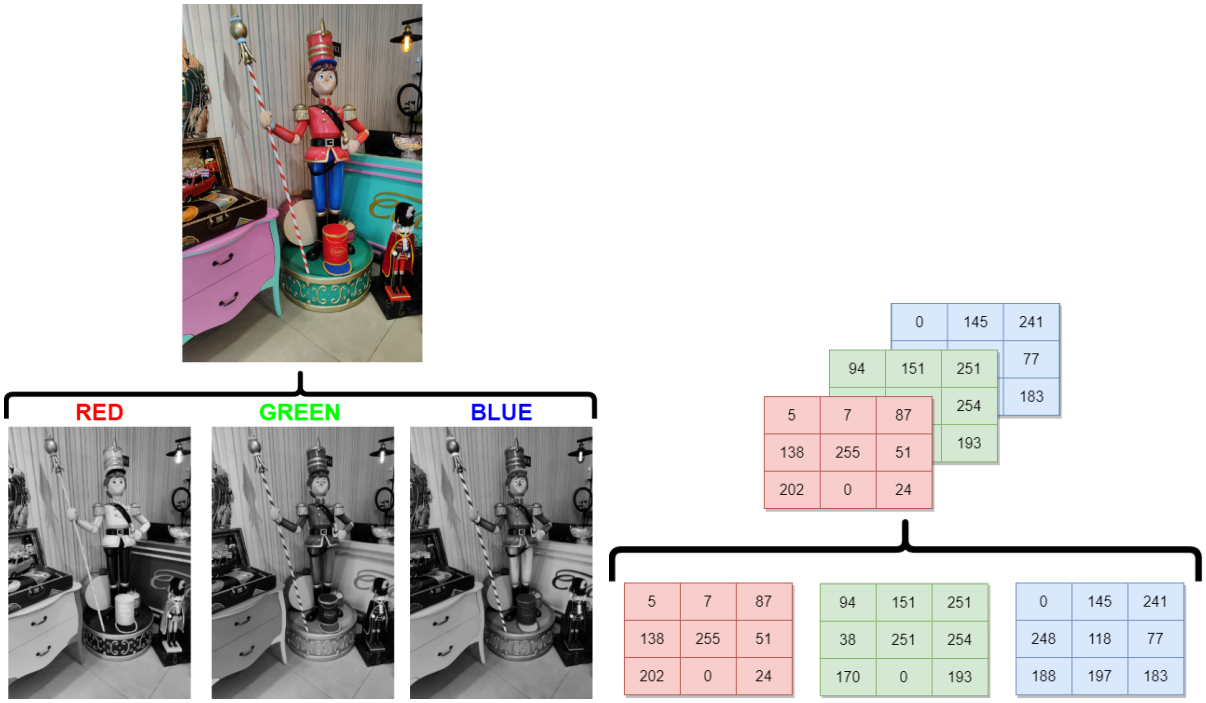


Fonte: Adaptado de Gonzalez e Woods (2002).

As imagens digitais comuns utilizam três canais contendo informações das cores que a compõem, sendo os canais: vermelho (*red)*, verde (*green*) e azul (*blue*), formando o padrão de cores RGB. Cada pixel da imagem contém essas três informações e as cores (não os valores) dos três canais se somam para compor a imagem colorida. A Figura 11 mostra

como é feita a decomposição das cores de uma imagem, onde cada canal monocromático (a) forma uma escala que começa no preto, tendo como limite superior o branco, formando uma escala de cinza (GONZALEZ; WOODS, 2002). Observando-se cada canal individual, as regiões mais claras indicam maior presença da cor do canal. Exemplificadamente, a roupa do personagem da figura é vermelha, aparecendo de forma branca no canal vermelho e de forma mais escura nos demais. A representação matricial em (b) mostra como os dados são numericamente os dados que compõem uma imagem, para um caso de quantização de 8 bits, cuja escala começa no zero (preto) e termina no branco (255).

Figura 11: Representação de uma imagem e seus canais.



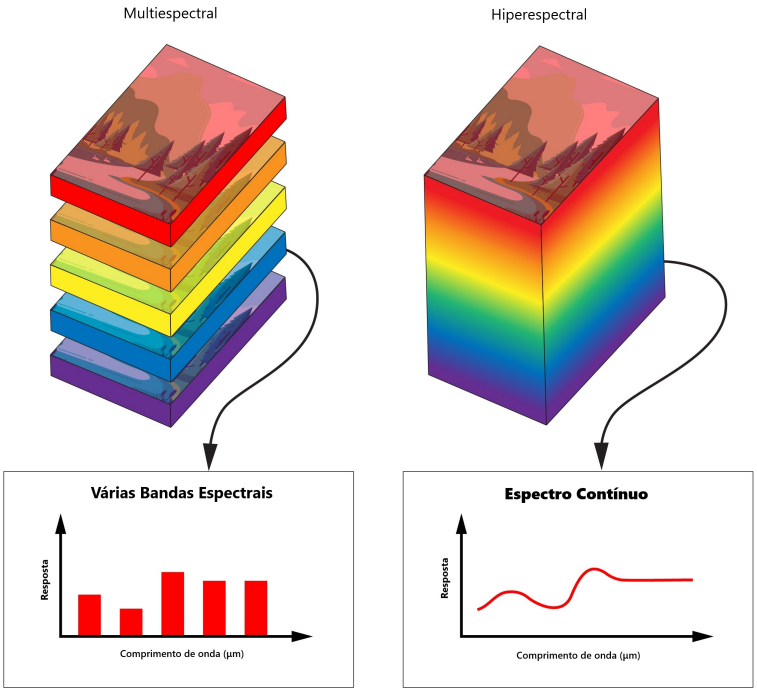
(a) Representação em forma de imagem. (b) Representação matricial.

Fonte: Elaborado pelo autor (2023).

As imagens hiperespectrais são utilizadas no âmbito do sensoriamento remoto e, diferente das imagens comuns, são representadas por uma maior quantidade de canais (espectros ou bandas). Enquanto imagens coloridas e multiespectrais possuem entre 3 e 10 bandas, imagens hiperespectrais podem ser representadas por centenas de bandas (CHANG, 2007). A Figura 12 mostra uma comparação visual entre MSIs e HSIs e nesta última, a cena é adquirida em um espectro "contínuo", com um menor espaçamento entre os comprimentos de onda coletados.

Uma forma de representação de uma HSI é através do cubo hiperespectral (Figura 13), com dimensões espaciais (x, y) e uma terceira dimensão correspondendo ao espectro

Figura 12: Diferenciação entre MSI e HSI



Fonte: Adaptado de Edmund Optics (c2023).

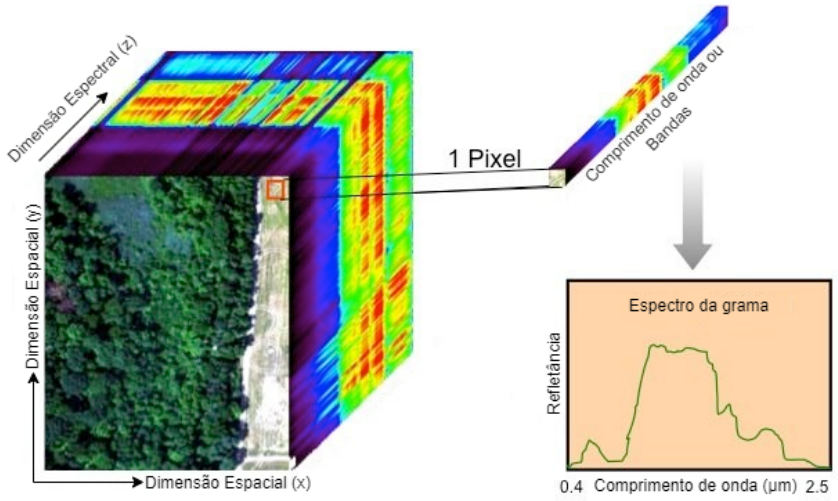
(z). Assim como uma imagem colorida, a HSI possui uma resolução espacial referente às dimensões espaciais e que diz respeito ao menor objeto que pode ser representado de forma distinguível, fazendo referência à quantidade de pixels que compõem a imagem. De forma análoga, a resolução espectral associada ao eixo z relaciona o intervalo de aquisição de comprimentos de onda e a quantidade de comprimentos coletados. Quanto menor o espaçamento espectral entre os comprimentos adquiridos, maior a resolução espectral (KHAN et al., 2018).

Enquanto em uma imagem colorida, um pixel contém a informação das cores RGB, em uma HSI, um pixel guarda todos os comprimentos de onda coletados pelo sensor (Figura 13), ou seja, há uma maior quantidade de dados, por exemplo, para se realizar inferências acerca do elemento da cena representado no pixel. No caso da Figura 13, há o exemplo do espectro que compõe a grama na cena, assim como a água ou a areia podem ter suas próprias assinaturas espectrais.

2.3.1 Classificação de Imagens Hiperespectrais

Considerando o cubo hiperespectral da Figura 13, modelos de aprendizado de máquina podem ser utilizados para a classificação de objetos na imagem. No processo de classifi-

Figura 13: Cubo hiperespectral e composição de um pixel.



Fonte: Adaptado de Ziemann (2015).

cação, podem ser utilizadas informações espectrais e/ou espaciais. Características como textura e formas são informações espaciais (ZHAO; YAN; HUANG, 2023), as quais podem ser extraídas através de redes neurais convolucionais, especificamente, camadas convolucionais do tipo 2D (Seção 2.1.2.1). CNNs 1D (Seção 2.1.2.3) podem ser usadas para explorar apenas a informação espectral, realizando a classificação de cada pixel. Analogamente, CNNs 3D podem ser utilizadas para a extração de informação espectral-espacial (VIEL et al., 2023).

3 Trabalhos Relacionados

Neste capítulo são apresentados os artigos relacionados ao tema de pesquisa deste trabalho (arquiteturas de hardware para a aceleração de CNNs-1D na borda). Esse capítulo apresenta a revisão sistemática realizada para a obtenção dos trabalhos a serem comparados, um breve resumo dos trabalhos selecionados e uma análise comparativa entre esses e o presente trabalho.

3.1 Revisão Sistemática

Esta seção apresenta a metodologia utilizada para a seleção dos trabalhos relacionados. Um protocolo de busca foi elaborado para filtrar os trabalhos retornados pelos repositórios eletrônicos dentro do escopo deste trabalho. Esse protocolo foi modelado considerando-se as perguntas de pesquisa mostradas a seguir. Além disso, foram utilizados critérios de seleção/exclusão para a redução da quantidade de resultados.

Perguntas de pesquisa

1. Qual o tipo de rede neural utilizada?
2. Qual o domínio de aplicação?
3. Qual plataforma de hardware foi utilizada para a aceleração dos algoritmos de classificação?
4. Quais as métricas de avaliação utilizadas?

Protocolo de Busca

O protocolo de busca foi baseado nas principais palavras-chave da pesquisa onde tentou-se reduzir o escopo a referências com conteúdos semelhantes à proposta desse

trabalho. A pesquisa foi conduzida completamente em inglês para ampliar os resultados obtidos. A expressão *booleana* que relaciona as palavras-chave da busca foi a seguinte:

"classification"AND "hardware"AND "embedded"AND "edge computing"AND ("convolutional"OR "deep"OR "neural"OR "network"OR "CNN") AND "accelerator"AND ("FPGA"OR "ASIC"OR "GPU"OR "CGRA") AND (NOT "survey") AND (NOT "review")

A plataforma Google Scholar (`https://scholar.google.com/`) foi utilizada como um meio de se observar os principais repositórios de artigos para se realizar a busca. A plataforma arXiv (`https://arxiv.org/`) reúne artigos na etapa de pré-impressão e que ainda não passaram pela revisão por pares, sendo utilizada para se buscar por títulos de artigos mais recentes, para que então fosse possível verificar se já haviam sido publicados para poderem ser considerados nesse protocolo de busca.

A expressão booleana foi então aplicada nas principais fontes de consulta encontradas, sendo as seguintes (com suas respectivas URLs):

- **ACM Digital Library:** `https://dl.acm.org/`
- **IEEE Xplore Digital Library:** `https://ieeexplore.ieee.org/`
- **ScienceDirect:** `https://www.sciencedirect.com/`

Para a redução da quantidade de artigos fornecidos pelos sites acima após a busca e procurando-se um maior alinhamento com o tema pesquisado nesse trabalho, foram aplicados critérios de seleção e exclusão, onde foram observados os títulos, resumos e palavras-chave. Os critérios utilizados foram os seguintes:

- **Critérios de seleção:**
  1. Trabalhos publicados entre 2019 e 2024;
  2. Trabalhos que utilizam soluções em hardware para a aceleração de CNNs;
  3. Trabalhos que foram citados 10 ou mais vezes.
- **Critérios de exclusão:**
  1. Trabalhos publicados em idiomas diferentes de inglês ou português;

2. Trabalhos diferentes de artigos, independente do meio de publicação;
3. Trabalhos focados apenas em otimizações a nível de software;
4. Trabalhos com ausência de palavras-chave no título.
5. Trabalhos desprovidos de informações completas para a comparação com este trabalho.

Resultados

Após a pesquisa com as palavras-chaves descritas anteriormente, foram encontrados 394 artigos distribuídos nos três repositórios (Tabela 2). Em seguida, foram aplicados os critérios de seleção e exclusão, os quais reduziram consideravelmente a quantidade de artigos. A consideração dos anos de publicação não reduziu de forma expressiva a quantidade de artigos, visto que a maioria dos trabalhos estava dentro da janela de tempo proposta para a seleção. Muitos foram descartados com base na análise dos títulos que fugiam do escopo da pesquisa. Em seguida, foram observados os resumos dos artigos para filtrar aqueles com maior relevância e similaridade com este trabalho. Por fim, foi observada a quantidade de vezes que o trabalho foi citado, resultando em apenas 5 artigos ao final do processo.

Tabela 2: Quantidade de artigos encontrados antes e depois da filtragem com os critérios de seleção e exclusão

Repositório	**Sem Filtragem**	**Com Filtragem**
ACM Digital Library	101	3
IEEE Xplore Digital Library	64	1
ScienceDirect	229	1

Fonte: Elaborado pelo autor (2023)

O processo de revisão sistemática resultou nos seguintes artigos:

- **ScienceDirect:** SparkNoC: An energy-efficiency FPGA-based accelerator using optimized lightweight CNN for edge computing (XIA et al., 2021)
- **ScienceDirect:** A Heterogeneous Processor Design for CNN-Based AI Applications on IoT Devices (LIU et al., 2020)
- **ScienceDirect:** CoNNa–Hardware accelerator for compressed convolutional neural networks (STRUHARIK et al., 2020)

- **IEEE Xplore Digital Library:** Fast and Efficient Convolutional Accelerator for Edge Computing Imagery (ARDAKANI; CONDO; GROSS, 2020)
- **ACM Digital Library:** Light-OPU: An FPGA-based Overlay Processor for Lightweight Convolutional Neural Networks (YU et al., 2020)

3.2 Trabalhos Selecionados

Nesta seção serão apresentados breves resumos acerca dos trabalhos selecionados.

3.2.1 Xia et al. (2021)

Xia et al. (2021) propõem uma nova estratégia para a compressão de redes neurais convolucionais para se obter melhor eficiência energética e desempenho para aplicações de computação na borda. Além dessa estratégia, é proposto um acelerador de hardware que toma vantagem da rede neural resultante, denominada SparkNet. O acelerador on-chip da SparkNet (SparkNoC) é realizado em FPGA e cada camada da rede é mapeada para um recurso de hardware próprio, realizando as operações de forma otimizada e possibilitando um *pipeline* para a execução de diferentes camadas em paralelo na arquitetura.

A arquitetura possui *buffers* para armazenamento das entradas, pesos e saídas das camadas da rede, bem como um controlador de memória que gerencia a troca de dados entre esses e uma memória externa. O processamento é feito em núcleos convolucionais contendo estruturas de armazenamento locais, uma árvore de multiplicadores e acumuladores, um bloco de função de ativação ReLU e um bloco de *pooling* responsável pela operação de *max-pooling*. Esses blocos são regidos por um controlador de *pipeline* responsável por garantir a execução de diferentes camadas da rede em diferentes núcleos, possibilitando a produção de múltiplas saídas da rede neural em paralelo. A arquitetura só lida com as operações de convolução, não executando camadas do tipo *dense*, por exemplo.

A arquitetura foi avaliada através da FPGA Intel Arria 10 GX 1150, sendo que dos *Digital Signal Processors* (DSPs) disponíveis, $96,9\%$ foram utilizados, assim como $15,2\%$ dos *Adaptative Logic Modules* (ALMs), $18,3\%$ dos registradores e $13,7\%$ de *Block Random Acess Memory* (BRAM). A arquitetura foi testada com um *clock* de 150 MHz, atingindo um desempenho de $337,21$ GOP/s para uma potência dissipada de $7,58$ W, correspondendo a uma eficiência de $44,8$ GOP/s/W.

3.2.2 Liu et al. (2020)

Para a aplicação em dispositivos de IoT, Liu et al. (2020) propõem uma arquitetura de hardware baseada em um processador heterogêneo para rodar aplicações que façam o uso de redes neurais convolucionais. A arquitetura possui um processador de uso geral com conjunto de instruções RISC-V acoplado a um acelerador de hardware, sendo capaz de processar de forma eficiente diferentes modelos de CNNs. Tanto o processador quanto o acelerador são realizados dentro de uma FPGA.

A arquitetura do acelerador em questão funciona com instruções que são executadas para a realização das operações das CNNs. Enquanto o acelerador está processando os dados, o processador para até o final do processo. O controlador do acelerador lida com as instruções recebidas, enquanto *buffers* são utilizados para armazenar informações como: entradas da rede, pesos e saídas, os quais possuem interface com uma memória externa ao acelerador através de barramentos. Um módulo de hardware é utilizado para separar a entrada da rede em múltiplas matrizes que são então processadas em um *array* de MACs 2D para a multiplicação de matrizes. A arquitetura também possui um bloco para soma e outro para multiplicação de vetores (elemento por elemento).

Para a avaliação da arquitetura, essa foi implementada na placa Xilinx VC709, possuindo a FPGA Xilinx XC7VX690T. Para a sintetização do processador RISC-V, foram utilizados 13% das *Lookup Tables* (LUTs) da placa, $2,1\%$ de registradores, 20% de BRAMs e 1% de DSPs. Já o acelerador ocupou, respectivamente, 53%, 33%, 27% e 23%. A avaliação de desempenho foi realizada através de um estudo de caso com a aplicação de detecção de faces, dividido em seis etapas, sendo quatro executadas pelo processador e duas (detecção de face e extração de características) pelo acelerador. Com a FPGA funcionando a 100 MHz e o processador a 20 MHz, foi possível realizar o processo de detecção de face em uma imagem em um intervalo de tempo de $723,8$ ms, sendo $76,4\%$ desse tempo ocupado pelo acelerador.

3.2.3 Struharik et al. (2020)

O trabalho de Struharik et al. (2020) propõe uma arquitetura de hardware para redes neurais convolucionais comprimidas, melhorando a eficiência de redes podadas e quantizadas. A arquitetura é implementada em FPGA com um nível de reconfiguração de alta granularidade, sendo capaz de mudar rapidamente a configuração para se adaptar a diferentes topologias de redes neurais. O hardware também suporta a execução de camadas

convolucionais, *pooling*, soma e camadas FC. Semelhante a outras arquiteturas do estado da arte, essa também evita a realização de computações ineficientes quando os pesos ou entradas são zeros.

O acelerador proposto possui quatro elementos principais, sendo o primeiro uma unidade reconfigurável de computação, responsável pelas operações das camadas das redes neurais convolucionais, dois blocos para gerenciamento da entrada e saída de dados na arquitetura e uma unidade de controle e configuração que se comunica com os demais blocos e com elementos externos da arquitetura. A unidade de computação é formada por blocos de processamento, os quais contêm uma memória local, um bloco que lida com os dados, permitindo a operação apenas entre valores cujo resultado será diferente de zero, um bloco que realizará as operações de fato e, por fim, um *buffer* de saída.

O hardware foi realizado na plataforma MultiProcessor System on Chip (MPSoC) Zynq Ultrascale+ ZU9. Os autores montam seis experimentos com quantidades diferentes de blocos de processamento operando a diferentes frequências, sendo o maior contendo 256 blocos de processamento a uma frequência de 140 MHz. Esse experimento é o que ocupa maiores recursos de hardware, sendo 244.464 LUTs, 596 BRAMs e 277 DSPs. Porém, foi alcançada uma frequência máxima de 214 MHz com 198 blocos. Esse experimento de maior *clock* alcançou um desempenho de $303,96$ GOP/s processando a rede neural VGG-16. A menor latência atingida por um dos experimentos foi de $14,20$ ms/frame para a rede neural AlexNet. Também foi observado que a presença de valores zerados nos filtros ou entradas das camadas aumenta o tempo de processamento da arquitetura, porém, quanto maior o tamanho do filtro, menor o aumento no tempo de processamento, mantendo-se abaixo de 10% para filtros de tamanho $3 \times 3 \times 1024$.

3.2.4 Ardakani, Condo e Gross (2020)

Ardakani, Condo e Gross (2020) desenvolveram um acelerador de CNNs capaz de evitar a realização de computações desnecessárias através de pesos ou entradas zeradas na rede neural. O acelerador é voltado para aplicações de computação na borda e possui suporte para filtros de diversos tamanhos, voltado principalmente aos filtros utilizados em redes neurais já consolidadas para classificação de imagens. Além da arquitetura, o trabalho também propõe um novo fluxo de dados a ser utilizado pelo hardware de forma a elevar o fator de utilização dos elementos de processamento da arquitetura.

O hardware projetado pelos autores possui *Convolutional Elements* (CEs), os quais possuem PEs em seu interior responsáveis pelas operações sobre os pesos e entradas,

bem como a aplicação da função de ativação ReLU. A quantidade de PEs por CE é reconfigurável, visto que o autor mostra que, para diferentes tamanhos de filtros de uma camada convolucional, tal quantidade influencia no desempenho. Além dos PEs, um CE também possui um bloco gerador de pesos, que direciona os pesos para os respectivos PEs para serem processados. Além disso, foi realizado um meio de evitar a realização de computações com valores zerados, substituindo diretamente os valores dos pesos zerados na memória por uma representação codificada para indicar que havia um determinado número de zeros sequenciais armazenados na memória, reduzindo assim tanto o acesso à memória quanto a realização de processamento.

A arquitetura em questão foi descrita em Verilog e projetada em silício com litografia de 65 nm com *clock* de 200 MHz. Comparada com outras arquiteturas do estado da arte, atingiu uma performance máxima de $128,8$ GOP/s processando a rede neural ResNet-50. Em termos de potência dissipada, a arquitetura atingiu um valor mínimo de 248 mW para a rede neural ResNet-18 e uma eficiência energética mínima de $240,9$ GOP/s/W para a rede VGG-16.

3.2.5 Yu et al. (2020)

Para lidar com CNNs do tipo *lightweight*, Yu et al. (2020) propõe a arquitetura Light-OPU em FPGA, visto que tais CNNs normalmente não tomam proveito da aceleração em GPUs. A arquitetura consiste em um processador configurável em tempo de execução através de instruções, com alta eficiência energética e baixa latência. Além da arquitetura, o autor apresenta uma *Instruction Set Architecture* (ISA) para sua arquitetura, bem como um compilador para gerar as instruções e alocar os dados nos respectivos recursos de hardware para a realização do processamento.

Tal como em outros aceleradores de CNNs, a arquitetura de Yu et al. (2020) possui *buffers* para armazenar os filtros e *feature maps* de entrada, os quais recebem os dados através de uma interface com uma memória externa. Um bloco para lidar com o direcionamento dos dados dos *buffers* para os respectivos recursos de processamento, sendo esse bloco configurado através da ISA. Além disso, há um bloco de computação contendo os PEs e responsável pelo processo de MAC, seguido por um bloco de pós-processamento, responsável por operações de concatenação, adição e quantização, por exemplo. A saída do pós-processamento pode ser armazenada em *buffers* de *feature maps* de saída, podendo ser operadas com *pooling* e função de ativação antes de serem enviadas para a interface de memória externa.

Light-OPU foi testada em uma FPGA Xilinx XC7K325T, ocupando 85, 14% de LUTs, 59, 16% de *flip-flops*, 43, 48% de BRAM e 83, 81% de DSPs, funcionando a uma frequência de 200 MHz. A quantização utilizada foi de 8$bits$ e atingindo um máximo de 460, 8 GOP/s. A eficiência energética alcançada foi de 52, 5 GOP/s/W. Em comparação a um ARM A57, a arquitetura conseguiu ser 30, 6× mais rápida, já em relação a um Intel i7-8700k, a aceleração foi de 4, 9×. Por fim, em comparação à GPU Jetson TX2, o resultado foi de 5, 5×.

3.3 Análise Comparativa

A partir dos trabalhos selecionados, foi realizado o Quadro 1 comparando-se este trabalho com os demais, considerando-se algumas métricas, sendo estas: as camadas da rede neural convolucional executadas na arquitetura, os tipos de funções de ativação aplicadas, o tipo de hardware alvo da arquitetura, as métricas observadas pelos autores e o tipo de aplicação para o qual os trabalhos são propostos.

Em relação às camadas que tomam proveito das arquiteturas propostas nos trabalhos selecionados, todas lidam com as camadas convolucionais e utilizam redes neurais consolidadas na área para a classificação de imagens, realizando operações de convoluções 2D. Trabalhos como o de Xia et al. (2021), Struharik et al. (2020) e Ardakani, Condo e Gross (2020) ainda lidam com redes neurais esparsas ou reduzidas. A camada de *pooling*, presente em muitas dessas redes neurais, não é realizada pela arquitetura de Ardakani, Condo e Gross (2020), a qual foca apenas nas camadas convolucionais. Além disso, a camada FC também não é abordada por Xia et al. (2021), que faz modificações na rede neural utilizada para a avaliação da arquitetura de forma a não ser necessário o uso de tal camada. Por fim, apenas a arquitetura de Liu et al. (2020) implementa a camada de *batch normalization*.

Quanto à função de ativação, todas as arquiteturas fazem o uso da função ReLU devido à sua simplicidade (Tabela 1). Além dessa, a arquitetura de Struharik et al. (2020) acrescenta a capacidade de se definir uma função de ativação arbitrária, armazenando-a em LUTs. Já Yu et al. (2020) realiza, além da ReLU, as funções de ativação *H-sigmoid* e *H-swish*, que são aproximações das funções sigmoide e *swish* utilizando a função ReLU como base.

Ardakani, Condo e Gross (2020) é o único trabalho dentre os selecionados que realiza a arquitetura em um ASIC, projetado em silício. Liu et al. (2020) apresenta uma arquitetura

Quadro 1: Análise comparativa com os trabalhos relacionados selecionados

Ref.	**Camadas**	**Ativação**	**Hardware**	**Métricas**	**Aplicação**
3.2.1	Conv. *Pooling*	ReLU	FPGA	Aceleração Desempenho Potência Recursos	Borda
3.2.2	Conv. *Pooling* FC *Batch norm.*	ReLU	FPGA	Desempenho Recursos	Borda
3.2.3	Conv. *Pooling* FC	ReLU Arbitrária	FPGA	Aceleração Desempenho Recursos	Geral/Borda
3.2.4	Conv.	ReLU	ASIC	Desempenho Potência Recursos	Borda
3.2.5	Conv. *Pooling* FC	ReLU *H-sigmoid* *H-swish*	FPGA	Aceleração Desempenho Potência Quantização Recursos	Borda
Este trabalho	Conv. 1D *Pooling* FC	ReLU Sigmoide mod.	FPGA	Aceleração Acurácia Desempenho Quantização Recursos	Geral/Borda

Fonte: Elaborado pelo autor (2024)

contendo uma CPU e um bloco reconfigurável, porém, ambos são implementados em uma FPGA. Os demais trabalhos são todos apresentados e avaliados em FPGA.

No âmbito destas arquiteturas, métricas comuns de análise são a utilização de recursos de hardware, seja em FPGA ou em ASIC, no caso de Ardakani, Condo e Gross (2020), onde o autor apresenta o tamanho do chip projetado, as memórias e a quantidade de unidades de MAC. Além disso, o desempenho em termos de quantidade máxima de operações por segundo também está presente em todos os trabalhos. Xia et al. (2021), Struharik et al. (2020) e Yu et al. (2020) fazem comparações com outras arquiteturas evidenciando a aceleração da execução dos processos. Ademais, Xia et al. (2021), Ardakani, Condo e Gross (2020) e Yu et al. (2020) fazem a análise da potência dissipada pelo hardware, bem como a eficiência energética, sendo um parâmetro importante em muitas aplicações de computação na borda. Por fim, apenas Yu et al. (2020) faz uma análise da quantização da arquitetura.

Em termos de domínio de aplicação, todos os trabalhos selecionados estão dentro do contexto de computação na borda, apesar de arquiteturas como a de Struharik et al. (2020) abrir margem para a utilização em aplicações gerais.

A arquitetura de hardware descrita neste trabalho se propõe a acelerar redes neurais convolucionais do tipo 1D, logo, as operações de convolução aplicadas pela arquitetura são todas do tipo 1D, havendo uma simplificação do hardware, visto que, dependendo da aplicação, a quantidade de operações e de pesos a serem processados é reduzida se comparada a convoluções de dimensionalidade superior. O tamanho dos filtros aplicados pelas camadas convolucionais é completamente parametrizável, desde que haja armazenamento suficiente na arquitetura para contê-los; o *stride* desses filtros também pode ser aplicado. Além disso, a camada de *pooling* também é realizada pela arquitetura, especificamente a função de *max-pool*, com *stride* e tamanho da janela parametrizáveis. As camadas *Fully Connected* também tomam proveito dos mesmos elementos de processamento das camadas convolucionais para serem processadas. Esse trabalho também utiliza a função de ativação ReLU, bem como aplica uma aproximação da função sigmoide por retas, visto que tal função pode ser utilizada na etapa de classificação na saída de CNNs. Devido à praticidade de prototipagem, testes e à possibilidade de reconfiguração, a arquitetura foi pensada para FPGAs. As métricas de análise são as comumente presentes em trabalhos do mesmo tipo, incluindo a análise da acurácia devido ao impacto da linearização da função sigmoide e do nível de quantização da arquitetura. Além disso, apesar da arquitetura ser validada em um domínio de aplicação de computação na borda, há margem para aplicação

da arquitetura em outras aplicações.

4 Proposta de Arquitetura de Hardware

Este trabalho propõe o desenvolvimento de um acelerador em hardware com objetivo principal de acelerar redes neurais convolucionais 1D, de forma que este capítulo apresentará uma descrição, contendo os requisitos funcionais e não funcionais, assim como diagramas de blocos descritivos da implementação desejada, em uma visão de alto nível de abstração.

4.1 Requisitos funcionais e não funcionais

Os Quadros 2 e 3 apresentam os requisitos funcionais e não funcionais para a execução da proposta de projeto aqui descrita.

Quadro 2: Requisitos funcionais

Código	**Objetivo**
RF01	O acelerador em hardware deve otimizar a execução de redes neurais convolucionais 1D
RF02	O acelerador deve lidar com filtros de qualquer tamanho desde que haja armazenamento o suficiente
RF03	O acelerador deve lidar com funções de ativação do tipo ReLU e uma versão linearizada da sigmoide
RF04	O acelerador deve realizar a operação de *max-pool*
RF05	O acelerador deve permitir a adição de novos elementos de processamento

Fonte: Elaborado pelo autor (2024)

4.2 HARS1DE

A arquitetura HARS1DE (*Hardware Architecture Reconfigurable and Scalable for CNN-1D processing on the Edge*) proposta (Figura 1) possui um bloco responsável pela classifi-

Quadro 3: Requisitos não funcionais

Código	**Objetivo**
RNF01	Deve atingir um desempenho mínimo de 10 GOP/s
RNF02	Deve ser capaz de melhorar o desempenho com o acréscimo de PEs
RNF03	Não deve exceder em 80% o uso de qualquer recurso de FPGA
RNF04	Deve operar com ponto fixo e representação de 16bits
RNF05	O acelerador deve possuir interface com uma memória externa
RNF06	O acelerador deve possuir interface de comunicação externa
RNF07	O acelerador deve ser parametrizável de acordo com os parâmetros de uma rede neural

Fonte: Elaborado pelo autor (2024)

cação de uma entrada através de uma rede neural convolucional 1D armazenada em uma memória (externa ou interna). A comunicação entre o acelerador e componentes externos deve se dar através de barramentos, cujas especificações dependerão do hardware em que a arquitetura for sintetizada. A memória deve ser capaz de conter todos os pesos e parâmetros descritivos da rede, os quais são utilizados para a configuração do acelerador.

A utilização de uma CPU em conjunto com o acelerador pode garantir o acréscimo de funcionalidades à arquitetura, como etapas de pré-processamento de dados, dentre outras. Também pode realizar o gerenciamento e envio dos dados para a arquitetura, fornecendo as entradas da rede neural.

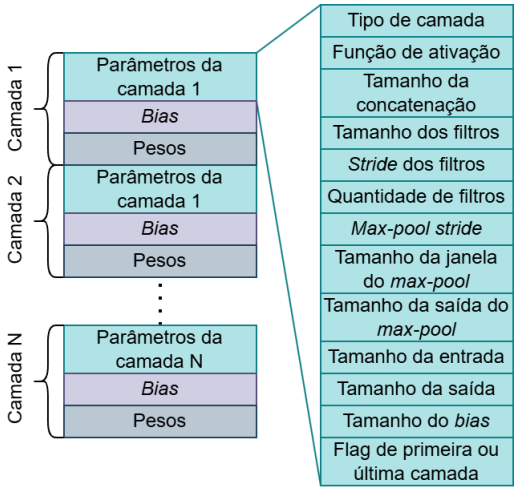
4.2.1 Organização da memória da CNN

Como já citado anteriormente, o acelerador depende de uma memória externa contendo os parâmetros descritivos da rede e seus valores. A Figura 14 mostra como se dá a organização da memória, onde antes dos valores de *bias* e dos pesos, os parâmetros de cada camada são armazenados para que possam ser lidos pela arquitetura e essa possa se configurar. Tais parâmetros são fundamentais para que a arquitetura tanto possa realizar as operações de forma apropriada quanto para garantir o fluxo de dados e saber quando um bloco terminou de processar completamente sua entrada.

4.2.2 Visão de alto nível da arquitetura

A Figura 15 mostra uma visão de alto nível da arquitetura. Os principais blocos que compõem a arquitetura, como o arranjo de PEs, *buffers*, bloco de *max-pool* dentre outros, podem ser encontrados em outras arquiteturas da literatura, como a Eyeriss (CHEN et al., 2017), a Cambricon-X (ZHANG et al., 2016), Simba (SHAO et al., 2021) dentre outras.

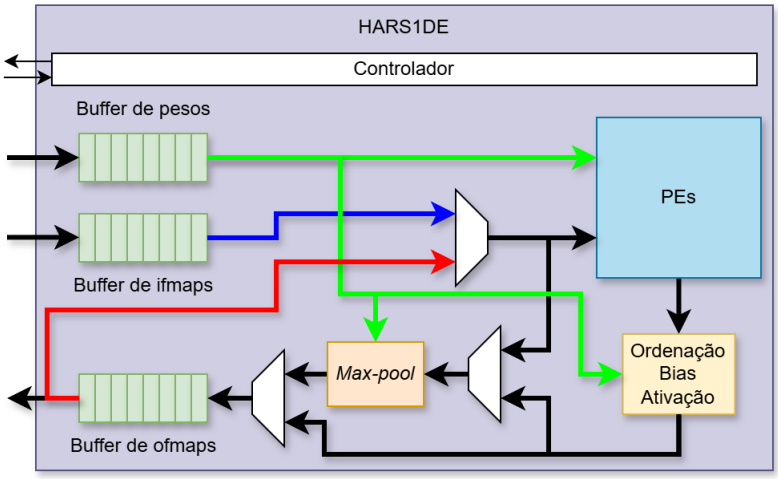
Figura 14: Organização da memória da rede neural



Fonte: Elaborado pelo autor (2024)

Tais blocos e os paradigmas associados a estes são explicados por Moolchandani, Kumar e Sarangi (2021).

Figura 15: Arquitetura do hardware de classificação espectral



Fonte: Elaborado pelo autor (2024)

Dessa forma, a arquitetura proposta no presente trabalho possui *buffers* para entrada e saída de dados, um arranjo de *Processing Elements*, um bloco para reordenar as saídas dos PEs, aplicação de *bias* e de funções de ativação nos resultados, um bloco que possibilita a realização do *max-pool*, e um controlador central para garantir o fluxo de dados. Os dados são processados e passados serialmente para o componente seguinte numa forma de *array* sistólico (KUNG; LEISERSON, 1979).

O acelerador realiza um *pipeline* no processamento de cada camada entre os blocos,

ou seja, diferentes blocos podem estar processando diferentes camadas da rede neural. Tal condição é garantida através de *buffers* em alguns sinais de controle (e.g. controle de *muxes*) e nos parâmetros de configuração dos blocos. O controlador principal e os blocos de processamento recebem parâmetros de configuração da camada em processamento (*stride*, número de entradas, tamanho dos filtros etc.) através do mesmo caminho de dados dos pesos, de modo a configurar seu comportamento. Cada um desses blocos será descrito nas seções a seguir.

4.2.2.1 *Buffers*

Os *buffers* são FIFOs (*First In, First Out*) para armazenamento temporário dos dados. Ao total, o hardware de classificação espectral possui três principais *buffers* com interface externa e alguns *buffers* menores dentro de cada bloco de processamento para garantir o fluxo de dados. Como citado anteriormente, também são utilizados em sinais de controle e de configuração para garantir um *pipeline* no processamento.

O *buffer* de pesos recebe os dados de uma memória externa ao acelerador e que armazena os pesos e parâmetros de cada camada de uma rede neural convolucional (Figura 14). Para cada camada, inicialmente são recebidos pela arquitetura os parâmetros de configuração, os quais são armazenados no controlador geral e enviados para os demais blocos para configurá-los. Em seguida, o *bias* é enviado para seu respectivo bloco e os pesos são enviados aos PEs. Visto que esse *buffer* recebe dados armazenados em uma memória externa, quanto maior seu tamanho e quanto antes os dados são carregados, menor a latência durante o processamento de uma entrada da rede. No caso de uma memória interna, o tamanho desse *buffer* pode ser mínimo, visto que a latência de acesso é menor.

O *buffer* de *ifmaps* possui interface externa para recebimento dos dados de entrada da rede neural. Vale ressaltar que a arquitetura suporta operações de concatenação da rede, ou seja, mais entradas podem ser utilizadas em camadas adiante na rede neural. O tamanho deste *buffer* depende do tipo de entrada utilizada na rede e, se possuir tamanho suficiente para manter as entradas armazenadas internamente, pode-se gerar ganhos de latência devido ao não acesso a elementos externos ao acelerador.

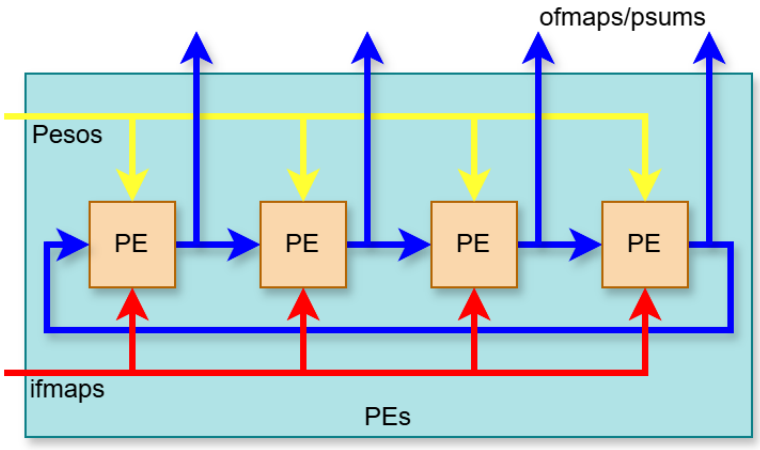
O *buffer* de *ofmaps* armazena os resultados parciais e finais da rede neural. Durante o processamento das camadas internas da rede, os resultados parciais devem voltar para os elementos de processamento para gerar os resultados das camadas seguintes, sendo o resultado da última camada enviado para fora da arquitetura. A arquitetura foi idealizada

de tal forma que resultados parciais de camadas intermediárias da rede não são enviados para fora da arquitetura, logo, esse *buffer* deve possuir espaço suficiente para armazenar todos os resultados de saída da maior camada da rede, consistindo no maior *buffer*.

4.2.2.2 Arranjo de *Processing Elements*

Para a realização das operações de convolução das camadas convolucionais e das camadas FC, a arquitetura possui um arranjo 1D de PEs, como mostra a Figura 16, onde os pesos e *ifmaps* são enviados para os respectivos elementos de processamento através de barramentos de dados e as somas parciais (*psums*) são passadas para o elemento seguinte formando um anel. Quando um PE possui o resultado completo da saída de um neurônio da camada (*ofmap*), o dado pode ser lido externamente. O arranjo pode processar um ou mais neurônios em paralelo, dependendo da quantidade de PEs disponíveis e da quantidade de entradas da camada (Seção 4.2.2.5).

Figura 16: Arranjo de PEs e caminhos de dados

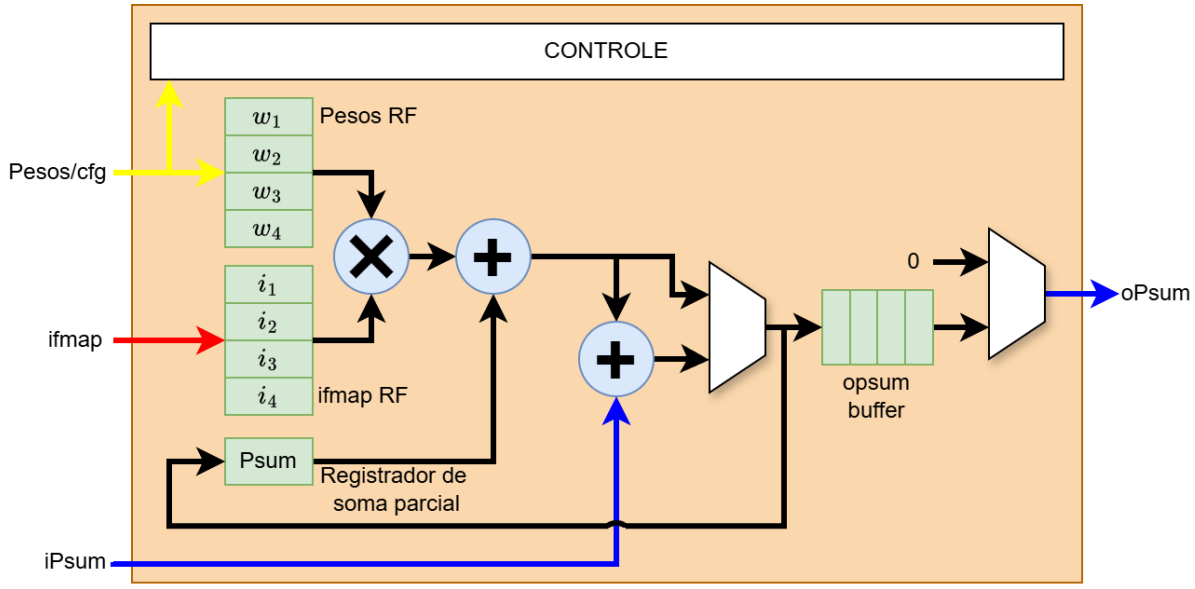


Fonte: Elaborado pelo autor (2024)

Internamente, cada PE é organizado como mostra a Figura 17. As entradas e os pesos são armazenados em *register files* (RFs), havendo um RF para armazenar uma ou mais entradas e outro para armazenar os filtros, sendo os filtros sobrescritos após seu uso. A entrada de pesos também é passada ao controlador, pois os parâmetros descritivos de cada camada também fazem uso desse mesmo caminho de dados; dessa forma, o bloco pode ser configurado.

Considerando-se a Equação 2.5, cada PE é capaz de realizar as operações dos somadores e produtos entre os elementos dos vetores de pesos e de entradas, em outras palavras, cada PE é capaz de realizar uma convolução a partir dos dados dos RFs, sendo que, a

Figura 17: Arquitetura interna de um PE



Fonte: Elaborado pelo autor (2024)

partir do momento em que há um dado disponível em cada RF, as operações já podem começar a ser realizadas, sem necessidade de esperar todos os dados estarem disponíveis. A convolução é realizada através do processo de multiplicação e acumulação, de forma que a arquitetura possui um multiplicador e um somador. A soma parcial é armazenada em um registrador interno e o somador pode, condicionalmente, receber o resultado da convolução do PE anterior, visto que a saída de um neurônio consiste na soma de múltiplas convoluções. O resultado é armazenado em um *buffer* de saída, cujos sinais de controle são compartilhados com o controlador do PE seguinte, com o controlador interno do PE e com o bloco de reordenação das saídas, que solicitam a leitura desses dados.

Apesar de não estarem representadas, existem *flags* internas para caracterizar um PE como sendo o primeiro dos que serão somados, logo, não utiliza a soma do PE anterior. Assim como caracteriza o último, que deve enviar a saída para fora do arranjo.

O mesmo *datapath* também pode ser utilizado para o cálculo das camadas FC de uma rede neural, visto que tal processo consiste em multiplicações e acumulações e que normalmente faz parte da parte de classificação da rede neural após as camadas convolucionais.

Outro recurso implementado no PE é a possibilidade de evitar computações desnecessárias, contando a quantidade de pesos ou entradas zeradas. Se uma janela do filtro a ser aplicada for composta completamente por zeros ou se uma entrada for composta também apenas por zeros, o resultado da convolução é zero sem a necessidade de ser processada.

Dessa forma, o *mux* da saída muda de estado e a saída do PE é zerada. Como os PEs são processados em sequência e havendo dependência de dados, a contagem de zeros só poupa a utilização de recursos, não garantindo ganhos significativos de desempenho.

4.2.2.3 Reordenação das saídas, aplicação do *bias* e da ativação

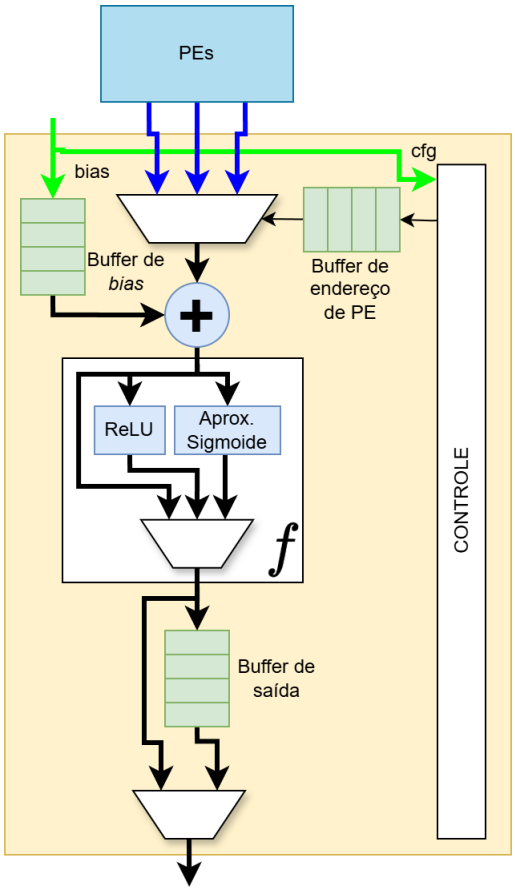
O bloco de reordenação (Figura 18) realiza as demais operações apresentadas na Equação 2.5, adicionando o bias e aplicando a função de ativação ao resultado gerado pelos PEs, resultando no final do processamento de uma camada convolucional. Esse bloco se comunica com todas as PEs, tendo acesso aos sinais que controlam a leitura dos *buffers* de saída e quando há dados disponíveis para leitura. Esses sinais só são lidos por esse bloco quando o processamento de um neurônio está na sua etapa final. Vale ressaltar que o processo de reordenação só ocorre quando mais de um neurônio está sendo processado em paralelo no PE. Quando o controlador recebe o sinal de que um dos *buffers* dos PEs possui um dado a ser lido, o endereço desse PE é armazenado em um *buffer* que controla o *mux* na entrada do reordenador. Quando todos os dados de um PE são lidos, o próximo endereço é atendido, até que todos os PEs tenham seus dados lidos em ordem. Os primeiros dados que entram nos PEs são os primeiros a saírem no processo de reordenação.

A saída do *mux* que lê os PEs passa por um somador que aplica o *bias* em cada saída de neurônio. O *bias* é armazenado em um *buffer*, cuja entrada é conectada à saída do *buffer* de pesos. Como esse *buffer* armazena também os parâmetros descritivos da camada, esse caminho de dados também é passado ao controlador para configurar o bloco.

Em seguida, há um bloco de aplicação da função de ativação, o qual implementa as funções ReLU e sigmoide exibidas na Tabela 1. O bloco também permite a não aplicação de funções de ativação. As funções consistem em blocos combinacionais, sendo a ReLU responsável por fazer a saída ser zero quando a entrada é menor ou igual a zero. Já o hardware responsável pela sigmoide implementa uma versão linearizada, aproximada por reta. Dependendo do intervalo em que a entrada do bloco se encontre, uma equação da reta é aplicada para a geração da saída. Visto que a função sigmoide é definida para valores reais entre zero e um, e a arquitetura não trabalha com ponto flutuante, os valores utilizados nas equações da reta dependem diretamente da quantização utilizada.

Por fim, o resultado da aplicação da ativação pode ser tanto armazenado em um *buffer* de saída como pode ser passado diretamente para o *buffer* de *ofmaps*. O último caso ocorre quando a função de *max-pool* não é realizada logo após a convolução e ocorre o *bypass* do

Figura 18: Bloco de reordenação de saídas, aplicação de *bias* e da ativação



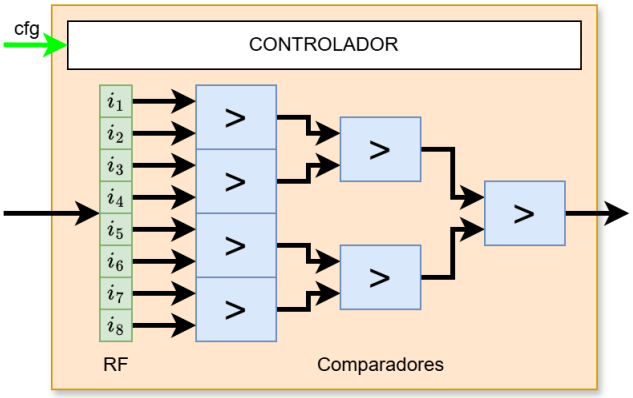
Fonte: Elaborado pelo autor (2024)

bloco direto para o *buffer* de *ofmap*.

4.2.2.4 *Max-pool*

O *max-pool* possui um RF para armazenar as entradas do bloco. O tamanho do RF consiste no tamanho máximo da janela de aplicação da operação de *max-pool* onde, para a aplicação atual, foi definido um tamanho de 8. Normalmente, cada RF contém o menor valor possível da representação de bits. Por exemplo, se for utilizada uma quantização de 8 bits para os dados, a representação com sinal fica no intervalo entre -128 e 127, logo, todo o RF mantém o valor -128 armazenado. Dessa forma, se for utilizada uma janela de *max-pool* de tamanho 3, os demais 5 valores não irão impactar nas comparações. A saída do RF é conectada a uma árvore de comparadores com duas entradas onde a saída consiste no maior valor entre todo o RF.

Figura 19: Bloco de *max-pool*



Fonte: Elaborado pelo autor (2024)

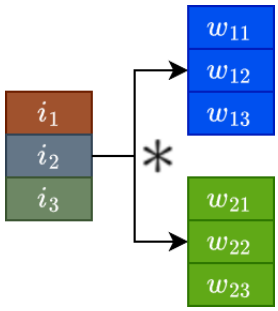
A camada de *max-pool* é parametrizada na arquitetura junto à camada convolucional que a precede, fazendo com que os dados saiam do arranjo de PEs para o *max-pool*, processando duas camadas antes do resultado final ser armazenado no *buffer* de saída.

4.2.2.5 Fluxo de dados

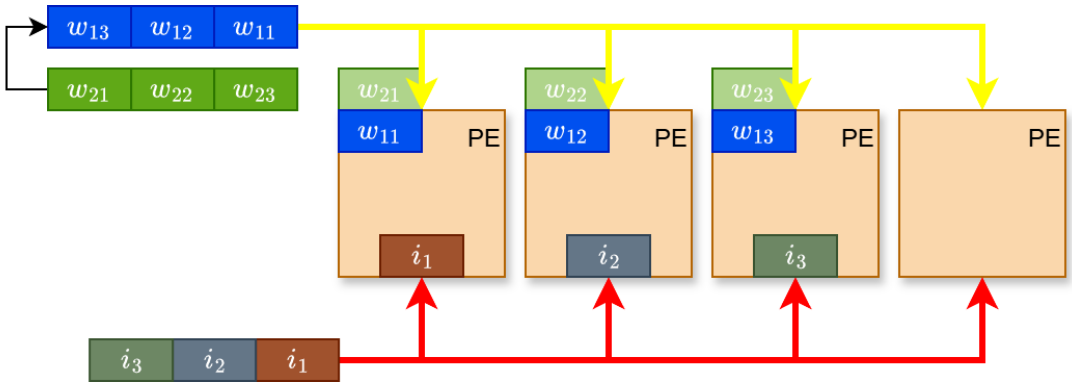
Apesar do fluxo de dados *Row-Stationary* possuir vantagens em relação aos demais, como explicado na Seção 2, sua aplicação na presente arquitetura se torna inviabilizada, pois este trabalho se limita à convolução 1D. O fluxo de dados RS faz o reuso das linhas do filtro e da *ifmap*, sendo que para a convolução 1D, o filtro e a *ifmap* possuem apenas uma linha, não havendo reuso.

Da mesma forma, analisando-se o modelo de pesos estacionários, o fato de os filtros e da entrada possuírem apenas uma dimensão, faz com que cada filtro da rede neural convolucional seja utilizado apenas uma vez, inviabilizando manter os pesos fixos dentro dos PEs. Já as entradas das camadas da rede são reutilizadas para vários filtros. Sendo assim, na proposta de arquitetura, as entradas são armazenadas estaticamente dentro dos PEs. A seguir serão mostradas três situações diferentes de alocação e movimentação dos dados nos elementos de processamento, sendo o primeiro caso o da Figura 20. Nessa situação, a quantidade de entradas da camada é maior do que a metade da quantidade de PEs. No exemplo, é considerada uma camada convolucional com três entradas e duas saídas, sendo o tamanho das entradas e pesos não representados. Os pesos em azul correspondem a um neurônio da camada e os em verde a outro.

Figura 20: Exemplo 1 de distribuição dos dados nos PEs



(a) Exemplo de camada convolucional com 3 entradas e duas saídas



(b) Alocação dos pesos e entradas nos PEs

Elaborado pelo autor(2024)

As entradas são lidas sequencialmente e armazenadas nos PEs também de forma sequencial, onde a primeira é alocada no primeiro PE, e assim por diante. Essas entradas são mantidas até o término do processo da camada. Já os pesos são processados um neurônio por vez nessa situação. Os pesos entram na arquitetura de forma sequencial também e são armazenados nos PEs para serem convolucionados com a entrada armazenada. O re-

sultado da convolução é passado para o PE seguinte para somá-lo, e assim por diante, até que o PE que armazena a última entrada gere o resultado do neurônio. Quando o peso é completamente utilizado, esse pode então ser substituído pelo do próximo neurônio, reiniciando o processo.

O segundo caso mostrado na Figura 21 ocorre quando a quantidade de entradas é maior do que a quantidade de PEs na arquitetura. As entradas e pesos são armazenados nos PEs de modo sequencial, analogamente ao caso anterior, porém, nesse caso um PE recebe mais de uma entrada. Após a escrita no último PE, o endereço de escrita volta para o primeiro PE e as entradas continuam sendo armazenadas. Na figura, os dois primeiros PEs armazenam duas entradas cada. O comportamento dos pesos funciona de forma análoga onde os pesos exibidos fora dos PEs são os que serão armazenados internamente nos ciclos seguintes de processamento. Também de forma análoga, as somas parciais de convolução são passadas para o PE seguinte, até o resultado final do neurônio ser processado.

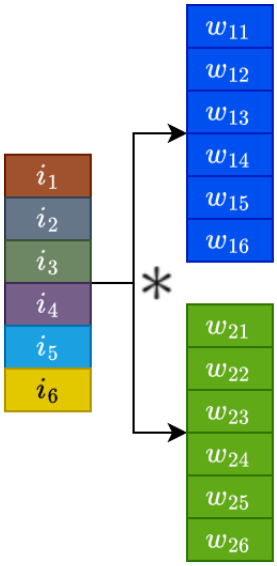
Por fim, o último caso de movimentação dos dados é mostrado na Figura 22, quando a quantidade de neurônios ou de entradas é menor ou igual a metade da quantidade de PEs. Nessa situação, as entradas são carregadas paralelamente em mais de um PE. A quantidade de PEs que recebem uma mesma entrada depende da quantidade de elementos de processamento em relação à quantidade de entradas.

No caso da figura, há quatro PEs e duas entradas, logo, essas são distribuídas nos PEs repetindo-se duas vezes. Em um caso com duas entradas e cinco PEs, o quinto PE se torna inativo, pois não é possível repetir o conjunto de duas entradas novamente. A quantidade de vezes que uma entrada se repete também é limitada pela quantidade de neurônios da camada. Se na figura houvesse apenas uma entrada i_1, os pesos seriam apenas w_{11} e $w_{21}htg$, não fazendo sentido repetir as entradas duas vezes.

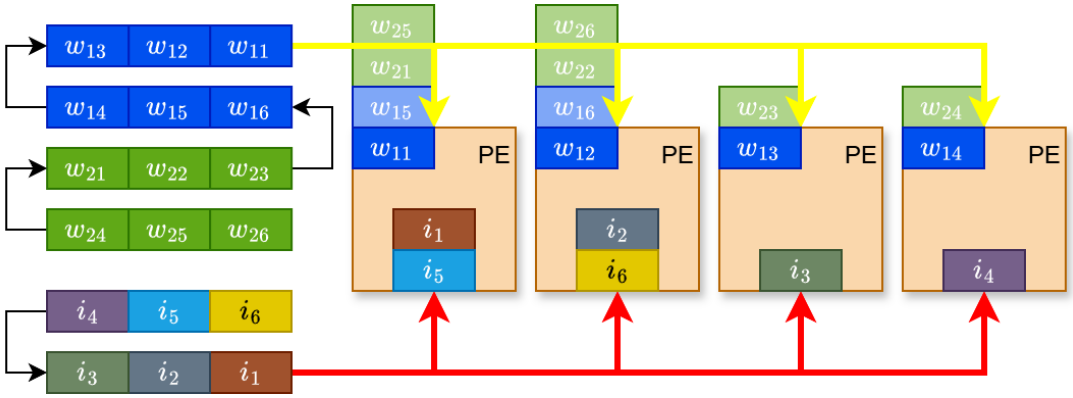
Nesse caso, mais de um neurônio pode ser processado em paralelo, havendo a necessidade da atuação do bloco de reordenação da Figura 18 para que a saída de cada neurônio seja lida por vez em ordem.

Em relação às camadas FC, os mesmos componentes de hardware são utilizados para a multiplicação dos pesos e das entradas. Nesse caso, os pesos de um neurônio são distribuídos igualmente em todos os PEs, assim como a entrada da camada. Internamente em cada PE, é processada uma "convolução"onde os dois operandos possuem a mesma quantidade de elementos, resultando em um produto escalar. O resultado do produto é passado adiante para os PEs seguintes até se ter a saída do neurônio. Em seguida, os pesos são substituídos pelos do próximo neurônio e o processo se repete. Por exemplo, os

Figura 21: Exemplo 2 de distribuição dos dados nos PEs



(a) Exemplo de camada convolucional com 6 entradas e duas saídas

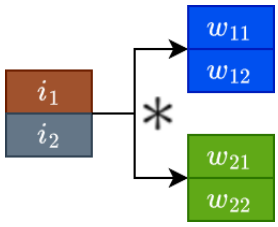


(b) Alocação dos pesos e entradas nos PEs

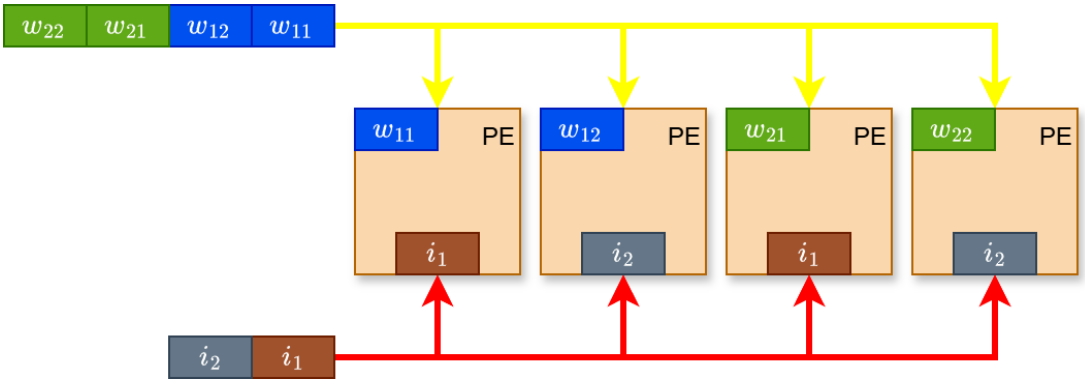
Elaborado pelo autor(2024)

parâmetros de uma camada FC são a quantidade de entradas e a quantidade de neurônios, onde, havendo uma arquitetura de 16 PEs, 1024 entradas e 128 neurônios, as entradas são divididas pela quantidade de PEs, resultando em 64 valores de entrada em cada elemento de processamento. Os pesos são então armazenados 64 em cada PE para a realização do produto escalar.

Figura 22: Exemplo 3 de distribuição dos dados nos PEs



(a) Exemplo de camada convolucional com 6 entradas e duas saídas



(b) Alocação dos pesos e entradas nos PEs

Elaborado pelo autor(2024)

5 Estudo de Caso na Classificação de Imagens Hiperespectrais

Atualmente, vive-se um cenário de constantes mudanças climáticas devido ao aquecimento global, onde as projeções mostram uma tendência crescente da temperatura do planeta e associação deste aumento com a presença de dióxido de carbono (CO_2) na atmosfera (SUPRAN; RAHMSTORF; ORESKES, 2023). Grande parte desse CO_2 depositado na atmosfera é proveniente da queima dos combustíveis fósseis, porém, há outra parcela importante que é devida ao desmatamento e degradação das florestas, sendo em 2009 responsável por 12% de todo o dióxido de carbono na atmosfera proveniente de atividades humanas (WERF et al., 2009). Diante disso, torna-se necessária a capacidade de se obter dados detalhados sobre a superfície terrestre, seja para mitigação dos impactos das mudanças ou acompanhamento dos danos causados através do monitoramento das alterações da superfície (e.g. redução da cobertura vegetal).

Nesse âmbito, tem-se o sensoriamento remoto, que consiste na aquisição de informação sobre uma determinada área (seja terrestre ou aquática) a partir de imageamento aéreo, sendo os dados coletados a partir da radiação eletromagnética refletida ou emitida pela superfície (CAMPBELL; WYNNE, 2011), ou seja, a aquisição é feita sem que haja contato direto com o objeto de estudo (FISCHER; HEMPHILL; KOVER, 1976). O sensoriamento possui vasta aplicação, sendo empregado na agricultura, monitoramento de oceanos, aferição da qualidade da água, estudos climáticos, dentre outras aplicações (NAVALGUND; JAYARAMAN; ROY, 2007). O sensor da aquisição pode ser embarcado em satélites, veículos aéreos e drones, bem como existem plataformas móveis que podem ser utilizadas em veículos terrestres e, por fim, também há a possibilidade de plataformas estáticas que coletam informações de um mesmo local (TOTH; JóźKóW, 2016).

O sensoriamento remoto pode se dar de forma ativa, quando o próprio sensor fornece energia ao meio e mede a reflexão desta na superfície terrestre (e.g. *Light Detection and Ranging* - LiDAR), e de forma passiva, quando o sensor captura a energia emitida pela

própria superfície analisada (FISCHER; HEMPHILL; KOVER, 1976). Nesta última categoria, tem-se as imagens multiespectrais (*Multispectral Images* - MSI) e as imagens hiperespectrais (*Hyperspectral Images* - HSI), as quais, diferentes de uma imagem colorida que é composta por três canais de cores (vermelho, verde e azul), são compostas por dezenas a centenas de canais, onde cada canal corresponde a um comprimento de onda do espectro eletromagnético que é adquirido pelo sensor. Esse tipo de imagem possui diversas aplicações, sendo utilizada na indústria alimentícia para aferição da qualidade dos alimentos, diagnósticos médicos de câncer, agricultura de precisão, recursos hídricos, entre outras aplicações (KHAN et al., 2018).

Cada pixel que compõe uma HSI é representado por todos os comprimentos de onda adquiridos, formando um espectro. Essa informação pode ser utilizada para a identificação dos elementos que compõem a superfície registrada pelo sensor onde, por exemplo, um pixel de vegetação possui uma assinatura espectral diferente de um pixel de uma construção. Segundo Paoletti et al. (2019, tradução nossa), "a tarefa de se rotular cada pixel contido em uma HSI é árdua e demorada, e geralmente requer um especialista". Uma das formas de se reduzir o tempo deste processo é através de técnicas de aprendizado de máquina para a criação de um modelo para classificação de pixels das HSIs. Vale ressaltar que alguns métodos de aprendizado não excluem a necessidade de um especialista para a classificação inicial dos pixels, pois os modelos podem necessitar de exemplos previamente classificados para uma melhor acurácia. Uma vez que o modelo seja criado, futuras HSIs coletadas podem fazer o seu uso para uma rápida classificação. Como um exemplo de aplicação, Thangavel et al. (2023) utilizaram técnicas de aprendizado de máquina aliadas a imagens hiperespectrais coletadas via satélite para a detecção de incêndios na natureza com acurácia de $97,83\%$ considerando a metodologia utilizada no trabalho.

Um dos grandes problemas associado às HSIs é o volume de dados gerados, visto que, devido à quantidade de comprimentos de onda coletados, o tamanho da imagem cresce consideravelmente. O desenvolvimento tecnológico dos sensores, aumentando a resolução espacial e espectral, também proporciona um aumento dos dados coletados. Por exemplo, a *National Aeronautics and Space Administration* (NASA) estimou que a missão HyspIRI (*Hyperspectral InfraRed Imager*) produziria $4,5$ TB de dados diariamente (BIOUCAS-DIAS et al., 2013). Além disso, a aplicação dos modelos de aprendizado de máquina depende diretamente do volume de dados a ser analisado, visto que o acesso à memória é um dos problemas envolvendo esta área (MOOLCHANDANI; KUMAR; SARANGI, 2021). No Capítulo 1, foram listadas múltiplas arquiteturas de hardware idealizadas para a aceleração da execução de redes neurais convolucionais, tanto para aplicações gerais quanto para a

classificação de HSIs, as quais tentam contornar tais desafios referentes ao processamento desses modelos.

Além disso, Viel et al. (2023) avaliam diferentes modelos para a classificação de HSIs e expõem que a utilização de CNNs do tipo 1D pode ser interessante para aplicações com restrições de recursos (de hardware), visto que sua utilização em detrimento de CNNs com maior dimensionalidade (2D e 3D) não gera perdas significativas de acurácia e necessita de menos operações e utilização de memória. Concomitantemente, essa perda de acurácia pode ser mitigada, visto que a utilização de informação espacial pode gerar ganhos de acurácia na classificação dos pixels (IMANI; GHASSEMIAN, 2020; FAUVEL et al., 2013; BORZOV; POTATURKIN, 2017). Junto ao trabalho de Viel et al. (2023), Souza (2023) apresenta a utilização de dados espaciais de uma HSI como entrada de uma rede neural 1D para a classificação de HSIs. O dado espectral é obtido através do algoritmo de segmentação *Superpixels Extracted via Energy-Driven Sampling* (SEEDS) e a fusão dos dados espectrais e espaciais na rede neural acarretou em um ganho de acurácia se comparada à solução apenas com informação espectral.

Sendo assim, considerando o volume de dados que podem ser gerados em aquisições de sensoriamento remoto, a utilização de modelos classificadores e a possibilidade de aceleração do processo de classificação através de arquiteturas de hardware dedicadas, pode-se utilizar o paradigma de computação na borda, que consiste em mover o processamento e armazenamento dos dados coletados para perto do usuário final, dispositivo ou sensor (KOUBAA et al., 2023), indo em um caminho oposto à tendência de computação na nuvem.

5.1 Materiais e Métodos

Esta seção apresenta os materiais e métodos utilizados no desenvolvimento e testes do projeto.

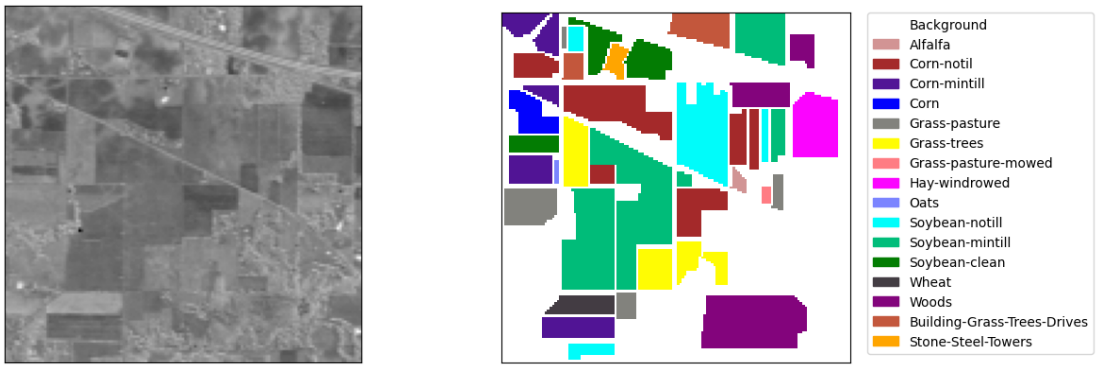
5.1.1 Conjuntos de dados

Para a obtenção dos resultados, são utilizadas três imagens hiperespectrais previamente classificadas para a aplicação de métodos de classificação supervisionados. Cada uma das imagens forma um conjunto de dados a ser analisado, os quais serão detalhados nas seções a seguir, mostrando as principais características de cada imagem.

5.1.1.1 Indian Pines (IP)

A cena de Indian Pines foi adquirida pelo sensor *Airbone Visible/Infrared Imaging Spectrometer* (AVIRIS) (GREEN et al., 1998) em 1992 no estado de Indiana e consiste de uma imagem com resolução de 145×145 pixels e 224 bandas espectrais entre 400 e 2500 nm. Dessas, 24 bandas são removidas devido à absorção de água ou por serem bandas nulas (PAOLETTI et al., 2019). A Figura 23 mostra em (a) a centésima banda que compõe a imagem hiperespectral, com coloração em escala de cinza. Em (b) tem-se o *ground truth* da cena, mostrando espacialmente a distribuição das classes. Há um total de 16 classes rotuladas, totalizando 10.249 pixels. Os demais pixels compõem o *background*, correspondendo a segmentos que não foram rotulados.

Figura 23: Indian Pines



(a) 100ª banda de Indian Pines (b) *Ground truth* de Indian Pines

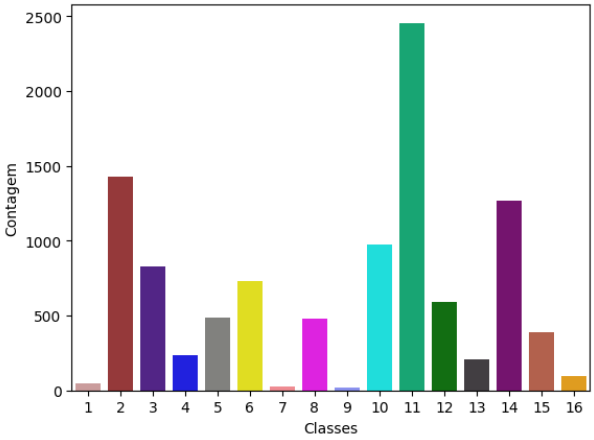
Fonte: Elaborado pelo autor (2023)

A Figura 24 apresenta a distribuição de frequência das classes rotuladas seguindo a mesma ordem e cores da Figura 23(b). É possível observar que o conjunto de dados obtido a partir da imagem é desbalanceado, sendo a classe menos frequente ocorrendo em 20 pixels, enquanto a mais frequente aparece em 2455 pixels. Tal fato se dá devido à baixa resolução da imagem e isso torna a aplicação dos algoritmos de aprendizado de máquina dificultada.

5.1.1.2 Salinas (SA)

Assim como Indian Pines, a cena de Salinas também foi adquirida através do sensor AVIRIS em Salinas Valley, na Califórnia. A imagem possui resolução de 512×217 pixels, com 224 bandas coletadas entre 400 e 2500 nm, com resolução espacial de $3,7$ metros por pixel. Das 224 bandas, 20 são descartadas devido à absorção de água e ruído (PAOLETTI

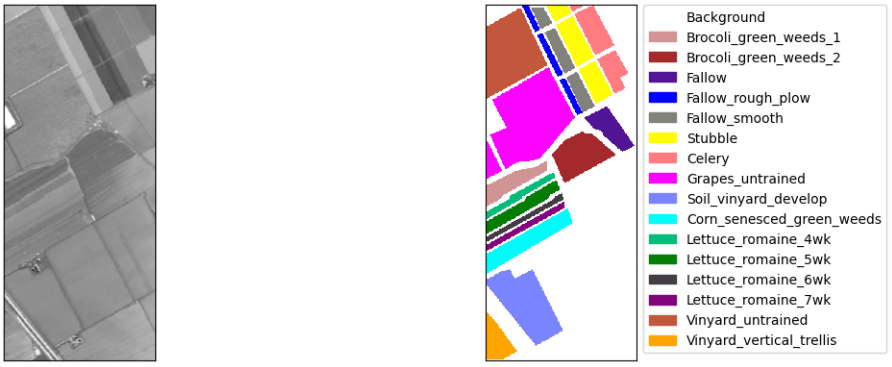
Figura 24: Distribuição de frequência das classes de Indian Pines



Fonte: Elaborado pelo autor (2023)

et al., 2019). A Figura 25 mostra a centésima banda da cena em (a) e em (b) o *ground truth*. A legenda mostra um total de 16 classes rotuladas, totalizando 54.129 pixels com classe definida dentro dos 111.104 que compõem a imagem.

Figura 25: Salinas

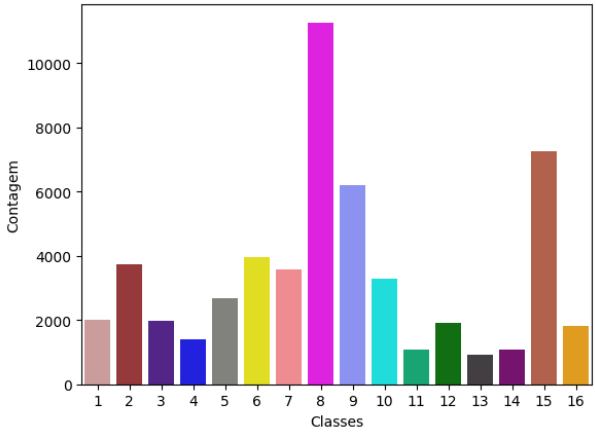


100ª banda de Salinas *Ground truth* de Salinas

Fonte: Elaborado pelo autor (2023)

A distribuição das classes pode ser vista na Figura 26. O conjunto de dados originário da imagem também apresenta características de desbalanceamento dos dados, porém, em menor grau se comparado à Indian Pines, visto que a cena de Salinas possui maior resolução. A classe menos frequente apresenta 916 instâncias e a mais frequente 11.271 instâncias.

Figura 26: Distribuição de frequência das classes de Salinas



Fonte: Elaborado pelo autor (2023)

5.1.1.3 Universidade de Pavia (UP)

A cena da Universidade de Pavia (Itália) foi adquirida com o sensor *Reflective Optics System Imaging Spectrometer* (ROSIS), o qual possui menor faixa espectral que o AVIRIS, com apenas 115 bandas entre 430 e 960 nm, porém, com resolução espectral superior (5 nm) (KUNKEL et al., 1988). A cena possui resolução de 610×340 pixels e resolução espacial de $1,3$ metros por pixel. Após o descarte de algumas das bandas, restam 103 para a análise da imagem entre 430 e 860 nm (PAOLETTI et al., 2019). A 50ª banda que compõe a imagem hiperespectral em questão pode ser vista na Figura 27(a) e na Figura 27(b) o *ground truth* correspondente, com um total de nove classes definidas, totalizando 42.776 pixels rotulados e 164.624 sem rótulo definido, formando o *background* da imagem.

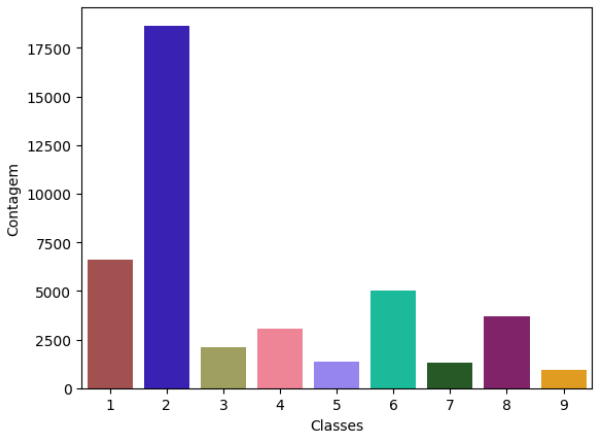
Figura 27: Cena da Universidade de Pavia

(a) 50ª banda da cena UP (b) *Ground truth* da cena UP

Fonte: Elaborado pelo autor (2023)

A distribuição das classes pode ser vista tanto na Figura 28. Por se tratar de uma imagem com maior resolução e menor quantidade de classes se comparada a Salinas e Indian Pines, apesar do desbalanceamento de classes, a classe menos frequente ainda possui uma quantidade maior de instâncias (947). Já a classe mais frequente aparece em 18.649 pixels.

Figura 28: Distribuição de frequência das classes da cena da Universidade de Pavia



Fonte: Elaborado pelo autor (2023)

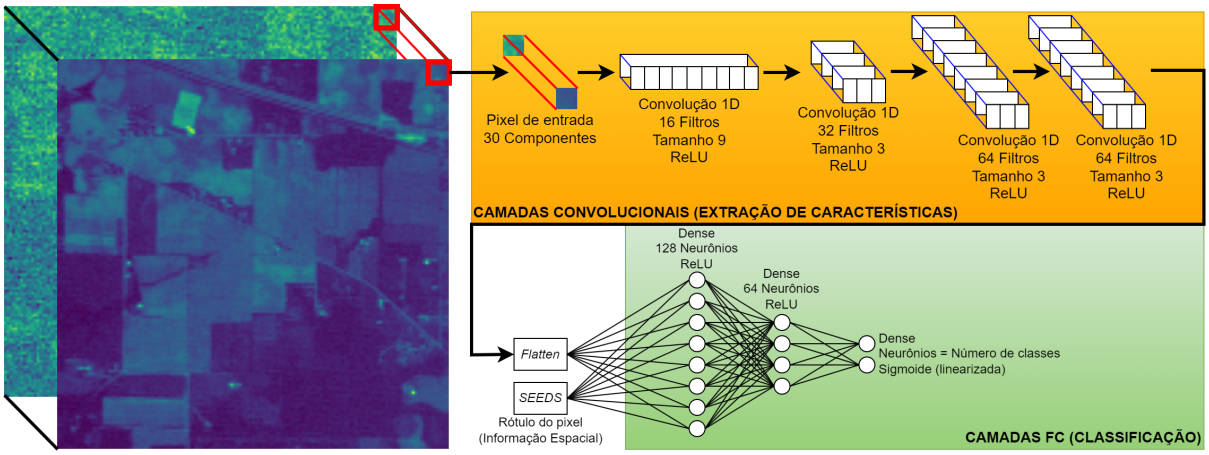
5.1.2 Arquitetura da rede neural utilizada

A arquitetura da rede neural utilizada foi inicialmente proposta por Viel et al. (2023) e modificada posteriormente por Souza (2023) para incluir informações espaciais na rede obtidas através do algoritmo de segmentação SEEDS. A Figura 29 mostra a arquitetura da rede neural que será utilizada para a geração e avaliação dos resultados, sendo igual à proposta por Souza (2023), porém, diferindo apenas na função de ativação da última camada, sendo a proposta pelo autor a função *softmax* e a aqui utilizada a sigmoide.

Considerando que a HSI passou pelo algoritmo PCA para a redução de dimensionalidade, são utilizadas as 30 principais componentes para a rede neural treinada para a HSI de Indian Pines e 15 para as demais. O algoritmo SEEDS funciona com três canais de imagem e, portanto, são utilizadas as três principais componentes provenientes do PCA. A seguir, será descrita a arquitetura da rede considerando o caso em que a entrada da rede possui tamanho 30.

O trecho em laranja da imagem apresenta as camadas convolucionais responsáveis pela extração de características espectrais do pixel. A primeira camada realiza a convolução da entrada com 16 filtros de tamanho 9, resultando em 16 saídas de tamanho 22. A

Figura 29: Arquitetura da rede neural em análise



Fonte: Elaborado pelo autor (2024)

segunda camada convolucional aplica 32 filtros (de profundidade 16, igual à quantidade de entradas) de tamanho 3 sobre cada um dos 16 vetores de saída da camada anterior, resultando em uma saída de 32 vetores de tamanho 20.

A camada convolucional seguinte faz o uso de 64 filtros de tamanho 3 e, de forma análoga à camada anterior, a saída da camada consiste em 64 vetores de tamanho 20. Por fim, a última camada convolucional usa os mesmos parâmetros da anterior, resultando ao final das camadas convolucionais em 64 vetores de tamanho 16. A operação de *flatten* transforma os 64 vetores de tamanho 16 em apenas um de tamanho 1024. Após isso, o rótulo de um pixel providenciado pelo algoritmo SEEDS é concatenado ao resultado do *flatten*, resultando em uma entrada de tamanho 1025.

Os 1025 valores de entrada passam por uma primeira camada de 128 neurônios, em seguida por uma de 64 e, por fim, uma camada cuja quantidade de neurônios depende da quantidade de classes do conjunto de dados utilizado (camada de saída). Com exceção da última, todas as camadas utilizam a função de ativação ReLU (Tabela 1), cuja implementação em hardware é mais simples devido à característica da função não aplicar operações matemáticas sobre o valor de entrada, realizando-se apenas uma comparação para verificar se o valor é maior que zero ou não.

No que diz respeito à camada de saída da rede, responsável pela classificação, Souza (2023) faz o uso da função de ativação *softmax*, cuja equação é exposta na Tabela 1. Tal função recebe os valores dos neurônios e os transforma em uma distribuição de probabilidade correspondente a cada classe. A soma da probabilidade (saída de cada neurônio da última camada) do pixel corresponder a cada classe é 100% e o neurônio com a saída de

maior valor corresponde à classe mais provável do pixel que entrou na rede. Observando a função *softmax*, a saída de um neurônio depende dos demais neurônios da mesma camada, além da utilização da função exponencial inúmeras vezes. Considerando que a saída dos neurônios finais pode não ser realizada em paralelo, tal dependência e a quantidade de operações realizadas aumentam o custo da implementação dessa em hardware.

Com isso, a *softmax* foi substituída pela sigmoide, função de ativação também utilizada em camadas finais de redes neurais e utilizada normalmente para a solução de problemas binários, onde só há duas classes. Porém, a *softmax*também pode ser aplicada para problemas multi-classes, onde quanto maior a saída de um determinado neurônio, maior a chance da entrada pertencer à classe correspondente em detrimento de todas as demais classes. Ou seja, o problema de múltiplas classes é resolvido realizando-se a comparação de cada saída com todas as outras. Além disso, a função sigmoide não possui dependência com os outros neurônios, sendo que as saídas podem ser calculadas uma por vez sem atraso. Apesar da expressão da função sigmoide possuir operações de divisão e exponencial, a função pode ser aproximada por segmentos de reta, cujo processo de linearização será apresentado na próxima seção.

As Tabelas 3, 4 e 5 mostram um resumo dos parâmetros de cada camada para cada modelo treinado a partir de cada imagem hiperespectral. Como já explicado anteriormente, os três modelos possuem camadas semelhantes, diferindo apenas nas camadas de entrada e de saída.

Tabela 3: Descrição do modelo para a HSI IP

Camada	**Formato de saída**	**Número de parâmetros**
input_1	(30,1)	0
conv1d_1	(22,16)	160
conv1d_2	(20,32)	1568
conv1d_3	(18, 64)	6208
conv1d_4	(16, 64)	12352
flatten	(1024)	0
input_2	(1)	0
concatenate	(1025)	0
dense1	(128)	131328
dense2	(64)	8256
dense3	(16)	1040

Fonte: Elaborado pelo autor (2024)

O modelo da HSI IP possui um total de 160.912 parâmetros treináveis com representação *float32* (4 Bytes), totalizando $628,56$ kB. Já para os outros dois modelos, tem-se

148, 56 kB e 146, 79 kB.

Tabela 4: Descrição do modelo para a HSI SA

Camada	**Formato de saída**	**Número de parâmetros**
input_1	(15,1)	0
conv1d_1	(7,16)	160
conv1d_2	(5,32)	1568
conv1d_3	(3, 64)	6208
conv1d_4	(1, 64)	12352
flatten	(64)	0
input_2	(1)	0
concatenate	(65)	0
dense1	(128)	8448
dense2	(64)	8256
dense3	(16)	1040

Fonte: Elaborado pelo autor (2024)

Tabela 5: Descrição do modelo para a HSI UP

Camada	**Formato de saída**	**Número de parâmetros**
input_1	(15,1)	0
conv1d_1	(7,16)	160
conv1d_2	(5,32)	1568
conv1d_3	(3, 64)	6208
conv1d_4	(1, 64)	12352
flatten	(64)	0
input_2	(1)	0
concatenate	(65)	0
dense1	(128)	8448
dense2	(64)	8256
dense3	(9)	585

Fonte: Elaborado pelo autor (2024)

5.1.2.1 Linearização da função sigmoide

A função de ativação sigmoide não é linear e é dada pela Equação 5.1. Para o processo de linearização, foram definidos os limites da função, visto que $\lim_{x\to\infty}\sigma(x) = 1$ e $\lim_{x\to-\infty}\sigma(x) = 0$. Como a função é simétrica em forma em torno de $x = 0$ e considerando o valor dos limites, foi escolhido inicialmente o intervalo $x \in [-4, 4]$, onde $\sigma(4) \approx 0,98$ e $\sigma(-4) \approx 0,02$, que são valores próximos dos limites tendendo a $\pm\infty$.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{5.1}$$

Para o intervalo $-4 \leq x \leq 4$, escolhe-se quantos segmentos de reta serão utilizados para a aproximação. Sendo N_r a quantidade de segmentos, há $P_r = N_r + 1$ pontos para formar os segmentos. Para redução de complexidade, os pontos foram considerados linearmente espaçados no eixo x, pois sabe-se que é possível realizar ajustes nas posições dos pontos para se obter melhores aproximações. Considerando cada ponto p_i (onde $i = 1, ..., P$) contendo coordenadas $(x; y)$, onde as coordenadas x são dadas pelo espaçamento entre os pontos e a coordenada y pela função sigmoide no ponto. Por exemplo, para p_0 é um ponto com coordenadas $(-4; 0,02)$ e p_P com $(4; 0,98)$. Os pontos intermediários são calculados de forma análoga.

Se o intervalo $[-4, 4]$ for aproximado por $N_r = 3$ retas, os pontos intermediários estarão nos valores de $x = \{-1,333; 1,333\}$. Calculando-se as coordenadas de todos os pontos, é possível utilizar dois pontos consecutivos para se definir a equação da reta que os une. A equação da reta substitui a função sigmoide no intervalo entre os pontos. O Apêndice A mostra o resultado da aproximação da função para 3, 5 e 7 segmentos de reta. O cálculo do erro quadrático médio mostra que, quanto maior a quantidade de segmentos, melhor a aproximação. O erro para 3 segmentos foi de $0,062\%$, $0,016\%$ para 5 e $0,007\%$ para 7 segmentos. As equações obtidas podem ser vistas no Apêndice B.

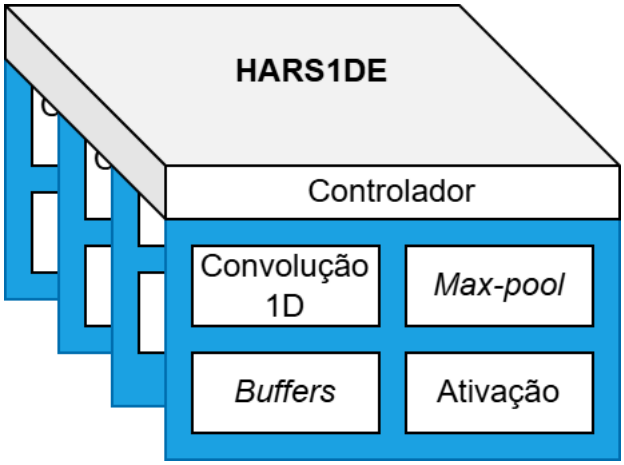
5.1.3 Topologias da arquitetura

A primeira topologia explorada neste trabalho é composta apenas de uma arquitetura da Figura 15. Nesse caso, uma entrada é processada por vez dentro da arquitetura, que executa a função de todas as camadas da rede. Essa topologia é denominada apenas como HARS1DE ou base.

A segunda topologia é feita para realizar o processamento de múltiplas entradas da rede neural de forma paralela. Como o comportamento da arquitetura é determinístico, pode-se utilizar o mesmo controlador e replicar o *datapath*, pois as mesmas operações serão realizadas em paralelo nos mesmos ciclos de *clock*. Dessarte, todas as entradas da rede devem estar disponíveis ao mesmo tempo para se iniciar o processo. Essa arquitetura é denominada HARS1DE-p.

Por fim, a última topologia consiste em conectar múltiplos blocos da Figura 15 em cascata, onde cada bloco fica responsável por executar uma camada da rede neural. Dessa forma, é possível fazer um *pipeline* da execução de cada camada, onde cada bloco processa uma camada diferente. A configuração é realizada apenas uma vez. Pontos negativos dessa proposta são o fato de que, por exemplo, se apenas a função *max-pool* for realizada, haverá

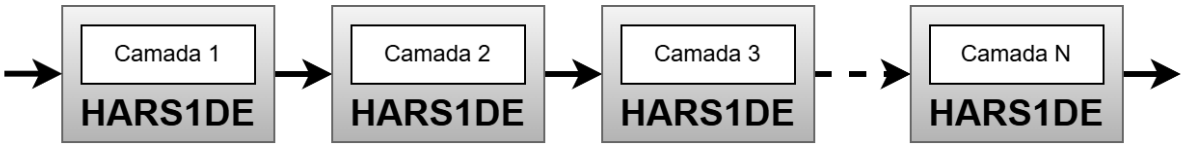
Figura 30: Topolgia de arquitetura em paralelo (HARS1DE-p)



Fonte: Elaborado pelo autor (2024)

estruturas de hardware vacantes em processamento, além da grande dependência de dados entre as camadas. Essa arquitetura é denominada HARS1DE-c.

Figura 31: Topologia de arquitetura em cascata (HARS1DE-c)



Fonte: Elaborado pelo autor (2024)

5.1.4 Métricas para análise de tempo

Para a realização de estimativas de tempo de processamento nos blocos da arquitetura proposta, é necessária uma modelagem do comportamento de cada bloco, considerando a disponibilidade das entradas e o processamento das saídas. Para o *array* de *Processing Elements*, considera-se que a primeira entrada e o primeiro peso estão disponíveis simultaneamente na entrada para serem lidos. Cada PE realiza uma convolução resultando em uma saída com tamanho (E) dado pela Equação 2.3. Para que cada saída seja computada, são realizadas uma quantidade de multiplicações igual ao tamanho do filtro (S). Logo, a quantidade de multiplicações e acumulações de um PE processando uma convolução é dada por:

$$MAC_{PE} = S \times E \tag{5.2}$$

Cada soma parcial leva S ciclos para ser computada mais dois de atraso para armazenamento no *buffer* de saída e demais modificações em *flags* internas ao componente. Logo, um PE processa uma convolução em:

$$T_{PE} = (S + 2) \times E \tag{5.3}$$

Sendo uma soma parcial passada para o PE seguinte, se necessário, após $S + 2$ ciclos. Apesar do PE seguinte necessitar da soma parcial do PE atual para computar sua primeira saída, não há atrasos gerados. Considerando o menor filtro que possa ser utilizado como possuindo tamanho 2 e a menor entrada que, ao ser convolucionada com esse filtro, produza uma saída de tamanho 2, tem-se uma entrada de tamanho 3, onde essa condição resulta na computação mais rápida de somas parciais, realizando apenas duas operações MAC. Considerando-se uma situação com dois PEs, analisando-se a distribuição das entradas e pesos nos PEs, o PE_1 começa seu processamento no primeiro ciclo devido à presença dos dados e, de forma análoga, o PE_2 começa seu processamento no ciclo 4, onde o primeiro valor da entrada de tamanho 3 é disponibilizado ao PE (o peso já está disponível desde o ciclo 3 devido ao seu tamanho). De acordo com a Equação 5.3, a primeira soma parcial é disponibilizada no ciclo $S + 2 = 4$ para o segundo PE, que pode ler a soma parcial em qualquer momento dentro do intervalo em que está realizando a primeira série de MAC, antes de deslizar o filtro, entre os ciclos 4 (início do processo) e 6 (final das primeiras operações de MAC). Sendo assim, sempre que um PE necessitar de uma soma parcial, essa estará disponível.

Tal modelagem de tempo dos PEs foi utilizada para a criação do modelo comportamental da arquitetura, considerando-se os ciclos para a distribuição dos dados na arquitetura, bem como a dependência entre PEs. Além disso, definiu-se a métrica do fator de utilização (f_u) dos PEs por ciclo como sendo:

$$f_u = \frac{\sum_{n=1}^{N_c} \frac{\#PE_{\mathrm{MAC}}}{\#PE_{\mathrm{act}}}}{N_c} \tag{5.4}$$

onde N_c é a quantidade de ciclos de processamento, $\#PE_{MAC}$ é a quantidade de PEs realizando operações de MAC no ciclo e $\#PE_{act}$ é a quantidade de PEs ativos no processamento. Essa métrica aponta, em média, quantos PEs estavam realizando operações de MAC durante cada ciclo de *clock*, tomando como referência a quantidade de PEs ativos no processamento da camada. Também pode ser observado como um indicador de

paralelismo, mostrando que múltiplas operações estão sendo realizadas ao mesmo tempo.

Já o bloco de *pooling* do tipo *max-pool* tem seu processamento dependente do *stride* e do tamanho da janela a ser aplicada. Seja E_m o tamanho da janela de *pooling* e U_m o *stride*, o processamento inicia após uma janela ter sido completamente preenchida, gerando E_m ciclos de atraso. Em seguida, cada saída é gerada em valores múltiplos de U_m, tendo-se a equação:

$$T_{mp} = E_m + U_m \times I_{mp} \tag{5.5}$$

onde I_{mp} corresponde ao tamanho da entrada a ser processada pelo bloco.

O bloco de reordenação pode receber múltiplas entradas e em paralelo para reordená-las. Considerando que o bloco recebe N_{re} entradas de tamanho I_{re} e que há dois ciclos de atraso para soma do *bias* e armazenamento da saída, tem-se:

$$T_{re} = 2 + N_{re} \times I_{re} \tag{5.6}$$

No geral, a arquitetura leva ainda 13 ciclos de configuração no início do processamento de cada camada e mais uma quantidade de ciclos igual ao tamanho do *bias* da camada antes de que os pesos comecem a ser processados. A arquitetura processa os blocos em paralelo, comportamento considerado no modelo representativo da arquitetura e utilizado para estimativas de tempo de processamento.

Outro parâmetro importante que pode ser abstraído teoricamente da arquitetura é o seu desempenho em termos de operações por segundo. Como um MAC consiste em duas operações e cada PE pode realizar um MAC por ciclo, o desempenho máximo que a arquitetura pode atingir é:

$$Des. = \#PE \times 2 \times f \tag{5.7}$$

onde f é a frequência de operação do hardware.

6 Resultados e Discussões

A arquitetura proposta foi sintetizada tendo-se a FPGA XC7Z020-CLG484 como base, sendo essa presente na placa de desenvolvimento ZedBoard Zynq-7000, contendo um *System on a Chip* (SoC) com um processador ARM, a qual pode vir a ser utilizada futuramente para a realização de testes físicos. Com a síntese, foi possível obter resultados referentes à utilização de recursos da FPGA. Além disso, a linguagem de programação Python foi utilizada para se observar resultados de acurácia das redes neurais quando adaptadas ao hardware, observando-se o nível de quantização e a aproximação da função sigmoide. Também foi realizada uma representação comportamental da arquitetura em Python para se observar resultados de tempo de execução para o processamento das redes neurais alvo do estudo de caso nas diferentes variações de arquitetura. Através de tal representação, também foram obtidas estimativas em relação ao uso dos PEs para diferentes parâmetros de configuração das CNNs. Para as estimativas de tempo, foi considerada a FPGA funcionando com um *clock* de 100 MHz, estando dentro dos limites da FPGA e dos limites encontrados na síntese da arquitetura. Para a realização das estimativas de tempo, foram assumidos alguns pressupostos, descritos a seguir.

A FPGA em questão possui 140 blocos de BRAM com $4,5$ kB cada, totalizando $630kB$. Considerando que o maior modelo treinado possui $628,56$ kB com uma representação de 32 bits, se esse for representado com uma quantização reduzida de 16 bits, a memória interna da FPGA pode conter os modelos. Sendo assim, assume-se que a memória interna contém o modelo completo e que a cada ciclo um valor da rede está disponível na entrada do acelerador.

Em relação às entradas da rede neural recebidas externamente, a maior possui tamanho 30+1 (devido à concatenação da informação espacial) e o processamento dessa entrada leva muitos ciclos, de modo que o atraso só impacta o início do processo. Mesmo com a latência de leitura de uma memória externa, o atraso é muito menor que a quantidade de ciclos de processamento na arquitetura, então tal valor de latência será desconsiderado nas estimativas de tempo para simplificação. Além disso, enquanto uma entrada é classificada,

outras podem ter sido acumuladas em *buffers* internos.

Para os resultados de tempo, foram utilizados como base de comparação duas execuções no Google Colab, fazendo o uso da CPU providenciada pela plataforma, sendo um AMD EPYC 7B12 com $2,25$ GHz. Através do Colab, também foi utilizada a GPU NVIDIA T4. Além disso, foi realizado o processamento localmente em uma máquina com um processador AMD Ryzen 7 7800X3D e frequência de $4,1$ GHz.

6.1 Acurácia

Para os resultados de acurácia das redes neurais do estudo de caso, foram realizados testes com a rede neural proposta por (SOUZA, 2023) para comparação e testes com versões modificadas dessa. Inicialmente, foi experimentada a mudança da função de ativação da camada de saída, sendo originalmente a *softmax*, onde avaliou-se a mudança para a sigmoide (Seção 5.1.2). Para cada base e função de ativação, foram realizados cinco treinamentos e calculou-se a média da acurácia. A Tabela 6 mostra a acurácia obtida com ambas as funções de ativação.

Tabela 6: Comparação da acurácia média com as funcções *softmax* e sigmoide

HSI	**Acurácia média(%)**	
	Softmax	**Sigmoide**
IP	92,05	91,07
SA	99,25	99,24
UP	98,52	98,40

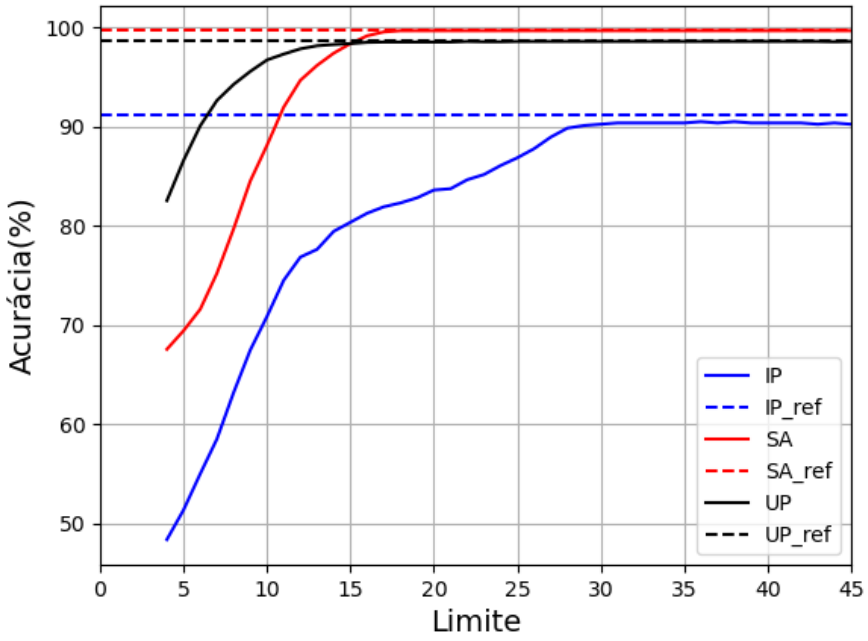
Fonte: Elaborado pelo autor (2025)

A partir dos resultados, é notável que, para as HSIs alvo de estudo desse trabalho e, consequentemente, as redes treinadas para cada uma, a mudança da função de ativação para sigmoide não representou mudanças significativas na acurácia, havendo uma diferença mais expressiva para a HSI de IP, com $0,98\%$. Para as demais, a diferença foi $0,01\%$ para a cena de SV e $0,12\%$ para a HSI de UP. Após tal verificação, prosseguiu-se para a implementação da função sigmoide aproximada por retas, bem como o impacto da substituição na acurácia.

Testes realizados mostraram que os limites impostos para a função sigmoide linearizada na Seção 5.1.2.1 ($[-4, 4]$) provocaram redução de acurácia e que limites maiores da função são tão impactantes quanto a quantidade de segmentos de retas da representação. Sendo assim, para cada HSI, foi treinado um modelo com uma acurácia de referência e

substituiu-se a função de ativação desse modelo, onde foi refeita a validação para se obter a acurácia para múltiplos valores de limite da função sigmoide. A Figura 32 mostra a acurácia em função dos limites da função sigmoide aproximada. Todos os testes foram realizados com uma aproximação por três retas, sendo a mais simples de ser implementada.

Figura 32: Evolução da acurácia em função dos limites da função sigmoide linearizada



Fonte: Elaborado pelo autor (2025)

Percebe-se que para intervalos maiores que $[-30, 30]$, não há impacto significativo na acurácia onde, para todas as HSIs, a acurácia converge para o valor de referência da função sigmoide original. O comportamento do gráfico pode ser explicado devido à distribuição de valores de saída da última camada FC antes de passar pela função de ativação. Para valores de intervalo pequenos, se a entrada da função for maior que o limite superior, a saída será igual a 1. Como a classe mais provável dentre os valores dos neurônios de saída é dada pelo maior valor, se múltiplas entradas da função de ativação forem maiores que o limite superior, tem-se múltiplas classes sendo apontadas igualmente como a mais provável de ser a correta. Quando a função utilizada em Python se depara com tal situação, o máximo é considerado no neurônio que aparece primeiro na lista de neurônios de saída, causando falsas classificações e reduzindo a acurácia. Ao aumentar o intervalo de forma a abranger todos os valores de entrada da camada de ativação, o segmento de reta é crescente tal como a função sigmoide, fazendo com que não haja valores iguais na saída e que os maiores valores de entrada serão os maiores valores de saída também.

No geral, a substituição da função sigmoide por sua aproximação através de segmentos de reta não gerou mudanças significativas no comportamento do modelo, sendo um ponto positivo para a portabilidade desses em hardware.

6.1.1 Quantização

Para a análise da quantização, foi realizado o processo de pós-quantização para os modelos já treinados. Para a verificação da quantidade de bits necessária para a representação dos dados na arquitetura, inicialmente investigou-se o comportamento das entradas dos modelos, das saídas de todas as camadas e dos pesos, observando os valores máximos e mínimos para eliminar menores níveis de quantização. A Tabela 7 mostra os valores máximos e mínimos dos modelos referentes a cada uma das HSIs.

Tabela 7: Máximos e mínimos dos valores dos modelos

		IP	**SA**	**UP**
Pesos	**Mínimo**	-1.22	-1.88	-1.54
	Máximo	0.89	1.21	0.81
Entradas	**Mínimo**	-5.64	-4.64	-5.77
	Máximo	78	91	44
Saídas	**Mínimo**	-163.62	-219.08	-117.83
	Máximo	88.12	126.74	55.41

Fonte: Elaborado pelo autor (2025)

A partir da tabela, observa-se que os pesos estão distribuídos em valores menores próximos a zero. Além disso, o valor máximo de entrada é referente à informação espacial proveniente do algoritmo de segmentação, que fornece o rótulo do pixel à rede neural. Considerando-se o mínimo $(-219,08)$ e o máximo $(126,74)$ da tabela, uma representação de 8 bits seria insuficiente, visto que mesmo considerando todos os dados como valores inteiros, a representação de 8 bits com sinal só atinge o intervalo de -128 a 127. O próximo valor de quantização a ser averiguado é de 16 bits, porém, representando não apenas valores inteiros. Para atestar a suficiência da representação, foram novamente treinados modelos para as imagens hiperespectrais e em seguida a representação foi reduzida de ponto flutuante com 32 bits para 16 bits, gerando os resultados de acurácia da Tabela 8.

Para os modelos testados, apenas no realizado para a HSI de Indian Pines houve uma degradação de $0,13\%$ na acurácia. Em seguida, verificou-se o impacto na acurácia devido à realização de uma quantização linear onde, para uma representação de 16 bits, foram considerados níveis de $0,1$ e $0,001$, visto que com esses valores, o intervalo de representação

Tabela 8: Comparação da acurácia para float32 e float16

HSI	**Acurácia**	
	float32	***float16***
IP	90,51	90,38
SA	99,68	99,68
UP	98,60	98,60

Fonte: Elaborado pelo autor (2025)

se torna, respectivamente, $[-3276,8; 3276,7]$ e $[-327,68; 327,67]$, abrangendo os valores da Tabela 7. Também foram testados os níveis de quantização de 10^{-6} e 2×10^{-7}, que seriam atingidos por uma representação de 32 bits com intervalos de $[-2.147,48; 2.147,48]$ e $[-429,50; 429,50]$. A Tabela 9 mostra os resultados de acurácia para os diferentes níveis de quantização em análise.

Tabela 9: Comparação entre diferentes níveis de quantização linear

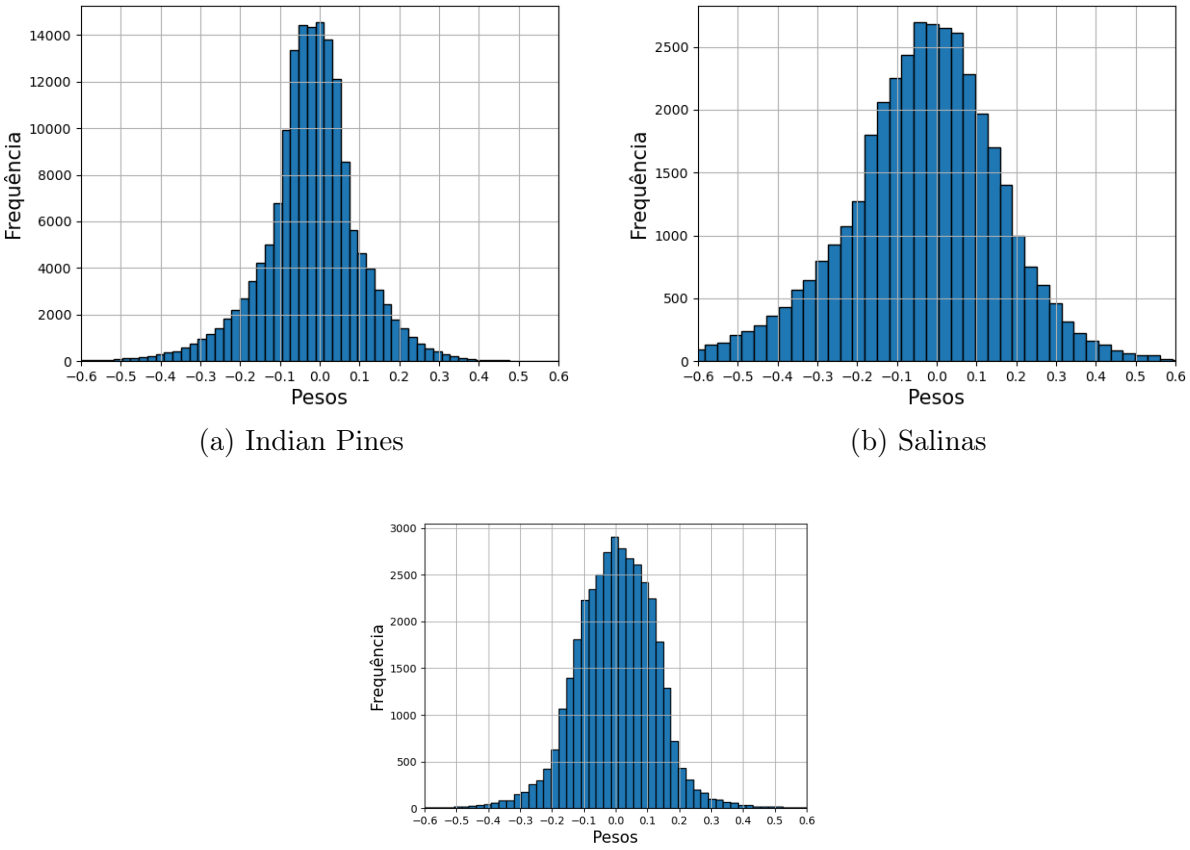
HSI	**Acurácia(%)**				
	Referência	$\mathbf{10^{-1}}$	$\mathbf{10^{-2}}$	$\mathbf{10^{-6}}$	$\mathbf{2 \times 10^{-7}}$
IP	90,51	68,79	78,02	89,47	90,25
SA	99,68	90,49	96,36	99,68	99,68
UP	98,60	84,11	98,10	98,57	98,60

Fonte: Elaborado pelo autor (2025)

Observa-se que para os modelos classificadores das HSIs de SA e UP, a representação de 16 bits com quantização linear de $0,01$ por bit não causou impacto significativo na acurácia. Com uma representação superior a partir de 10^{-6}, pode-se dizer que não há perda de acurácia para esses modelos. Já para a cena de IP, mesmo com uma quantização de 2×10^{-7}, não foi possível atingir o valor de referência. Analisando-se a distribuição de pesos dos modelos (Figura 33), para a base de IP, há uma grande quantidade de pesos próximos ao zero, que são mais impactados pela quantização, pois, dependendo do nível adotado, estes são zerados, excluindo-se sinapses da rede.

Ademais, a partir da Figura 33 também pode-se concluir que, devido à maior concentração de pesos ao redor do zero e menor para valores maiores, uma quantização não-linear pode ser eficiente na representação, como a utilizada pelo tipo de variável *float16*, possuindo intervalos menores para menores valores a serem quantizados, podendo-se atingir acurácias próximas à referência com representação em ponto flutuante com 32 bits.

Figura 33: Distribuição dos pesos entre os modelos para cada HSI



(a) Indian Pines

(b) Salinas

(c) Universidade de Pavia

Fonte: Elaborado pelo autor (2024)

6.2 Utilização de PEs

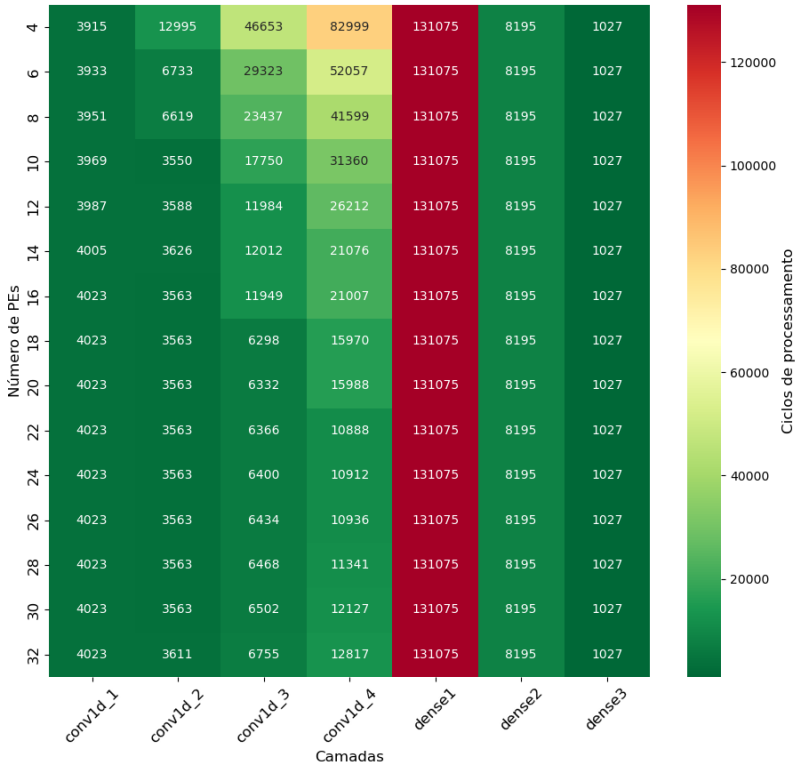
Para os resultados de desempenho, primeiramente é necessário se definir a quantidade de PEs a serem utilizados na arquitetura, pois essa impacta diretamente no tempo de processamento das camadas das redes neurais alvo deste estudo. Como a arquitetura se configura a partir dos parâmetros da camada a ser processada, o comportamento também depende desses. Como há diversos parâmetros de configuração, focou-se apenas nos parâmetros já definidos das redes testadas e observou-se o fator de utilização dos PEs no processamento, bem como a quantidade de ciclos. Foram observadas quantidades de elementos de processamento entre 4 e 32 apenas para o modelo treinado através da HSI de Indian Pines, que é o de maior quantidade de parâmetros. São consideradas apenas as camadas que realizam algum processamento, sendo essas as de convolução e as FC.

A Figura 34 mostra a quantidade de ciclos de processamento apenas do arranjo de PEs para cada camada estimados a partir das equações da Seção 5.1.4. Primeiro, nota-se que a camada que realiza mais processamento é justamente a com a maior quantidade de parâmetros, sendo a primeira camada *dense*, com uma entrada de tamanho 1025 e 128 neurônios. Nesse caso, as entradas e os pesos são distribuídos pelos PEs. Como os pesos entram um a um e são utilizados apenas uma vez em uma operação de multiplicação, isso faz com que não haja ganho de tempo, mesmo com quantidades maiores de PEs.

Em relação às camadas convolucionais, observando-se a Tabela 3, na maioria das situações, ocorre um aumento da quantidade de ciclos com o passar das camadas devido ao aumento de parâmetros. Na primeira camada convolucional que aplica 16 filtros em uma entrada, a entrada é repetida em todos os PEs até o ponto em que a quantidade de PEs é igual à quantidade de filtros (16), onde, a partir desse ponto, não há mais alteração na quantidade de ciclos. Nesse caso, nota-se que a estratégia de repetição das entradas ocasionou uma redução da quantidade de ciclos. A partir da segunda camada, onde a quantidade de entradas tende a crescer, quanto menor o número de PEs, mais entradas são armazenadas em um mesmo PE e, devido à dependência de dados, atrasos maiores são criados. Nessas camadas também é observado o comportamento de redução da quantidade de ciclos até o ponto em que as entradas não são repetidas mais nos PEs.

A Figura 35 mostra o fator de utilização dos PEs para cada camada estimado através da Equação 5.4. Percebe-se um comportamento inverso se comparado à quantidade de ciclos. Para as camadas convolucionais, quanto menor a quantidade de PEs, maior o fator de utilização, pois os PEs estarão com uma carga maior de dados acumulados para

Figura 34: Quantidade de ciclos para múltiplos PEs no modelo treinado para IP



Fonte: Elaborado pelo autor (2025)

processar, logo, múltiplos PEs estarão ativos por ciclo de *clock*. O comportamento das camadas *dense* se dá devido à distribuição dos pesos nos PEs e ao fato de que as operações de MAC são realizadas junto à chegada dos dados, resultando em uma execução em ordem, um PE por vez, reduzindo assim o fator de utilização.

Figura 35: Fator de utilização para múltiplos PEs no modelo treinado para IP

Fonte: Elaborado pelo autor (2025)

Com esses resultados, foi escolhido um arranjo de tamanho 18, o qual consegue manter um fator de utilização semelhante a quantidades maiores de PEs, bem como possui uma quantidade de ciclos semelhante para as primeiras camadas e não causa uma degradação considerável da quantidade de ciclos para as camadas convolucionais seguintes. Tal quantidade será utilizada como valor máximo, visto que HARS1DE-c pode ser configurada com múltiplos arranjos de PEs de tamanhos diferentes.

6.3 Utilização de recursos

Para a estimativa de utilização de recursos em FPGA, foram realizadas parcialmente as três topologias descritas no Capítulo 5, visto que nem todas as funções do controlador principal proposto no Capítulo 4 foram implementadas, logo, os resultados de recursos

são aproximações. HARS1DE foi realizada com um arranjo de 18 PEs. Cada *buffer* foi dimensionado considerando a quantidade máxima de dados que este pode armazenar durante a execução. Os que estão localizados entre os componentes possuem dimensões menores, pois armazenam os dados momentaneamente para garantir o fluxo de dados. As estruturas que de fato acumulam valores são os *buffers* principais da arquitetura, o de *bias* e os RFs dos PEs. A Tabela 10 mostra a quantidade de recursos da FPGA ocupados com a primeira arquitetura, com uso máximo de $23,43\%$ de LUTs, $3,61\%$ de *Flip-Flops* (FFs), $10,46\%$ de DSPs e $9,38\%$ de *Buffered Global Clock* (BUFG).

Tabela 10: Utilização de recursos da FPGA XC7Z020-CLG484 para HARS1DE

Recurso	**Estimativa**	**Disponível**	**Uso(%)**
LUT	12463	53200	23,43
FF	3844	106400	3,61
DSP	23	220	10,46
BUFG	3	32	9,38

Fonte: Elaborado pelo autor (2025)

Já para a arquitetura em paralelo, essa foi realizada com a replicação do *datapath* da arquitetura base quatro vezes, mantendo-se o mesmo controlador, para se manter a quantidade de uso de recursos abaixo do estipulado nos requisitos. A Tabela 11 mostra a utilização de recursos nessa situação, utilizando aproximadamente três vezes mais LUTs, DSPs e FFs em relação à solução anterior, e o uso de BUFG se manteve o mesmo.

Tabela 11: Utilização de recursos da FPGA XC7Z020-CLG484 para HARS1DE-P

Recurso	**Estimativa**	**Disponível**	**Uso(%)**
LUT	40287	53200	75,73
FF	14605	106400	13,73
DSP	74	220	33,64
BUFG	3	32	9,38

Fonte: Elaborado pelo autor (2025)

Por fim, para o último caso, a arquitetura foi parametrizada com uma quantidade de PEs variável para cada camada, sendo o máximo de 18 PEs. Tomando-se a Figura 34 como referência, a primeira camada convolucional possui 4 PEs, a segunda possui 10, a terceira e quarta 18. As camadas *dense* foram realizadas com apenas 4 PEs, totalizando 62 PEs. A Tabela 12 mostra a utilização de recursos nesse caso, onde a quantidade de LUTs excedeu o limite estipulado no Quadro 3 e houve um aumento dos demais recursos em comparação com as arquiteturas anteriores, visto que é a mais complexa.

Tabela 12: Utilização de recursos de FPGA para HARS1DE-c

Recurso	Estimativa	Disponível	Uso(%)
LUT	44882	53200	84,37
FF	14692	106400	13,81
DSP	93	220	42,27
BUFG	12	32	37,50

Fonte: Elaborado pelo autor (2025)

6.4 Desempenho

Considerando a quantidade de PEs escolhida, o modelo comportamental da arquitetura foi utilizado para a obtenção dos resultados estimados de tempo. Considerando a arquitetura HARS1DE e as equações na Seção 5.1.4, cada camada é processada por vez e apenas dois blocos são utilizados: arranjo de PEs e reordenação de saídas. O bloco de reordenação inclui apenas dois ciclos de atraso no processamento das saídas dos PEs e as escritas no *buffer* de saída também incluem mais um ciclo de atraso, de forma que é razoável considerar que a quantidade de ciclos para processar uma entrada é aproximadamente a quantidade de ciclos executados pelo arranjo de PEs para os modelos em teste. Já a arquitetura HARS1DE-p processa quatro vezes mais entradas por ciclo e a em cascata possui dependência de dados, sendo necessária a modelagem dos atrasos no *pipeline*.

A Tabela 13 mostra o desempenho teórico máximo atingido por cada uma das arquiteturas testadas, sendo calculado a partir da Equação 5.7 considerando a quantidade de PEs em cada arquitetura e um *clock* de 100 MHz. O desempenho da arquitetura em paralelo é 4× maior em relação à base devido ao paralelismo. Já o da arquitetura HARS1DE-c, apesar de ser similar ao da em paralelo, pode levar mais tempo para processar as entradas, até mesmo em relação à arquitetura base.

Tabela 13: Desempenho máximo teórico de cada topologia de arquitetura

	Desempenho(GOP/s)
HARS1DE	3,6
HARS1DE-p	14,4
HARS1DE-c	12,4

Fonte: Elaborado pelo autor (2025)

HARS1DE-c possui uma grande dependência de dados entre os blocos, pois a execução de uma camada depende do processamento da camada anterior, o que faz com que surja um atraso na realização da primeira camada *dense* (Figura 34) que leva mais tempo para ser

processada. Por exemplo, enquanto a arquitetura base levaria aproximadamente 340.000 ciclos para processar duas entradas (170.000 ciclos cada), a arquitetura em cascata, de acordo com a modelagem dos atrasos, levaria aproximadamente 424.000. O desempenho máximo da arquitetura só ocorre sob circunstâncias específicas no início do processamento. Após isso, a cada entrada processada, o tempo em que cada bloco de processamento de camada espera o processamento do bloco anterior tende a crescer.

Considerando que HARS1DE-p possui maior paralelismo e satisfaz o requisito de ocupação de recursos da FPGA e que a arquitetura em cascata não se mostrou eficiente, os resultados de comparação de tempo serão feitos apenas com a arquitetura em paralelo. A Tabela 14 mostra o tempo médio de execução para a classificação das HSIs completas, sendo a HSI de IP com 21.025 pixels, a de SA com 111.104 e de UP com 207.400 pixels. A classificação de cada imagem foi realizada cinco vezes para o cálculo do tempo médio e desvio padrão. Para tornar a comparação justa, visto que HARS1DE-p processa 4 entradas em paralelo, os modelos foram aplicados nas demais arquiteturas com um *batch* de tamanho 4. A estimativa de tempo da arquitetura HARS1DE-p foi realizada de acordo com os pressupostos do início deste capítulo e a modelagem do comportamento esperado pela arquitetura.

Tabela 14: Comparação do tempo de classificação das HSIs

HSI	**Tempo médio(s) (Desvio padrão)**			**Estimativa(s)**
	T4 (Colab)	**CPU (Colab)**	**Ryzen 7 7800X3D**	**HARS1DE-p**
IP	15,73 (4,48)	9,47 (0,77)	6,10 (0.07)	8,94
SA	60,65 (0,49)	39,75 (1,73)	31,19 (0.33)	9,22
UP	142,00 (0,01)	76,74 (4,81)	57,59 (0.47)	16,98

Fonte: Elaborado pelo autor (2025)

A partir da tabela, para a HSI de IP, o menor tempo foi no processador Ryzen 7. Como as redes neurais para SA e UP são semelhantes, o tempo de processamento em cada arquitetura se torna aproximadamente proporcional à quantidade de entradas classificadas. Visto que o modelo para IP é o maior, o tempo de processamento na arquitetura HARS1DE-p se aproxima do para a HSI SA, mesmo com uma quantidade maior de pixels desta. Como a quantidade de dados que entra na rede é muito menor que a quantidade de parâmetros da rede em si, e como a arquitetura proposta mantém o modelo de classificação armazenado internamente, não há acesso excessivo de memória para ler os parâmetros da rede e, consequentemente, o tempo de processamento varia em uma proporção menor.

Com isso, para a primeira HSI, a estimativa de aceleração máxima ocorre em relação

à GPU da plataforma Colab, sendo para cada HSI, respectivamente, $1,76\times$, $6,58\times$ e $8,36\times$. Além disso, dentre as arquiteturas testadas e comparadas, HARS1DE-p possui a flexibilidade de poder ser embarcada, visto que as demais são voltadas a plataformas de computador.

7 Considerações Finais

A computação na borda é um paradigma que tem ganhado força e, aliado a isso, pode-se ter a necessidade da aplicação de modelos de *machine learning* em dispositivos com limitações de recursos. Com isso, esse trabalho apresentou uma proposta de arquitetura de hardware voltada à aceleração de redes neurais convolucionais do tipo 1D, incluindo o processamento de camadas de *pooling* e *dense* e das funções de ativação ReLU e uma versão modificada da sigmoide.

A validação da arquitetura foi feita através de um estudo de caso na classificação de imagens hiperespectrais e foram experimentadas variações da proposta. Para cada rede neural analisada, foi observado o comportamento em cada arquitetura em função da quantidade de PEs utilizados, visto que uma quantidade maior de PEs pode melhorar o desempenho em termos de operações por segundo; porém, definiu-se o fator de utilização de PEs para se observar o quanto os elementos de processamento são utilizados na execução de cada camada das redes neurais.

O hardware (HARS1DE) foi idealizado como um bloco único capaz de realizar os processos descritos anteriormente, sendo capaz de computar uma CNN-1D completamente. Porém, foram testadas variações que incluem o processamento de múltiplas entradas em paralelo (HARS1DE-p), dado que o comportamento da arquitetura é determinístico, então uma das variações consiste em um bloco controlador com múltiplos *datapaths* de processamento. A última variação foi idealizada para processamento em cascata (HARS1DE-c), na tentativa de realizar um *pipeline*, onde cada bloco seria responsável pelo processamento de uma cada da CNN.

Dentre as três variações testadas, a execução em cascata se mostrou infrutífera devido à dependência de dados entre as camadas onde, se camadas interiores demoram muito tempo para realizar seu processamento, surge um atraso que se propaga para os demais processamentos, inviabilizando esse modelo. A variação que mostrou melhores resultados teóricos foi a HARS1DE-p, sendo capaz de processar até quatro instâncias ao mesmo

tempo.

Os testes foram realizados tanto em VHDL quanto em Python e, comparada com outras, a arquitetura proposta demonstrou um potencial de aceleração máxima de $8,36\times$ na classificação de uma das imagens hiperespectrais alvo de estudo. Ademais, o processo foi feito com utilização máxima de recursos de hardware abaixo do estipulado ($75,73\%$) e, por fim, mesmo com níveis de quantização reduzidos (16 bits), a arquitetura pode realizar o processamento sem perdas significativas de acurácia. Com isso, baseando-se nas hipóteses apresentadas na Seção 1.1.1, tem-se o aceite da hipótese H0.

7.1 Principais contribuições

Este trabalho se soma aos já existentes relacionados à arquiteturas aceleradoras de redes neurais, sendo esse especificamente voltado à redes neurais convolucionais do tipo 1D, voltada à aplicações de computação na borda e com margem para demais domínios de aplicação. Através dos testes realizados, foi constatado o potencial da arquitetura, visto que essa, mesmo funcionando com um nível de quantização reduzido e adaptações nos modelos de classificação, pode ser capaz de atingir acurácias semelhantes à execução em outras plataformas de hardware e estima-se ainda a capacidade de realizar o processo de forma acelerada fazendo o uso de recursos de FPGA em acordo com os requisitos propostos. Além disso, a arquitetura é escalável em termos de elementos de processamento e tem seu comportamento reconfigurável de acordo com cada camada a ser processada, garantindo maior flexibilidade na execução de CNNs diversas.

7.2 Trabalhos futuros

Devido ao fato deste trabalho ter se limitado a resultados obtidos de forma estimada, o principal trabalho futuro é a realização de experimentos práticos para a constatação dos resultados obtidos. Além da possibilidade de se incluir a análise do consumo de energia, que pode ser essencial em muitas aplicações de computação na borda. Adicionalmente, há a realização dos blocos externos à arquitetura, como controladores de acesso à memória e *buffers* para a distribuição dos dados nos casos das arquiteturas derivadas da base. Por fim, garantir o envio de valores parciais para fora da arquitetura pode ser uma exploração a ser realizada, de forma a possibilitar a aceleração de modelos classificadores maiores e a redução de *buffers* internos.

Referências

APICELLA, A. et al. A survey on modern trainable activation functions. *Neural Networks*, v. 138, p. 14–32, 2021. ISSN 0893-6080. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0893608021000344>.

ARDAKANI, A.; CONDO, C.; GROSS, W. J. Fast and efficient convolutional accelerator for edge computing. *IEEE Transactions on Computers*, v. 69, n. 1, p. 138–152, 2020.

ASIM, M. et al. A review on computational intelligence techniques in cloud and edge computing. *IEEE Transactions on Emerging Topics in Computational Intelligence*, v. 4, n. 6, p. 742–763, 2020.

BIOUCAS-DIAS, J. M. et al. Hyperspectral remote sensing data analysis and future challenges. *IEEE Geoscience and Remote Sensing Magazine*, v. 1, n. 2, p. 6–36, 2013.

BORZOV, S. M.; POTATURKIN, O. I. Efficiency of the spectral-spatial classification of hyperspectral imaging data. *Optoelectronics, Instrumentation and Data Processing*, v. 53, n. 1, p. 26–34, 01 2017. ISSN 1934-7944. Disponível em: <https://doi.org/10.3103/S8756699017010058>.

CAMPBELL, J. B.; WYNNE, R. H. *Introduction to Remote Sensing*. 5. ed. London: The Guilford Press, 2011.

CHANG, C.-I. *Hyperspectral Data Exploitation*: Theory and applications. 1. ed. Baltimore: Wiley, 2007.

CHEN, Y. et al. Diannao family: Energy-efficient hardware accelerators for machine learning. *Commun. ACM*, Association for Computing Machinery, New York, NY, USA, v. 59, n. 11, p. 105–112, 10 2016. ISSN 0001-0782. Disponível em: <https://doi.org/10.1145/2996864>.

CHEN, Y.-H.; EMER, J.; SZE, V. Using dataflow to optimize energy efficiency of deep neural network accelerators. *IEEE Micro*, v. 37, n. 3, p. 12–21, 2017.

CHEN, Y.-H. et al. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE Journal of Solid-State Circuits*, v. 52, n. 1, p. 127–138, 2017.

CHI, P. et al. Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory. In: . Seoul: [s.n.], 2016. p. 27–39.

Edmund Optics. *Hyperspectral and Multispectral Imaging*. c2023. `https://www.edmundoptics.com/knowledge-center/application-notes/imaging/hyperspectral-and-multispectral-imaging/`.

FABELO, H. et al. An intraoperative visualization system using hyperspectral imaging to aid in brain tumor delineation. *Sensors*, v. 18, n. 2, 2018. ISSN 1424-8220. Disponível em: <https://www.mdpi.com/1424-8220/18/2/430>.

FAUVEL, M. et al. Advances in spectral-spatial classification of hyperspectral images. *Proceedings of the IEEE*, v. 101, n. 3, p. 652–675, 2013.

FISCHER, W. A.; HEMPHILL, W. R.; KOVER, A. Progress in remote sensing (1972-1976). *Photogrammetria*, v. 32, n. 2, p. 33–72, 1976. Disponível em: <https://doi.org/10.1016/0031-8663(76)90013-2>.

FOLEY, J. D.; DAM, A. V. *Fundamentals of interactive computer graphics*. 1. ed. [s.l.]: Addison-Wesley, 1982.

FURIAN, P. H. *Espectro Eletromagnético*. c2023. `https://www.infoescola.com/fisica/espectro-eletromagnetico/`.

GHIMIRE, D.; KIL, D.; KIM, S.-h. A survey on efficient convolutional neural networks and hardware acceleration. *Electronics*, v. 11, n. 6, 2022. ISSN 2079-9292. Disponível em: <https://www.mdpi.com/2079-9292/11/6/945>.

GONZALEZ, R. C.; WOODS, R. E. *Digital Image Processing*. 2. ed. New Jersey: Prentice-Hall, Inc., 2002.

GREEN, R. O. et al. Imaging spectroscopy and the airborne visible/infrared imaging spectrometer (aviris). *Remote Sensing of Environment*, v. 65, n. 3, p. 227–248, 1998. ISSN 0034-4257. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0034425798000649>.

GURUNATH, R. et al. An overview: Security issue in iot network. In: *2018 2nd International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC), 2018 2nd International Conference on*. [S.l.: s.n.], 2018. p. 104–107.

GYANESHWAR, D.; NIDAMANURI, R. R. A real-time fpga accelerated stream processing for hyperspectral image classification. *Geocarto International*, Taylor & Francis, v. 37, n. 1, p. 52–69, 2022. Disponível em: <https://doi.org/10.1080/10106049.2020.1713231>.

HALLIDAY, D.; RESNICK, R.; WALKER, J. *Fundamentals of physics*. 9. ed. Baltimore: Wiley, 2011.

HUA, H. et al. Edge computing with artificial intelligence: A machine learning perspective. *ACM Comput. Surv.*, Association for Computing Machinery, New York, NY, USA, v. 55, n. 9, jan. 2023. ISSN 0360-0300. Disponível em: <https://doi.org/10.1145/3555802>.

IMANI, M.; GHASSEMIAN, H. An overview on spectral and spatial information fusion for hyperspectral image classification: Current trends and challenges. *Information Fusion*, v. 59, p. 59–83, 2020. ISSN 1566-2535. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1566253519307857>.

KAELBLING, L. P.; LITTMAN, M. L.; MOORE, A. W. Reinforcement learning: A survey. *Journal of artificial intelligence research*, v. 4, p. 237–285, 1996.

KHAN, M. J. et al. Modern trends in hyperspectral image analysis: A review. *IEEE Access*, v. 6, p. 14118–14129, 2018.

KIRANYAZ, S. et al. 1d convolutional neural networks and applications: A survey. *Mechanical Systems and Signal Processing*, v. 151, p. 107398, 2021. ISSN 0888-3270. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0888327020307846>.

KOUBAA, A. et al. Aero: Ai-enabled remote sensing observation with onboard edge computing in uavs. *Remote Sensing*, v. 15, n. 7, 2023. ISSN 2072-4292. Disponível em: <https://www.mdpi.com/2072-4292/15/7/1873>.

KUNG, H. T.; LEISERSON, C. E. Systolic arrays (for vlsi). In: SOCIETY FOR INDUSTRIAL AND APPLIED MATHEMATICS PHILADELPHIA, PA, USA. *Sparse Matrix Proceedings 1978*. [S.l.], 1979. v. 1, p. 256–282.

KUNKEL, B. et al. Rosis (reflective optics system imaging spectrometer): A candidate instrument for polar platform missions. *Optoelectronic Technologies for Remote Sensing from Space*, v. 868, p. 134–142, 1988.

LECUN, Y.; BENGIO, Y.; HINTON, G. Deep learning. *Nature*, v. 521, n. 7553, p. 436–444, 05 2015. ISSN 1476-4687. Disponível em: <https://doi.org/10.1038/nature14539>.

LIU, Z. et al. A heterogeneous processor design for cnn-based ai applications on iot devices. *Procedia Computer Science*, v. 174, p. 2–8, 2020. ISSN 1877-0509. 2019 International Conference on Identification, Information and Knowledge in the Internet of Things. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1877050920315611>.

MARTINS, L. A. et al. An svm-based hardware accelerator for onboard classification of hyperspectral images. In: . New York, NY, USA: Association for Computing Machinery, 2019. (SBCCI '19). ISBN 9781450368445. Disponível em: <https://doi.org/10.1145/3338852.3339869>.

MAZZIA, V. et al. Real-time apple detection system using embedded systems with hardware accelerators: An edge ai application. *IEEE Access*, v. 8, p. 9102–9114, 2020.

MICHIE, D.; SPIEGELHALTER, D. J.; TAYLOR, C. C. *Machine Learning, Neural and Statistical Classification*. [S.l.]: Prentice Hall, 1994. (Ellis Horwood Series in Artificial Intelligence).

MOOLCHANDANI, D.; KUMAR, A.; SARANGI, S. R. Accelerating cnn inference on asics: A survey. *Journal of Systems Architecture*, v. 113, 2021.

NASTESKI, V. An overview of the supervised machine learning methods. *Horizons. b*, v. 4, p. 51–62, 2017.

NAVALGUND, R. R.; JAYARAMAN, V.; ROY, P. Remote sensing applications: An overview. *Current Science*, v. 93, n. 12, p. 1747–1766, 2007.

PAOLETTI, M. et al. Deep learning classifiers for hyperspectral imaging: A review. *ISPRS Journal of Photogrammetry and Remote Sensing*, v. 158, p. 279–317, 2019. ISSN 0924-2716. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0924271619302187>.

ROSENBLATT, F. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, American Psychological Association, v. 65, n. 6, p. 386, 1958.

SAH, S. Machine learning: A review of learning types. *Preprints*, Preprints, 7 2020. Disponível em: <https://doi.org/10.20944/preprints202007.0230.v1>.

SATYANARAYANAN, M. The emergence of edge computing. *Computer*, v. 50, n. 1, p. 30–39, 2017.

SHADRIN, D. et al. Designing future precision agriculture: Detection of seeds germination using artificial intelligence on a low-power embedded system. *IEEE Sensors Journal*, v. 19, n. 23, p. 11573–11582, 2019.

SHAFIEE, A. et al. Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars. In: . Seoul: [s.n.], 2016. p. 14–26.

SHAO, Y. S. et al. Simba: Scaling deep-learning inference with chiplet-based architecture. *Commun. ACM*, Association for Computing Machinery, New York, NY, USA, v. 64, n. 6, p. 107–116, 5 2021. ISSN 0001-0782. Disponível em: <https://doi.org/10.1145/3460227>.

SILVANO, C. et al. *A Survey on Deep Learning Hardware Accelerators for Heterogeneous HPC Platforms*. 2023. `https://arxiv.org/pdf/2306.15552.pdf`.

SONG, L. et al. Pipelayer: A pipelined reram-based accelerator for deep learning. In: . Austin: [s.n.], 2017. p. 541–552.

SOUZA, S. M. de. *Avaliação de Fusão de Dados Espaciais com Espectrais para Classificação de Imagens Hiperespectrais*. 99 f. Monografia (Trabalho de Conclusão de Curso) — Universidade do Vale do Itajaí, Itajaí, 2023.

STRUHARIK, R. J. et al. Conna–hardware accelerator for compressed convolutional neural networks. *Microprocessors and Microsystems*, v. 73, p. 102991, 2020. ISSN 0141-9331. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0141933119300158>.

SUPRAN, G.; RAHMSTORF, S.; ORESKES, N. Assessing exxonmobil's global warming projections. *Science*, v. 379, n. 6628, 2023. Disponível em: <https://www.science.org/doi/abs/10.1126/science.abk0063>.

SZE, V. et al. Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE*, v. 105, p. 2295–2329, 2017. Disponível em: <https://api.semanticscholar.org/CorpusID:3273340>.

THANGAVEL, K. et al. Autonomous satellite wildfire detection using hyperspectral imagery and neural networks: A case study on australian wildfire. *Remote Sensing*, v. 15, n. 3, 2023. Disponível em: <https://www.mdpi.com/2072-4292/15/3/720>.

Tom Bawden. *Global warming*: Data centres to consume three times as much energy in next decade, experts warn. 2016. `https://www.independent.co.uk/climate-change/news/global-warming-data-centres-to-consume-three-times-as-much-energy-in-next-decade-experts-warn-a6830086.html`.

TOTH, C.; JóźKóW, G. Remote sensing platforms and sensors: A survey. *ISPRS Journal of Photogrammetry and Remote Sensing*, v. 115, p. 22–36, 2016. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0924271615002270>.

VARGHESE, B. et al. Challenges and opportunities in edge computing. In: *2016 IEEE International Conference on Smart Cloud (SmartCloud)*. [S.l.: s.n.], 2016. p. 20–26.

VIEL, F. et al. Hyperspectral image classification: An analysis employing cnn, lstm, transformer, and attention mechanism. *IEEE Access*, v. 11, p. 24835–24850, 2023.

WERF, G. R. van der et al. Co2 emissions from forest loss. *Nature Geoscience*, v. 2, n. 11, p. 737–738, 11 2009. Disponível em: <https://doi.org/10.1038/ngeo671>.

XIA, M. et al. Sparknoc: An energy-efficiency fpga-based accelerator using optimized lightweight cnn for edge computing. *Journal of Systems Architecture*, v. 115, p. 101991, 2021. ISSN 1383-7621. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1383762121000138>.

YU, Y. et al. Light-opu: An fpga-based overlay processor for lightweight convolutional neural networks. In: *Proceedings of the 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. New York, NY, USA: Association for Computing Machinery, 2020. (FPGA '20), p. 122–132. ISBN 9781450370998. Disponível em: <https://doi.org/10.1145/3373087.3375311>.

ZHANG, A. et al. *Dive into Deep Learning*. [S.l.]: Cambridge University Press, 2023. `https://D2L.ai`.

ZHANG, S. et al. Cambricon-x: An accelerator for sparse neural networks. In: . Taipei: [s.n.], 2016. p. 1–12.

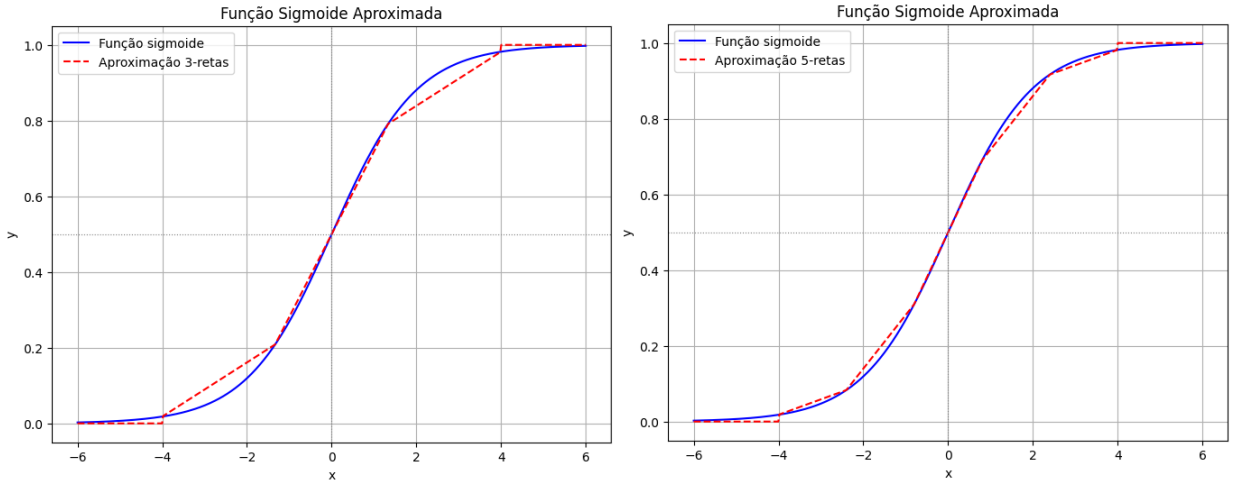
ZHANG, T. Application of ai-based real-time gesture recognition and embedded system in the design of english major teaching. *Wireless Networks*, 07 2021. ISSN 1572-8196. Disponível em: <https://doi.org/10.1007/s11276-021-02693-0>.

ZHAO, J.; YAN, H.; HUANG, L. A joint method of spatial–spectral features and bp neural network for hyperspectral image classification. *The Egyptian Journal of Remote Sensing and Space Science*, v. 26, n. 1, p. 107–115, 2023. ISSN 1110-9823. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1110982322001259>.

ZIEMANN, A. *A manifold learning approach to target detection in high-resolution hyperspectral imagery*. Tese (Doutorado) — Rochester Institute of Technology, New York, 04 2015.

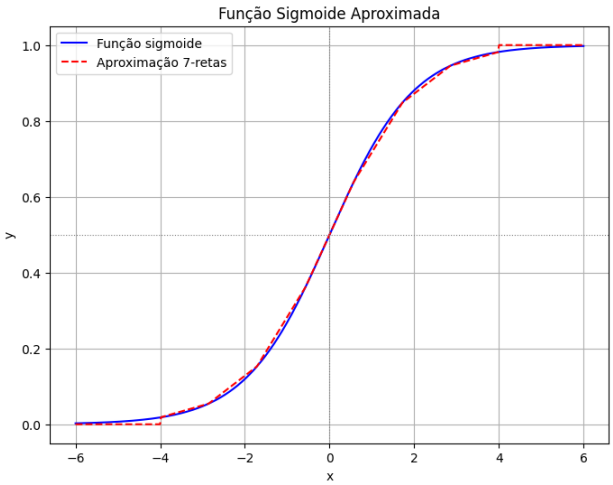
APÊNDICE A – Plot das funções aproximadas

Figura 36: Aproximação da função sigmoide por 3, 5 e 7 retas



(a) 3 segmentos de retas

(b) 5 segmentos de retas



(c) 7 segmentos de retas

Fonte: Elaborado pelo autor (2024)

APÊNDICE B – Funções sigmoide aproximadas

$$\sigma_{l3}(x) = \begin{cases} 0, \text{ para } x < 4 \\ 0,07x + 0,30, \text{ para } -4 \leq x \leq -1,33 \\ 0,22x + 0,50, \text{ para } -1,33 < x \leq 1,33 \\ 0,07x + 0,70, \text{ para } 1,33 < x \leq 4 \\ 1, \text{ para } x > 4 \end{cases} \tag{B.1}$$

$$\sigma_{l5}(x) = \begin{cases} 0, \text{ para } x < 4 \\ 0,04x + 0,18, \text{ para } -4 \leq x \leq -2,4 \\ 0,14x + 0,42, \text{ para } -2,4 < x \leq -0,8 \\ 0,24x + 0,5, \text{ para } -0,8 < x \leq 0,8 \\ 0,14x + 0,58, \text{ para } 0,8 < x \leq 2,4 \\ 0,04x + 0,82, \text{ para } 2,4 < x \leq 4 \\ 1, \text{ para } x > 4 \end{cases} \tag{B.2}$$

$$\sigma_{l7}(x) = \begin{cases} 0, \text{ para } x < 4 \\ 0,03x + 0,15, \text{ para } -4 \leq x \leq -2,86 \\ 0,09x + 0,30, \text{ para } -2,86 < x \leq -1,71 \\ 0,18x + 0,47, \text{ para } -1,71 < x \leq -0,57 \\ 0,24x + 0,50, \text{ para } -0,57 < x \leq 0,57 \\ 0,18x + 0,53, \text{ para } 0,57 < x \leq 1,71 \\ 0,09x + 0,70, \text{ para } 1,71 < x \leq 2,86 \\ 0,03x + 0,85, \text{ para } 2,86 < x \leq 4 \\ 1, \text{ para } x > 4 \end{cases} \tag{B.3}$$