UNIVERSIDADE FEDERAL DE SANTA CATARINA
CAMPUS FLORIANÓPOLIS
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Felipe Viel

Exploring Hardware Acceleration of a Transformer-Based Architecture for Hyperspectral Image Classification

Florianópolis
2025

Felipe Viel

Exploring Hardware Acceleration of a Transformer-Based Architecture for Hyperspectral Image Classification

Tese submetida ao Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Santa Catarina para a obtenção do título de Doutor em Engenharia Elétrica.
Orientador: Prof. Eduardo Augusto Bezerra, Dr.
Coorientador: Prof. Cesar Albenes Zeferino, Dr.

Florianópolis
2025

Ficha catalográfica gerada por meio de sistema automatizado gerenciado pela BU/UFSC.
Dados inseridos pelo próprio autor.

Viel, Felipe
Exploring Hardware Acceleration of a Transformer-Based Architecture for Hyperspectral Image Classification / Felipe Viel ; orientador, Eduardo Augusto Bezerra, coorientador, Cesar Albenes Zeferino, 2025.
124 p.

Tese (doutorado) - Universidade Federal de Santa Catarina, Centro Tecnológico, Programa de Pós-Graduação em Engenharia Elétrica, Florianópolis, 2025.

Inclui referências.

1. Engenharia Elétrica. 2. Aceleração em Hardware. 3. Rede Neural Transformer. 4. Imagens Hiperespectrais. 5. GPU embarcada. I. Bezerra, Eduardo Augusto. II. Zeferino, Cesar Albenes. III. Universidade Federal de Santa Catarina. Programa de Pós-Graduação em Engenharia Elétrica. IV. Título.

Felipe Viel

Exploring Hardware Acceleration of a Transformer-Based Architecture for Hyperspectral Image Classification

O presente trabalho em nível de doutorado foi avaliado e aprovado por banca examinadora composta pelos seguintes membros:

Prof. Joceli Mayer, Phd.
Universidade Federal de Santa Catarina (UFSC)

Prof. Luigi Carro, Dr.
Universidade Federal do Rio Grande do Sul (UFRGS)

Prof. Jorge Luis Victoria Barbosa, Dr.
Universidade do Vale do Rio do Sinos (Unisinos)

Prof. Marcio Eduardo Kreutz, Dr.
Universidade Federal do Rio Grande do Norte (UFRN)

Certificamos que esta é a **versão original e final** do trabalho de conclusão que foi julgado adequado para obtenção do título de Doutor em Engenharia Elétrica.

Documento assinado digitalmente
Carlos Renato Rambo
Data: 25/11/2025 10:09:53-0300
CPF: ***.748.198-**
Verifique as assinaturas em https://v.ufsc.br

Prof. Carlos Renato Rambo, Dr.
Coordenador do Programa de
Pós-Graduação

Documento assinado digitalmente
Eduardo Augusto Bezerra
Data: 24/11/2025 12:11:59-0300
CPF: ***.851.577-**
Verifique as assinaturas em https://v.ufsc.br

Prof. Eduardo Augusto Bezerra, Dr.
Orientador

Florianópolis, 2025.

ACKNOWLEDGEMENTS

Inicialmente, gostaria de agradecer aos meus pais Doralite e Pedro por me criarem, incentivar, amar e propiciar vivenciar todo esse caminho que resultou neste trabalho. Gostaria também de agradecer ao meu irmão Fabio que em todos esses anos agiu como amigo e também protetor e, juntamente aos nossos pais, influenciou e auxiliou nessa jornada. Agradeço também a minha companheira de vida e esposa Roberta por estar ao meu lado, incentivar-me e me apoiar nessa etapa de vida.

Ao meu orientador professor Eduardo Augusto Bezerra, minha gratidão e minha admiração pelo conhecimento, conselhos, direcionamentos, sabedoria e dedicação na minha orientação durante estre trabalho. Ao professor Cesar Albenes Zeferino, fica minha gratidão e admiração pelos conselhos, incentivo e dedicação para que mais essa etapa da minha jornada acadêmica e de vida se concluísse.

Aos pesquisadores do SpaceLab (Laboratório de Pesquisa em Sistemas Espaciais) da Universidade Federal de Santa Catarina, LEDS (Laboratório de Sistemas Embarcados e Distribuídos) da Universidade do Vale do Itajaí e IoTec Lab da Universidade do Vale do Itajaí, fica também minha gratidão em auxiliar na minha jornada durante o doutorado. Em especial, gostaria de agradecer os professores Douglas Rossi de Melo e Mauricio de Campos pelos conselhos e incentivos para que essa etapa da minha vida se concluísse.

Também agradeço a todos os professores da Universidade Federal de Santa Catarina e Universidade do Vale do Itajaí que de alguma forma, mesmo que momentânea, me auxiliaram e permitiram que esse doutorado se concretizasse. Também agradeço ao CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico) e a FAPESC (Fundação de Amparo à Pesquisa e Inovação do Estado de Santa Catarina) que auxiliaram financeiramente as pesquisas deste trabalho.

“A mente do homem é o instrumento básico de sua sobrevivência. Seu corpo lhe é concedido, mas não o seu sustento. Sua mente lhe é concedida, mas não o seu conteúdo. Para permanecer vivo o homem tem de agir e para isso ele tem de pensar”

Ayn Rand, A Revolta de Atlas

RESUMO

O aumento da demanda por desempenho no processamento de grandes volumes de dados tem impulsionado a adoção de sistemas-em-chip heterogêneos, que combinam diferentes tecnologias de processamento e circuitos. Esses sistemas visam atender a requisitos como alta performance, baixo consumo de energia e resposta em tempo real, os quais são aspectos fundamentais em aplicações como sensoriamento remoto e sistemas espaciais. Nesse contexto, os aceleradores em hardware se destacam como alternativa viável para substituir tarefas computacionalmente custosas executadas em software, como aquelas envolvidas em redes neurais artificiais (RNAs) aplicadas ao processamento de imagens hiperespectrais (HSIs), as quais exigem alta capacidade de processamento devido à riqueza de informações espectrais por pixel. Entre os principais desafios está a classificação de HSIs, tarefa essencial em diversos cenários de aplicação. Apesar da eficácia de arquiteturas espectro-espaciais, este trabalho investigou e demonstrou o potencial de classificadores que operam exclusivamente no domínio espectral, visando à sua viabilidade para implementação em hardware embarcado. Após uma revisão sistemática da literatura e uma análise experimental detalhada, observou-se que a arquitetura Transformer, mesmo operando apenas com dados espectrais, pode alcançar desempenho comparável às abordagens tradicionais, com menor complexidade computacional e uso de memória. Com base nesses achados, esta tese propõe uma arquitetura Transformer reduzida e otimizada, chamada Reduced Transformer, voltada à classificação espectral de HSIs e acelerada via GPU. A arquitetura demonstrou-se eficiente tanto em CPUs quanto em GPUs, com destaque para seu desempenho escalável em plataformas embarcadas de baixo consumo, como o NVIDIA Jetson Orin Nano. Os experimentos evidenciaram que o modelo obtém um bom compromisso entre acurácia, tempo de inferência, consumo energético e uso de memória, superando arquiteturas de referência como a 3D-CNN em diferentes cenários de volume de dados. Embora não tenha sido implementada como um núcleo IP para FPGA, a solução proposta mostrou-se adequada para futuras implementações em hardware dedicado, indicando sua aplicabilidade em sistemas reais de sensoriamento remoto com restrições computacionais severas. Assim, esta tese contribui com uma alternativa viável, confiável e eficiente para classificação de HSIs em contextos embarcados, reforçando o potencial do uso de Transformers em arquiteturas otimizadas para Deep Learning embarcado.

Palavras-chave: Aceleração em Hardware; Rede Neural Transformer; Imagens Hiperespectrais; Sensoriamento Remoto; GPU embarcada.

ABSTRACT

The increasing demand for high-performance processing of large volumes of data has driven the adoption of heterogeneous systems-on-chip (SoCs) that combine multiple processor and circuit technologies. These systems aim to meet requirements such as high performance, low power consumption, and real-time responsiveness, which are key to applications such as remote sensing and space systems. In this context, hardware accelerators have emerged as a viable solution for replacing computationally intensive tasks traditionally executed in software, such as those involving artificial neural networks (ANNs) applied to hyperspectral image (HSI) processing, which demand significant computational resources due to the high spectral resolution per pixel. Among the primary challenges in this domain is the classification of HSIs, a critical task across numerous application scenarios. Although spectral-spatial architectures have proven effective, this work investigates and demonstrates the potential of classifiers that operate exclusively in the spectral domain, aiming to assess their feasibility for embedded hardware implementation. Through a systematic literature review and comprehensive experimental analysis, it was observed that the Transformer architecture, even when operating solely on spectral data, can achieve performance comparable to traditional approaches while requiring lower computational complexity and memory usage. Based on these findings, this thesis proposes a reduced and optimized Transformer-based architecture, the Reduced Transformer, for spectral HSI classification and GPU acceleration. The architecture proved to be highly efficient on both CPUs and GPUs, with particular emphasis on its scalable performance on low-power embedded platforms such as the NVIDIA Jetson Orin Nano. Experimental results showed that the model offers an excellent balance between accuracy, inference time, energy consumption, and memory usage, outperforming reference architectures such as the 3D-CNN across various data volume scenarios. Although the proposed solution has not yet been implemented as an IP core for an FPGA, it demonstrated strong potential for future deployment on dedicated hardware, indicating its applicability in real-world remote sensing systems operating under strict computational constraints. In summary, this thesis contributes a viable, reliable, and efficient alternative for HSI classification in embedded contexts, reinforcing the potential of Transformer-based models in optimized architectures for embedded Deep Learning.

Keywords: Hardware Acceleration; Transformer Neural Network; Hyperspectral Image; Remote Sensing; Embedded GPU.

RESUMO EXPANDIDO

Introdução

O sensoriamento remoto por utilizando imagens hiperespectrais tem adquirido importância estratégica em aplicações científicas, agrícolas, ambientais e espaciais devido à sua elevada resolução espectral por pixel. Essa elevada quantidade de informação espectral, entretanto, impõe desafios significativos de processamento, memória e consumo de energia, especialmente em sistemas embarcados e plataformas utilizadas em aplicações espaciais, como satélites de pequeno porte. Alinhado a essa questão devido a seu uso junto a imagens hiperespectrais, principalmente para classificação, a crescente adoção de inteligência artificial e, em especial, de técnicas de aprendizado profundo, intensifica ainda mais as demandas computacionais sobre computadores de bordo e subsistemas de missão. Ao mesmo tempo, diversas plataformas e iniciativas vêm promovendo o uso de hardware especializado, como GPUs embarcadas, SoCs heterogêneos e FPGAs, para acelerar tarefas complexas e possibilitar autonomia a satélites e veículos aéreos de alta altitude ao mesmo tempo que permite atender restrições de operação imposta pelas aplicações. Entre as arquiteturas de aprendizado profundo aplicadas à classificação de imagens hiperespectrais, destacam-se CNNs, LSTMs e, mais recentemente, Transformer. Apesar do amplo uso de arquiteturas que usam dados espaciais e espectrais, modelos que operam apenas no domínio espectral têm ganhado relevância devido ao menor custo computacional e permitirem alcançar requisitos com implementações em hardware embarcado. Nesse contexto, está tese demonstra que arquiteturas baseadas em Transformer podem alcançar acurácia competitiva mesmo operando somente com informações espectrais, reduzindo significativamente a complexidade e viabilizando aceleração em dispositivos de baixo consumo energético. Dessa forma, o trabalho se insere na interseção entre classificação de imagens hiperespectrais, otimização arquitetural e aceleração em hardware, propondo, avaliando e validando uma arquitetura Transformer reduzida projetada para execução eficiente em GPUs embarcadas.

Objetivos

Diante da problemática exposta, esse trabalho busca como objetivo principal investigar arquiteturas de aprendizado profundo para classificação de imagens hiperespectrais em nível espectral e desenvolver um acelerador de hardware baseado em uma arquitetura de rede neural artificial adaptada para HSI. A solução proposta busca equilibrar precisão, demanda de recursos de hardware e confiabilidade para processamento embarcado em sistemas espaciais. Além disso, tem como objetivos específicos: (*i*) Caracterizar arquiteturas de redes neurais artificiais que fornecem classificação em nível espectral para imagens hiperespectrais; (*ii*) identificar técnicas e arquiteturas de RNAs para classificação de dados unidimensionais projetadas para aceleradores de hardware; e (*iii*) desenvolver uma arquitetura de aceleração de hardware para classificação de HSI em nível espectral baseada em RNAs.

Metodologia

A metodologia compreendeu quatro etapas principais. Inicialmente, foi realizada uma revisão sistemática da literatura sobre aceleradores de hardware para aprendizado profundo e classificação de imagem hiperespectral, abrangendo CNNs, LSTMs e Transformers, além de trabalhos recentes focados em aplicações embarcadas e plataformas espaciais. Em seguida, foram selecionados seis conjuntos de dados amplamente utilizados na área: Indian Pines, Salinas Valley, Pavia Center, University of Pavia, Kennedy Center e Botswana. Esses conjuntos de dados permitem uma comparação abrangente e padronizada entre diferentes arquiteturas de rede. Para a análise experimental, foram implementadas arquiteturas 1D-CNN, 2D-CNN, 3D-CNN, LSTM e Transformer, seguindo configurações de referência da literatura. Os experimentos incluíram: (*i*) avaliação da acurácia de classificação; (*ii*) comparação do número de parâmetros e demanda de memória; (*iii*) medição dos tempos de inferência em CPU e GPU; (*iv*) análise energética e de utilização de recursos computacionais. A partir dessas análises, iniciou-se o processo de exploração arquitetural da arquitetura Transformer, buscando identificar camadas e hiperparâmetros críticos para desempenho e custo computacional. Isso permitiu propor o modelo Reduced Transformer ou Transformer-R, uma versão compacta com redução de parâmetros, profundidade e custo de inferência. Por fim, foram conduzidos experimentos extensivos em três plataformas: um notebook com GPU dedicada, um servidor com GPU NVIDIA A100 e um NVIDIA Jetson Orin Nano de baixo consumo. Para cada plataforma e modelo, foram medidos tempo de inferência, uso de GPU, uso de memória e consumo energético, com análises para diferentes volumes de dados, 3.075 e 18.450 amostras, e diferentes tamanhos de conjuntos a serem processados paralelamente. Além disso, a arquitetura Transformer-R foi compara com uma arquitetura 3D-CNN de referência da literatura a fim de demonstrar os benefícios de uso de arquiteturas reduzidas e que usam somente informação espectral.

Resultados e Discussão

Os resultados confirmam que arquiteturas baseadas exclusivamente no domínio espectral podem alcançar acurácia comparável às abordagens espectro-espaciais, desde que adequadamente otimizadas. A arquitetura Transformer original apresentou desempenho competitivo, porém com elevado custo computacional e memória. A arquitetura Transformer-R proposta obteve significativa redução no número de parâmetros e no uso de memória, oferecendo desempenho mais estável e previsível em dispositivos embarcados. Para conjuntos menores, a inferência em CPU superou a GPU, devido ao sobrecusto de comunicação entre CPU e GPU. Entretanto, para volumes maiores de dados, a GPU demonstrou aceleração expressiva, especialmente na GPU NVIDIA Jetson Orin Nano, onde o Transformer-R apresentou tempo de inferência inferior ao de arquitetura CNN de referência, como a 3D-CNN HybridSN. Além disso, enquanto modelos como a 3D-CNN esbarraram em limitações de lote de amostras, a Transformer-R operou de forma eficiente mesmo com grandes volumes de amostras. As análises de energia e potência demonstraram que o Transformer-R manteve baixas demandas de memória e consumo de energia significativamente inferior ao de arquiteturas comparadas. Na GPU embarcada, o modelo atingiu um bom compromisso entre acurácia, tempo de inferência, paralelismo e eficiência energética. A arquitetura mostrou também maior escalabilidade, tornando-se adequada para operações embarcadas em satélites

e drones com restrições de recursos computacionais e de energia.

Considerações Finais

O estudo demonstrou que classificadores baseados em Transformer, operando apenas no domínio espectral, são eficientes, competitivos e adequados a cenários embarcados com restrições de recursos, como memória e energia. O modelo Transformer-R apresentou acurácia comparável a arquiteturas que usam informação espacial e espectral e são muito mais custosas quanto a recursos computacionais, ao mesmo tempo que reduziu significativamente a complexidade computacional e os requisitos de hardware. A validação em plataformas reais, incluindo o NVIDIA Jetson Orin Nano, evidencia a aplicação prática do modelo para missões espaciais e sistemas autônomos baseados em HSI. Embora a implementação em FPGA ainda não tenha sido realizada, os resultados indicam forte potencial para síntese futura, especialmente devido à natureza modular e compacta da arquitetura reduzida. Assim, o trabalho contribui para o avanço da classificação hiperespectral em dispositivos embarcados, reforçando o papel de Transformers otimizados como alternativa promissora para Deep Learning em aplicações remotas, críticas e energeticamente restritas.

Palavras-chave: Aceleração em Hardware; Rede Neural Transformer; Imagens Hiperespectrais; Sensoriamento Remoto; GPU embarcada.

LIST OF FIGURES

Figure 1 – Exemplification of the electromagnetic spectrum and sampling done for Red-Green-Blue (RGB) images. 24
Figure 2 – Exemplification of the electromagnetic spectrum and sampling done for Hyperspectral Image (HSI). 25
Figure 3 – Exemplification of the electromagnetic spectrum and sampling done for HSI. 25
Figure 4 – Example of the Different Types of Representation Spaces for an HSI. 27
Figure 5 – Spatial resolution comparison between (a) HyMap (2.8 m) and (b) Hyperion (30 m). 28
Figure 6 – Division of ML into categories (a) Classification and (b) Regression. 29
Figure 7 – Representation of an Artificial Neural Network (ANN) composed of a structure with an input layer, only a hidden layer, and an output layer with Rectified Linear Unit (ReLU) type activation function. 32
Figure 8 – 1D, 2D and 3D convolutional layers added to ANNs, forming the CNN. 34
Figure 9 – Standard Recurrent Neural Network (RNN) and original Long Short-Term Memory (LSTM) cell architectures. 36
Figure 10 – Transformer Neural Network Architecture. 38
Figure 11 – Multi-head Attention layer architecture. 39
Figure 12 – Organization of the ARM Mali-G761 Embedded GPU. 42
Figure 13 – Microarchitecture of the NVIDIA Ada Lovelace High-Performance GPU. 42
Figure 14 – Organization of the CLB present in Versal devices from the manufacturer Xilinx. 43
Figure 15 – Development Flow for FPGA Using Vitis HLS and Vivado Design Suite Tools from Xilinx. 44
Figure 16 – MAX78000 Systems-on-Chip (SoC) with a hardware accelerator for executing CNN models. 46
Figure 17 – Overview of the AI Engine Organization in the Xilinx Versal Device. . 46
Figure 18 – Proposed dataset-splitting approach to avoid contamination between training and testing datasets. 59
Figure 19 – 1D-CNN architecture based in Roy et al. (2020). We named this architecture 1D-CNN-O (-O stands for original architecture). 62
Figure 20 – 2D-CNN architecture based on Roy et al. (2020). 66
Figure 21 – HybridSN proposed by Roy et al. (2020). 66
Figure 22 – LSTM Architecture with Attention Layer. 68
Figure 23 – Transformer Architecture proposed in Ntakouris (2021). 69
Figure 24 – Comparison between Transformer default (a) and Transformer reduced (b). 80

Figure 25 – Segmentation Map Types. 89
Figure 26 – Overview of the architecture implemented for segmentation evaluation 90
Figure 27 – KC dataset (a) and segmentations by (b) color map and by (c) boundaries. 92
Figure 28 – Inference time performance comparison on GPU using the Default and Reduced models. The graphics demonstrate the models behavior to (a) 3,075 samples and (b) 18,450 samples. 94
Figure 29 – Inference time performance comparison on GPU of the Default, Reduced, and 3D-CNN HybridSN models running on the Test Notebook whith (a) 3,075 samples and (b) 18,450 samples. 94
Figure 30 – Inference time performance comparison on GPU of the Default, Reduced, and 3D-CNN HybridSN models running on the NVIDIA Jetson Orin Nano whit (a) 3,075 samples and (b) 18,450 samples. 95
Figure 31 – Comparison of GPU power consumption during inference on the Test Notebook with (a) 3,075 samples and (b) 18,450 samples. 102
Figure 32 – Comparison of GPU power consumption during inference on the NVIDIA Jetson Orin Nano with (a) 3,075 samples and (b) 18,450 samples. 102
Figure 33 – Comparison of GPU power consumption during inference in the power-saving versions of the platforms with (a) 3,075 and (b) 18,450 samples.103
Figure 34 – Comparison of GPU power consumption during inference in the max performance versions of the platforms with (a) 3,075 and (b) 18,450 samples. 103

LIST OF TABLES

Table 1 – Summary of Convolutional Neural Network (CNN)-based architectures for HSI classification. 50
Table 2 – Summary of LSTM-based architectures works for classification. . . . 52
Table 3 – Summary of Transformer-based architectures works for classification. 53
Table 4 – Summary of architectures works for classification in compare with this work. 55
Table 5 – 1D-CNN for the IP and KC scenes. 62
Table 6 – 1D-CNN for the SA, PU, CP, and BT. The values of the last layer are 2064 (16 classes) for the SA, 1161 (9 classes) for UP and CP, 1677 (13 classes) for the KC, and 1806 (14 classes) for the BT. Dense 6-layer is 524,544 for the KC scene and 33,024 for others. 63
Table 7 – Reduced 1D-CNN without Attention layer. 63
Table 8 – Reduced 1D-CNN with Attention layer. 64
Table 9 – 2D-CNN for the IP scene. 65
Table 10 – 2D-CNN for the SA, PU, CP, and BT scenes. The values of the last layer are 2064 (16 classes) for the SA scene, 1161 (9 classes) for the UP and CP scenes, 1677 (13 classes) for the KC scene, and 1806 (14 classes) for the BT scene. 65
Table 11 – 3D-CNN for the IP scene. 67
Table 12 – 3D-CNN for the SA, PU, CP, and BT scenes. The values of the last layer are 2064 (16 classes) for the SA scene, 1161 (9 classes) for the UP and CP scenes, 1677 (13 classes) for the KC scene, and 1806 (14 classes) for the BT scene. 67
Table 13 – LSTM without Attention layer. 68
Table 14 – LSTM with Attention layer. 69
Table 15 – Transformer layers. Layers 2 to 9 are repeated 4 times. 70
Table 16 – Classification performance of the three CNN architectures using the new dataset-splitting methodology. 70
Table 17 – Accuracy of the two dataset-splitting methodologies for 1D-CNN. . . . 71
Table 18 – Performance of the 1D architectures. 72
Table 19 – Number of parameters and memory required by the three CNN architectures. 73
Table 20 – Number of parameters and consumed memory of the 1D classification architectures. 74
Table 21 – Processing time for the three CNN architectures running on different infrastructures and using the new dataset-splitting methodology. . . . 75

Table 22 – Processing time for the 1D architectures running on the CPU+NVIDIA A100 GPU infrastructure. 76
Table 23 – Transformer architecture components optimized for embedded implementation 78
Table 24 – Architecture, layers, output shape, and parameters from Transformer-based models. 81
Table 25 – Performance of the Transformer-based architectures. 85
Table 26 – Number of parameters and consumed memory of the 1D Transformer-based architectures. 86
Table 27 – Parameters defined for each dataset. 89
Table 28 – Performance comparison of Transformer-R vs. Transformer-R+SEEDS architecture 91
Table 29 – Inference time for 3,075 samples on GPU across different models and platforms 93
Table 30 – Inference time for 18,450 samples on GPU across different models and platforms. 93
Table 31 – Inference time for 3,075 samples on CPU across different models and platforms 95
Table 32 – Inference time for 18,450 samples on CPU across different models and platforms 95
Table 33 – Comparasion and Acceleration GPU and CPU - 3,075 samples . . . 96
Table 34 – Comparation and Acceleration GPU and CPU - 18,450 samples . . . 96
Table 35 – Time comparasion to a one sample processing 97
Table 36 – GPU Utilization in % with 3,075 samples 99
Table 37 – GPU Utilization in % with 18,450 samples 99
Table 38 – GPU Memory Utilization - 3,075 samples 100
Table 39 – GPU Memory Utilization - 18,450 samples 100
Table 40 – GPU and memory Utilization to lowest inference time 101
Table 41 – GPU power (in Watts) comparison between platforms to 3,075 samples.101
Table 42 – GPU power (in Watts) comparison between platforms to 18,450 samples.101
Table 43 – Electrical Power and Current to the lowest inference time. 103
Table 44 – NVIDIA Jetson input electrical power and current comparison to the lowest inference time. 104
Table 45 – Comparison between platforms to lowest inference time to 3,075 samples. 105
Table 46 – Comparison between platforms to lowest inference time to 18,450 samples. 106

LIST OF ABBREVIATIONS AND ACRONYMS

AA	Average Accuracy
ACM	Association for Computing Machinery
Adam	Adaptive Moment Estimation
AI	Artificial Intelligence
AIS	Airborne Imaging Spectrometer
ANN	Artificial Neural Network
API	Application Programming Interface
ASIC	Application-Specific Integrated Circuit
ASIP	Application Specific Instruction-set Processor
AVIRIS	Airborne Visible-Infrared Imaging Spectrometer
BRAM	Block Random Access Memory
BT	Botswana
CLB	Configurable Logic Block
CNN	Convolutional Neural Network
CP	Center of Pavia
CPU	Central Processing Unit
dB	decibel
DL	Deep Learning
DSPs	Digital Signal Processors
EM	Electromagnetic
EO-1	Earth Observing One
FPGA	Field-Programmable Gate Array
FWHM	Full Width Half Max
GD	Gradiente Descendente
GIFOV	Ground-projected Instantaneous Field of View
GPP	General Purpose Processor
GPU	Graphic Processing Unit
GRU	Gated Recurrent Unit
GSD	Ground Sample Distance
HA	Hardware Accelerator
HLS	High-Level Synthesis
HSI	Hyperspectral Image
IEEE	Institute of Electrical and Electronics Engineers
IFOV	Instantaneous Field of View
IOB	Input/Output Block
IP	Indian Pines
JPL	Jet Propulsion Laboratory

KC	Kennedy Center
LSTM	Long Short-Term Memory
LUT	Look-Up Table
MDPI	Multidisciplinary Digital Publishing Institute
ML	Machine Learning
NASA	National Aeronautics and Space Administration
OA	Acurácia Geral
OBC	On-Board Computer
PC	Principal Component
PCA	Principal Component Analysis
PPGEEL	Programa de Pós-Graduação em Engenharia Elétrica
ReLU	Rectified Linear Unit
RGB	Red-Green-Blue
RNA	Redes Neurais Artificiais
RNN	Recurrent Neural Network
ROSIS	Reflective Optics System Imaging Spectrometer
SA	Salinas Valley
SGD	Stochastic GD
SIMD	Single Instruction, Multiple Data
SNR	Signal-to-Noise Ratio
SoC	Systems-on-Chip
SPP	Specific Purpose Processor
SWIR	Short Wave InfraRed
TIR	Thermal InfraRed
UAV	Unmanned Aerial Vehicle
UFSC	Universidade Federal de Santa Catarina
UP	University of Pavia
VHDL	VHSIC Hardware Description Language
ViT	Vision Transformer
VNIR	Visible and Near-InfraRed
VPU	Vision Processing Unit

CONTENTS

1	**INTRODUCTION**	**19**
1.1	MOTIVATION	20
1.2	RESEARCH QUESTIONS	21
1.3	OBJECTIVES	22
1.3.1	**General Objective**	**22**
1.3.2	**Specific Objectives**	**22**
1.4	SCIENTIFIC CONTRIBUTION	22
1.5	DOCUMENT STRUCTURE	23
2	**BACKGROUND**	**24**
2.1	HYPERSPECTRAL IMAGING	24
2.1.1	**Definition of Hyperspectral Image**	**24**
2.1.2	**Characteristics of Hyperspectral Images**	**26**
2.1.3	**Features of HSI Systems**	**27**
2.2	ARTIFICIAL NEURAL NETWORKS	28
2.2.1	**Introduction to Machine Learning**	**28**
2.2.2	**Artificial neural networks**	**32**
2.2.2.1	Convolutional Neural Networks	34
2.2.2.2	Recurrent Neural Networks	36
2.2.2.3	Transformer Neural Network	37
2.3	HARDWARE ACCELERATORS	40
2.3.1	**Processor Technology**	**40**
2.3.2	**Hardware Accelerators**	**40**
2.4	CONSIDERATIONS	47
3	**REVIEW OF THE STATE OF THE ART OF DEEP LEARNING ARCHITECTURES HARDWARE ACCELERATION FOR HSI CLASSIFICATION**	**48**
3.1	CNN-BASED HARDWARE ACCELERATORS	49
3.2	LSTM	51
3.3	TRANSFORMER	52
3.4	DISCUSSION	53
4	**PROPOSAL OF AN ANN-BASED ARCHITECTURE FOR HARDWARE-ACCELERATED HSI CLASSIFICATION IN EMBEDDED DEVICES**	**56**
4.1	HSI CLASSIFICATION ANALYSIS: 1D, 2D, AND 3D	56
4.1.1	**Dataset**	**56**
4.1.2	**Materials for comparison between architectures**	**57**
4.1.3	**Methods for comparing architectures**	**58**
4.2	CNN ARCHITECTURE	60

4.2.1 1D-CNN . . . 61
4.2.2 2D-CNN . . . 64
4.2.3 3D-CNN . . . 66
4.3 LSTM ARCHITECTURE . . . 66
4.4 TRANSFORMER ARCHITECTURE . . . 67
4.5 EXPERIMENT RESULTS . . . 69
4.5.1 Classification Performance . . . 69
4.5.2 Memory Demand . . . 71
4.5.3 Time for Inference . . . 72
4.5.4 Discussion of the Results of Analysis of the Architectures . . . 73
4.6 ANALYSIS AND EXPLORATION OF TRANSFORMER-BASED ARCHITECTURE DESIGN . . . 77
4.6.1 Architecture Exploration . . . 77
4.6.2 Motivations for Reducing Transformer Architecture . . . 79
4.7 MATERIALS AND METHODS FOR ANALYSIS OF THE REDUCED MODEL IN CLASSIFICATION AND INFERENCE USING GPU HARDWARE ACCELERATION . . . 82
4.7.1 Materials . . . 82
4.7.2 Methods . . . 83
5 RESULTS . . . 85
5.1 CLASSIFICATION PERFORMANCE . . . 85
5.1.1 Memory Utilization . . . 86
5.1.2 Analysis of the Results of the Original Architecture with Variation in the Number of Bands . . . 87
5.2 DATA FUSION ANALYSIS FOR HSI CLASSIFICATION WITH NEURAL NETWORKS BASED ON 1D DATA . . . 88
5.2.1 Segmentation Algorithm . . . 88
5.2.2 Spectral-Spacial Fusion . . . 89
5.2.3 1D Transformer Performance with SEEDS . . . 90
5.3 EXPLORATORY AND COMPARATIVE ANALYSIS OF THE ARCHITECTURES ON CPU AND GPU WITH DIFFERENT SAMPLE VOLUMES . . . 92
5.3.1 Time Analysis . . . 92
5.3.2 GPU and Memory utilization . . . 98
5.3.3 Power and Current Comparasion . . . 100
5.3.4 General comparison . . . 104
6 CONCLUSIONS . . . 107
References . . . 110

1 INTRODUCTION

The increasing performance gains in processing large volumes of data are closely associated with the growing integration of cores into a single chip, or System-on-Chip SoC. However, the enhanced integration of processing cores and the heightened complexity of SoCs do not entirely address the current processing demands. As a result, according to Geist et al. (2019), there is a pursuit of heterogeneous SoCs that amalgamate the strengths of diverse processor and circuit technologies, with an emphasis on achieving high performance and processing large volumes of data while meeting real-time requirements, which is something unattainable through conventional Central Processing Unit (CPU). As a solution, the development and integration of dedicated processing units in SoC enable meeting these requirements (Ashenden, 2007; Cardoso; Coutinho; Diniz, 2017). The incorporation of dedicated processing cores, known as Hardware Accelerator (HA), specialized processors, or coprocessors, facilitates the fulfillment of computationally demanding processing tasks and computations, as outlined in previous literature (Cardoso; Coutinho; Diniz, 2017; Ashenden, 2007; Geist et al., 2019).

Hardware Accelerators find utility in applications that require substantial processing power, low power consumption, real-time responsiveness, and the ability to replace computationally intensive software tasks (Ashenden, 2007; Cardoso; Coutinho; Diniz, 2017). Among the domains imposing these requirements, hyperspectral image (HSI) processing deserves mention, as it requires handling large volumes of data. Consequently, there has been a research focus on leveraging accelerators to fulfill the processing demands inherent to hyperspectral imaging (Liu et al., 2019; Pereira et al., 2019; Rodriguez et al., 2019; Viel et al., 2020; Martins et al., 2024). As a targeted application area for accelerator deployment, aligned with HSI utilization, remote sensing of the Earth through satellites and high-altitude aircraft stands out.

Numerous investigations in recent years have been conducted in relation to processing large volumes of data from space applications, particularly those involving on-board processing (Davidson; Bridges, 2018; Pérez et al., 2020). These investigations demonstrate the demand and requirements for the use of more complex On-Board Computer (OBC), as well as processing capacity in payload subsystems on spacecraft, which have been undergoing continuous evolution in various aspects of their architecture and organization (Davidson; Bridges, 2018; Melo et al., 2019; Viel et al., 2020; Martins et al., 2024). In conjunction with this issue and the escalating demand for onboard decision-making in space vehicles, the utilization of hardware accelerators is under investigation for the automation and implementation of Artificial Intelligence (AI) within space systems (Kothari; Liberis; Lane, 2020; Furano et al., 2020; Sabogal; George, 2021).

Recent works presented in the literature have reported the implementation of AI techniques in OBC and subsystems of space systems. Applications employing AI strive to achieve autonomy and decision-making capabilities in these systems, with AI being harnessed at various processing stages (Tipaldi et al., 2020; Sabogal; George, 2021; Thangavel et al., 2024). In this context, machine learning algorithms, especially those based on Deep Learning (DL), have gained prominence, featuring various neural network architectures (Sisodiya; Dube; Thakkar, 2020; Kothari; Liberis; Lane, 2020; Furano et al., 2020; Sabogal; George, 2021; Spiller et al., 2022b; Mateo-Garcia et al., 2023). Furthermore, the application of artificial intelligence has begun to excel and is now a subject of investigation and validation, as evidenced by platforms such as the PhiSat-1 satellite (ESA, 2020).

The application of Machine Learning (ML) in OBC and subsystems aligns with the use of HAs due to the increased computational demands posed by ML algorithms and the growing volume of data captured and processed in space systems, such as the substantial amount of data inherent in hyperspectral imaging (Furano et al., 2020; Kothari; Liberis; Lane, 2020; Sabogal; George, 2021; Tzanos; Kachris; Soudris, 2019). These two concerns converge in the integration of ML and hardware accelerators. By combining the trends of onboard ML utilization alongside the increased adoption of hardware accelerators, an ideal scenario emerges for applying onboard HSI classification in satellites (Caba et al., 2020; Furano et al., 2020; Giuffrida et al., 2020; González et al., 2013; Kothari; Liberis; Lane, 2020; Ciardi et al., 2023). Architectures that stand out for HSI classification include ANN,

1.1 MOTIVATION

The increasing demand for computational systems capable of delivering high processing performance to handle large volumes of data has become a critical focus in space exploration (Geist et al., 2019; Kothari; Liberis; Lane, 2020; Furano et al., 2020; Sabogal; George, 2021). Several platforms aim to develop and validate these technologies, including SpaceCube v3.0 (Geist et al., 2019), Phisat-1 (Giuffrida et al., 2021b), OPS-SAT (Bahl et al., 2019), ScOSA (Schwenk; Ulmer; Peng, 2018), and GPU@SAT (Ciardi et al., 2023). Moreover, various space applications, such as Earth remote sensing, have adopted these solutions, which encompass image processing, artificial intelligence, and fault tolerance (Kothari; Liberis; Lane, 2020; Furano et al., 2020; Sabogal; George, 2021; Sun et al., 2022c).

In the context of these applications, design requirements and constraints, such as performance, power consumption, real-time operation, and fault tolerance that have guided and will continue to guide the development of new processor architectures that form the next OBCs and subsystems of space platforms, such as satellites (Kothari; Liberis; Lane, 2020; Furano et al., 2020; Kosmidis et al., 2020; Sabogal; George, 2021;

Spiller et al., 2022b; Ciardi et al., 2023; Růžička; Markham, 2024). Various solutions for space systems have been proposed to enhance performance through processor technologies (Sabogal; George; Crum, 2019; Lentaris et al., 2019; Ciardi et al., 2023) and circuit technologies (Yan et al., 2019; Giuffrida et al., 2021b; Ciardi et al., 2023). Furthermore, there is significant convergence in the use of DL techniques applied to remote sensing with HSI (Audebert; Le Saux; Lefevre, 2019; Kothari; Liberis; Lane, 2020; Furano et al., 2020; Spiller et al., 2022b; Ciardi et al., 2023; Růžička; Markham, 2024).

In the application of DL to HSIs, Convolutional Neural Networks, Recurrent Neural Networks, and Transformer architectures stand out. These architectures for HSI classification leverage spectral, spatial, and spectral-spatial data (Wang et al., 2022b; Ahmad et al., 2022; Růžička; Markham, 2024). Unlike other types of images, HSIs contain a significant amount of information in a single pixel due to their high spectral resolution. This characteristic enables per-pixel scene classification (at the spectral level) without relying on spatial structures (at the spatial level). However, factors such as memory requirements, processing time, hardware resource usage, and architectural complexity make spectral-level classification more suitable for hardware implementation than spectral-spatial classification. A counterpoint to spectral-level HSI classification, often addressed with a focus on spectral-spatial classification, is accuracy (Li et al., 2019; Audebert; Le Saux; Lefevre, 2019; Alcolea et al., 2020b; Yin et al., 2021b; Wang et al., 2022b; Ahmad et al., 2022; Růžička; Markham, 2024). In this context, this proposal is positioned and motivated by the goal of enabling the application of DL onboard remote sensing platforms utilizing HSI.

This project is driven by the increasing demand for autonomy, on-board computing, and reliability in space systems through the application of DL and HSI, which are of significant interest to the scientific community, as previously discussed and demonstrated. From a technical perspective, efficient and reliable approaches to designing and developing hardware accelerators tailored for onboard operations are continually under research and development.

The project also aligns with ongoing research and initiatives within the Information Processing Concentration Area and the Embedded Systems Research Line of the Programa de Pós-Graduação em Engenharia Elétrica (PPGEEL) at Universidade Federal de Santa Catarina (UFSC), particularly the work carried out by Spacelab. It is closely connected to current research projects at UFSC, including the Payload-XL and Constelação Catarina initiatives.

1.2 RESEARCH QUESTIONS

The onboard classification of HSIs using HAs based on ML algorithms addresses critical requirements such as energy efficiency and processing time. However, achieving

parallelism, real-time operation, low power consumption, and fault mitigation remains a challenge for computational systems used in space environments. These challenges stem from the harsh conditions to which the HA will be exposed and the demanding computational requirements it must fulfill.

In this context, this project aims to address the following main research questions:

- Can the classification of HSIs using ANNs at the spectral level alone achieve accuracy in inference performance?
- Which ANN architectures are efficient in accuracy for classifying HSIs solely at the spectral level while achieving performance comparable to spectral-spatial classification?
- Do hardware accelerators for HSI classification at the spectral level using ANN architectures strike an effective balance between required hardware resource utilization and classification performance?

1.3 OBJECTIVES

The general objective and the specific objectives of this work are described below.

1.3.1 General Objective

This work aims to investigate deep learning architectures for spectral-level HSI classification and develop a hardware accelerator based on an ANN architecture tailored for HSI. The proposed solution seeks to balance accuracy, hardware resource demands, and reliability for onboard processing in space systems.

1.3.2 Specific Objectives

1. Characterize ANN architectures that provide spectral-level classification for HSI.
2. Identify techniques and ANN architectures for one-dimensional data classification designed for hardware accelerators.
3. Develop a hardware acceleration for spectral-level HSI classification architecture based on ANN.

1.4 SCIENTIFIC CONTRIBUTION

This document presents evidence that the use of ANNs for spectral-level HSI classification can achieve accuracy comparable to that of spectral-spatial architectures,

while requiring significantly fewer parameters and enabling faster inference.

Furthermore, this thesis presents a new architecture for HSI classification that relies solely on spectral information and a Transformer encoder. Although this architecture has not been implemented as an IP core for Field-Programmable Gate Array (FPGA)s, its performance has been extensively evaluated using acceleration-based Graphic Processing Unit (GPU). The proposed model demonstrated high efficiency on embedded GPU, particularly in scenarios with large data volumes, without experiencing batch size constraints, unlike other architectures, which frequently encounter this issue.

Empirical experiments indicate that CPU-based inference achieves superior performance compared to GPU-based inference for smaller datasets. This result underscores the adaptability of CPU-based methods in environments with limited computational resources. The findings suggest that simple architectures can meet low-power requirements, rendering GPU hardware acceleration unnecessary for small-scale data processing. GPU acceleration provides benefits only when data volumes are large enough to compensate for the increased communication overhead between the processor and acceleration cores.

These findings underscore the scientific and technical contributions of this thesis. An exhaustive analysis of works indexed in bibliographic databases further supports this conclusion. This analysis emphasizes the relevance of HAs for HSI and the growing adoption of Transformer-based architectures for classifying this type of imagery.

Although implementing the Reduced Transformer as a hardware IP core for FPGA is promising, this thesis focuses on its feasibility and effectiveness in GPU-based embedded systems. Using the NVIDIA Jetson platform, a low-power embedded device, the study validates the model's performance under computational and energy constraints. This ensures practical evaluation of model scalability and real-world applicability in embedded AI scenarios.

1.5 DOCUMENT STRUCTURE

The remainder of this document is structured into five chapters. Chapter 2 introduces fundamental concepts related to hyperspectral imaging, artificial neural networks, and hardware accelerators. Chapter 3 provides a review of related works on hardware accelerators based on DL for data processing and HSI. Chapter 5 presents the analysis of data obtained from evaluations and tests, highlighting the contributions of the thesis. Finally, Chapter 6 presents the conclusions of the work, answering the research questions that guided this thesis, highlighting its contributions, and indicating future work.

2 BACKGROUND

2.1 HYPERSPECTRAL IMAGING

In space exploration during the 1970s, National Aeronautics and Space Administration (NASA) initiated a project to capture multiple electromagnetic spectrum bands and facilitate image creation. This initiative led to the inception of the Airborne Imaging Spectrometer (AIS) by the Jet Propulsion Laboratory (JPL) in 1979. AIS could generate images utilizing as many as 128 bands spanning from the 1.2 μm to 2.4 μm range of the electromagnetic spectrum. Building upon the success of AIS, in 1986, JPL embarked on the development of the Airborne Visible-Infrared Imaging Spectrometer (AVIRIS) project, which further expanded the spectral acquisition capacity ranging from 380nm to 2510 nm (Green, 2022; Feldman; Taranik; Mouat, 1985). Consequently, a novel approach to remote sensing was ushered in, giving rise to what we now refer to as hyperspectral imaging (or imagery).

2.1.1 Definition of Hyperspectral Image

Hyperspectral images (HSIs) are images characterized by containing multiple bands across different wavelengths of the electromagnetic spectrum. A defining feature of HSIs is their fine spectral resolution. This fine spectral resolution means that the electromagnetic wavelengths captured for a sensed scene typically span 5 nm to 10 nm. Consequently, the intervals between each band in an HSI display minute differences in the order of a few nanometers. In contrast, for reference, an RGB image comprises three bands with differences spanning tens or hundreds of nanometers (Lee et al., 2015; Wang; Zhao, 2016). Figures 1 and 2 provide a visual comparison illustrating this distinction. It is evident that hyperspectral sensors, particularly those based on the AVIRIS technology, capture images across both the visible light and infrared spectra.

Figure 1 – Exemplification of the electromagnetic spectrum and sampling done for RGB images.

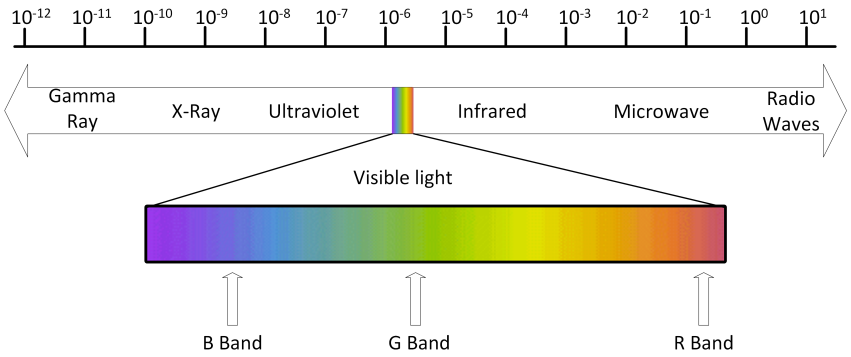
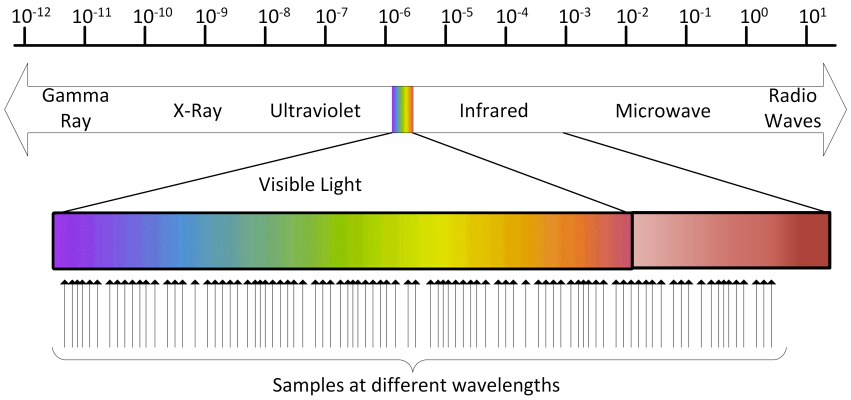


Figure 2 – Exemplification of the electromagnetic spectrum and sampling done for HSI.



The amalgamation of these images captured at varying wavelengths culminates in a hypercube, a term used to visually depict hyperspectral imaging. Figure 3 illustrates the hyperspectral hypercube and the concept of spectral signature pertaining to a pixel within the scene represented by the hyperspectral imaging. The notion of a spectral signature stems from the premise that numerous values delineate the scene at the designated pixel for each wavelength (band) captured by the hyperspectral sensor. This representation facilitates the seamless coverage of the electromagnetic spectrum (Wang; Zhao, 2016).

Figure 3 – Exemplification of the electromagnetic spectrum and sampling done for HSI.

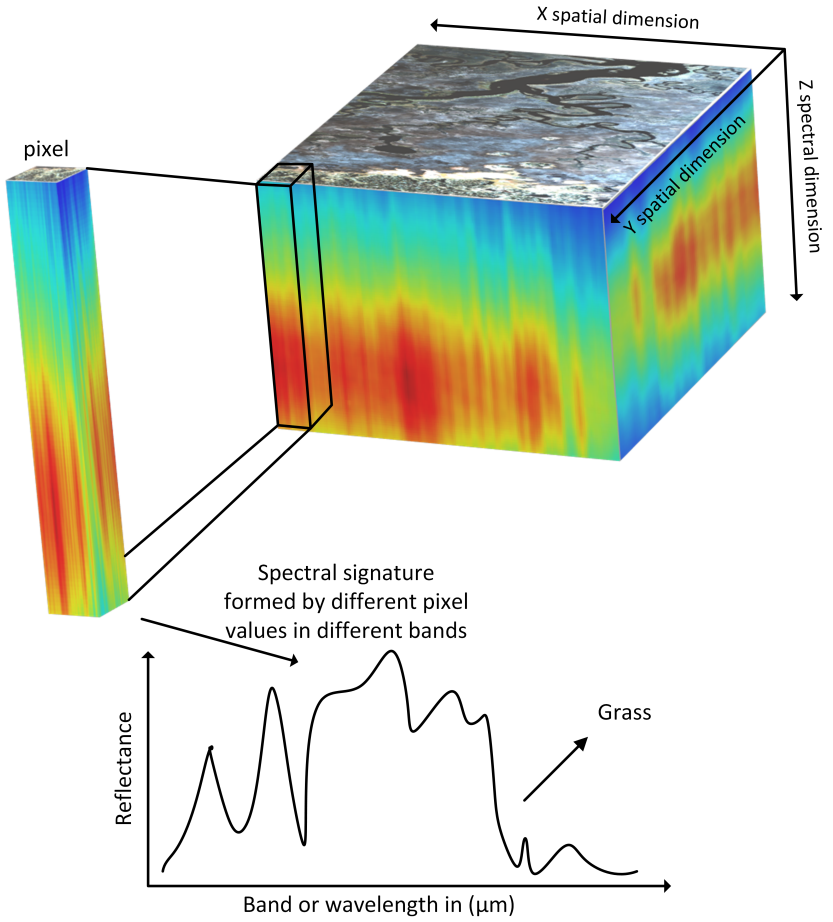


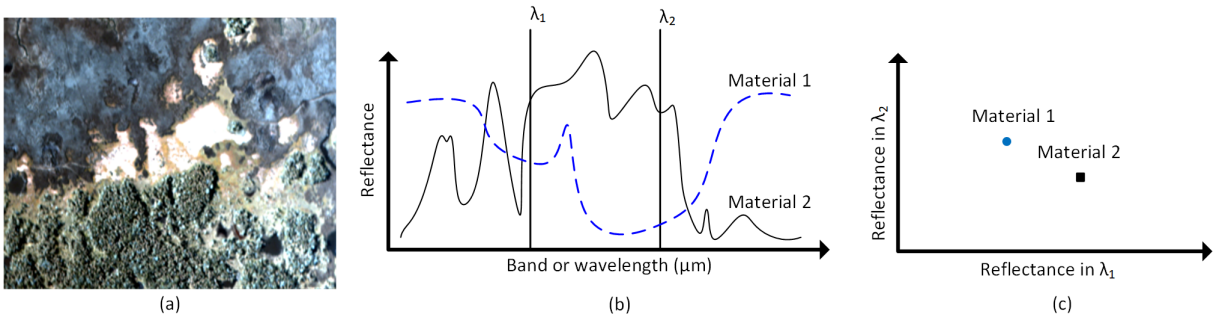
Figure 3 illustrates various representation approaches and the discernible aspects within a scene using HSIs. As depicted, it is evident that each pixel autonomously contains sufficient scene information at its designated location, obviating the need for neighboring pixels to contribute data. However, data within a single pixel delineate only the reflectance of individual scene points, thereby precluding the identification of spatial information. The amalgamation of band values within a pixel facilitates the generation of a spectral signature, visually manifested as a graph depicting fluctuating reflectance signals $\times$ wavelength. This spectral signature (graph depicted in Figure 3) exhibits diverse behaviors across bands due to the inherent nature of the analyzed material or compound. Leveraging this signature, precise analysis and classification of the content represented by the pixel surpass the capabilities of alternative imaging employed in remote sensing. Nonetheless, owing to the sensor's construction, which emphasizes the acquisition of multiple bands, the spatial information encapsulated within a pixel can transcend a mere compound/material class (Lee et al., 2015; Wang; Zhao, 2016).

2.1.2 Characteristics of Hyperspectral Images

The representation of HSIs, as stated by Wang and Zhao (2016), can be categorized into three forms based on their characteristics:

- Image Space: This two-dimensional image representation emphasizes the spatial information within the image. It enables the recognition of spatial structures and features. Figure 3 illustrates the image space through the spatial dimensions X and Y, while Figure 4 represents this space in image (a).
- Spectral Space: This representation of HSI is based on the spectral response. The spectral response is a function of energy (or reflectance) as a function of the wavelength generated from the EM spectrum. Figure 3 illustrates the spectral space through the variance graph and the spectral dimension Z, while Figure 4 represents it in image (b).
- Feature Space or Attribute Space: This representation conveys information as an *n*-dimensional vector, where each dimension (or *n*) corresponds to a band of the captured Electromagnetic (EM) spectrum. Thus, each vector contains *n* dimensions of the pixel, with each dimension representing spectral information. Figure 4 illustrates the feature space in image (c).

Figure 4 – Example of the Different Types of Representation Spaces for an HSI.



Source: Adapted from Wang and Zhao (2016).

2.1.3 Features of HSI Systems

Spectral regions and resolutions

Hyperspectral imaging (HSI) capture systems typically span the electromagnetic spectrum most relevant to the intended application. Remote sensing systems predominantly utilize bands ranging from 400 nm to 2500 nm. Some sensors operate exclusively in the Visible and Near-InfraRed (VNIR) and Short Wave InfraRed (SWIR) regions. More modern sensors may also encompass the Thermal InfraRed (TIR) region. The VNIR region typically operates within the range of 400 nm to 1050 nm, the SWIR region covers 820 nm to 2500 nm, and the TIR region complements the others, featuring a reduced number of bands with variations of either 400 nm or 700 nm to 1300 nm (Lee et al., 2015; Varshney; Arora, et al., 2004).

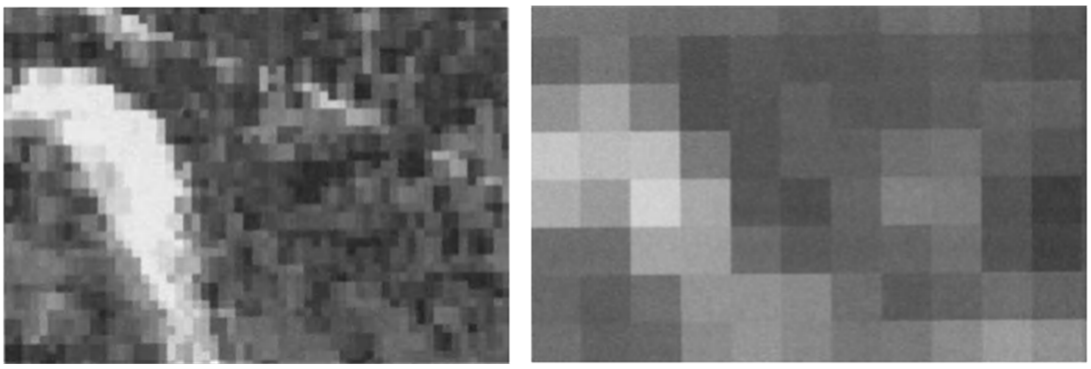
Another vital parameter for HSIs is the Full Width Half Max (FWHM). This parameter pertains to the spectral resolution of an HSI concerning the bandwidth used to measure the radiation (or reflectance) within a band. Typically, the FWHM hovers around 10 nm, either slightly more or less, and is determined by the specific target application. Additionally, spectral sampling refers to the interval between continuously captured samples (bands). The spacing between consecutive samples generally ranges from 2 nm to 3 nm (Varshney; Arora, et al., 2004).

Spatial Resolution and Coverage

The spatial resolution of an HSI pertains to its capability to detect and observe objects within a scene. The spatial representation of a scene is a composite of multiple pixels, with each pixel representing a distinct area. This representation is expressed in meters and is fundamentally influenced by the Instantaneous Field of View (IFOV) as well as the Ground-projected Instantaneous Field of View (GIFOV) or Ground Sample Distance (GSD) (Schowengerdt, 2006; Varshney; Arora, et al., 2004). Owing to sensor altitude and design, the spatial resolution of HSIs can vary from a few meters to tens of

meters. Examples include the HyMap imaging system with a resolution of 2.8 meters and the AVIRIS system with a resolution of 30 meters (Varshney; Arora, et al., 2004). Figure 5 illustrates the disparity in spatial resolutions between the HyMap and Hyperion systems.

Figure 5 – Spatial resolution comparison between (a) HyMap (2.8 m) and (b) Hyperion (30 m).



Source: Adapted from Varshney, Arora, et al. (2004).

Radiometric Resolution

The radiometric resolution can also be defined as quantization. Quantization refers to the range of reflectance values that can be observed and the number of bits used to represent them digitally. HSIs typically have radiometric resolutions of 10 to 12 bits. An illustration of this is the AVIRIS sensor, which operates with a radiometric resolution of 10 to 12 bits (Varshney; Arora, et al., 2004).

A factor that is assessed and influenced by radiometric resolution is the Signal-to-Noise Ratio (SNR). This metric is expressed in decibel (dB) or the SNR. The SNR metric enables us to evaluate the capacity to distinguish the desired signal, object, or data from the noise or background values. Values above 0 dB or 1:1 are desirable for a reliable SNR estimation. For instance, the HyMap sensor exhibits a 500:1 ratio at 2200 nm and a 1000:1 ratio in the VNIR region (Varshney; Arora, et al., 2004).

2.2 ARTIFICIAL NEURAL NETWORKS

2.2.1 Introduction to Machine Learning

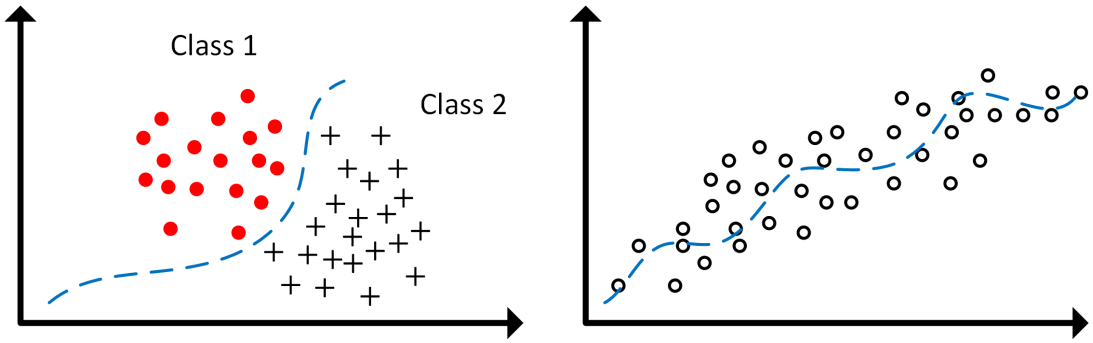
ML is a field of Computer Science that employs statistical techniques so that, without explicit programming rules, computers can learn from data. Unlike conventional computer programming, where developers deduce execution rules, in ML, rules are inferred by models (or algorithms) based on the presented data (Mitchell, 1997). Mitchell (1997) describes the learning process of algorithms in ML as a computer program that

learns from experience E with respect to a specific class of tasks T and a performance measure P, such that its performance on tasks in T, as measured by P, improves with experience E.

According to Murphy (2012), ML can be divided into two categories:

- Classification: It can be defined as the mapping (inference or prediction) from inputs x to outputs y learned through training. In this category, $y \in 1,..., C$, where C is the number of classes in the problem. When C equals 2, it is called binary classification, while when C exceeds 2, it is called multi-class classification. When presented with an unseen input (referred to as generalization), the algorithm estimates the class (or label) to which the input belongs. Figure 6(a) depicts a binary classification and the function $f(x)$ demarcating the separation between classes 1 and 2.
- Regression: This pertains to the inference of a continuous value ($y_i \in \mathbb{R}$) from input data ($x_i \in \mathbb{R}$). Regression can indeed be regarded as a classification with continuous values and without predefined classes. Figure 6(b) showcases a regression and the function $f(x)$. The function $f(x)$ is designed to capture the data behavior and allow inferences from new input values.

Figure 6 – Division of ML into categories (a) Classification and (b) Regression.



One of the fundamental mathematical concepts in ML is the loss function. A loss function is a measure of how well an ML model predicts the expected output based on a given set of inputs. Given a set of input-output pairs, the goal in ML is to find a function that minimizes the loss function. A commonly used loss function, also called a cost function, for training ML algorithms is cross-entropy loss. This function measures the distance between the model's predicted probability distribution and the true probability distribution (Mitchell, 1997; Murphy, 2012). Equation (1) mathematically expresses this loss function:

$$L = -\sum_{i=1}^{n} y_i \log \hat{y}_i \tag{1}$$

where L is the cross-entropy loss function, n is the number of classes, y_i is the true probability of class i, and $\hat{y}_i$ is the probability predicted by the model for class i. This equation can be further adapted for one-hot encoding, where the target values are represented as vectors of 0's and 1's, with the number of positions equal to the number of classes. This adaptation is known as categorical cross-entropy. The difference is that y_i becomes Y_i, and the output assigns a value of 1 to the classified class and 0 to the remaining classes.

Another important concept in ML is the gradient. The gradient of a function is a vector of partial derivatives that indicates the direction of the steepest ascent. In ML, we commonly use Gradiente Descendente (GD), an optimization algorithm that follows the negative gradient of a loss function to find the model parameters that minimize this function. The gradient descent update equation is used to adjust the weights of a neural network based on the gradient descent method (Mitchell, 1997; Murphy, 2012). It is given by:

$$w^{(t+1)} = w^{(t)} - \eta \nabla J(w^{(t)}) \tag{2}$$

where $w^{(t)}$ represents the set of weights at iteration t, η is the learning rate, and $\nabla J(w^{(t)})$ is the gradient of the cost function J with respect to the weights w at iteration t.

A variation of the GD algorithm is Stochastic GD (SGD). SGD is a gradient-based optimization method that updates the model's weights using the derivative of the loss function with respect to each weight. It performs this iteratively, one step at a time, until the loss function reaches a minimum or the number of iterations reaches a pre-defined limit. SGD is simple to implement and efficient for large models, but can be prone to getting stuck in local minima and may struggle to converge to the global minimum Mitchell (1997). The basic equation for SGD is as follows:

$$w_{t+1} = w_t - \eta \cdot \frac{\partial L}{\partial w_t} \tag{3}$$

where w_t is the weight vector at step t, L is the loss function, and η is the learning rate that controls the size of the optimization step. The partial derivative $\frac{\partial L}{\partial w_t}$ is the gradient of the loss function with respect to the weights at step t, indicating the direction in which the weights should be updated to minimize the loss function.

Another popular optimizer is Adaptive Moment Estimation (Adam). Adam is a variant of SGD that incorporates information about past gradients' moments to dynamically adjust the optimization step sizes. This allows Adam to automatically adapt to the problem's conditions, often achieving more accurate solutions more quickly than SGD (Kingma; Ba, 2017; Shao et al., 2024). The Adam algorithm involves more complex equations and includes two additional moments, m_t and v_t, which are updated at each optimization step and in each iteration, the g_t is the gradient of the loss function with respect to the parameters and is calculated:

$$g_t = \nabla_\theta f_t \theta_t \tag{4}$$

$$m_t = \beta_1 \cdot m_{t-1} + (1-\beta_1) g_t \tag{5}$$

$$v_t = \beta_2 \cdot v_{t-1} + (1-\beta_2) g_t^2 \tag{6}$$

where β_1 and β_2 are decay parameters that control the influence of past moments. Adam optimization uses bias correction to address initial bias in moment estimates when decay rates are small:

$$\hat{m}_t = \frac{m_t}{1 - B_1^t} \tag{7}$$

$$\hat{v}_t = \frac{v_t}{1 - B_2^t} \tag{8}$$

These moments are then used to update the weights as follows:

$$\theta_{t+1} = \theta_t - \frac{\alpha}{\sqrt{\hat{v}_t} + \varepsilon} \cdot \hat{m}_t \tag{9}$$

where α is the learning rate, and ε is a small value added to prevent division by zero. The square root of v_t serves as a scaling adjustment, helping regularize the optimization process.

According to Chollet (2018) and Mitchell (1997), the types of existing algorithms in ML, based on the type of training, are:

- Supervised Learning: This approach uses labeled data to train the model. The output labels that correspond to the correct responses are included in the training data.
- Unsupervised Learning: In this approach, the data received by the model is not labeled for training. The training stage allows the model to learn and find patterns in the data through techniques such as clustering.
- Semi-Supervised Learning: This approach is somewhere between supervised and unsupervised learning. The model receives both labeled and unlabeled training data.
- Reinforcement learning: An agent, a term used to refer to the model, learns through interaction with the application environment. During training, the agent receives rewards or punishments for specific actions. Training aims to teach the agent a policy that optimizes, with the objective of maximizing, the reward over time.

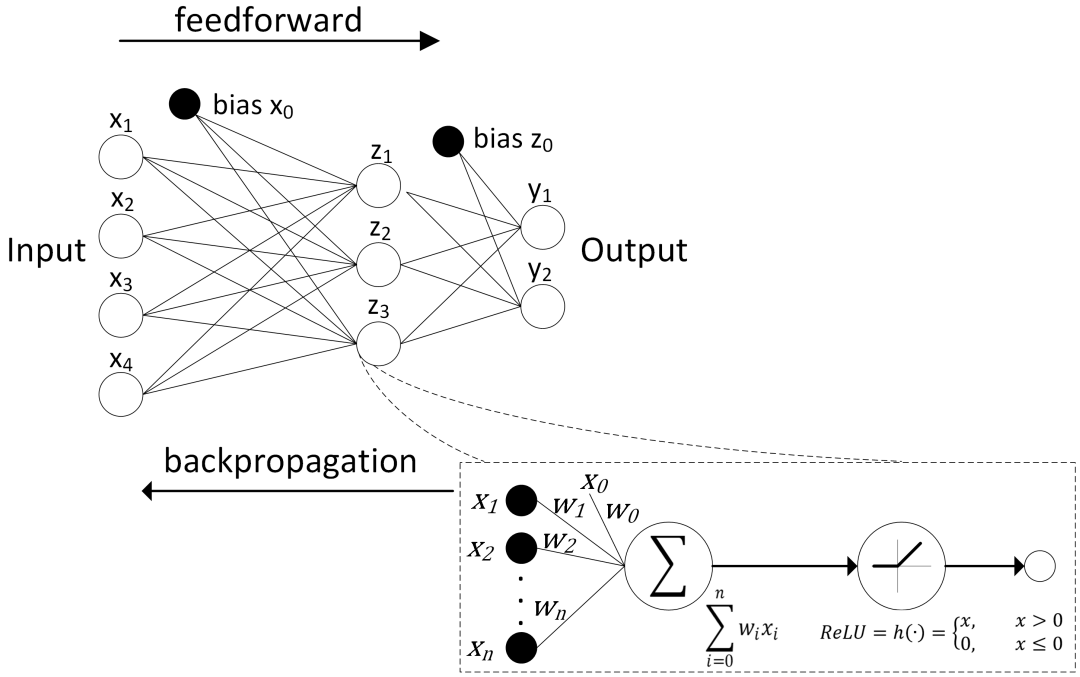
Another pivotal aspect of machine learning is the observation of generalization. This attribute identifies whether the model, during its learning process, has encountered underfitting, fitting, or overfitting. The term 'underfitting' refers to a model that is too simplistic to accurately capture the underlying pattern in the data, leading to poor performance on both the training and test data. A model suffering from underfitting will exhibit high bias, indicating it makes the same types of errors across different data

subsets. 'Fitting,' also known as a good fit or an ideal fit, occurs when a model accurately captures the pattern in the data. A well-fitted model will perform well on both training and test data. In 'overfitting,' a model fits the training data very closely but performs poorly on unseen test data, displaying high variance and becoming overly sensitive to minor changes in the training data.

2.2.2 Artificial neural networks

Within the context of machine learning, the human brain is used as inspiration and structure for the ANN algorithm. ANNs comprise interconnected nodes called neurons, which process and transmit information, as depicted in Figure 7 (Bishop et al., 1995; Mitchell, 1997; Rojas, 2013).

Figure 7 – Representation of an ANN composed of a structure with an input layer, only a hidden layer, and an output layer with ReLU type activation function.



Source: Adapted from Bishop et al. (1995) and Mitchell (1997).

Among ANNs, one of the most common is the feedforward neural network. This network is composed, at its base, of an input layer, one or more hidden layers, and an output layer. In this type of ANN, information flows unidirectionally from the input layer through the hidden layers to the output layer. In each layer, each neuron receives information from the outputs of other neurons, which are generated by a non-linear activation function (Bishop et al., 1995; Mitchell, 1997; Rojas, 2013). According to Mitchell (1997), the neuron output is defined by Eq. (10):

$$\hat{y}(x) = h(\sum_{i=1}^{n} w_i x_i + b) \tag{10}$$

where the output of the neuron is defined as y, the activation function is given by h, the weights between connections are represented by w_i, i-th indicates the connection between the input neuron and the current neuron, x_i represents is the output of the i-th input neuron and b is the bias of the current neuron. Using $\hat{y}(x)$, a partial y, means that each neuron has its own output that can be propagated forward. The term b can also be replaced by w_0 due to the common occurrence of x_0 being 1, making b exclusively dependent on the variation of w_0 (Bishop et al., 1995; Mitchell, 1997; Rojas, 2013).

The activation function $h(\cdot)$ can be any non-linear function. Examples of activation functions include the sigmoid function, the hyperbolic tangent function (also known as tanh), and the ReLU. The choice of activation function depends on the specific task and the data characteristics. Redes Neurais Artificiais (RNA) is widely used in various applications, including image and speech recognition, natural language processing, value prediction tasks, image classification, and pattern detection in time series data (Bishop et al., 1995; Mitchell, 1997; Rojas, 2013).

The weights and biases of the connections between neurons can be adjusted to optimize the network's performance for a given task. This process, known as training, involves adjusting the weights and biases to minimize the difference between the network's predicted and actual outputs. The backpropagation algorithm is a commonly used method for training ANNs, especially feedforward neural networks (Rojas, 2013; Bishop et al., 1995; Mitchell, 1997). Equation (11) shows the equation used to update the weights in the hidden layers:

$$w_{i,j}^{(k)} = w_{i,j}^{(k)} + \eta \delta_j^{(k)} x_i^{(k-1)} \tag{11}$$

where $w_{i,j}^{(k)}$ is the weight of the connection between neuron i in layer $k-1$ and neuron j in layer k, η is the learning rate, $\delta_j^{(k)}$ is the error of neuron j in layer k, and $x_i^{(k-1)}$ is the output of neuron i in layer $k-1$. The terms η, $\delta_j^{(k)}$, and $x_i^{(k-1)}$ together form $\Delta w_{i,j}^{(k)}$, the weight variation (Rojas, 2013; Bishop et al., 1995; Mitchell, 1997).

Equation (12) is used to update the weights in the output layer and is similar, but it uses the predicted output y instead of x:

$$w_{i,j}^{(k)} = w_{i,j}^{(k)} + \eta \delta_j^{(k)} y_i^{(k-1)} \tag{12}$$

where $w_{i,j}^{(k)}$ is the weight of the connection between neuron i in layer $k-1$ and neuron j in layer k, η is the learning rate, $\delta_j^{(k)}$ is the error of neuron j in layer k, and $y_i^{(k-1)}$ is the predicted output of neuron i in layer $k-1$. The terms η, $\delta_j^{(k)}$, and $y_i^{(k-1)}$ form the term $\Delta w_{i,j}^{(k)}$, which represents the weight variation (Rojas, 2013; Bishop et al., 1995; Mitchell, 1997).

Mitchell (1997) describes the δ terms for the output layer and the hidden layer in

Equation (13) and Equation (14):

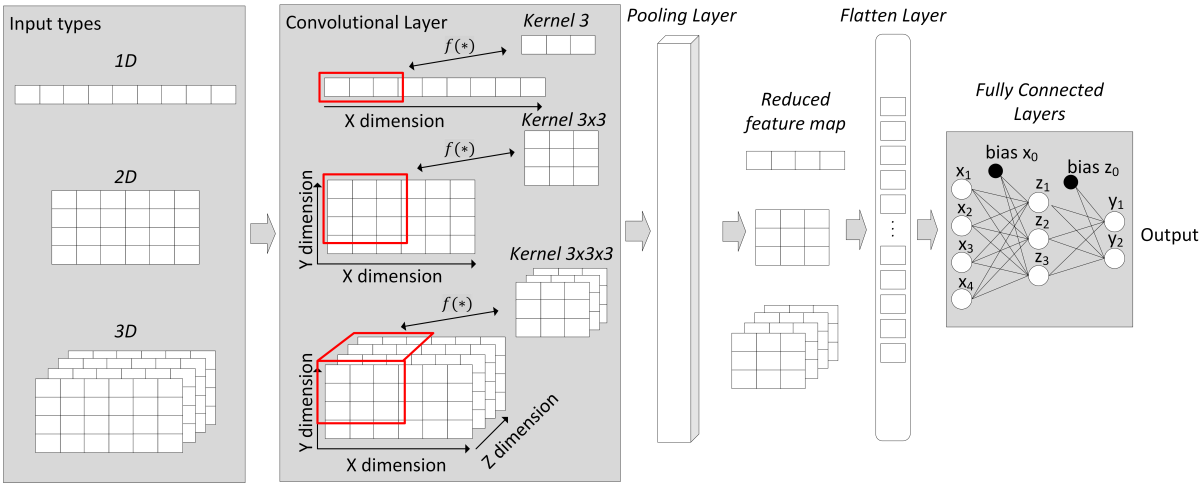
$$\delta_k = o_k(1 - o_k)(t_k - o_k) \tag{13}$$

$$\delta_h = o_h(1 - o_h) \sum_{k \in \text{outputs}} w_{hk}\delta_k \tag{14}$$

2.2.2.1 Convolutional Neural Networks

When compared to humans, who have limitations in presenting patterns and characteristics in images, ANNs are particularly suitable for image classification tasks as they are able to overcome these limitations. CNN is a type of ANN commonly used in this application domain because it is designed to process data with a matrix-like topology, such as images. Another accepted definition is that a CNN is an ANN that employs convolution in at least one layer instead of general matrix multiplication. CNNs can operate on one-dimensional, two-dimensional, or three-dimensional data, referred to as 1D-CNNs, 2D-CNNs, and 3D-CNNs, respectively. Figure 8 illustrates a basic model with a convolutional layer, a pooling layer, the flatten layer, and fully connected layers for 1D-, 2D-, and 3D-CNNs.

Figure 8 – 1D, 2D and 3D convolutional layers added to ANNs, forming the CNN.



According to Goodfellow, Bengio, and Courville (2016) and Venkatesan and Li (2017), we can describe the 1D-, 2D- and 3D-CNNs as:

- 1D-CNN: is a variant of CNNs that process data with a single dimension, such as audio signals or text data. The filters applied by 1D convolution layers have *kernels* that vary with the specificity of the application and start from size 1.
- 2D-CNN: is the most common type of CNN used to process data with two spatial dimensions, such as images. Convolutional layers apply a set of filters, or *kernels*, that vary with the application and typically start at 1x1. 2D-CNNs have achieved high performance in a variety of applications involving image

classification and have been applied to various other tasks in computer vision, including object detection, segmentation, and tracking.

- 3D-CNN: is a variant of CNNs used to process 3D data such as video, medical images, or HSI. 3D-CNNs combine convolutional and fully connected layers, among others. The main layers, the convolutional layers, apply a set of filters, or *kernels*, to the input data, which are adapted to the application and have minimum dimensions of 1x1x1.

CNN consists of multiple layers, including convolutional, pooling, and fully connected layers (similar to classical ANNs). Additionally, depending on the application, it may be necessary to vectorize the output of convolutional and pooling layers using a flatten layer, which adapts the data for the fully connected layers. Convolutional layers apply a set of filters to the input image, each designed to detect a specific pattern or feature. Convolution, described by Equation (15), is a mathematical operation that combines the input data and the filter to produce a feature map representing the presence of patterns or features detected by the filters (Goodfellow; Bengio; Courville, 2016):

$$s(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a) = x * w \tag{15}$$

where $s(t)$ is the convolved value, a is the number of steps (or window) in the convolution, x represents the input values, w is the kernel values, and t is the time index. The operator $*$ denotes the convolution operation.

According to Goodfellow, Bengio, and Courville (2016), convolution can be extended to higher dimensions, as shown in Equation (16):

$$S(i,j) = \sum_{m} \sum_{n} I(i-m, j-n)K(m,n) \tag{16}$$

where S is the convolved value in two dimensions, m is the number of steps (or window) in one dimension, and n is the number of steps in the other dimension, I is the input value at point i and j, and K represents the kernel values.

The convolutional layer also includes hyperparameters, such as stride and padding, which control how the filters are applied to the input data. The stride hyperparameter specifies the step size for each convolution operation, while padding determines whether virtual information, such as virtual edges in images, is added to preserve the dimensions of the feature maps at the output of the convolutional layer. The values in the convolutional layer also pass through an activation function.

Pooling layers reduce the resolution of the feature maps by applying a pooling operation, such as max pooling or average pooling, to groups of adjacent pixels. This reduces the size of the feature maps and helps decrease the model's complexity.

Fully connected layers, also known as dense layers, perform the final classification of the image based on the features extracted by the convolutional and pooling

layers. The output of the fully connected layer is a set of class probabilities, with each probability representing the likelihood that the input image belongs to a specific class.

2.2.2.2 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are a type of ANN well-suited for processing sequential data, such as text, audio, video, time series, or natural language processing. The term “recurrent" is used because of the way in which information flows between one stage and the next through the connections between neurons. This feature allows you to retrieve and store information from previous steps to make decisions in the present, adding context (Goodfellow; Bengio; Courville, 2016; Yu et al., 2019).

RNNs are composed of layers of neurons, and connections through weights are made with the next and previous layers. Examples of RNNs variations are LSTM and Gated Recurrent Unit (GRU) (Goodfellow; Bengio; Courville, 2016; Yu et al., 2019). According to Yu et al. (2019), Equations (17) and (18) define the default cell of an RNN:

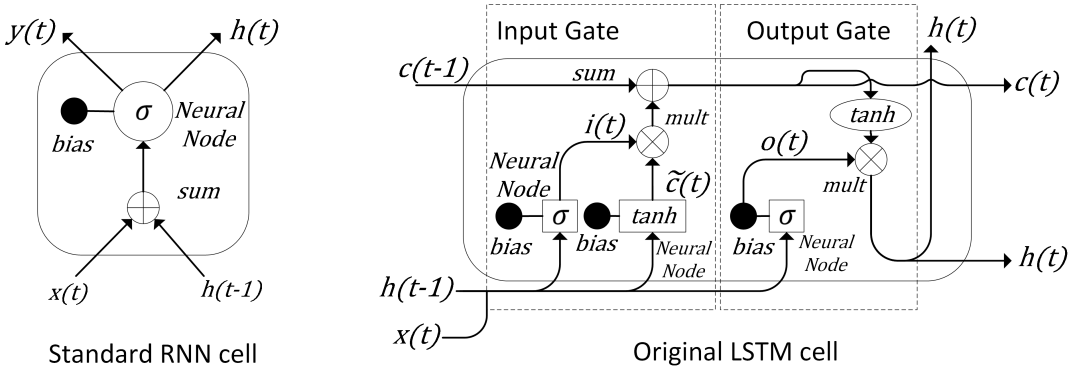
$$h_t = \sigma(W_h h_{t-1} + W_x x_t + b) \tag{17}$$

$$y_t = h_t \tag{18}$$

where the recurring information is given by h_t, the input is given by x_t, and the cell output at time t is given by y_t. W_h and W_x are the weights and b is the bias.

The LSTM type of RNN is designed to address the vanishing gradient problem, introducing a set of memory cells that store information for long periods of time for reference. In addition, there are also gates that control the flow of information, whether into or out of cells (Goodfellow; Bengio; Courville, 2016; Yu et al., 2019). Figure 9 illustrates a standard RNN cell compared to the original LSTM cell (or unit).

Figure 9 – Standard RNN and original LSTM cell architectures.



Source: Adapted from Yu et al. (2019).

The LSTM network is composed of LSTM cells. The LSTM cell receives an input x_t in time slot t and a hidden state h_{t-1} from the previous time slot. This produces an output h_t and a cell state c_t. The output weight matrix W_o, forgetting weight matrix W_f,

and input weight matrix W_i are also part of the LSTM cell (Yu et al., 2019; Goodfellow; Bengio; Courville, 2016; Yu et al., 2019).

The LSTM cell calculates the following quantities at each time interval:

$$i_t = \sigma(W_{ih}h_{t-1} + W_{ix}x_t + b_i) \tag{19}$$

$$\tilde{c}_t = \tanh(W_{\tilde{c}h}h_{t-1} + W_{\tilde{c}x}x_t + b_{\tilde{c}}) \tag{20}$$

$$c_t = c_{t-1} + i_t \cdot \tilde{c}_t \tag{21}$$

$$o_t = \sigma(W_{oh}h_{t-1} + W_{ox}x_t + b_o) \tag{22}$$

$$h_t = o_t \cdot \tanh(c_t) \tag{23}$$

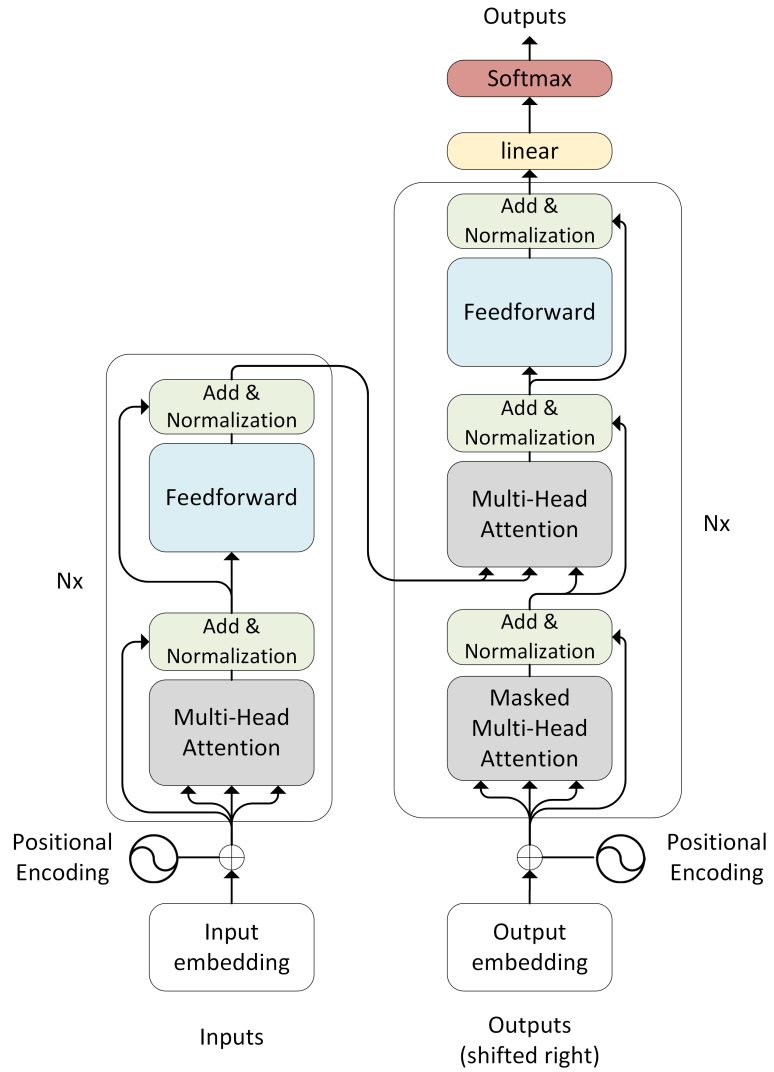
where the bias terms are b_i, $b_{\tilde{c}}$, and b_o and sigmoid activation function is given by $\sigma(\cdot)$. The input gate i_t controls which input information x_t can enter the cell. $\tilde{c}t$ corresponds to the output of the hyperbolic tangent type activation function. The variable f_t corresponds to the forget gate and is responsible for controlling which information from the previous state cell $(ct-1)$ is discarded. o_t corresponds to the output gate and is responsible for controlling which information from the current state cell, c_t, can be passed as h_t. The previous state cell and the current input are combined by the state cell, filtered by the input and forget gates (Goodfellow; Bengio; Courville, 2016; Yu et al., 2019).

2.2.2.3 Transformer Neural Network

The Transformer is a neural network architecture introduced by Vaswani et al. (2017). It is primarily used for natural language processing tasks, such as machine translation, language modeling, and text summarization, but it can also be applied to time series processing. One of the key features of the Transformer is the use of the Self-Attention Mechanism, or simply the Attention Mechanism, which allows the model to focus on different parts of the input sequence in parallel, rather than employing a sequential processing approach like traditional RNNs (Bahdanau; Cho; Bengio, 2014; Luong; Pham; Manning, 2015; Vaswani et al., 2017).

Figure 10 illustrates the basic architecture of the Transformer.

Figure 10 – Transformer Neural Network Architecture.



Source: Adapted from Vaswani et al. (2017).

However, the Transformer architecture can also be applied to time-series processing. The Self-Attention Mechanism, also referred to as the Attention Mechanism, is one of the main features of this architecture as it allows the model to serve different parts of the data input sequence in parallel. This feature differs from traditional RNNs that rely on sequential processing (Bahdanau; Cho; Bengio, 2014; Luong; Pham; Manning, 2015; Vaswani et al., 2017).

In the model, the importance of each part of the input for each output is determined by a set of weights, learned during training, introduced by the Attention Mechanism. The learned weights are treated as coefficients applied to the input data before it passes through the rest of the architecture. One approach to applying the Attention Mechanism, among the many that exist, is to use a separate Attention layer that, based on an importance or relevance metric, takes the input data and calculates weights that are then applied to the data before it is processed. continue model architecture

(Bahdanau; Cho; Bengio, 2014; Luong; Pham; Manning, 2015).

The Equation (24) shows the operation of calculating the dot product between each key vector and the query vector in the input data:

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{24}$$

where V is the set of value vectors, K is the set of key vectors, and Q is the query vector. A normalization by the square root of the dimension of the key vector d_k is applied to the dot product between each key vector and the query vector. After this operation, the softmax function uses the resulting dot products to produce a set of weights that scale the value vectors. The output of the Attention Mechanism is given by the weighted sum of the value vectors (Vaswani et al., 2017).

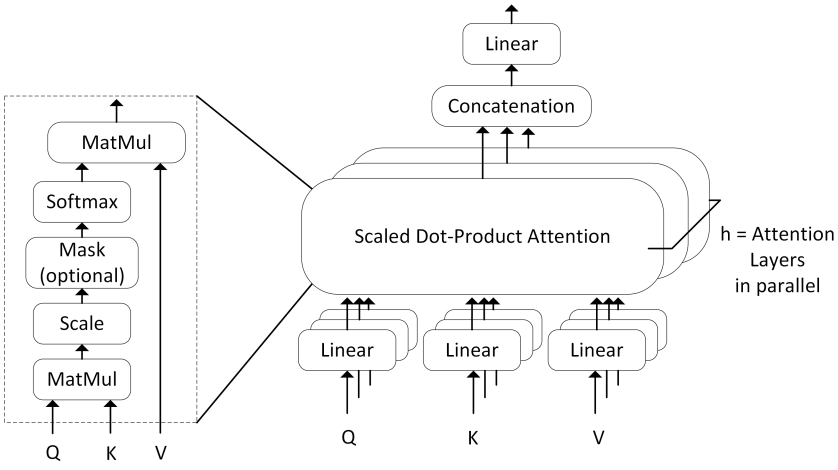
To allow simultaneous attention to different parts of the data input sequence, the Transformer architecture uses Multi-Head Attention (Vaswani et al., 2017). Figure 11 illustrates Multi-Head Attention. This can be expressed as shown in Equations (25) and (26):

$$MultiHead(Q, K, V) = Concat(head_1, ..., head_h)W^O \tag{25}$$

$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V) \tag{26}$$

where is applied h times, for each Attention head, the function $Attention(Q, K, V)$ and with different linear projections for K, V and Q together with the projection matrix W^O (Vaswani et al., 2017).

Figure 11 – Multi-head Attention layer architecture.



Source: Adapted from Vaswani et al. (2017).

Finally, feedforward networks along with residual connections and layer normalizations are also applied in the Transformer architecture (Vaswani et al., 2017).

Equations (27) and (28) express this:

$$FFN(x) = \max(0, xW_1 + b_1)W_2 + b_2 \tag{27}$$

$$LayerNorm(x) = \frac{\gamma_i(x - \mu_i)}{\sqrt{\sigma_i^2 + \varepsilon}} + \beta_i \tag{28}$$

where $FFN(x)$ is the feedforward network, W_1 and W_2 are trainable weight matrices, and b_1 and b_2 are the trainable biases. The layer that implements the normalization function is given by $LayerNorm(x)$, where μ is the mean of the input x, σ is the standard deviation of the input x, γ is the scale parameter, and β is the offset parameter. The parameters γ and β can be learned (Vaswani et al., 2017; Xiong et al., 2020).

2.3 HARDWARE ACCELERATORS

2.3.1 Processor Technology

Processor technology refers to the design and development of digital circuits. Processor technology can be categorized into three types: (*i*) General Purpose Processor (GPP); (*ii*) Application Specific Instruction-set Processor (ASIP); and (*iii*) Specific Purpose Processor (SPP) (Vahid; Givargis, 2001). According to Vahid and Givargis (2001), these three types of technologies determine the types of applications, algorithms, or computations that can be executed.

- GPP: This processor technology allows any class of computational application to be executed, always considering constraints. However, GPPs are general-purpose processors that offer moderate performance. Despite this, they can execute any application due to their wide instruction set.
- ASIP: This processor technology targets a more limited class of applications. It retains programmability but features an instruction set and organization specifically designed to execute low-complexity algorithms or those belonging to a particular application class.
- SPP: This technology focuses on the optimized design and implementation of an algorithm that was initially executed as software but later transformed into a processor. Key factors driving the construction of SPPs include improvements in computation time, energy consumption, or transmission parallelism.

2.3.2 Hardware Accelerators

The concept of processor technology pertains to the design and development of digital circuits. Hardware accelerators are described as SPPs, which are designed to optimize the implementation of algorithms that were initially executed in software and

later transformed into a processor (Cardoso; Coutinho; Diniz, 2017; Vahid; Givargis, 2001; Wang, 2017).

The design and development of SPPs align with the goal of improving or optimizing algorithm execution directly in digital hardware. The use of the performance metric of computation time guides SPPs and makes them HAs. The focus of HAs is to reduce the execution time of an algorithm by leveraging hardware parallelism, avoiding the overhead of executing instructions found in GPPs and ASIPs. By reducing the execution time, it is possible to improve other hardware metrics such as power dissipation, energy consumption, and development cost, although these are not guiding metrics like computation time (Cardoso; Coutinho; Diniz, 2017; Vahid; Givargis, 2001; Wang, 2017).

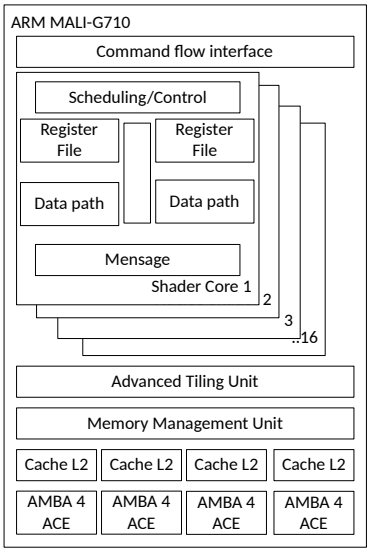
Generally, algorithms or tasks (algorithm stages) are candidates for transformation into HAs when they are computationally intensive or iterative. Optimizations and approximations are also accepted for HA construction, such as parallelism in task and operator execution, as well as simplified representations of operands and operators (approximate computing). Representing operators and operands in approximate or simplified formats can reduce computation time, often at the cost of precision and logical correctness. For instance, traditional GPPs use IEEE-754 floating-point representation for fractional operand computations, whereas HAs might employ fixed-point representation for computations, resulting in simpler operands and faster operators (Cardoso; Coutinho; Diniz, 2017; Vahid; Givargis, 2001; Wang, 2017). Several technologies are employed for hardware acceleration. Commonly, we can highlight the use of GPU, programmable logic, Application-Specific Integrated Circuit (ASIC), and SoC (Cardoso; Coutinho; Diniz, 2017; Vahid; Givargis, 2001; Wang, 2017).

GPU

The GPUs offer flexibility and support for software-level programming. Initially developed for graphics-related applications, they are also used for diverse tasks, including scientific applications and other forms of digital signal processing. The hardware acceleration provided by GPUs stems from their internal organization, consisting of numerous simple processing cores (shader cores) and internal memory. This design is developed to provide native data parallelism, implementing the Single Instruction, Multiple Data (SIMD) paradigm (Cardoso; Coutinho; Diniz, 2017; Wang, 2017).

Construction characteristics of GPUs vary depending on their end application and the platform into which they will be integrated (Cardoso; Coutinho; Diniz, 2017; Wang, 2017). GPUs designed for embedded systems feature simpler attributes, such as fewer memory levels, global data sharing, and a simplified internal distribution of shader cores (Cardoso; Coutinho; Diniz, 2017; Wang, 2017). Figure 12 illustrates the organization of a GPU tailored for embedded systems, highlighting these characteristics in a Mali-G710 GPU from ARM.

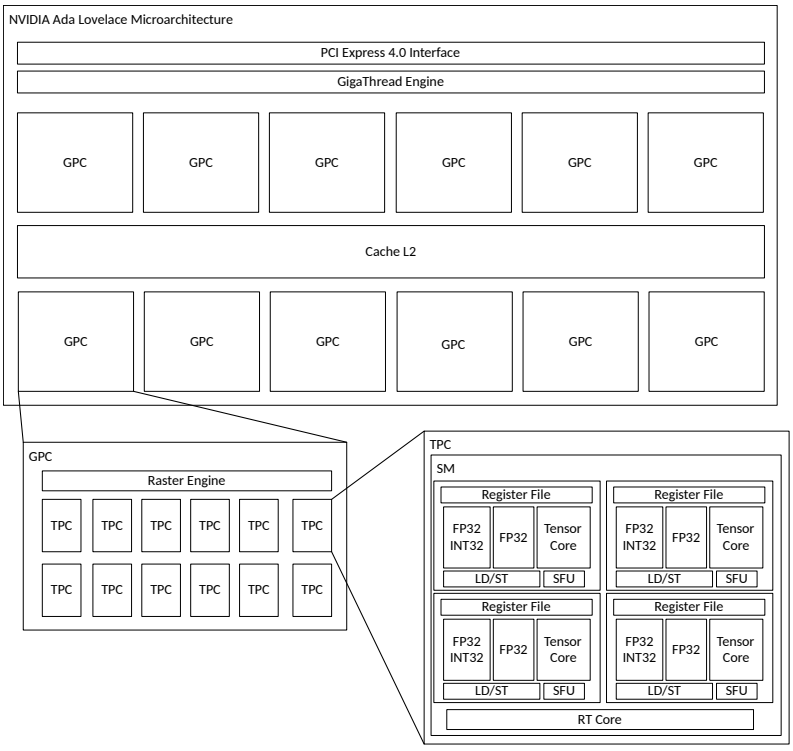
Figure 12 – Organization of the ARM Mali-G761 Embedded GPU.



Source: Adapted from ARM (2022).

High-performance GPUs are composed of clusters of shader cores, which include internal memory levels that are partially shared with other clusters and partially reserved for internal use. Figure 13 illustrates a high-performance GPU, showcasing the key features mentioned in a GPU with the Ada Lovelace microarchitecture (or organization) from NVIDIA.

Figure 13 – Microarchitecture of the NVIDIA Ada Lovelace High-Performance GPU.



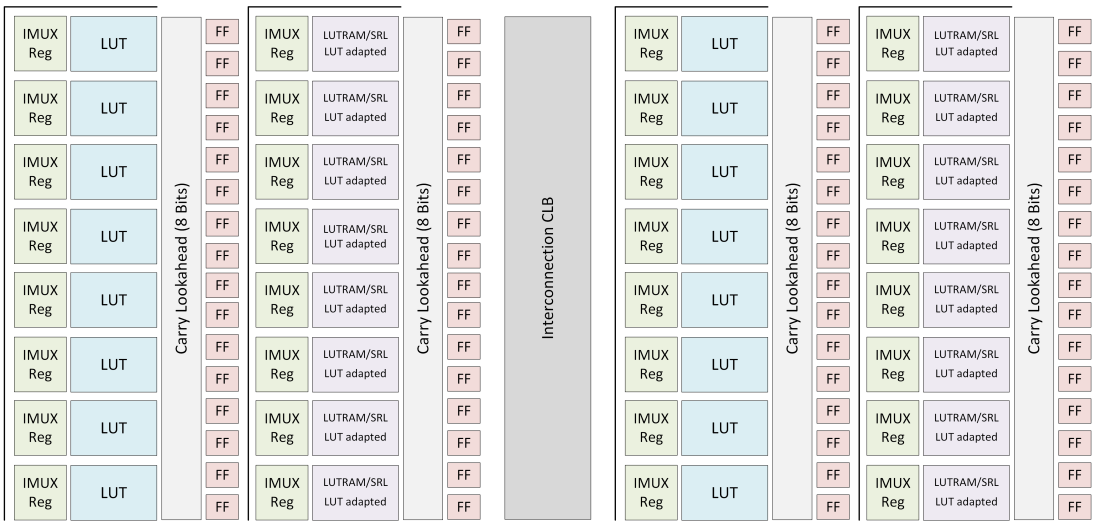
Source: Adapted from NVIDIA (2022).

Furthermore, the number of shader cores and operational capacity are enhanced in this type of GPU, including optimizations tailored for specific application classes, such as neural networks (NNs) (Cardoso; Coutinho; Diniz, 2017; Wang, 2017).

Programmable Logic - Field-Programmable Gate Array

Programmable logic, or reconfigurable hardware, is widely employed for hardware acceleration. Notably, Field-Programmable Gate Arrays (FPGAs) stand out as platforms widely used for applications that require or are ported to/from HAs. FPGAs are specifically designed as a matrix of elements that can be dynamically configured based on the description of the desired solution. Elements within an FPGA include Configurable Logic Block (CLB), also known as Logic Cells, Input/Output Block (IOB), Digital Signal Processors (DSPs), Block Random Access Memory (BRAM), and interconnection elements that link these blocks (referred to as a switching matrix). CLBs encompass components such as Look-Up Table (LUT), flip-flops, and multiplexers. These elements come in varying sizes, dependent on the manufacturer and device family (Cardoso; Coutinho; Diniz, 2017; Skliarova; Sklyarov, 2019; Wang, 2017). Figure 14 provides a general illustration of the organization and blocks within a CLB from Xilinx, specifically their Versal FPGA family.

Figure 14 – Organization of the CLB present in Versal devices from the manufacturer Xilinx.

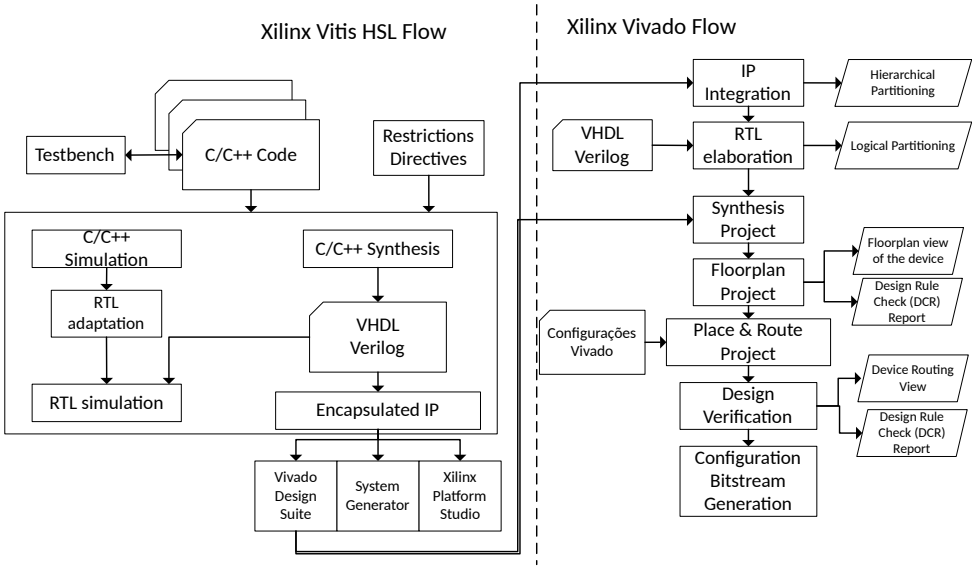


Source: Adapted from Xilinx (2021).

The accelerator to be implemented in programmable logic is realized using hardware description or programming languages. Typically used languages for this purpose are VHSIC Hardware Description Language (VHDL) and Verilog for hardware description and C/C++ for logic inference from high-level languages, known as High-Level Synthesis (HLS). In addition to frameworks like FINN, which provides an end-to-end solution for the inference of quantized machine learning models (Cardoso; Coutinho;

Diniz, 2017; Wang, 2017; Blott et al., 2018; Skliarova; Sklyarov, 2019). Figure 15 illustrates the development flow for both cases using the Vitis HLS and Vivado Design Suite tools from Xilinx.

Figure 15 – Development Flow for FPGA Using Vitis HLS and Vivado Design Suite Tools from Xilinx.



Source: Adapted from Xilinx (2022b,c).

In Figure 15, it is evident that even when designing in HLS, the final development and generation of the FPGA configuration file are performed using the Vivado Design Suite tool. Accelerator designers describe the behavior of the algorithm or task in the programming language of their choice, utilizing tools provided by programmable logic device manufacturers. The tool generates a configuration file called a bitstream. The bitstream is used to interconnect and configure the elements within the device, enabling them to replicate the behavior of a digital circuit that executes the algorithm or task (Cardoso; Coutinho; Diniz, 2017; Wang, 2017; Skliarova; Sklyarov, 2019).

The flexibility of inference when describing an algorithm or task enables the optimization of various design variables. It allows for the creation of more efficient mathematical operators in terms of computation time and hardware resources, appropriate quantization tailored to the essential data being processed, multi-access capabilities for memory reads and writes, and other features. This flexibility allows computations performed in programmable logic to outperform operations executed on GPUs and CPUs, whose fixed architectures and organizations are only programmable at the instruction level (software) (Cardoso; Coutinho; Diniz, 2017; Wang, 2017; Skliarova; Sklyarov, 2019).

However, FPGAs, for instance, must adapt their organization and architecture to support the inferred digital circuits, leading to challenges such as limited resource capacity, necessitating larger devices and lower operational frequencies, and requiring

designers with advanced expertise in hardware description and physical hardware. As a result, these devices are ideal for projects aiming to achieve better performance than CPUs or even GPUs, but they typically underperform compared to ASICs or fully customized circuits. Factors such as cost and time-to-market make programmable logic devices more advantageous than ASICs (Cardoso; Coutinho; Diniz, 2017; Wang, 2017; Skliarova; Sklyarov, 2019).

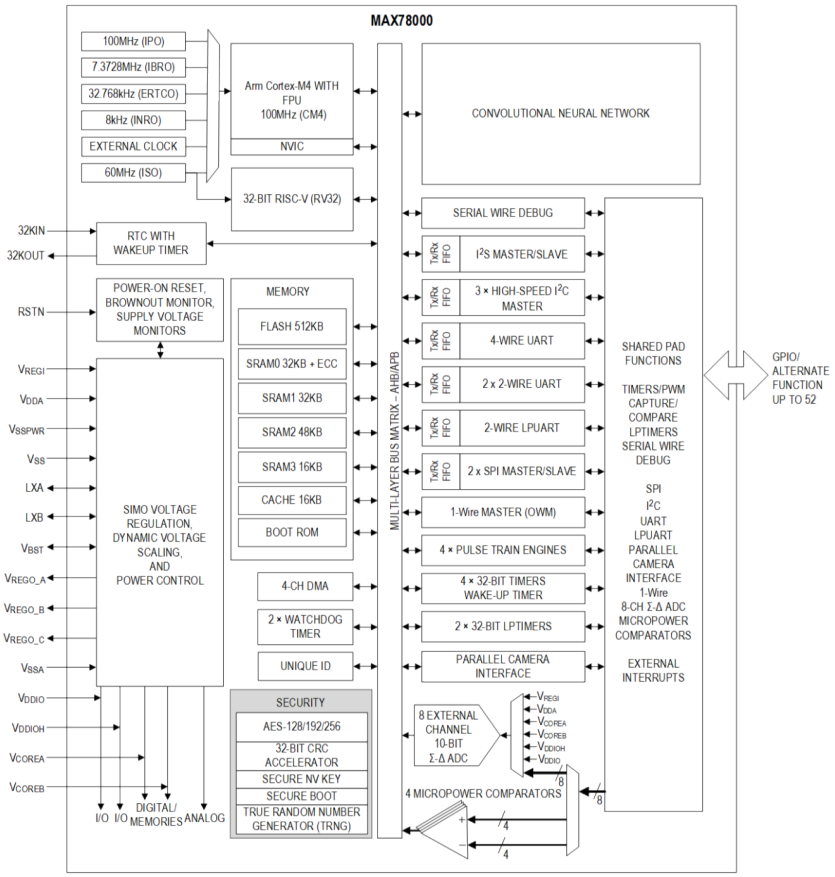
ASIC, full custom, and SoC

The design and implementation of accelerators can also be carried out directly using ASIC technology, which is fully customizable, or in SoC devices (constructed using ASIC or fully customizable components) that incorporate accelerators into their architecture or support inference via programmable logic areas. Hardware accelerators developed in ASIC with full customization help mitigate the overhead of programmable logic. This consideration arises from the fact that the digital circuit employed generates the algorithm's or task's behavior with the minimum required or near-minimum. On the other hand, inference on SoCs equipped with programmable logic can alleviate part of the costs associated with control, communication, and other peripherals used or needed for computation (Vahid; Givargis, 2001; Cardoso; Coutinho; Diniz, 2017; Wang, 2017).

Developing hardware accelerators directly on ASICs or fully customizable circuits requires significantly more time for design, specialization, development, and production compared to GPUs or programmable logic devices. However, these characteristics result in smaller circuits operating at higher frequencies, leading to shorter computation times, lower per-unit costs (at high production volumes), and the potential for lower energy consumption (Vahid; Givargis, 2001; Cardoso; Coutinho; Diniz, 2017; Wang, 2017).

Figure 16 illustrates an SoC from Maxim that incorporates a hardware accelerator based on a CNN algorithm. While this approach offers advantages in terms of application development time, the hardware-implemented CNN is limited in customization and applicability (Integrated, 2021).

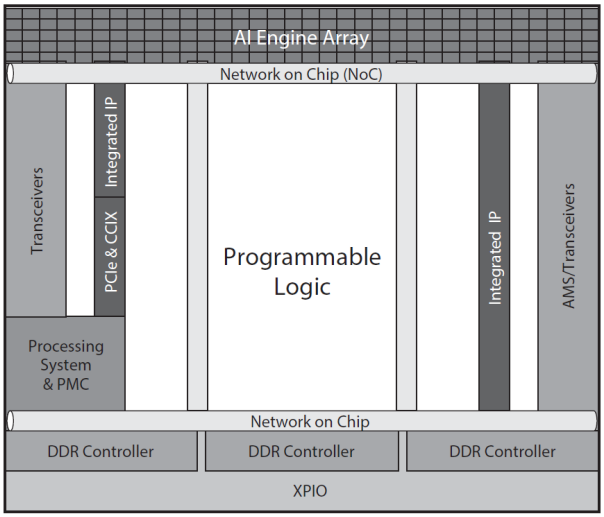
Figure 16 – MAX78000 SoC with a hardware accelerator for executing CNN models.



Source: Adapted from Integrated (2021).

To mitigate the design and development challenges of accelerators, some SoCs include programmable logic areas. For instance, Figure 17 illustrates the internal organization of the Xilinx Versal device, which features a MultiProcessor SoC with a hardware accelerator for executing CNNs as well as a programmable logic area.

Figure 17 – Overview of the AI Engine Organization in the Xilinx Versal Device.



Source: Adapted from Xilinx (2022a).

Various computational resources are provided using ASIC or fully customizable technology. The programmable logic area available enables the optimization of algorithms or tasks in hardware, requiring the same level of design and development effort as purely programmable logic devices (Cardoso; Coutinho; Diniz, 2017; Wang, 2017).

2.4 CONSIDERATIONS

The concepts presented as theoretical foundations are used to introduce the knowledge necessary to conduct this research. The topic of HSI provides an understanding of how this type of imagery can be utilized for classification, the types of data and information it contains, and how they can be extracted. The concepts of ANN and their architectural variations are introduced to explore the key aspects of ANNs and how this type of machine learning algorithm relates to and can be employed for extracting information from HSI. Finally, the topic of HA is included to understand how its concepts, such as benefits, technologies, and definitions, can be leveraged to reduce computational time for tasks or software, thereby decreasing energy consumption and enabling real-time responses.

3 REVIEW OF THE STATE OF THE ART OF DEEP LEARNING ARCHITECTURES HARDWARE ACCELERATION FOR HSI CLASSIFICATION

This section presents a state-of-the-art review of hardware acceleration for HSI classification and similar data structures based on ANN architectures. The research focused on hardware accelerators used for onboard processing in remote sensing, as space applications. Recent contributions in the field were analyzed to address the following questions:

- Which ANN architectures are commonly used for HSI classification?
- Which ANN architectures are implemented as hardware accelerators for HSI classification?
- How is GPU-level HSI classification implemented for embedded systems?

The questions outlined guided a systematic review of works indexed in various bibliographic and reference databases, such as Institute of Electrical and Electronics Engineers (IEEE) Xplore, Association for Computing Machinery (ACM) Digital Library, and Google Scholar. These databases enable searches in fields related to Computer Science and Electrical Engineering. Google Scholar was used for its broader indexing of databases beyond IEEE and ACM, including Elsevier, SPIE, and Multidisciplinary Digital Publishing Institute (MDPI). The main inclusion criteria were a focus on acceleration using hardware, the use of complete neural network architectures, a focus on HSI (both images and vector data), the use of ASICs, GPUs, or FPGAs, applications for remote sensing or space systems, and publications from 2016 onward. Exclusion criteria included works not written in English or Portuguese, natural language processing studies, and those using GPUs solely as auxiliary components for training or inference.

ANN architectures that are most relevant to HSI classification were investigated to refine the systematic review. CNN, RNN using LSTM, and more recently, the Transformer architecture stood out. As a result, CNN, LSTM, and Transformer were explored as foundational architectures, with hardware accelerators for HSI and various image classification tasks. Time-series data formats were included due to their similarity to HSI pixels.

The queries used to carry out the systematic review initially focused on the architectures that obtained the most returns in the searches. For the CNN architecture, the query used was *("CNN" OR "convolution") AND ("neural network" OR "network" OR "DNN" OR "machine learning") AND ("hardware accelerator" OR "hardware acceleration" OR "FPGA") AND ("hyperspectral" OR "multispectral")*. For the LSTM architecture, the query used was *("RNN" OR "LSTM" OR "GRU") AND ("neural network" OR "network" OR "DNN" OR "machine learning") AND ("hardware accelerator" OR "hardware acceleration" OR "FPGA") AND ("hyperspectral" OR "multispectral")*.

The search queries for transformer architecture were *"transformer" AND ("neural*

network" OR "network" OR "DNN" OR "machine learning") AND ("hardware accelerator" OR "hardware acceleration" OR "FPGA") and *("vision transformer" OR "ViT") AND ("neural network" OR "network" OR "DNN" OR "machine learning") AND ("hardware accelerator" OR "hardware acceleration" OR "FPGA") AND ("remote sensing" OR "hyperspectral" OR "multispectral")*. The use of two queries for the transformer architecture was due to problems with the search, including work on an electric transformer device.

Additionally, to emphasize the use of GPUs, the following search query was employed: “Hyperspectral” AND “GPU” AND “Jetson” AND (“CNN” OR “Transformer” OR “LSTM”). This query was designed to identify publications that lie at the intersection of hyperspectral image classification, hardware acceleration, and modern deep learning architectures, with a focus on GPU-based processing.

The main inclusion criteria for HA works were: (i) focus on hardware acceleration; (ii) use of complete ANN architectures; (iii) relevance to HSI – processing images and time series data; (iv) utilization of ASICs, GPUs, or FPGAs; (v) application to remote sensing or space systems; and (vi) published from 2016 onwards. The main exclusion criteria for works were: (i) natural language processing topics; (ii) utilization of GPUs solely for training or GPU that not are for embedded applications; (iii) finished works; (iv) description of the implemented architecture; (v) metrics used for classification and hardware accelerator performance; (vi) covers non-English or Portuguese language papers.

Initially, considering the CNN, LSTM, and Transformer architectures, 128 works were identified that focused on hardware acceleration for HSI classification, image, or vector data. After applying inclusion and exclusion criteria, only 57 works focused on the acceleration concept of the investigated architectures were selected.

3.1 CNN-BASED HARDWARE ACCELERATORS

The focus of works on hardware accelerators based on CNNs was primarily on processing vector data for HSI. In total, 21 works were selected. Compared to LSTM and Transformer architectures, CNN is the architecture with the most hardware-accelerator-related works focused on HSI classification.

The main acceleration technologies identified were GPUs and FPGAs. Specifically, 12 works used GPUs (six of which employed NVIDIA Jetson embedded devices), four studies used fully customized circuits, and five employed FPGA-based accelerators. Regarding data type, 12 studies focused exclusively on spectral-level data, while nine incorporated spectral-spatial imagery. The application domains included general remote sensing (19 works) and specific spatial applications (two works).

Among the circuit technologies used, the works by Giuffrida et al. (2020, 2021a), Rapuano et al. (2021) and Spiller et al. (2022a) employ technologies utilizing fully customizable circuits. The works by Alcolea et al. (2020a), Dang et al. (2022), Zhu,

Du, and Zhang (2020), Huang et al. (2024), Sun et al. (2023), Spiller et al. (2022a), Thangavel et al. (2023), Paoletti et al. (2021), De Lucia, Lapegna, and Romano (2023), Carbone et al. (2023), Chen and Huang (2023) and Shinde and Patidar (2022) use GPUs, while the works by Chen et al. (2018), Raita-Hakola et al. (2022), Zang et al. (2022), Reddy et al. (2025) and Liu et al. (2025) utilize FPGAs.

Focusing on HSI classification, we can highlight the works of Huang et al. (2024), Sun et al. (2023), Spiller et al. (2022a), Thangavel et al. (2023), Paoletti et al. (2021), De Lucia, Lapegna, and Romano (2023), Carbone et al. (2023), Chen and Huang (2023) and Shinde and Patidar (2022), which employ GPUs, particularly NVIDIA Jetson devices, and mainly emphasize spectral-spatial data exploration. On the other hand, the studies in Reddy et al. (2025) and Liu et al. (2025) investigate HSI classification on FPGAs, with a strong focus on spectral-spatial processing.

Table 1 summarizes the main findings regarding CNN-based HSI classification. The works that most closely match the proposal of this thesis are highlighted.

Table 1 – Summary of CNN-based architectures for HSI classification.

Work	Focus	Datasets	Hardware	Key Point
Paoletti et al. (2021)	3D-CNN with mixed precision (FP16/FP32)	IP, PU, SA, and BT	Jetson Xavier, Titan RTX	Lower cost, memory via FP16
Shinde and Patidar (2022)	Lightweight DCNN with 6 layers (spectral)	Indian Pines	CPU/GPU	Simple, accurate CNN (98%)
Spiller et al. (2022)	**1D-CNN for wildfire segmentation onboard**	**PRISMA (fires 2019)**	**Jetson Movidius**	**Edge accelerators for alerts**
De Lucia et al. (2023)	**PCA + CNN for edge HSI**	**Generic UAV**	**Edge GPUs**	**Band reduction saves energy**
Chen and Huang (2023)	Band-based CNN for AQI	AQI (urban/traffic/ind.)	FPGA vs Jetson Nano	Adaptive band selection on FPGA
Carbone et al. (2023)	**1D-CNN vs. 3D-CNN (wildfire detection)**	**PRISMA (fires 2019)**	**Jetson TX2**	**Domain adaptation for wildfires**
Thangavel et al. (2023)	**1D-CNN for wildfire onboard detection**	**PRISMA (fires 2019)**	**Intel Movidius, Jetson Nano, Jetson TX2**	**Trusted autonomous satellite operation**
Sun et al. (2023)	Dual Siamese CNN (spatial + spectral) for tracking for HSI	WHISPERS 2021	Jetson Xavier NX	Real-time HSI tracking
Huang et al. (2024)	Custom CNN for AQI (HSI input)	AQI (HSI)	FPGA UAV	Trustworthy UAV with FPGA crypto

Studies such as Paoletti et al. (2021), Shinde and Patidar (2022) and Spiller et al. (2022a) compared GPU-based inference between Jetson Nano, TX2, and Intel Movidius for 1D- and 3D-CNN models, analyzing energy efficiency and inference time

for edge deployment. Chen and Huang (2023) and Huang et al. (2024) extended these analyses to FPGA implementations, demonstrating energy reductions with minimal accuracy loss.

3.2 LSTM

The works that address LSTM architectures implemented as hardware accelerators exhibit particularities compared to CNN. In total, 13 works were selected, and among these works, Fouda (2022) using FPGA and Zhu et al. (2024) using GPU present the use of LSTM focused on HSI. The 13 projects were separated by the technology used for acceleration, the type of data processed, and the area of application.

Among the 13 selected projects, three implement acceleration in ASICs or full custom solutions, nine implement it in FPGAs, and one implements it in an embedded GPU.. The works by Cao et al. (2019), Ding et al. (2018), Fouda (2022), Jiang et al. (2019), Kouris, Venieris, and Bouganis (2019), Que et al. (2019), Rybalkin et al. (2021), Torti et al. (2020, 2021) and Yin et al. (2021a) implement the accelerator in FPGA technology, while the works by Ding et al. (2018), Liu et al. (2022) and Wu, Liu, and Chen (2021) use fully-customize circuit technology or ASIC. The work Zhu et al. (2024) is the only one that uses an embedded GPU to accelerate the LSTM model.

Regarding the type of data processed, two studies use HSI, four use regular images, and seven are based on time series. The works of Cao et al. (2019), Jiang et al. (2019), Kouris, Venieris, and Bouganis (2019), Que et al. (2019), Torti et al. (2021, 2020) and Wu, Liu, and Chen (2021) use time series as input for the HA, while Ding et al. (2018), Rybalkin et al. (2021), Yin et al. (2021a), Fouda (2022), Liu et al. (2022) and Zhu et al. (2024) use images as input for the LSTM. As previously mentioned, authors Fouda (2022) and Zhu et al. (2024) use HSI as the data to be processed.

As for the end application, only two works focus on the space sector, while the other eleven are terrestrial and common applications. The works of Jiang et al. (2019), Kouris, Venieris, and Bouganis (2019), Fouda (2022), Liu et al. (2022) and Zhu et al. (2024) target remote sensing, Fouda (2022) and Zhu et al. (2024) also aims at space systems, and Ding et al. (2018), Cao et al. (2019), Que et al. (2019), Torti et al. (2020, 2021), Rybalkin et al. (2021), Wu, Liu, and Chen (2021) and Yin et al. (2021a) focus on applications outside the space domain.

Table 2 summarizes the main findings regarding LSTM-based classification works. The works that most closely match the proposal of this thesis are highlighted.

Table 2 – Summary of LSTM-based architectures works for classification.

Work	Focus	Datasets	Hardware	Key Point
Fouda (2022)	LSTM for compression of satellite imagery in ScOSA architecture for CubeSat application	BIROS (IR), MACS (VIS), Sentinel (VIS/IR), DESIS (HSI)	GPU training and HA on FPGA Zynq-7020 and Zynq UltraScale+ ZU7EV	Onboard encoder, high compression ratios while preserving reconstruction quality and meeting the 10 W power and 1 Hz throughput constraints
Zhu et al. (2024)	**Multi-agent reinforcement learning (MaHSI) for adaptive band selection to optimize communication, energy, and accuracy**	**IP, PU, SA, KC, BT, Houston, HyRANK Dioni, and HyRANK Loukia**	**GPU Training: NVIDIA RTX 2070 Super GPU Test: NVIDIA Jetson TX2**	**Band selection onboard to save bandwidth/energy, transmitting only informative bands and maintain accuracy**

The listed works on LSTM show a greater focus on the use of FPGAs, while exploration with embedded GPUs is demonstrated in only one work, in (Zhu et al., 2024). However, despite being a single work, it demonstrates the applicability of LSTM models with embedded GPUs for on-board processing in satellites.

3.3 TRANSFORMER

The Transformer architecture has been gaining prominence in recent years for processing time-series data and has been replacing RNN architectures such as LSTM for various tasks. Due to its performance, this architecture has been utilized for HSI classification, particularly at the spectral-spatial level.

In total, 23 (HA) works based on the Transformer architecture were selected. Among these 23 works, La Salvia et al. (2024), Torti et al. (2023), Assudani and Krishna (2025), Růžička and Markham (2025) and Lu et al. (2025) focus on classifying HSI. Five works employed GPU acceleration, one utilized ASIC, and 17 adopted FPGAs. Regarding data type, 17 works processed 2D or 3D images, while six handled time-series data. Furthermore, in the context of application, 22 out of 23 papers target diverse applications, while only the paper by Yoo et al. (2022) targets space applications with HSI.

The works Dong et al. (2022), Hu et al. (2021), Huang et al. (2022), Kong et al. (2021, 2022), Li et al. (2022a), Li et al. (2022b), Liang et al. (2022), Sun et al. (2022a), Wang and Chang (2022), Wang et al. (2022a) and Yoo et al. (2022) focus on accelerating image processing, while Fan, Ferianc, and Luk (2022), Jiao et al. (2021), Qi et al. (2021), Song et al. (2022), Sun et al. (2022b) and Wojcicki et al. (2022) focus on accelerating time series data.

Focusing on HSI and the use of embedded GPUs, we can highlight the studies by La Salvia et al. (2024), Torti et al. (2023), Assudani and Krishna (2025), Růžička and Markham (2025) and Lu et al. (2025).

Table 3 summarizes the main findings regarding Transformer-based classification works. The works that most closely match the proposal of this thesis are highlighted.

Table 3 – Summary of Transformer-based architectures works for classification.

Work	Focus / Architecture	Datasets	Hardware	Key Point
Torti et al., (2023)	Parallel Vision Transformer for skin cancer HSI	Medical HSI (skin lesions)	Jetson Nano (5W), RTX 2080 GPUs	Low-power GPU feasibility, portable diagnostics
La Salvia et al. (2024)	Edge/Cloud Vision Transformer for HSI cancer diagnosis	Medical HSI (skin cancer datasets)	Low-power GPUs, desktop GPUs, HPC clusters	Edge focusing low energy, Cloud focusing faster inference trade-off
Assudani et al. (2025)	**Fusion: EfficientNet (RGB) + 1D-CNN (HSI) + ViT**	**Plant disease datasets (public + custom)**	**Jetson Nano**	**Accuracy, time, and optimized with pruning/distillation**
Lu et al. (2025)	Graph Neural Network + Attention for nitrogen estimation	Field maize HSI datasets (growth stages)	RS platforms (CPU/GPU)	Temporal-spatial nitrogen estimation
Růžička and Markham (2025)	**HyperVit, Adapted Transformer (SegFormer, EfficientViT) with spectral layers + upscaling**	**OxHyperSyntheticCH4, OxHyperRealCH4, OxHyperMinerals (from EMIT)**	**Jetson AGX Xavier, Raspberry Pi, Unibap SpaceCloud (ION-SCV proxy).**	**Methane detection vs prior DL, faster inference than baselines general-purpose foundation models for HSI onboard**

Regarding the technology, only the work by Wang and Chang (2022) does not use FPGA and has a design focused on customizable hardware, while all other listed works use FPGA. Regarding applicability, only the work by Yoo et al. (2022) addresses the context of space systems, whereas the other works have various applications. Several works, such as those by La Salvia et al. (2024), Torti et al. (2023), Assudani and Krishna (2025), Růžička and Markham (2025) and Lu et al. (2025), show the use of Transformer-based architectures running on embedded GPUs, but none of them explore reduced architectures and the energy consumption, as well as the communication latency of different dataset sizes.

3.4 DISCUSSION

In the present study, considering hardware acceleration (HA), it is noteworthy that the literature shows a predominance of the use of CNN for spectral-level HSI classification, whereas only two studies employ the LSTM architecture for HSI processing, and five studies adopt the Transformer architecture. This observation contrasts with the initial refinement search, which indicated a considerable number of studies employing all three architectures for HSI classification; however, those results were primarily focused on software implementations. When narrowing the scope to hardware acceleration, the number of implementations is significantly reduced, despite the well-known benefits of hardware-based acceleration.

Nonetheless, when analyzing the Transformer architecture, there is a noticeable trend toward its adoption for hardware-accelerated image processing, where it is commonly referred to as Vision Transformer (ViT). This conclusion is supported by the selected studies published within the last five years.

Despite the growing number of works employing ViT, all focus on processing two- or three-dimensional images, addressing spatial or spectral-spatial characteristics in the context of HSI classification. The emphasis on Transformers stems from their emerging role in replacing RNNs, such as LSTM, and even CNNs for one-dimensional data processing.

It is worth noting that although studies have implemented ANN architectures for HSI classification on both FPGA and GPU platforms, none have analyzed the behavior of the Transformer architecture for spectral-only HSI classification. The reviewed studies predominantly emphasize the exploitation of spatial or spectral-spatial features.

The literature review conducted allows us to answer the questions posed at the beginning of this chapter:

- Which ANN architectures are commonly used for HSI classification?
  - A greater adherence to CNN, LSTM, and Transformer architectures has been identified within ANN architectures. More recent work even focuses on the Transformer.
- Which ANN architectures are implemented as hardware accelerators for HSI classification?
  - All three highlighted architectures, CNN, LSTM, and Transformer, are hardware-accelerated. The technologies commonly employed are FPGA, ASIC/Full Custom, and embedded GPUs. There are also works that mention the use of Vision Processing Unit (VPU), but these works were indicated as ASIC/Full Custom.
- How is GPU-level HSI classification implemented for embedded systems?
  - 1D, 2D, and 3D data level using NVIDIA Jetson to explore edge AI with low-power computation using CNN, LSTM, and Transformer, but 1D-only classification level (spectral) is not explored with Transformer-based architectures.

Table 4 summarizes the main findings regarding all highlighted works among CNN, LSTM, and Transformer-based architectures.

Table 4 – Summary of architectures works for classification in compare with this work.

Work	Focus / Architecture	Datasets	Hardware	Key Point
Spiller et al. (2022)	1D-CNN for wildfire segmentation onboard	PRISMA (fires 2019)	Jetson Movidius	Edge accelerators for alerts
De Lucia et al. (2023)	PCA + CNN for edge HSI	Generic UAV	Edge GPUs	Band reduction saves energy
Carbone et al. (2023)	1D-CNN vs. 3D-CNN (wildfire detection)	PRISMA (fires 2019)	Jetson TX2	Domain adaptation for wildfires
Thangavel et al. (2023)	1D-CNN for wildfire onboard detection	PRISMA (fires 2019)	Intel Movidius, Jetson Nano, Jetson TX2	Trusted autonomous satellite operation
Zhu et al. (2024)	Multi-agent reinforcement learning (MaHSI) for adaptive band selection to optimize communication, energy, and accuracy	IP, PU, SA, KC, BT, Houston, HyRANK Dioni, and HyRANK Loukia	GPU Training: NVIDIA RTX 2070 Super GPU Test: NVIDIA Jetson TX2	Band selection onboard to save bandwidth/energy, transmitting only informative bands and maintain accuracy
Assudani et al., (2025)	Fusion: EfficientNet (RGB) + 1D-CNN (HSI) + ViT	Plant disease datasets (public + custom)	Jetson Nano	Accuracy, time, and optimized with pruning/distillation
Růžička and Markham (2025)	HyperVit, Adapted Transformer (SegFormer, EfficientViT) with spectral layers + upscaling	OxHyperSyntheticCH4, OxHyperRealCH4, OxHyperMinerals (from EMIT)	Jetson AGX Xavier, Raspberry Pi, Unibap SpaceCloud (ION-SCV proxy).	Methane detection vs prior DL, faster inference than baselines general-purpose foundation models for HSI onboard
Viel (2025)	Transformer-R vs. 1D-CNN, 2D-CNN, 3D-CNN, and LSTM	IP, PU, PC, SA, KC, BT	CPU (i9 and Cortex A78), NVIDIA RTX 4070, and NVIDIA Jetson Orin Nano	Transformer-R to show tradeoff among accuracy, time, power, GPU utilization, and memory demands

In conclusion, this study has enabled the answering of three research questions, specifically, the second and the last two, contributing to the definition and development of the proposed design and implementation of this work.

Upon analyzing the works after answering the research questions, we noted a gap in the exploration of Transformer-based architectures that exploit only spectral data, specifically 1D data, using an embedded GPU. Furthermore, there is a lack of exploration of parameter-reduced models that minimize memory, energy, and processing time while maintaining acceptable accuracy. This observation demonstrates the contribution of this thesis to the field's literature.

4 PROPOSAL OF AN ANN-BASED ARCHITECTURE FOR HARDWARE-ACCELERATED HSI CLASSIFICATION IN EMBEDDED DEVICES

To support the research questions and guide the design of the proposed architecture, a classification study of HSI in 1D, 2D, and 3D was conducted using CNN, LSTM, and Transformer architectures. The comparison that will be performed allows us to evaluate the different architectures raised in the literature review. This evaluation will support the choice of the target architecture to be accelerated in this work.. Furthermore, the Attention mechanism used in the Transformer architecture was integrated into the developed CNN and LSTM architectures. This integration of the Attention mechanism aimed to assess whether its use, applied in a simplified, isolated manner as a single layer, can improve the accuracy of CNN and LSTM architectures.

Based on the preliminary results from this software-level analysis, a ANN architecture will be proposed. This architecture will be implemented as a HA for spectral-level classification of HSI.

4.1 HSI CLASSIFICATION ANALYSIS: 1D, 2D, AND 3D

In this section, the classification performance of HSI is evaluated using LSTM, CNN, and Transformer architectures. These architectures are compared using metrics that assess classification performance, model storage requirements, and inference (prediction) time. For the CNN architecture, the evaluation compares the performance of 1D, 2D, and 3D CNN-based approaches. For the CNNs based on the methodology proposed by Roy et al. (2020), a data splitting strategy is also applied to avoid contamination among hypercubes used in 2D and 3D HSI classification.

Finally, a comparison is also made to evaluate the impact of adding layers with the Attention Mechanism to 1D-CNN and LSTM architectures.

This comparative analysis enables us to determine, using various metrics, whether architectures that perform classification solely at the spectral level (per pixel) can outperform or match those that handle classification at the spatial and spectral-spatial levels. This comparison is relevant because spectral-level architectures require less data structuring and storage, whereas spatial and spectral-spatial architectures are widely recognized for achieving higher classification accuracy.

4.1.1 Dataset

The benchmark datasets used are from the literature and can be found in (M Graña MA Veganzons, 2021). These datasets show aerial views from three different sensors: the ROSIS (Reflective Optics System Imaging Spectrometer), Hyperion on NASA Earth Observing One (EO-1), and NASA/JPL AVIRIS. Indian Pines, the University of Pavia, the Center of Pavia, the Kennedy Space Center, Botswana, and the

Salinas Valley served as HSIs that served as benchmarks and datasets. Images of the Kennedy Space Center, Salinas Valley, and Indian Pines were taken by the AVIRIS sensor. The scenes of the University of Pavia and the Center of Pavia were recorded by the Reflective Optics System Imaging Spectrometer (ROSIS) sensor. On the other hand, the Botswana scene was captured by the NASA EO-1 satellite's Hyperion sensor (M Graña MA Veganzons, 2021).

There are 224 spectral bands in the Kennedy Center (KC), Salinas Valley (SA), and Indian Pines (IP) scenes that span the electromagnetic spectrum from 400 to 2500 nm. Nevertheless, 24 bands are eliminated since they are located in areas where IP and SA absorb water. Following water absorption, 176 bands were used for the HSI KC analysis, and low SNR bands were removed. With a spatial dimension of 145×145 pixels, the IP scene is composed of one third forest or other perennial natural vegetation and two thirds agriculture. A railway line, two major dual-lane highways, some low-density housing, other built structures, more minor roads, and some corn and soybean crops in their early stages of growth are all present in this scene. This HSI contains a $145\times145\times224$ hypercube with 16 previously mapped classes. With a spatial resolution of 512×217 pixels and a hypercube of $512\times217\times224$, the SA HSI depicts the California region of the same name. This HSI contains 16 classes, including vineyards, bare soil, and vegetation. The 512×614 pixels of the KC scene include 13 classes of vegetation species' distinct spectral fingerprints, which correspond to the different types of land cover that exist in this ecosystem (M Graña MA Veganzons, 2021).

The Botswana dataset is represented by the Botswana (BT) scene, which has a spatial resolution of 1476×256 pixels and 242 bands that span the 400–2500 nm spectrum in 10 nm windows. Only 145 bands remain after preprocessing to reduce the effects of poor detectors, inter-detector miscalibration, and occasional anomalies. The different types of land cover in the distant seasonal wetlands, sporadic swamps, and dry forests are represented by their observations from 14 recognized classes. Lastly, the scenes from the University of Pavia (UP) and the Center of Pavia (CP) also depict the Italian region of Pavia. The hypercube of CP is $1096\times1096\times102$, and its spatial resolution is 1096×1096 pixels and 102 bands. With a hypercube of $610\times610\times103$, UP has a spatial resolution of 610×610 pixels and 103 spectral bands. These HSIs have already been mapped and categorized into nine classes, which range from bare soil to asphalt (M Graña MA Veganzons, 2021).

4.1.2 Materials for comparison between architectures

Python 3.8, along with NumPy, Keras, TensorFlow, Spectral, and Time libraries, was used to develop the machine learning models. Building machine learning models is made easy with Keras, a high-level, user-friendly Application Programming Interface (API) built on top of TensorFlow. TensorFlow, on the other hand, is a powerful library

devoted to DNN serving, inference, and training. CNN models use the NumPy library to generate tensors and manipulate arrays. The HSIs used as the dataset are manipulated using spectral analysis. Lastly, the execution time of the prediction can be captured using the Time library.

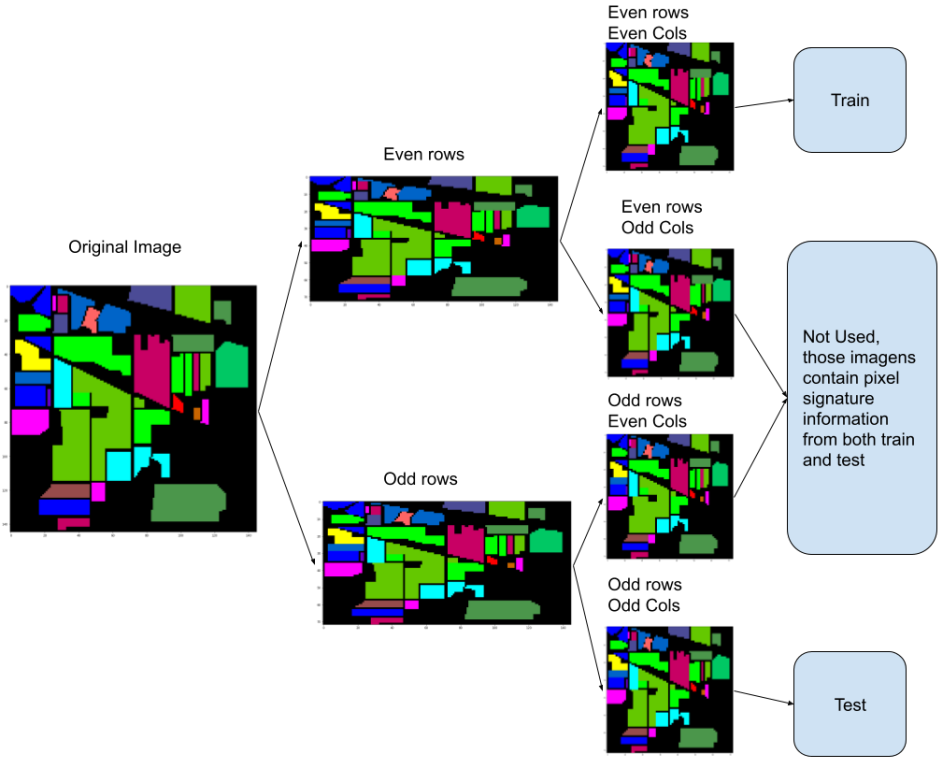
We trained, inferred, and evaluated using the Google Collaboratory Pro cloud service platform. Three configurations were employed: CPU + NVIDIA A100 GPU, CPU + NVIDIA Tesla T4 GPU, and CPU only. Three Intel Xeon processors operating at 2.2 GHz, 25 GB of main memory, and 225 GB of disk storage are available in the CPU-only setup. Three 2.2 GHz Intel Xeons, 25 GB of main memory, 166 GB of disk space, and an NVIDIA Tesla T4 GPU with 15 GB of memory are all included in the CPU+NVIDIA Tesla T4 GPU setup. Last but not least, the CPU+NVIDIA A100 GPU infrastructure provides a computational infrastructure with an Intel Xeon CPU running at 2.2 GHz and 11 processors, 83.78 GB of main memory, 166 GB of disk storage, and an NVIDIA A100-SXM4 offboard graphics card with 40 GB of memory. This setup enables high-performance experiments, mainly due to the use of a dedicated graphics card.

4.1.3 Methods for comparing architectures

Roy et al. (2020) served as a baseline for the CNN architectures. Additionally, a unique methodology for splitting the HSI dataset is suggested. The goal is to enable splits that do not contaminate training data with testing data, and vice versa. By avoiding overly optimistic model metrics, this splitting methodology seeks to improve classification performance and increase confidence in the precision results reported in Roy et al. (2020). The authors of Roy et al. (2020) employ a data split methodology for testing and training, which results in data leakage between the testing, validation, and training sets when constructing the spatial-spectral cubes. The macro-precision of the prediction in the test set on production in real-world applications is highly mistrusted due to this contamination.

The HSI is divided into even and odd rows using this suggested data division procedure, with the odd rows being used for testing and the even rows for training. It creates a column division following this row division. Following this division, the odd rows and columns are designated for testing, while the even rows and columns are designated for training. The suggested data division methodology is shown in Figure 18. It is crucial to remember that 1D-CNN classifies pixels, and this methodology is used to keep the three architectures comparable. But the scikit-learn library's *train_test_split* function was also utilized.

Figure 18 – Proposed dataset-splitting approach to avoid contamination between training and testing datasets.



Source: Viel et al. (2023).

Additionally, a transformation using the Principal Component Analysis (PCA) algorithm is applied to reduce the number of HSIs used, thereby achieving effective training and classification while minimizing memory requirements. The fundamental concept of PCA is to reduce the original data into a lower-dimensional set of orthogonal (i.e., uncorrelated) variables known as Principal Component (PC), which are then ranked according to how important they are in explaining the variance in the data. The direction in the data that captures the most variance is the first principal component (PC1), followed by the direction that captures the second-greatest variance (PC2), and so on. The training and testing sets are subjected to the PCA transformation (Farrell; Mersereau, 2005). Class balancing was used exclusively for the 1D-CNN on the Indian Pines image, as described in (Roy et al., 2020). Because the IP HSI has classes with nearly 2,000 samples and others with fewer than 10 during training, the class-balancing was designed to ensure equal training across the HSI classes. Furthermore, to observe the prediction results without jeopardizing the testing set, a validation set was used to train the architectures.

Acurácia Geral (OA), Average Accuracy (AA), Cohen Kappa Score (kappa), prediction time, and model memory usage were the metrics used to extract results. The results were extracted using the Python libraries scikit-learn (model accuracy),

Keras (model size in bytes), and Time (prediction time). Out of all the samples, the OA will calculate the proportion of correctly classified samples. A statistical measurement metric called kappa gives information about how well the classification map and the ground truth map agree. The classwise classification accuracy mean is denoted by AA. Each metric is defined by the following equations:

$$OA = \frac{tp + tn}{tp + tn + fp + fn} \tag{29}$$

$$AA = \frac{\sum_{i=1}^{n_{\text{classes}}} AccuaracyClass_i}{n_{\text{classes}}} \tag{30}$$

$$Kappa = \frac{p_o - p_e}{1 - p_e} \tag{31}$$

where *tp* are true positives, *tn* are true negatives, *fp* are false positives, *fn* are false negatives, $n_classes$ indicates the number of classes present in the HSI, $AccuracyClass_i$ indicates the individual accuracy of the class. In Kappa, p_o is the observed relative agreement between raters (identical to precision), and p_e is the hypothetical probability of chance agreement, using the observed data to calculate the probability that each observer randomly sees each category.

4.2 CNN ARCHITECTURE

Convolution is based on the fundamental principle of taking a small matrix of numbers (referred to as a "kernel" or "filter"), summing the results, and performing an element-wise multiplication between the input data and the kernel's entries. A fresh feature map matrix is produced by repeating this procedure for every kernel position. The term "convolutional operation" refers to the method of performing element-wise multiplication and summing by sliding the kernel over the input data. A CNN's convolutional layer is where a kernel is applied to an input matrix. Usually, a CNN has several convolutional layers. To add nonlinearity to the model, a nonlinear activation function (e.g., a ReLU) is applied after each layer. Before the final output layer that generates predictions, the convolutional layers' outputs are frequently passed through one or more fully connected layers, also referred to as "dense layers" (Goodfellow; Bengio; Courville, 2016).

We used the model described in Roy et al. (2020) as a guide for implementing the 1D-, 2D-, and 3D-CNN architectures. To implement the suggested architecture for this work, the CNNs were modified. We employed dense layers and 1D convolutional layers for the 1D-CNN. Next, we implemented the 2D-CNN using dense layers and both 1D and 2D convolutional layers. Lastly, we constructed the 3D-CNN using the same architecture as in (Roy et al., 2020).

In addition, all CNNs have the Flatten layers to vectorize the data and Dropout layers to prevent overfitting by randomly setting a fraction of the activations in the network to zero during training.

All architectures were subjected to the Adam optimizer with a learning rate ranging from 0.001 to 0.0001. The first moment, which is an exponentially weighted moving average of the gradients, and the second moment, which is an exponentially weighted moving average of the squares of the gradients, are the two moving averages that the Adam optimizer employs to calculate the adaptive learning rates (Bock; Goppold; Weiß, 2018; Bock; Weiß, 2019). The following is how the adaptive learning rates are calculated using these moving averages:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t \tag{32a}$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \tag{32b}$$

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t} \tag{32c}$$

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t} \tag{32d}$$

$$w_t = w_{t-1} - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \varepsilon} \tag{32e}$$

where m_t is the first moment at time t, v_t is the second moment at time t, g_t is the gradient at time t, w_t is the weight at time t, α is the learning rate, β_1 and β_2 are the exponential decay rates for the first and second moments, respectively, and epsilon is a small constant used to prevent division by zero.

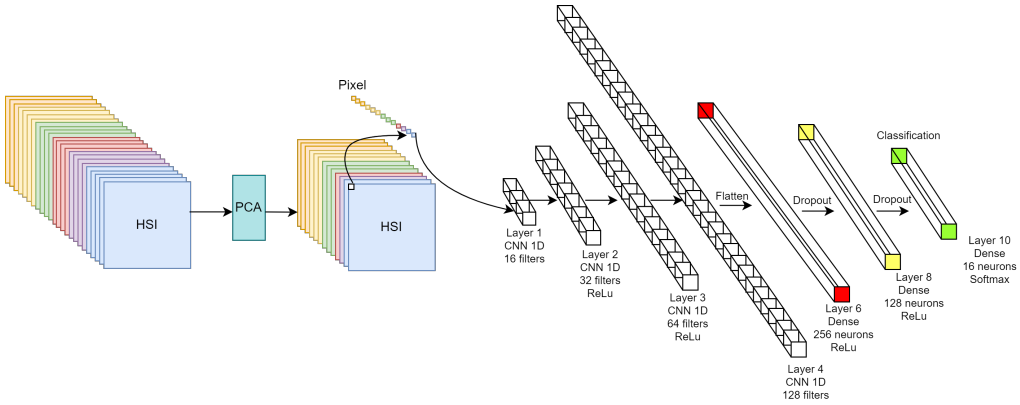
Below, the architectures for HSI classification are described in more detail.

4.2.1 1D-CNN

1D-CNN is designed to perform a pixel-wise classification, focusing solely on the spectral characteristics of the HSIs. After using PCA, we developed an architecture better suited to the two types of images. This strategy results from the IP HSI using a different number of PCs than the SA and UP HSIs. For the IP and KC scenes, we used 30 components; for the SA, UP, CP, and BT scenes, we used 15. The transformation using the PCA algorithm produced the PCs in both situations. The kernel sizes used in the convolutional layers vary with the number of components, but the number of layers remains constant. For example, the kernels used in the Conv 1D layers (1, 2, 3, and 4) for the IP HSI were 9, 7, 5, and 2, respectively, while for the SA, UP, CP, KC, and BT HSIs were 7, 5, 3, and 3, respectively.

Figure 19 illustrates the architecture of the developed 1D-CNN. Tables 5 and 6 describe the characteristics of the layers of the models created, respectively, for the IP scene and the SA, UP, CP, KC, and BT scenes. In addition, two methodologies for dividing training and testing data were used, as mentioned in Section 4.1.3.

Figure 19 – 1D-CNN architecture based in Roy et al. (2020). We named this architecture 1D-CNN-O (-O stands for original architecture).



Source: Viel et al. (2023).

Table 5 – 1D-CNN for the IP and KC scenes.

Layer	**Output Shape**	**Number of Parameters**
Input	(1,30,1)	0
Conv1d_1	(1,22,16)	160
Conv1d_2	(1,16,32)	3,616
Conv1d_3	(1,12,64)	103,304
Conv1d_4	(1,11,128)	16,512
Flatten	(1024)	0
Dense_6	(256)	360,704
Dropout	(256)	0
Dense_8	(128)	32,896
Dropout	(128)	0
Dense_10	(16)	2,064

Later, the Attention layer and MaxPooling layers were added to the suggested 1D-CNN architecture. Furthermore, to determine whether accuracy can be achieved with fewer storage resources, the 1D-CNN-R and 1D-CNN-RA architectures investigate the use of fewer resources, specifically parameters. The R in 1D-CNN-R stands for Reduced. RA in 1D-CNN-RA stands for Reduced with Attention. The only distinction between 1D-CNN-RA and 1D-CNN-R is the absence of the Attention layer. The features of the developed architectures are displayed in Tables 7 and 8.

It is important to note that the reduced architectures reduce the number of components to 30 following the PCA. The purpose of adding the layers is to observe the effects, both positive and negative, primarily caused by the addition of the Attention layer. Instead of employing a fixed weighting or treating every component of the input equally, the attention mechanism in a neural network, for instance, enables the model to selectively focus on specific portions of the input data during processing (Bahdanau; Cho; Bengio, 2014; Luong; Pham; Manning, 2015).

Table 6 – 1D-CNN for the SA, PU, CP, and BT. The values of the last layer are 2064 (16 classes) for the SA, 1161 (9 classes) for UP and CP, 1677 (13 classes) for the KC, and 1806 (14 classes) for the BT. Dense 6-layer is 524,544 for the KC scene and 33,024 for others.

Layer	**Output Shape**	**Number of Parameters**
Input	(1,15,1)	0
Conv1d_1	(1,9,16)	128
Conv1d_2	(1,5,32)	2,592
Conv1d_3	(1,3,64)	6,208
Conv1d_4	(1,1,128)	24,704
Flatten	(128)	0
Dense_6	(256)	33,024/524,544
Dropout	(256)	0
Dense_8	(128)	32,896
Dropout	(128)	0
Dense_10	(16)/(9)/(13)/(14)	2,064/1,161/1,677/1,806

Table 7 – Reduced 1D-CNN without Attention layer.

Layer	**Output Shape**	**Number of Parameters**
Input	(1,30,1)	0
Conv1d_1	(16,120)	1,920
Conv1d_2	(16,104)	12,584
Conv1d_3	(14,24)	7,512
MaxPolling1d_4	(7,24)	0
Conv1d_5	(6,16)	784
MaxPolling1d_6	(3,16)	0
Attention_7	(32)	1,280
Flatten	(48)	0
Dense_8	(96)	3,168
Dropout	(96)	0
Dense_9	(96)	9,312
Dropout	(96)	0
Dense_10	(16)	1,552

This feature is helpful when working with lengthy data sequences, like sentences in natural language, where some words or phrases may be more pertinent to the task at hand than others (Bahdanau; Cho; Bengio, 2014; Luong; Pham; Manning, 2015). The kernels used in the Conv 1D layers 1, 2, 3, and 5 for the IP HSI were 15, 1, 3, and 2, respectively. The choice of hyperparameters, kernel sizes, numbers of filters for the convolutional layers, the number of units for the Attention layer, the number of neurons for the Dense layers, and the learning rate used were determined via Hyperband in the keras_tuner library.

A set of "attention weights" that represent the relative importance of each component of the input to the model's output is introduced as part of the attention mechanism's operation. You can think of these weights as a collection of coefficients that are applied

Table 8 – Reduced 1D-CNN with Attention layer.

Layer	Output Shape	Number of Parameters
Input	(1,30,1)	0
Conv1d_1	(16,120)	1,920
Conv1d_2	(16,104)	12,584
Conv1d_3	(14,24)	7,512
MaxPolling1d_4	(7,24)	0
Conv1d_5	(6,16)	784
MaxPolling1d_6	(3,16)	0
Flatten	(48)	0
Dense_7	(96)	4,704
Dropout	(96)	0
Dense_8	(96)	9,312
Dropout	(96)	0
Dense_9	(16)	1,552

to the input data before it passes through the rest of the network. They are learned during training. A neural network can incorporate attention mechanisms in various ways. Using a distinct "attention layer" that evaluates the input data and determines attention weights based on a relevance or importance metric is a popular strategy. The input data is then weighted prior to its passage through the remainder of the network (Luong; Pham; Manning, 2015; Bahdanau; Cho; Bengio, 2014).

This process involves calculating the dot product between the query vector and each key vector in the input data, as:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{33}$$

where, Q is the query vector, K is the set of key vectors, and V is the set of value vectors. The dot product between the query vector and each key vector is normalized by the square root of the key vector dimension, d_k. The resulting dot products are then passed through a softmax function to produce a set of weights, which are used to weight the value vectors. The weighted sum of the value vectors is returned as the attention mechanism's output.

4.2.2 2D-CNN

The architecture of the model proposed in (Roy et al., 2020) serves as the basis for the 2D-CNN developed in this work. The model, in contrast to that work, consists of three dense layers, one 1D layer, and three 2D convolutional layers. The model is more than just 2D thanks to this architecture. This design enables exploration of the spatial properties of the HSI while reducing the number of parameters. For the same number of convolutional layers, a purely 2D model produced a significant number of parameters (75 times more). The dense layers have the same number of units as the proposal in

(Roy et al., 2020), which is the same architecture used in the 3D approach. In addition, we used a 19×19 spatial filtering window, which is smaller than the one applied in the reference work (i.e., 25×25).

Tables 9 and 10 provide a more detailed description of the architectures for the IP scene and the SA and UP scenes, which also differ in the number of components generated by the PCA algorithm.

Unlike the methodology used in (Roy et al., 2020), the division of training and testing data was done using the methodology proposed in this work (described in Section 4.1.3).

Table 9 – 2D-CNN for the IP scene.

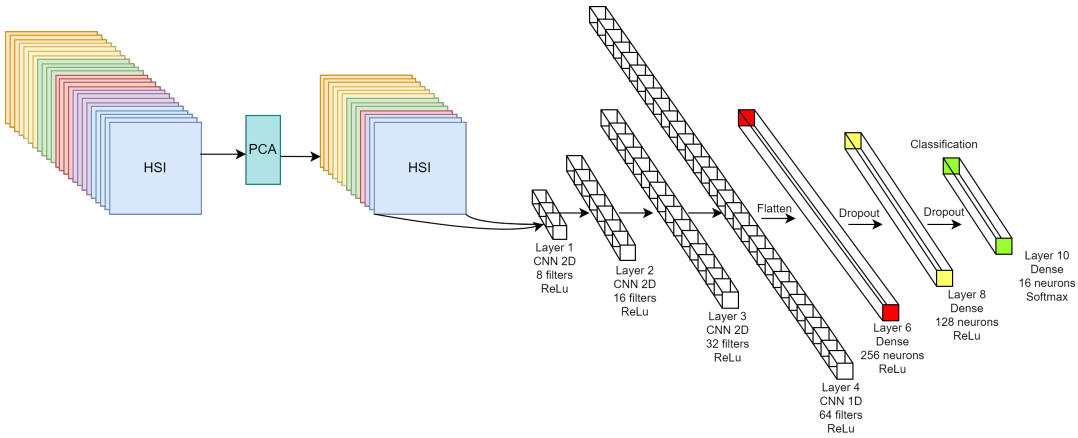
Layer	**Output Shape**	**Number of Parameters**
Input	(19,19,30,1)	0
Conv2d_1	(19,13,24,8)	400
Conv2d_2	(19,9,20,16)	3216
Conv2d_3	(19,7,18,32)	4640
Reshape	(19,7,576)	0
Conv1d_4	(19,5,64)	110656
Flatten	(6080)	0
Dense_5	(256)	1556736
Dropout_6	(256)	0
Dense_7	(128)	32896
Dropout_8	(128)	0
Dense_9	(16)	2064

Table 10 – 2D-CNN for the SA, PU, CP, and BT scenes. The values of the last layer are 2064 (16 classes) for the SA scene, 1161 (9 classes) for the UP and CP scenes, 1677 (13 classes) for the KC scene, and 1806 (14 classes) for the BT scene.

Layer	**Output Shape**	**Number of Parameters**
Input	(19,19,15,1)	0
Conv2d_1	(19,13,9,8)	400
Conv2d_2	(19,9,5,16)	3216
Conv2d_3	(19,7,3,32)	4640
Reshape	(19,7,96)	0
Conv1d_4	(19,5,64)	18496
Flatten	(6080)	0
Dense_5	(256)	1556736
Dropout_6	(256)	0
Dense_7	(128)	32896
Dropout_8	(128)	0
Dense_9	(16)/(9)/(13)/(14)	2,064/1,161/1,677/1,806

Figure 20 illustrates the developed 2D-CNN architecture designed to work primarily with spatial characteristics.

Figure 20 – 2D-CNN architecture based on Roy et al. (2020).

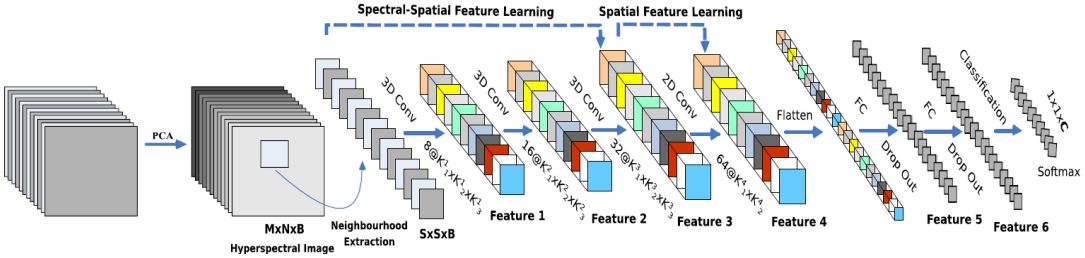


Source: Viel et al. (2023).

4.2.3 3D-CNN

The 3D architectures described in this work were the 3D-CNN, which was proposed and developed in (Roy et al., 2020). In contrast to the reference architecture, the authors' suggested methodology (described in Section 2) was used to split the training and test data. Once more, this methodology keeps the training and testing datasets clean. We employed a smaller spatial filtering window than the one used in the reference work, as previously mentioned. The architecture suggested by Roy et al. (2020) was illustrated by

Figure 21 – HybridSN proposed by Roy et al. (2020).



Source: Roy et al. (2020).

Tables 11 and 12 provide a more detailed description of the architectures for the IP scene for the SA and UP scenes, respectively, which also differ in the number of components generated by the PCA algorithm. Unlike 1D-CNN, 30 components are used for the IP scene.

4.3 LSTM ARCHITECTURE

As in 1D-CNN, we design two LSTM architectures to apply HSI classification at the pixel level. The architecture of the suggested LSTM with Attention Layer (LSTM-A) is shown in Figure 22.

Table 11 – 3D-CNN for the IP scene.

Layer	Output Shape	Number of Parameters
Input	(19,19,30,1)	0
Conv3d_1	(17,17,24,18)	512
Conv3d_2	(15,15,20,16)	5776
Conv3d_3	(13,13,18,32)	13856
Reshape_3	(13,13,576)	0
Conv2d_4	(11,11,64)	331840
Flatten	(7744)	0
Dense_6	(256)	1982720
Dropout	(256)	0
Dense_8	(128)	32896
Dropout	(128)	0
Dense_10	(16)	2064

Table 12 – 3D-CNN for the SA, PU, CP, and BT scenes. The values of the last layer are 2064 (16 classes) for the SA scene, 1161 (9 classes) for the UP and CP scenes, 1677 (13 classes) for the KC scene, and 1806 (14 classes) for the BT scene.

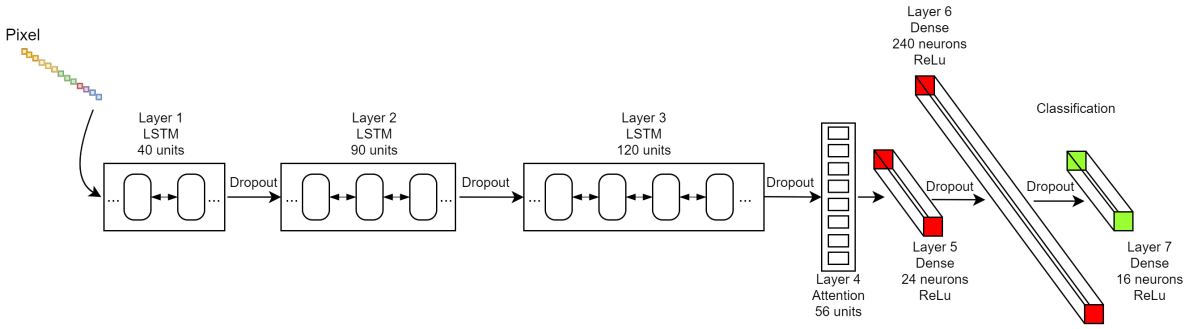
Layer	Output Shape	Number of Parameters
Input	(19,19,15,1)	0
Conv3d_1	(17,17,24,18)	512
Conv3d_2	(15,15,20,16)	5776
Conv3d_3	(13,13,18,32)	13856
Reshape_3	(13,13,576)	0
Conv2d_4	(11,11,64)	55360
Flatten	(7744)	0
Dense_6	(256)	1982720
Dropout	(256)	0
Dense_8	(128)	32896
Dropout	(128)	0
Dense_10	(16)/(9)/(13)/(14)	2,064/1,161/1,677/1,806

A more thorough explanation of the LSTM architectures can be found in Tables 13 and 14. The suggested LSTMs use 30 components in PCA for the three HSI images used in this work, just as in the 1D-CNN architecture proposed, with and without the Attention layer. Using Hyperband from the keras_tunner library, the hyperparameters kernel size, number of convolutional layer filters, number of Attention layer units, number of Dense layer neurons, number of LSTM units, and learning rate were determined.

4.4 TRANSFORMER ARCHITECTURE

The implementation by Ntakouris (2021) served as the foundation for the Transformer architecture that was employed. This architecture illustrates how to use Trans-

Figure 22 – LSTM Architecture with Attention Layer.



Source: Viel et al. (2023).

Table 13 – LSTM without Attention layer.

Layer	Output Shape	Number of Parameters
Input	(1,30,1)	0
LSTM_1	(30,40)	6720
Dropout	(30,40)	0
LSTM_2	(30,90)	47160
Dropout	(30,90)	0
LSTM_3	(120)	101280
Dropout	(120)	0
Dense_4	(24)	2904
Dropout	(24)	0
Dense_5	(240)	6000
Dropout	(240)	0
Dense_6	(16)	3856

formers for time-series classification. The architecture's input was changed to accommodate HSI classification at the pixel level, as 1D-CNN does. Both the IP HSI and the SA and UP HSIs in the model use a pixel with 30 PCs. The Transformer architecture used for the experiments is shown in Figure 23.

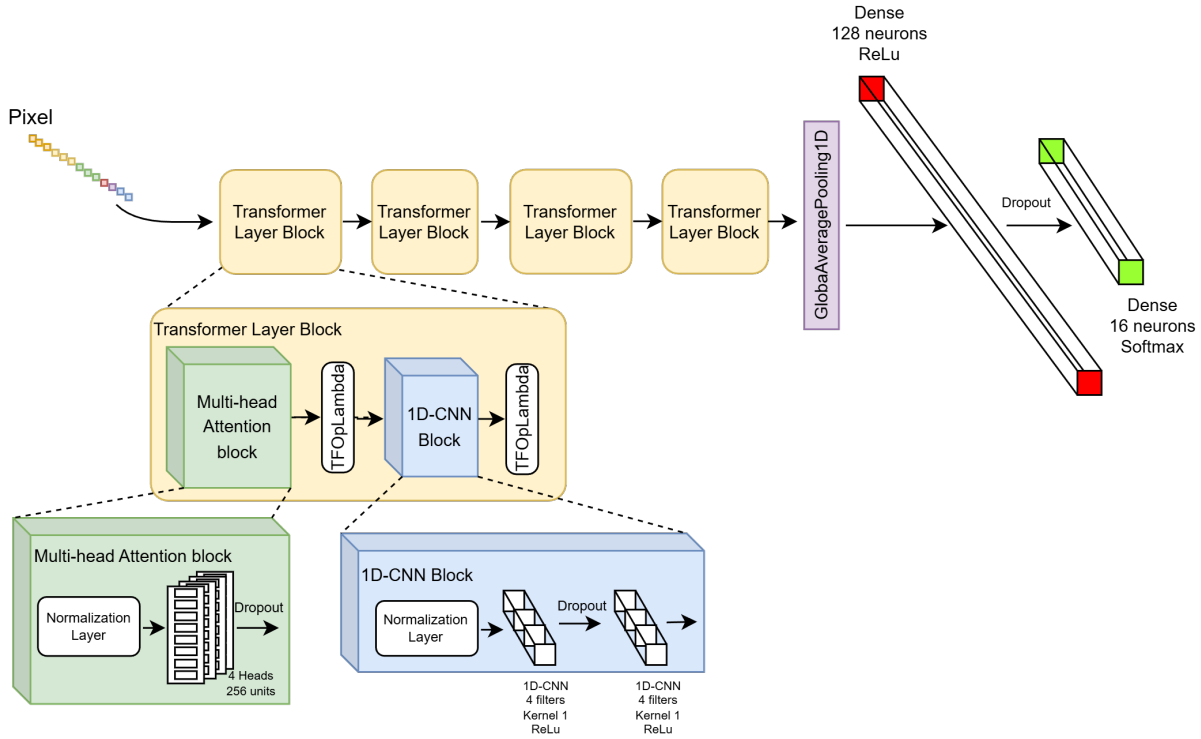
The architecture consists of four blocks, each named a Transformer Layer. Internally, each block is divided into a Multi-head Attention block and a 1D-CNN block. Multi-head Attention blocks comprise four *heads* of 128 units and a normalization layer. The 1D-CNN block comprises two convolutional layers with 4 filters and a kernel size of 1, along with a normalization layer. Between the blocks is the TOFpLambda layer, also called the Lambda layer, which serves as an adapter for sequential data between layers and is inserted by the TensorFlow library.

A more thorough explanation of the Transformer architectures can be found in Table 15.

Table 14 – LSTM with Attention layer.

Layer	Output Shape	Number of Parameters
Input	(1,30,1)	0
LSTM_1	(30,40)	6720
Dropout	(30,40)	0
LSTM_2	(30,90)	47160
Dropout	(30,90)	0
LSTM_3	(30,120)	101280
Dropout	(30,120)	0
Attention	(56)	27840
Dense_4	(24)	1368
Dropout	(24)	0
Dense_5	(240)	6000
Dropout	(240)	0
Dense_6	(16)	3856

Figure 23 – Transformer Architecture proposed in Ntakouris (2021).



Source: Viel et al. (2023).

4.5 EXPERIMENT RESULTS

As a result, the performance metrics for the proposed architectures were based on prediction time, precision, and size (in bytes).

4.5.1 Classification Performance

Table 16 illustrates the results obtained in prediction (classification) with the testing data set. This table presents the results for the six HSIs used with the proposed

Table 15 – Transformer layers. Layers 2 to 9 are repeated 4 times.

Layer	**Output Shape**	**Number of Parameters**
Input	(1,30,1)	0
LayerNormalization	(30,1)	2
Multi-head Attention	(30,1)	7,169
Dropout	(30,90)	0
Lambda	(30,120)	0
LayerNormalization	(30,120)	2
Conv1D	(30,4)	8
Dropout	(30,4)	0
Conv1D	(30,1)	5
Lambda	(30,1)	0
–	–	–
GlobalAveragePooling	(30)	0
Dense	(128)	3,968
Dense	(16)	2,064

division methodology and the size of the data sets used for training and testing.

Table 16 – Classification performance of the three CNN architectures using the new dataset-splitting methodology.

HSI	**CNN Arch.**	**OA** (%)	**AA** (%)	**Kappa** ($\times$100)	**Training Dataset**	**Testing Dataset**
IP	1D	74.23	66.69	70.72	2.560	2.569
	2D	97.00	93.05	96.58	2.560	2.569
	3D	97.58	98.45	97.25	2.560	2.569
SA	1D	94.21	97.45	93.55	13.580	13.490
	2D	99.92	99.97	99.91	13.580	13.490
	3D	100.00	100.00	100.00	13.580	13.490
UP	1D	93.20	90.92	90.98	10.647	10.729
	2D	99.35	99.30	99.14	10.647	10.729
	3D	99.92	99.91	99.90	10.647	10.729
CP	1D	99.08	97.14	98.69	37,109	36,959
	2D	99.94	99.81	99.92	37109	36959
	3D	99.71	98.70	99.59	37109	36959
KC	1D	92.64	90.10	91.81	1,308	1,305
	2D	91.03	87.19	90.01	1308	1305
	3D	88.73	82.29	87.42	1308	1305
BT	1D	93.91	92.97	93.51	817	802
	2D	99.87	99.88	99.86	817	802
	3D	99.25	99.12	99.18	817	802

The process of dividing the HSIs used in the tests was also evaluated for division using the proposed methodology and the *train_test_split* function from the scikit-learn library. Table 17 presents the results obtained.

Table 18 presents the accuracy difference only of the architectures that process the spectral information, comparing the 1D-CNN based on (Roy et al., 2020), the proposed CNN without and with Attention, and the LSTM architecture with and without the Attention layer.

Table 17 – Accuracy of the two dataset-splitting methodologies for 1D-CNN.

HSI	**Divison Method**	**OA** (%)	**AA** (%)	**Kappa** ($\times$100)	**Train. Size**	**Test. Size**
IP	Proposed	75.36	70.45	72.01	2.560	2.569
	scikit-learn	83.17	84.96	80.78	8.199	2.050
SA	Proposed	94.21	97.45	93.55	13.580	13.490
	scikit-learn	96.24	98.19	95.81	43.303	10.826
UP	Proposed	93.20	90.92	90.98	10.647	10.729
	scikit-learn	95.17	93.61	93.59	34.220	8.556
CP	Proposed	99.08	97.14	98.69	37,109	36,959
	scikit-learn	99.16	97.38	98.82	118,521	29,631
KC	Proposed	92.64	90.10	91.81	1,308	1,305
	scikit-learn	96.06	94.10	95.62	4,168	1,043
BT	Proposed	93.91	92.97	93.51	817	802
	scikit-learn	96.92	97.26	96.66	2,598	650

It is observed that the proposed CNN and Attention architectures present adequate results with fewer parameters. However, they do not affect the classification performance with the three metrics, showing a general result worse than their use. However, the architectures require a large number and variety of training samples, as observed in the SA and UP datasets.

4.5.2 Memory Demand

Model storage is a feature that must be considered for computational systems with limited memory available. This requirement is essential for any application to run on these platforms. Therefore, the amount of memory (space) used by the model, in bytes (excluding the dataset), was also measured. To provide a more accurate comparison, the number of parameters used by each architecture in each dataset was also obtained.

Table 19 presents the memory required by the three CNN architectures and by the three datasets based on (Roy et al., 2020).

There is a difference in batch size between the architectures: 1D-CNN uses 32, and 2D-CNN and 3D-CNN use 256. This difference was used based on the size of the training samples.

The comparison of required memory and the number of parameters has also been extended to 1D architectures, as shown in Table 20. It is observed that the memory demands of the 1D-CNN-R and -RA architectures, in joint analysis with classification performance results, show that these architectures offer a good trade-off. This feature is essential for use in the final application, subject to the constraints imposed by computational resources, such as embedded systems. By default, we use the split method in scikit-learn because it provides more samples, and 1D architectures do not suffer from the problem of separating training and test samples, unlike 2D and 3D architectures.

Table 18 – Performance of the 1D architectures.

HSI	Architecture	OA (%)	AA (%)	Kappa (×100)
IP	1D-CNN-O	83.17	84.96	80.78
	1D-CNN-R	77.34	73.97	74.03
	LSTM	75.70	73.08	72.28
	1D-CNN-RA	73.49	71.36	69.80
	LSTM-A	75.48	76.73	72.22
	Transformer	82.56	79.13	80.00
SA	1D-CNN-O	96.24	98.19	95.81
	1D-CNN-R	95.04	97.47	94.47
	LSTM	94.76	97.59	94.17
	1D-CNN-RA	94.39	96.95	93.75
	LSTM-A	94.77	97.48	94.16
	Transformer	96.23	98.22	95.80
UP	1D-CNN-O	95.17	93.61	93.59
	1D-CNN-R	92.16	89.94	89.63
	LSTM	93.99	92.39	92.02
	1D-CNN-RA	89.28	86.34	85.75
	LSTM-A	94.43	91.81	92.59
	Transformer	95.10	93.05	93.50
CP	1D-CNN-O	99.16	97.38	98.82
	1D-CNN-R	98.92	96.85	98.47
	LSTM	99.13	9747	9877
	1D-CNN-RA	98.41	94.95	97.75
	LSTM-A	99.04	97.15	98.65
	Transformer	99.20	97.34	98.87
KC	1D-CNN-O	96.06	94.10	95.62
	1D-CNN-R	94.11	91.43	93.45
	LSTM	91.24	87.11	90.24
	1D-CNN-RA	94.56	91.83	93.94
	LSTM-A	91.56	88.82	90.61
	Transformer	92.45	88.32	91.59
BT	1D-CNN-O	96.92	97.26	96.66
	1D-CNN-R	91.89	91.64	91.21
	LSTM	95.28	95.92	94.89
	1D-CNN-RA	75.38	75.46	73.32
	LSTM-A	95.79	96.43	95.44
	Transformer	97.12	97.23	96.88

4.5.3 Time for Inference

The impact of classifying HSIs using 1D, 2D, and 3D data formats was observed by measuring the prediction time of CNNs with various architectures at the testing dataset and sample levels. This method enables us to determine which architecture, such as a CPU or a SoC with low-power cores, best fits when processing time is a constraint. The results of predicting the various architectures using the NVIDIA A100 GPU, NVIDIA Tesla T4 GPU, and CPU-only infrastructures are shown in Table 21. For 1D, 2D, and 3D CNNs, the number of samples used for testing and training is displayed in Table 16; for 1D architectures, it is displayed in Table 17. Scikit-learn uses the size in Table 17 for 1D architectures.

This comparison of time results across various GPUs shows how the computing system and communication resource usage affect runtime. As can be observed, the communication latency of an A100 GPU renders a 1D or 3D sample inconsequential relative to the bandwidth available between the CPU and GPU. But when we use the Tesla T4 in both CPU-only and CPU+GPU modes, the sample size is already large enough. Additionally, we can observe that the acceleration is significantly higher when

Table 19 – Number of parameters and memory required by the three CNN architectures.

HSI	CNN Architecture	Size (bytes)	Number of Parameters
IP	1D	1,095,952	426,256
	2D	287,543,824	1,710,608
	3D	360,030,336	2,369,664
SA	1D	298,224	100,616
	2D	78,786,064	1,618,448
	3D	97,266,816	2,093,184
UP	1D	296,425	100,713
	2D	78,777,993	1,617,545
	3D	97,258,745	2,092,281
CP	1D	296,425	100,713
	2D	78,777,993	1,617,545
	3D	97,258,745	2,092,281
KC	1D	297,453	101,229
	2D	78,782,605	1,618,061
	3D	97,263,357	2,092,797
BT	1D	297,710	101,358
	2D	78,783,758	1,617,545
	3D	97,264,510	2,092,926

processing files with the CPU and more than two$\times$ with the T4 Tesla GPU if we divide the total sample time by the total time to process the dataset by the number of samples. Therefore, even with the A100 GPU, we will obtain a result similar to that of the Tesla T4 GPU if we calculate the processing time per sample as the dataset's processing time divided by the number of samples. The calculated processing time came to mitigate this effect and evaluate how much the inference costs (in the processor that will make it – CPU or GPU). We approximated the processor time for the entire dataset by dividing it by the number of samples. This operation minimizes the impact of latency caused by the above factors and allows a closer look beyond the processed inference.

Table 22 expands the processing time results to sort the complete and sample-only dataset. The presented results compare different classification architectures at the pixel level, using only spectral information. The low time required is a consequence of the low number of parameters and memory, and the lack of data structuring, as in the 2D-CNN and 3D-CNN architectures.

4.5.4 Discussion of the Results of Analysis of the Architectures

When comparing only the 1D architectures, we observe that the processing performance remains relatively consistent. The variation observed during full-dataset inference is primarily due to differences in the number of samples to be processed. Notably, regardless of the 1D architecture, in the case CNN with or without the Attention layer, LSTM with or without the Attention layer, or Transformer, the per-sample processing time remains in the range of 40 to 90 milliseconds. This consistency is likely influenced by the processing infrastructure offered by Google Colaboratory, which tends

Table 20 – Number of parameters and consumed memory of the 1D classification architectures.

HSI	CNN Architecture	Size (bytes)	Number of Parameters
IP	1D-CNN-O	1,095,952	426,256
	1D-CNN-R	637,408	38,368
	LSTM	1,266,672	167,920
	1D-CNN-RA	639,200	38,112
	LSTM-A	2,191,024	194,224
	Transformer	307,672	34,776
SA	1D-CNN-O	298,224	101,616
	1D-CNN-R	637,408	38,368
	LSTM	1,266,672	167,920
	1D-CNN-RA	639,200	38,112
	LSTM-A	2,191,024	194,224
	Transformer	307,672	34,776
UP	1D-CNN-O	296,425	100,713
	1D-CNN-R	635,833	37,689
	LSTM	1,264,089	166,233
	1D-CNN-RA	637,625	37,433
	LSTM-A	2,188,441	192,537
	Transformer	305,873	33,873
CP	1D-CNN-O	296,425	100,713
	1D-CNN-R	635,833	37,689
	LSTM	1,264,089	166,233
	1D-CNN-RA	637,625	37,433
	LSTM-A	2,188,441	192,537
	Transformer	275,153	33,873
KC	1D-CNN-O	297,453	101,229
	1D-CNN-R	636,733	38,077
	LSTM	1,265,565	167,197
	1D-CNN-RA	638,525	37,821
	LSTM-A	2,189,917	193,501
	Transformer	276,181	34,389
BT	1D-CNN-O	297,710	101,358
	1D-CNN-R	636,958	38,174
	LSTM	1,265,934	167,438
	1D-CNN-RA	638,750	37,918
	LSTM-A	2,190,286	193,742
	Transformer	276,438	34,518

to standardize execution times regardless of the number of layers.

However, it is important to note that memory consumption showed greater variation, which may affect deployment on computing platforms with different resource constraints than those used in this study. When analyzing processing time using only the CPU and the CPU combined with the NVIDIA Tesla T4 GPU, we observe that 1D gls CNNs provide nearly 100$\times$ acceleration over 2D and 3D CNNs in CPU-only scenarios, and more than 2$\times$ when using the CPU+GPU setup. This acceleration becomes noticeable only when the impact of data communication overhead is minimized. To achieve this, we divided the total dataset inference time by the number of samples, isolating the per-sample processing time.

In scenarios that rely on pixel-level sampling, the 1D-CNN architecture offers a

Table 21 – Processing time for the three CNN architectures running on different infrastructures and using the new dataset-splitting methodology.

HSI	CNN Arch.	Test dataset size	Time all dataset A100 (s)	Time 1 sample capt. A100 (ms)	Time 1 sample capt. A100 (ms)	Time all dataset T4 (s)	Time 1 sample T4 capt. (ms)	Time 1 sample T4 calc. (ms)	Time all dataset CPU (s)	Time 1 sample CPU capt. (ms)	Time 1 sample CPU calc. (ms)
IP	1D	2.569	0.62	87.48	0.24	0.3	193.1	0.12	0.6	55.41	0.24
	2D	2.569	0.77	87.57	0.30	0.7	238.3	0.29	9.3	55.26	3.63
	3D	2.569	0.54	84.78	0.21	0.7	242.6	0.30	32.3	61.29	12.58
SA	1D	13.490	1.66	81.59	0.12	1.1	191.0	0.08	1.2	60.95	0.09
	2D	13.490	2.21	92.15	0.16	1.7	229.5	0.12	15.3	46.13	1.13
	3D	13.490	1.85	87.71	0.14	1.6	204.3	0.12	46.8	49.98	3.47
UP	1D	10.729	1.33	65.83	0.12	0.9	211.4	0.08	0.9	54.43	0.09
	2D	10.729	1.75	68.86	0.16	1.3	173.6	0.12	12.3	46.56	1.15
	3D	10.729	1.47	69.83	0.14	1.3	130.0	0.12	36.8	50.04	3.43
CP	1D	36,959	3.51	78.86	0.09	2.9	87.95	0.08	3.8	56.72	0.10
	2D	36,959	5.56	63.89	0.34	6.8	75.27	0.20	44.3	54.41	1.43
	3D	36,959	4.35	51.66	0.11	5.6	63.95	0.15	128.3	63.03	3.47
KC	1D	1,305	0.43	63.47	0.33	0.3	47.85	0.25	0.4	55.97	0.28
	2D	1,305	0.40	59.08	0.35	0.4	60.05	0.22	1.7	52.62	1.46
	3D	1,305	0.24	74.47	0.18	0.2	57.66	0.17	4.6	63.49	3.52
BT	1D	802	0.37	59.95	0.46	0.3	46.41	0.43	0.3	57.03	0.43
	2D	802	0.37	60.79	0.47	0.3	57.67	0.44	1.2	57.14	1.57
	3D	802	0.15	55.81	0.18	0.2	54.39	0.24	2.8	71.63	3.54

clear advantage. It enables classification without requiring storage of multiple samples or a spatial window (e.g., 19$\times$19), allowing classification to occur immediately after the sensor acquires each pixel. Additionally, it is worth noting the limited control over the underlying hardware in Google Colaboratory's SaaS model, where the computational configuration is not explicitly known and reported times are approximate estimates.

Regarding classification accuracy, 1D-CNN models perform comparably to 2D and 3D-CNNs when the training dataset is well-balanced and contains a sufficient number of samples. This behavior is particularly observed in the IP scene, where poor results are obtained regardless of the data-split methodology, due to its highly imbalanced class distribution. In contrast, for the SA, UP, CP, KC, and BT scenes, 1D-CNNs do not handle the class imbalance as effectively as 2D and 3D CNNs. Therefore, in HSI datasets with significant class imbalance, often due to the sensing region's characteristics, classification should preferably be performed using 2D or 3D CNNs. This recommendation stems from the greater capacity of these architectures to relate spatial information, which helps mitigate class imbalance. Consequently, when accuracy is the most critical requirement, 2D or 3D CNNs should be prioritized. These architectures also tend to perform better even in balanced datasets. Another factor to consider is that the new data-splitting strategy adopted in this study results in fewer training samples, directly affecting the network's generalization capability.

Nonetheless, the results in Tab. 16 and Tab. 18 show a direct, nearly linear correlation between the number of training samples and classification accuracy, often exceeding 99%. These results suggest that using spectral-spatial information in 3D architectures can have an effect similar to increasing dataset size and balancing

Table 22 – Processing time for the 1D architectures running on the CPU+NVIDIA A100 GPU infrastructure.

HSI	1D Architecture	Time test dataset (ms) captured	Time 1 test sample (ms) captured	Time 1 test sample (ms) calculated
IP	1D-CNN-O	423.36	51.69	0.13
	1D-CNN-R	444.42	66.91	0.14
	LSTM	1,742.48	56.35	0.56
	1D-CNN-RA	555.94	50.84	0.18
	LSTM-A	1,630.76	54.56	0.53
	Transformer	897.95	65.62	0.29
SA	1D-CNN-O	1,132.53	51.66	0.06
	1D-CNN-R	1,472.39	49.31	0.09
	LSTM	3,642.60	51.56	0.22
	1D-CNN-RA	2,060.54	52.62	0.12
	LSTM-A	4309.86	53.10	0.26
	Transformer	2,705.60	56.11	0.16
UP	1D-CNN-O	945.23	50.09	0.07
	1D-CNN-R	1,207.85	49.86	0.09
	LSTM	3,077.71	51.26	0.23
	1D-CNN-RA	1,657.27	52.59	0.12
	LSTM-A	3,659.06	54.99	0.28
	Transformer	2,231.02	54.80	0.17
CP	1D-CNN-O	3029.52	67.30	0.10
	1D-CNN-R	3948.82	73.13	0.08
	LSTM	9153.98	62.29	0.20
	1D-CNN-RA	5421.70	65.52	0.12
	LSTM-A	11946.48	81.16	0.26
	Transformer	7937.89	82.39	0.17
KC	1D-CNN-O	415,41	60.81	0.39
	1D-CNN-R	369.36	61.13	0.23
	LSTM	1633.70	65.37	1.04
	1D-CNN-RA	453.93	59.66	0.29
	LSTM-A	1768.09	68.27	1.13
	Transformer	706.98	68.59	0.45
BT	1D-CNN-O	337.75	57.35	0.51
	1D-CNN-R	275.59	58.89	0.28
	LSTM	2564.81	68.81	2.63
	1D-CNN-RA	341.33	60.23	0.52
	LSTM-A	1674.84	71.07	1.71
	Transformer	621.34	65.54	0.63

in 1D architectures, as observed in the CP scene. This leads to the conclusion that architectures such as the Transformer, which achieve the best average performance among 1D models, can effectively replace 3D CNNs, provided they are trained on a sufficiently large and well-balanced dataset. Additionally, 1D architectures offer the benefit of significantly reduced processing time and memory usage.

Focusing on the impact of the Attention mechanism within the 1D models, we observe that its inclusion had no significant effect on CNN performance, and only a modest effect on LSTM performance. As previously mentioned, the IP dataset is especially affected by sample scarcity and class imbalance. However, 1D architectures show robust performance on more balanced datasets such as SA and UP, achieving

classification accuracies that approach or exceed 95% while requiring substantially fewer parameters, lower memory usage, and shorter processing times.

In summary, HSI datasets with sufficient spectral information at the pixel level can be effectively classified using 1D architectures, particularly when spatial features or scene structure are not crucial to the application. In such cases, 1D models offer an advantageous balance between classification performance, resource efficiency, and computational feasibility.

4.6 ANALYSIS AND EXPLORATION OF TRANSFORMER-BASED ARCHITECTURE DESIGN

In this section, we explore the Transformer-based architecture previously tested and compared in the preceding section, aiming to reduce it to enable deployment on embedded systems and to investigate hardware acceleration. The choice of the Transformer architecture is justified by the results, discussions, and analyses presented earlier.

4.6.1 Architecture Exploration

The Transformer-based architecture used in the experiments has a relatively large number of parameters (34,000) in its version designed for datasets with more classes. This observation motivates an investigation into the potential reduction of the network's parameter count and complexity. Accordingly, the following questions guide this exploration:

- Is it possible to reduce the number of parameters, and what is the threshold below which the network is no longer able to maintain the accuracy achieved in the original tests?
- Can the number of layers and operations in the architecture be reduced to minimize the computational complexity in terms of the operations required for inference?

With these questions in mind, exploratory and manual tests were conducted to reduce the architecture's depth, computational operations, and parameter count.

The initial architecture, as described in the code below, consisted of four Transformer blocks with a 256-dimensional multi-head attention and 4 attention heads, followed by convolutional layers with a feed-forward projection dimension of 4. After the encoder blocks, a GlobalAveragePooling1D layer and an MLP with 128 units completed the network, resulting in an architecture with approximately 34,000 parameters. Although this configuration achieved good accuracy in the experiments, it poses significant implementation challenges for embedded systems due to its computational complexity and parameter volume.

To address this and make hardware acceleration and deployment on resource-constrained platforms feasible, a manual, incremental reduction process was initiated. This reduction was guided by criteria such as minimizing the number of parameters, network depth, and computational operations, without drastically compromising model performance. The architecture resulting from this process is significantly more compact. It now includes only two Transformer blocks, implemented manually to allow greater control over operations. The attention dimension was reduced to 30, with five attention heads maintained. The feed-forward block was simplified to include only two internal units, and normalization layers were removed, reducing the number of vector operations. The GlobalAveragePooling layer was replaced with a Flatten operation, followed by a single Dense layer with 128 neurons prior to the output layer.

This reduced version, while much simpler, still achieved satisfactory results in initial tests, demonstrating that it is possible to preserve model efficiency even under severe structural constraints. Therefore, the proposed reduced version of the Transformer architecture represents a key step in adapting the architecture to embedded systems, while maintaining a balance between performance, accuracy, and computational feasibility. Table 23 summarizes the reductions made in the default Transformer architecture.

Table 23 – Transformer architecture components optimized for embedded implementation

Component	**Reduction Strategy**	**Justification**
Number of Transformer blocks	Reduced from 4 to 2	Decreases network depth, reducing latency and total parameter count
Attention head dimension (key_dim)	Reduced from 256 to 30	Lowers computational cost in the attention mechanism, requiring fewer array multiplications
Number of attention heads	Change from 4 to 5	Decreases memory required for storing parameters and the number of computations needed
Normalization layers (LayerNorm)	Removed	These are costly operations and complex to implement in digital logic
Final aggregation operation	Replaced *GlobalAveragePooling* with *Flatten + Dense*	Provides greater control and facilitates hardware conversion

The reduction in the total number of model parameters, from approximately 34,776 to 8,148, represents a significant advancement toward enabling deployment on embedded platforms with limited resources, such as FPGAs and low-power GPUs (e.g., NVIDIA Jetson devices). This reduction of over 75% in parameter count directly impacts

several critical aspects of embedded applications. First and foremost, the memory savings are substantial. In environments such as FPGAs, the available internal memory (BRAM) and external memory are limited, and efficient inference execution depends heavily on the ability to store model weights locally. This avoids frequent access to external memory, which is typically slower and consumes more energy.

Moreover, reducing the number of parameters also reduces the number of arithmetic operations required during inference, particularly multiplications and additions. This results in faster inference times and lower power consumption, which are essential in real-time applications and systems with constrained energy budgets, such as autonomous vehicles, drones, smart sensors, and satellites.

In the context of embedded GPUs, such as those found in the NVIDIA Jetson family, this reduction is equally relevant. These platforms, while offering GPU acceleration, still operate under power and thermal constraints. A smaller model with fewer parameters reduces GPU workload and memory transfers, lowering latency and energy consumption, enabling high-performance inference while preserving the device's thermal envelope and operational autonomy. Therefore, the proposed parameter reduction not only enhances the feasibility of deploying the Transformer architecture in constrained environments but also aligns with the design goals of edge AI systems that demand both computational efficiency and energy-awareness. Figure 24 shows the differences between the Transformer default and the Transformer reduced.

Table 24 presents a comparison between the two architectures proposed for use in HSI classification based on the architecture adapted for the IP dataset.

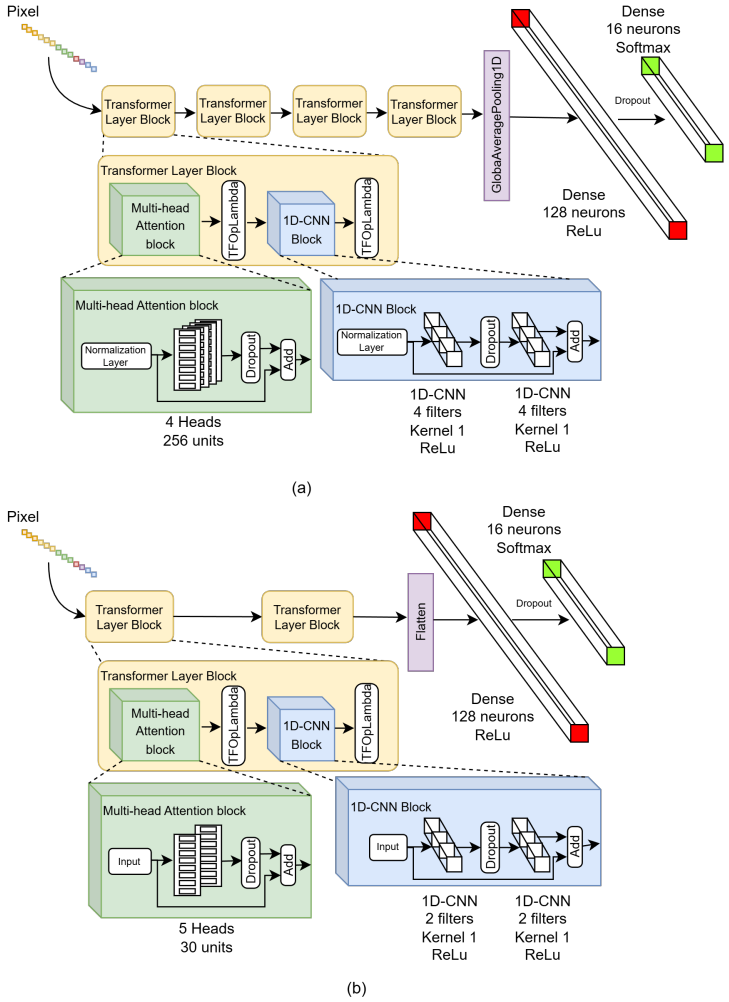
The reduction of layers in the Transformer-R (Reduced) architecture compared to the standard version is evident. This allows us to initially conclude on the performance gain of the proposed architecture across the metrics that guide this work.

4.6.2 Motivations for Reducing Transformer Architecture

The reduction of the Transformer architecture aims to adapt it for efficient execution on embedded platforms with limited resources, particularly those equipped with embedded GPUs, such as the NVIDIA Jetson family. Although these platforms offer considerable computational capabilities, they operate under constraints related to memory, power consumption, and thermal performance, requiring targeted optimizations to enable real-time on-device inference.

A key motivation lies in the computational resource limitations of these devices. Embedded GPUs such as the Jetson Orin, TX2, and Xavier offer significantly less RAM, memory bandwidth, and CUDA cores compared to data center GPUs. These constraints limit the size and complexity of deployable models. Furthermore, energy efficiency is critical, especially for battery-powered systems. Reducing the depth of the Transformer architecture, along with the attention and feed-forward dimensions, results

Figure 24 – Comparison between Transformer default (a) and Transformer reduced (b).



in a lower computational load and lower energy consumption during inference.

Inference latency is another central concern. Embedded applications such as onboard computer vision, mobile robotics, drones, and autonomous vehicles require fast and predictable responses. Optimized architectures with fewer parameters and simpler operations contribute significantly to achieving real-time performance. From the GPU execution standpoint, reduced models allow better CUDA core utilization and more efficient use of shared memory and cache, minimizing bottlenecks and improving parallelism. Additionally, simpler structures facilitate the application of optimization techniques such as operator fusion, INT8 quantization, and hardware acceleration via libraries like TensorRT.

Compact models also lead to shorter training, validation, and redeployment cycles, which accelerate development and support more agile experimentation in constrained environments, which is an advantage during prototyping and edge-focused model tuning.

In summary, reducing the Transformer architecture not only enables deployment on embedded GPU platforms but also enhances energy efficiency, reduces latency,

Table 24 – Architecture, layers, output shape, and parameters from Transformer-based models.

Layer Default	Output shape Default	Parameters Default	Layer Reduced	Output shape Reduced	Parameters Reduced
Input Layer	(30,1)	0	Input Layer	(30,1)	0
Layer Normalization	(30,1)	2	Multihead Attention	(30,1)	1,051
Multihead Attention	(30,1)	7,169	Dropout	(30,1)	0
Dropout	(30,1)	0	Add	(30,1)	0
Add	(30,1)	0	Conv1D	(30,2)	4
Layer Normalization	(30,1)	2	Dropout	(30,2)	0
Conv1D	(30,4)	8	Conv1D	(30,1)	3
Dropout	(30,4)	0	Add	(30,1)	0
Conv1D	(30,1)	5	Multihead Attention	(30,1)	1,051
Add	(30,1)	0	Dropout	(30,1)	0
Layer Normalization	(30,1)	2	Add	(30,1)	0
Multihead Attention	(30,1)	7,169	Conv1D	(30,2)	4
Dropout	(30,1)	0	Dropout	(30,2)	0
Add	(30,1)	0	Conv1D	(30,1)	3
Layer Normalization	(30,1)	2	Add	(30,1)	0
Conv1D	(30,4)	8	Flatten	(30)	0
Dropout	(30,4)	0	Densa	(128)	3,968
Conv1D	(30,1)	5	Dropout	(128)	0
Add	(30,1)	0	Densa	(16)	2,064
Layer Normalization	(30,1)	2			
Multihead Attention	(30,1)	7,169			
Dropout	(30,1)	0			
Add	(30,1)	0			
Layer Normalization	(30,1)	2			
Conv1D	(30,4)	8			
Dropout	(30,4)	0			
Conv1D	(30,1)	0			
Add	(30,1)	2			
Layer Normalization	(30,1)	2			
Multihead Attention	(30,1)	7,169			
Dropout	(30,1)	0			
Add	(30,1)	0			
Layer Normalization	(30,1)	2			
Conv1D	(30,4)	8			
Dropout	(30,4)	0			
Conv1D	(30,1)	5			
Add	(30,1)	0			
Global Polling 1D	(30)	0			
Densa	(128)	3,968			
Dropout	(128)	0			
Densa	(16)	2,064			

facilitates quantization, and accelerates development. This adaptation is essential for edge applications that require responsive, low-power, and reliable on-device inference, expanding the feasibility of using Transformer models beyond high-performance computing environments.

While the Reduced Transformer architecture is inherently well-suited for hardware-level deployment, such as FPGA implementation using IP cores, the experiments in this study aimed to validate its efficiency through GPU acceleration on an embedded

platform. Using the NVIDIA Jetson enabled a realistic evaluation of energy consumption, inference time, and memory usage under typical embedded constraints. The results demonstrated that the proposed model maintains a satisfactory trade-off between ranking performance and hardware resource requirements, confirming its suitability for embedded scenarios and potentially for future hardware-specific implementations.

4.7 MATERIALS AND METHODS FOR ANALYSIS OF THE REDUCED MODEL IN CLASSIFICATION AND INFERENCE USING GPU HARDWARE ACCELERATION

To evaluate the classification performance of the reduced model, a comparison was conducted against the previously explored datasets and the original Transformer architecture, now referred to as *Default* or simply *Transformer*. The newly proposed architecture is referred to as *Transformer-R* or simply *Reduced*.

Following this analysis, a technique developed during this work was also explored. It focuses on fusing spatial data extracted using the SEEDS segmentation algorithm with features obtained from Multihead-Attention and 1D-CNN layers, concatenated at the MLP stage of the architecture. This approach enables parallel extraction of spatial and spectral information, reducing the strong spatial-spectral dependence typically observed in 3D-CNN architectures.

After validating the classification capability of the reduced architecture, this study employed two computational devices with distinct characteristics to evaluate the performance of Transformer-based neural network models, both the Default and Reduced versions, as well as a 3D convolutional architecture (3D-CNN), used as a baseline for comparison.

4.7.1 Materials

To compare the classification performance between the Default and Reduced models, and to demonstrate the viability of the data fusion technique, the following setup was used:

- Google Colaboratory Pro with 12-core Intel Xeon at 2.20 GHz, NVIDIA A100 GPU with 40 GB of VRAM, and 83.5 GB of main memory;
- Programming language: Python 3.11;
- Datasets: IP, SA, UP, CP, KC, and BT.

For GPU-based model execution, two platforms/devices were employed. The first device is a notebook Avell Storm One 470, equipped with:

- Processor: Intel Core i9-14900HX (24 cores, 32 threads, up to 5.8GHz);
- RAM: 64GB DDR5;

- GPU: NVIDIA GeForce RTX 4070 with 8GB of GDDR6 dedicated memory.

The second is a NVIDIA Jetson Orin Nano, model ASUS PE1100N. This device is an embedded system with a heterogeneous architecture optimized for real-time inference applications, running JetPack 6.0 with TensorRT, CUDA, and support for the TensorFlow C API. Its main specifications are:

- CPU: 6-core Arm Cortex-A78AE;
- GPU: Ampere architecture with 1024 CUDA cores;
- Memory: 8GB unified LPDDR5;
- Configurable power modes: 7W, 15W without Jetson Clocks, and 15W with Jetson Clocks enabled.

The Jetson Clocks mode ensures that the CPU, GPU, and other processing units operate at their maximum permitted frequencies, eliminating performance fluctuations caused by dynamic frequency scaling.

Both devices were used to execute models exported in TensorFlow format, previously trained in Python (TensorFlow/Keras), and loaded via the TensorFlow C API. Execution was supported in both GPU and CPU-only modes, enabling a comparative evaluation of both operational scenarios.

For model execution on both platforms/devices, the following environment setup was adopted:

- Training language: Python 3.10 on the NVIDIA Jetson and Python 3.12 on the notebook;
- TensorFlow version: TensorFlow 2.16 on the NVIDIA Jetson and TensorFlow 2.19 on the notebook;
- Inference language: C, using the TensorFlow C API (version 2.16 for NVIDIA Jetson and 2.18 for the notebook);
- Operating Systems: Ubuntu 22.04 on the NVIDIA Jetson and Kubuntu 24.04 on the notebook.

4.7.2 Methods

To evaluate model performance, the same methodologies described throughout this chapter for the various architectures were applied. This ensured a fair and consistent comparison across models.

For the inference performance analysis on the two target platforms, three architectures were evaluated:

- The original Transformer-based architecture;
- The reduced Transformer-based architecture, with lower depth, fewer attention heads, and narrower feed-forward blocks;

- The HybridSN architecture as a baseline.

All models accept input tensors in the format $(19, 19, 30, 1)$ and were exported to preserve 1:1 equivalence with their Python versions, ensuring functional consistency between implementations.

The primary metric of interest was inference time, evaluated under three conditions:

1. Full batch inference on datasets with 18,450 and 3,075 samples;
2. Inference of a single sample;
3. Inference with varying *batch sizes*:
   1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, and 8192.

Additionally, in GPU usage scenarios, the following metrics were recorded:

- Peak GPU utilization (%);
- Total GPU memory usage (in GiB);
- Free GPU memory before execution;
- GPU power consumption during execution (in Watts).

For the Jetson Orin Nano, additional energy consumption metrics were collected using the `tegrastats` tool, including:

- VDD_IN (W): total power consumed by the device, including all active components;
- VDD_CPU_GPU_CV (W): combined power consumed by CPU, GPU, and computer vision (CV) units;
- Current VDD_IN (A): total input electrical current;
- Current VDD_CPU_GPU_CV (A): current drawn specifically by the processing subsystems.

These variables are critical for analyzing the energy efficiency of embedded models. The VDD_IN parameter represents the total energy supplied to the device, while VDD_CPU_GPU_CV isolates the energy consumption of the core computational units (CPU, GPU, CV), enabling a more precise estimation of the inference energy cost. The relationship between energy consumption, performance, and varying batch size helps to identify the optimal operating point for each architecture under test.

5 RESULTS

This chapter presents the results obtained from running and comparing the Transformer Default, Transformer Reduced, and baseline 3D-CNN architectures. The results were divided into analyses of classification performance, adding data fusion techniques to improve classification performance, and the reduced model's performance on an embedded GPU. The embedded GPU analysis considered time, parallelism exploitation, GPU utilization, memory demand, and energy metrics.

5.1 CLASSIFICATION PERFORMANCE

Table 25 presents the accuracy difference only for the architectures that process spectral information, comparing the Transformer-based Default (Transformer) and Reduced (Transformer-R). The results reveal that Transformer-R maintains comparable, and in some cases superior, performance despite its significantly reduced complexity. Notably, on the IP and CP datasets, Transformer-R slightly outperforms the original architecture in all three metrics: Overall Accuracy (OA), Average Accuracy (AA), and Kappa coefficient. For example, in the CP dataset, the Transformer-R achieves an OA of 99.35%, an AA of 98.01%, and a Kappa of 99.07, surpassing the Default Transformer by small but consistent margins. Similarly, on the IP dataset, Transformer-R improves OA by 0.69 percentage points and Kappa by 0.81, indicating better generalization even under dataset imbalance.

Table 25 – Performance of the Transformer-based architectures.

HSI	Architecture	OA (%)	AA (%)	Kappa ($\times$100)
IP	Transformer	82.56	79.13	80.00
	Transformer-R	83.25	79.83	80.81
SA	Transformer	96.23	98.22	95.80
	Transformer-R	96.13	98.18	95.69
UP	Transformer	95.10	93.05	93.50
	Transformer-R	95.05	92.88	93.43
CP	Transformer	99.20	97.34	98.87
	Transformer-R	99.35	98.01	99.07
KC	Transformer	92.45	88.32	91.59
	Transformer-R	94.24	91.00	93.59
BT	Transformer	97.12	97.23	96.88
	Transformer-R	95.69	95.33	95.77

In the remaining datasets (SA, UP, KC, and BT), the performance differences between the two architectures are marginal, generally within 1–2 percentage points. On SA and UP, the Transformer-R retains nearly identical performance to the original Transformer, indicating that architectural simplification did not hinder its learning capacity. The KC dataset showcases a more notable improvement with Transformer-R, which outperforms the default Transformer by approximately 1.8% in OA and 2.68% in AA. The only exception appears in the BT dataset, where the original Transformer slightly outperforms the reduced version, with a Kappa of 96.88 versus 95.77. These

results collectively confirm that the Transformer-R strikes an excellent balance between efficiency and predictive performance, validating its suitability for deployment in resource-constrained scenarios while maintaining competitive classification accuracy.

5.1.1 Memory Utilization

Model storage is a feature that must be considered for computational systems with limited memory available. This requirement is essential for any application to run on these platforms. Therefore, the amount of memory (space) used by the model in bytes (without the dataset) was also measured using the value obtained by `summary()` from the TensorFlow command. To provide a more accurate comparison, the number of parameters used by each architecture in each dataset was also obtained.

Table 26 highlights the substantial reduction in model size and number of parameters achieved by the proposed *Reduced* Transformer architecture (Transformer-R) across all evaluated HSI datasets. On average, the number of parameters decreased by approximately 76% when compared to the original Transformer architecture, dropping from over 34,000 parameters to fewer than 8,200 in all cases. This reduction is consistent across datasets such as IP, SA, KC, and BT, with Transformer-R maintaining less than one-quarter of the parameters of the default model. This compression directly translates to a significant decrease in memory usage, with model sizes reduced from approximately 135 KB to under 32 KB in most configurations. For instance, in the IP and SA datasets, the model's memory footprint dropped from 135.84 KB to 31.83 KB, making it notably more suitable for deployment on memory-constrained embedded systems.

Table 26 – Number of parameters and consumed memory of the 1D Transformer-based architectures.

HSI	CNN Architecture	Size (KB)	Number of Parameters
IP	Transformer	135.84	34,776
	Transformer-R	31.83	8,148
SA	Transformer	135.84	34,776
	Transformer-R	31.83	8,148
UP	Transformer	132.32	33,873
	Transformer-R	28.30	7,245
CP	Transformer	132.32	33,873
	Transformer-R	28.30	7,245
KC	Transformer	134.33	34,389
	Transformer-R	30.32	7,761
BT	Transformer	134.84	34,518
	Transformer-R	30.82	7,890

This architectural simplification does not compromise generalizability, as it retains compatibility with a wide variety of datasets. For example, in the UP and CP datasets, the original model required over 33,000 parameters and 132.32 KB of stor-

age, while the reduced version needed only 7,245 parameters and 28.30 KB. Similarly, in the KC and BT datasets, the Transformer-R reduced the memory footprint by more than 75%, with only marginal variation in the number of required parameters (7,761 and 7,890, respectively). These results emphasize the efficiency of the Transformer-R architecture and demonstrate its potential for real-time inference in energy-constrained environments. This parameter and memory efficiency make Transformer-R highly appealing for edge computing scenarios, including on-board satellite processing, Unmanned Aerial Vehicle (UAV), or mobile robotics, where storage and compute budgets are critical design constraints.

5.1.2 Analysis of the Results of the Original Architecture with Variation in the Number of Bands

To further evaluate the original Transformer architecture, additional experiments were conducted on the Pavia Centre (PC), Kennedy Space Center (KC), and Botswana (BT) datasets, using two distinct input configurations: 30 and 60 spectral bands. The goal was to assess the impact of doubling the input bands on model performance and to evaluate the architecture's sensitivity to this increase in spectral dimensionality.

The results indicate that, overall, increasing the number of spectral bands did not yield a significant improvement in performance metrics such as Overall Accuracy (OA), Average Accuracy (AA), or the Kappa coefficient. In some cases, slight positive variations were observed; however, these remained within the expected statistical margin of error, suggesting that the architecture exhibits a saturation point in leveraging the additional spectral information provided.

Moreover, increasing the number of input bands directly affected the model's complexity. As a result of processing more features in the initial layers, the total number of parameters increased by approximately 4,000, compared to the original 30-band configuration. This growth implies not only higher memory usage, but also greater computational cost during both training and inference phases—factors that may be limiting in resource-constrained environments such as embedded systems.

Therefore, the results suggest that, for the tested original architecture, doubling the number of input spectral bands is not fully justified from a cost-benefit perspective. The additional complexity did not yield substantial performance gains, reinforcing the importance of considering not only accuracy but also the number of parameters and computational feasibility when selecting and tuning the network's input configurations.

5.2 DATA FUSION ANALYSIS FOR HSI CLASSIFICATION WITH NEURAL NETWORKS BASED ON 1D DATA

This section presents a solution that reduces data dependency in the extraction of both spatial and spectral information while achieving strong performance in image classification. The objective is to demonstrate the feasibility of improving hyperspectral image (HSI) classification accuracy under class imbalance without compromising memory and processing time, as typically observed in 2D and 3D architectures. This is achieved by avoiding data dependency in the creation of the segmentation map and the one-dimensional feature vector extracted by architectures such as CNNs and Transformers, as well as in the fusion method through concatenation. Furthermore, the Superpixel Extracted via Energy-Driven Sampling (SEEDS) segmentation algorithm is explored as part of a solution that introduces no data dependencies until the fusion stage. This algorithm stands out for its reduced processing time when compared to other segmentation algorithms (Yuan et al., 2020).

The proposed architecture combines the 1D Transformer-R architecture with the SEEDS segmentation algorithm and a data concatenation technique to achieve a solution that improves hyperspectral image classification performance while maintaining low processing time and a reduced number of parameters.

In developing this solution, the standard HSI datasets commonly used in the literature and previously explored in this work were employed: IP, SA, UP, CP, KC, and BT.

5.2.1 Segmentation Algorithm

The SEEDS segmentation algorithm applies hill-climbing optimization to assign pixels to segments, starting from a complete initial segmentation and iteratively refining the segments. Initially, superpixels (SPs) are generated on a regular grid that covers the entire image. The algorithm then refines these SPs at each iteration by adjusting their boundaries to better match the image's edges and features (Van den Bergh et al., 2012).

The SEEDS algorithm uses Equation ((34)) to determine the set s of pixels belonging to a superpixel (SP). The term $H(s)$ computes the pixel neighborhood based on the color distribution, $G(s)$ defines the segment boundaries, and γ balances these two factors.

$$E(s) = H(s) + \gamma G(s) \tag{34}$$

The SEEDS algorithm used in this work is available in the OpenCV library and allows adjustment of several parameters. These parameters include the image dimensions, the number of superpixels (SPs), the size of the initial grid—which influences the

emphasis on pixels—and the γ parameter, which controls the balance between color and contours, as well as the value that adjusts the color granularity (Bradski, 2000).

Figure 25 shows the output types of the SEEDS algorithm (either edge maps or labeled region maps). We opted for the labeled region map because it facilitates the 1D CNN's identification of segments in the concatenation layer, while prioritizing distinct segmentation within regions belonging to the same class and maintaining homogeneity within regions.

Figure 25 – Segmentation Map Types.

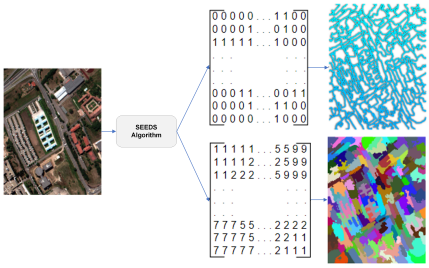


Table 27 presents the configuration parameters of the SEEDS algorithm from OpenCV, which were manually and empirically adjusted for each of the three datasets.

Table 27 – Parameters defined for each dataset.

	Datasets		
Parameters	**IP**	**SV, KC, and BT**	**UP, CP**
Iterations	200	150	150
Priority	2	2	2
Double Check	Yes	Yes	Yes
Estimated SPs	150	150	150
Levels	10	3	3
Bins	5	15	15

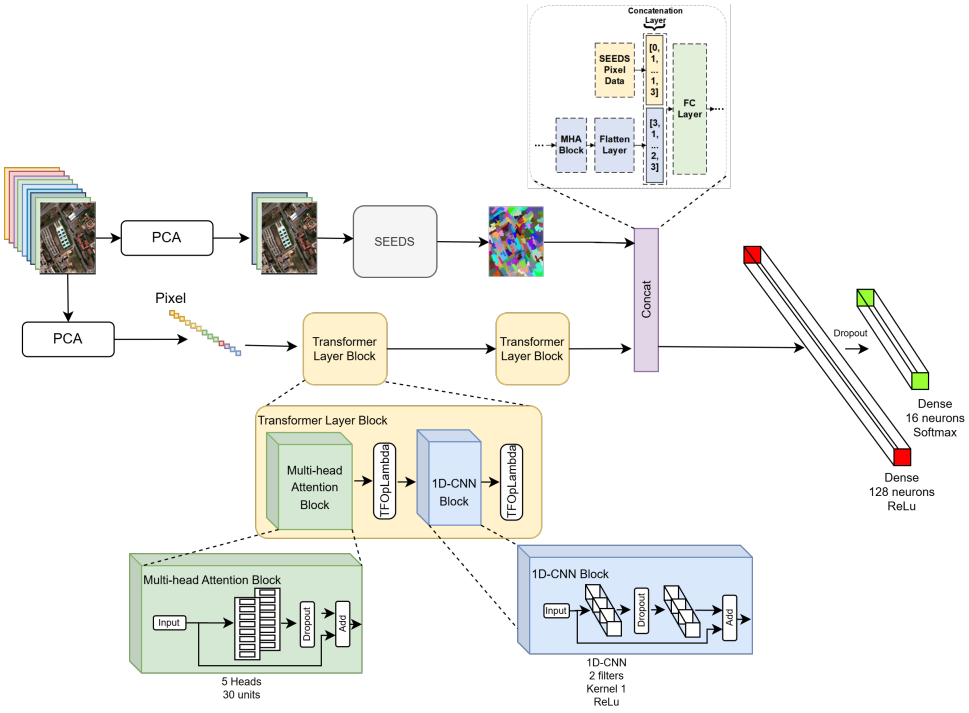
5.2.2 Spectral-Spacial Fusion

The fusion of spectral data extracted by the 1D CNN with spatial data processed by the SEEDS algorithm is performed using data concatenation. In the literature, this technique is applied in different contexts, such as the fusion of heterogeneous data sources (Yadav et al., 2022; Gao; Lim; Jia, 2018) and the fusion of parallel layers (Fırat et al., 2023; Géron, 2023). Figure 26 illustrates the proposed solution architecture for data fusion using concatenation.

In order for concatenation to be successful, the values to be concatenated must have the same number of dimensions, and all dimensions, except the concatenation axis, must match (Abadi et al., 2015). The processing flow follows the standard approach found in most related works. The HSI serves as the input, with processing divided into two streams: spatial and spectral.

The PCA algorithm is applied to reduce the number of HSI bands for both the SEEDS and 1D Transformer-R algorithms. PCA reduces the image to three bands,

Figure 26 – Overview of the architecture implemented for segmentation evaluation



which are then combined into an RGB image, since only spatial information is relevant for SEEDS.

The final classification is performed by the dense layers stage, with the segmentation being integrated at the concatenation layer to generate the class map. Although Figure 26 suggests that classification and segmentation are processed in parallel, the SEEDS algorithm is applied prior to the 1D Transformer.

5.2.3 1D Transformer Performance with SEEDS

Table 28 presents a performance comparison between the Transformer-R and Transformer-R+SEEDS architectures across different hyperspectral datasets, based on Overall Accuracy (OA), Average Accuracy (AA), and the Kappa coefficient. In general, integrating the SEEDS algorithm into the Transformer-R architecture results in significant performance improvements across most datasets analyzed, indicating a significant contribution from the segmentation algorithm's spatial refinement.

In the Indian Pines (IP) dataset, SEEDS yields substantial improvements across all metrics. OA increases from 83.25% to 92.58%, representing a relative gain of approximately 11.2%. AA improves from 79.83% to 91.02%, a gain of around 14.1%, while the Kappa coefficient rises from 80.81 to 91.53, which corresponds to a 13.2% improvement. These results demonstrate a significant enhancement in robustness and generalization capability when spatial information is incorporated.

Similar behavior is observed in the Salinas (SA) dataset, where performance was already high with the baseline Transformer-R and improved further with SEEDS. OA

increased from 96.13% to 99.60% (+3.6%), AA rose from 98.18% to 99.68% (+1.5%), and Kappa improved from 95.69 to 99.56 (+4.0%). These results reinforce the advantage of spatial segmentation even in high-performing scenarios. For the Pavia University (UP) dataset, gains are also evident. OA increases from 95.10% to 98.47% (+3.5%), AA from 93.05% to 97.90% (+5.2%), and Kappa from 93.50 to 97.97 (+4.8%). Although performance in the Pavia Centre (CP) dataset was already high, SEEDS still provides incremental improvements. OA increases from 99.35% to 99.70% (+0.35%), AA from 98.01% to 98.88% (+0.9%), and Kappa from 99.07 to 99.57 (+0.5%).

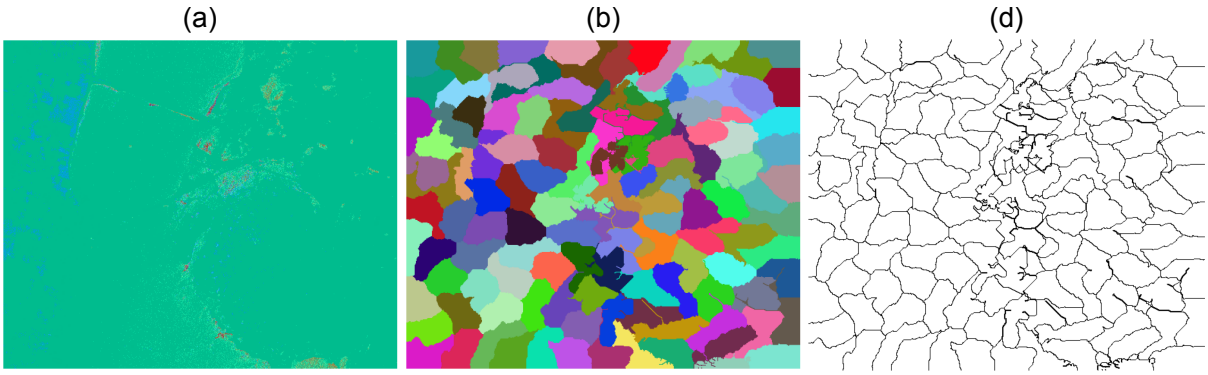
In the Botswana (BT) dataset, the improvements are similarly consistent. OA rises from 95.69% to 99.18% (+3.6%), AA from 95.33% to 99.11% (+4.0%), and Kappa from 95.77 to 99.11 (+3.5%), highlighting the effectiveness of SEEDS in structured environments.

Table 28 – Performance comparison of Transformer-R vs. Transformer-R+SEEDS architecture

Dataset	**Arquitetura**	**OA(%)**	**AA(%)**	**Kappa ($\times$ 100)**
IP	Transformer-R	83.25	79.83	80.81
	Transformer-R+SEEDS	92.58	91.02	91.53
SA	Transformer-R	96.13	98.18	95.69
	Transformer-R+SEEDS	99.60	99.68	99.56
UP	Transformer-R	95.10	93.05	93.50
	Transformer-R+SEEDS	98.47	97.90	97.97
CP	Transformer-R	99.35	98.01	99.07
	Transformer-R+SEEDS	99.70	98.88	99.57
KC	Transformer-R	94.24	91.00	93.59
	Transformer-R+SEEDS	83.63	82.22	84.33
BT	Transformer-R	95.69	95.33	95.77
	Transformer-R+SEEDS	99.18	99.11	99.11

However, this positive trend is not universal. In the Kennedy Space Center (KC) dataset, SEEDS integration leads to a performance drop across all metrics. OA falls from 94.24% to 83.63%, a relative decrease of 11.3%; AA drops from 91.00% to 82.22% (–9.6%); and the Kappa coefficient declines from 93.59 to 84.33 (–9.9%). These results suggest that in datasets with greater spectral complexity or less spatial homogeneity, SEEDS may induce over-segmentation or disrupt discriminative spectral patterns, ultimately harming classification performance. Figure 27 illustrates the KC dataset and the segmentations generated. As can be seen, the segmentation has little effect because the classes are represented by small clusters relative to the entire scene. This suggests that segmentation techniques should be avoided when class homogeneity is high across the scene.

Figure 27 – KC dataset (a) and segmentations by (b) color map and by (c) boundaries.



In summary, the Transformer-R+SEEDS architecture achieves considerable performance gains across most datasets, especially on those with strong spatial coherence. However, its effectiveness is dataset-dependent. In certain cases, such as the KC dataset, spatial post-processing can lead to negative outcomes. Therefore, the use of SEEDS should be carefully evaluated based on the spectral-spatial characteristics of the data, ensuring that segmentation enhances rather than hinders model performance.

5.3 EXPLORATORY AND COMPARATIVE ANALYSIS OF THE ARCHITECTURES ON CPU AND GPU WITH DIFFERENT SAMPLE VOLUMES

In this section, the performance exploration of the proposed reduced Transformer architecture, referred to as *Reduced*, is presented in comparison to the original Transformer architecture from which it was derived, referred to as *Default*, and the 3D-CNN HybriSN architecture proposed by Roy et al. (2020), used here as a baseline. The comparisons followed the methodology described at the end of the architecture design chapter and were conducted using a GPU-equipped notebook (P0 and P1) and the NVIDIA Jetson Orin Nano under three GPU power configurations (J0, J1, and J2). The comparisons include inference time for both dataset sizes (3,075 and 18,450 samples) on GPU and CPU, GPU utilization percentage, memory usage during processing, GPU power dissipation, and the electric current drawn by the NVIDIA Jetson Orin Nano during inference. A compiled analysis based on the best inference times for each architecture and platform concludes the evaluation.

5.3.1 Time Analysis

The execution of the Reduced model on GPUs demonstrated significantly superior performance when compared to the standard Transformer model and the 3D-CNN architecture. This performance gain is evident in both the 3,075-sample and 18,450-sample tests, particularly under configurations with larger batch sizes (e.g., 512). The execution times are reported in Table 29 and Table 30. Moreover, Figures 28, 29, and

30 visually illustrate the execution time evolution as the batch size increases. On the GPU-equipped notebook (P1), the Reduced model achieved a minimum inference time of 343.84 milliseconds for 3,075 samples and 462.50 milliseconds for 18,450 samples.

Table 29 – Inference time for 3,075 samples on GPU across different models and platforms

Batch	P0 - 1D Default (ms)	P0 - 1D Reduced (ms)	P1 - 1D Default (ms)	P1 - 1D Reduced (ms)	P0 - 3D-CNN (ms)	P1 - 3D-CNN (ms)	J0 - 1D Default (ms)	J0 - 1D Reduced (ms)	J1 - 1D Default (ms)	J1 - 1D Reduced (ms)	J2 - 1D Default (ms)	J2 - 1D Reduced (ms)	J0 - 3D-CNN (ms)	J1 - 3D-CNN (ms)	J2 - 3D-CNN (ms)
1	5818.43	1055.59	2519.11	1055.59	4511.00	2204.53	35527.86	13773.09	34997.69	13818.45	21014.24	6971.90	17274.00	12439.84	7876.77
2	3353.01	742.28	1634.53	742.28	3019.33	1571.63	19059.72	6951.29	18988.07	6912.53	11245.42	3946.92	14272.99	9412.58	5917.20
4	2264.38	557.85	1177.00	557.85	2043.52	1255.70	11398.35	4385.59	10700.89	3579.28	6903.35	3142.22	10142.37	5975.19	4608.88
8	1574.03	427.02	927.07	427.02	1631.57	1134.12	8078.50	2558.05	6465.15	2673.01	4338.06	1475.81	9480.27	5134.64	4257.59
16	1301.79	395.98	846.95	395.98	1499.52	1042.57	7582.90	1784.38	4461.28	1569.95	3548.21	1051.47	9277.89	4993.83	4133.71
32	1235.59	394.39	767.62	394.39	1472.08	1081.60	7120.07	1546.02	3883.77	1123.96	3567.72	870.93	11108.60	5091.65	4293.94
64	1125.41	371.26	751.68	371.26	1486.74	1131.06	6973.29	1405.86	3734.82	995.62	3481.39	827.88	12538.51	5442.62	4833.66
128	1185.76	356.05	769.47	356.05	1610.23	1279.95	6882.30	1341.78	3630.72	953.98	3394.12	802.91	14893.01	6465.82	6071.13
256	1248.70	361.06	795.62	361.06	1848.08	1504.93	6862.61	1359.64	3659.38	883.34	3405.42	813.24	22263.69	9083.77	8689.94
512	1241.24	343.84	804.25	343.84	2314.75	1948.20	6953.50	1343.34	3765.67	898.08	3421.34	811.75	43902.49	14497.18	14007.39
1024	1213.09	349.74	840.22	349.74	3332.14	2972.62	7177.61	1363.99	3902.43	986.59	3595.72	810.76	66023.51	24761.15	23796.31
2048	1243.84	351.59	832.76	351.59	6894.60	6626.83	7324.51	1385.04	4045.61	943.49	3724.78	896.80	146917.15	51990.49	51346.41
4096	1226.54	353.38	822.56	353.38	5800.97	5378.77	7338.70	1370.40	4317.01	980.24	4013.51	812.16	121218.56	41841.02	41836.68
8192	1446.68	372.29	1033.48	372.29	5705.09	5377.06	7590.83	1359.34	4302.45	985.81	3994.59	825.73	121074.07	41782.97	41673.51

On the NVIDIA Jetson platform, even under energy and computational constraints, the *Reduced* model achieved competitive inference times, such as 1341.78 ms (3,075 samples, batch size 128 in J0) and 1510.09 ms (18,450 samples, batch size 1024 in J1). These results demonstrate that the *Reduced* model is highly scalable: as the sample volume increases, it maintains stable, efficient performance while better utilizing the GPU's parallel resources.

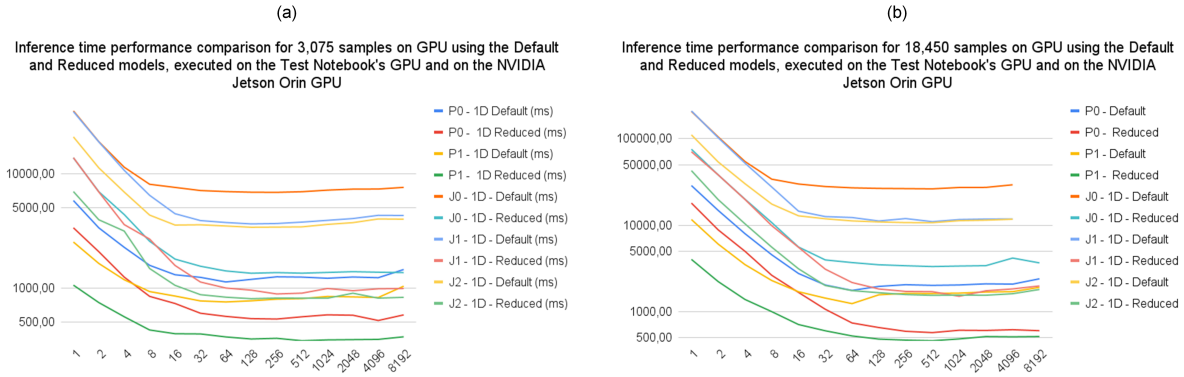
Table 30 – Inference time for 18,450 samples on GPU across different models and platforms.

Batch	P0 - 1D Default (ms)	P0 - 1D Reduced (ms)	P1 - 1D Default (ms)	P1 - 1D Reduced (ms)	P0 - 3D-CNN (ms)	P1 - 3D-CNN (ms)	J0 - 1D Default (ms)	J0 - 1D Reduced (ms)	J1 - 1D Default (ms)	J1 - 1D Reduced (ms)	J2 - 1D Default (ms)	J2 - 1D Reduced (ms)	J0 - 3D-CNN (ms)	J1 - 3D-CNN (ms)	J2 - 3D-CNN (ms)
1	28772.56	18051.83	11653.64	4031.18	21910.89	9475.35	207274.55	75661.82	210077.89	70776.06	110918.71	42790.76	88263.47	67334.36	37529.53
2	14843.16	8845.38	6055.29	2232.85	11204.79	5158.89	104894.15	38211.23	102764.33	38249.20	53943.82	19792.43	49885.25	33394.50	21184.94
4	7998.30	5003.65	3510.52	1388.18	6398.66	3281.17	54612.50	20023.87	52308.73	19883.46	30244.66	10397.22	34398.73	19510.93	14523.86
8	4558.76	2640.54	2297.82	1003.21	4093.32	2344.24	34107.28	10742.86	27702.48	9898.73	17580.54	5584.06	27645.20	14488.32	11367.35
16	2769.14	1673.88	1703.93	711.66	2863.62	1819.92	29969.64	5604.41	14554.95	5561.71	12858.82	3138.10	24838.96	12092.10	10369.95
32	2039.19	1069.04	1438.78	602.69	2563.17	1896.39	28037.44	3972.82	12598.91	3107.00	11882.07	2009.77	24862.56	11049.02	9809.35
64	1765.16	741.88	1241.63	525.27	2336.45	1786.77	27089.66	3702.36	12287.69	2178.15	11225.75	1756.16	26979.87	10947.36	10073.24
128	1964.34	656.93	1570.27	482.94	2583.90	2136.75	26677.34	3493.24	11201.29	1833.16	10925.62	1660.36	30141.99	11972.89	11101.79
256	2054.69	594.49	1629.80	470.25	2785.55	2314.64	26523.71	3404.23	11950.74	1719.03	10734.52	1580.45	36103.48	14552.36	13757.77
512	2019.09	573.98	1628.82	462.50	3097.44	2757.83	26356.90	3332.14	11006.61	1706.65	10705.80	1547.95	52442.01	19673.65	20261.36
1024	2039.73	611.16	1641.17	485.12	4349.34	3916.88	27344.17	3376.73	11638.69	1510.09	11290.66	1558.39	82791.28	30428.35	29798.77
2048	2103.29	607.95	1692.05	517.01	6011.21	5552.19	27395.33	3410.49	11792.64	1748.04	11447.20	1553.41	95375.82	35686.70	35170.14
4096	2087.11	620.04	1708.19	513.40	11288.79	10803.02	29369.10	4176.62	11820.49	1837.12	11742.50	1616.18	216737.55	80900.49	74821.89
8192	2403.38	603.94	1915.13	517.21				3671.51		1986.92		1810.19			

The 3D-CNN architecture is well known for its ability to extract complex spatial and temporal information. However, this level of sophistication comes at a cost: processing is considerably slower, especially on platforms with limited computational resources. The comparison between the Reduced model and the 3D-CNN reveals a significant performance gain. On the Jetson platform (J0), with 18,450 samples, the Reduced model was approximately 7.45 times faster than the 3D-CNN.

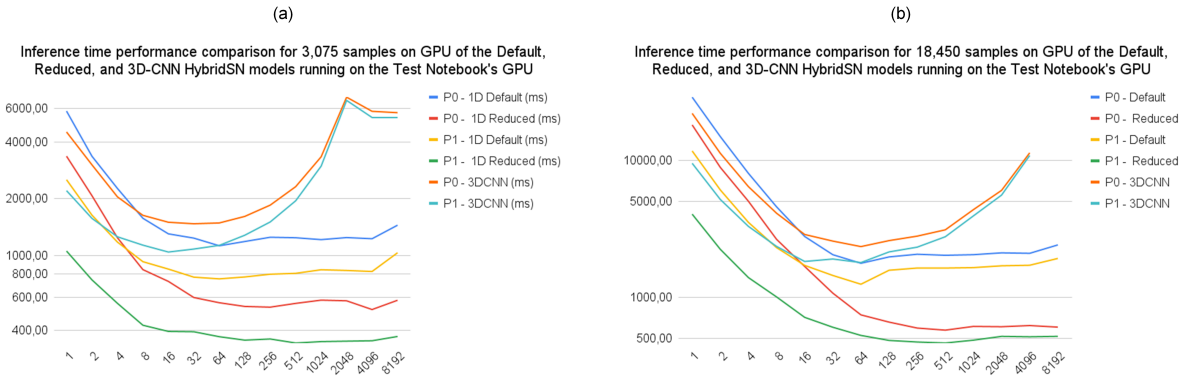
On the CPU (J2), with 3,075 samples, the improvement was even more pronounced: the Reduced model was up to 16.38 times faster than the 3D-CNN, with execution times of 721 ms versus 11,815 ms, respectively. On the notebook (P1), the acceleration was also substantial, reaching up to 5.44$\times$ improvement in execution time. These results reinforce that, although the 3D-CNN offers advantages in terms of representing complex data structures, the Reduced model is a significantly more efficient

Figure 28 – Inference time performance comparison on GPU using the Default and Reduced models. The graphics demonstrate the models behavior to (a) 3,075 samples and (b) 18,450 samples.



choice when computational performance and speed are priorities, without compromising the model's ability to infer relevant patterns.

Figure 29 – Inference time performance comparison on GPU of the Default, Reduced, and 3D-CNN HybridSN models running on the Test Notebook whith (a) 3,075 samples and (b) 18,450 samples.



The comparison between CPU and GPU reveals a particularly interesting behavior, where it is observed that the Reduced model behaves differently depending on the data volume. In low-data-volume scenarios, such as 3,075 samples used in tests, the CPU was found to be as efficient as, or even faster than, the GPU. This behavior can be observed in Table 31 and Table 32.

As observed in the previous tables, in low-data-volume scenarios, in this case, with 3,075 samples, the CPU was found, in some cases, to be as efficient as, or even faster than, the GPU. This occurs primarily because the initial cost of transferring data to the GPU (transfer overhead) may not be offset by the gains from parallelization. On the P1 notebook, for instance, the Reduced model with a batch size of 512 achieved an inference time of 153.94 ms on the CPU, whereas the GPU recorded 343.84 ms. In other words, the CPU was more than twice as fast in this specific configuration.

This scenario is particularly relevant for real-time applications with intermittent

Figure 30 – Inference time performance comparison on GPU of the Default, Reduced, and 3D-CNN HybridSN models running on the NVIDIA Jetson Orin Nano whit (a) 3,075 samples and (b) 18,450 samples.

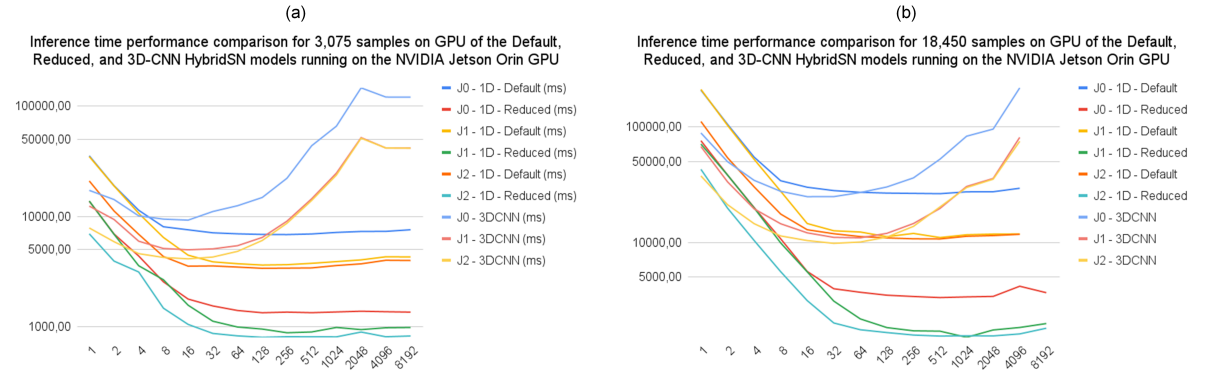


Table 31 – Inference time for 3,075 samples on CPU across different models and platforms

Batch	P0 - 1D Default (ms)	P0 - 1D Reduced (ms)	P1 - 1D Default (ms)	P1 - 1D Reduced (ms)	P0 - 3D-CNN (ms)	P1 - 3D-CNN (ms)	J0 - 1D Default (ms)	J0 - 1D Reduced (ms)	J2 - 1D Default (ms)	J2 - 1D Reduced (ms)	J0 - 3D-CNN (ms)	J2 - 3D-CNN (ms)
1	14831.97	6052.20	4891.53	2023.32	15059.65	2957.74	24633.43	5983.71	13886.24	3180.87	42760.57	22891.38
2	8892.42	3584.88	3009.35	1174.46	7613.31	1575.63	17997.87	4051.48	9958.96	2253.41	37345.96	18052.66
4	5576.25	2047.33	1903.38	713.71	4696.51	1208.11	15278.55	2813.38	8860.34	1637.97	32349.06	15669.19
8	3669.66	1188.37	1295.38	434.13	3297.63	1025.34	12400.54	2250.01	7193.11	1247.46	29818.00	14374.23
16	2806.32	817.18	859.10	309.97	2643.99	981.67	11425.11	1858.99	6400.12	1040.14	28942.68	12743.30
32	2318.54	583.24	608.57	234.20	2193.16	1053.23	10444.16	1520.38	6218.29	890.16	29179.95	12525.93
64	1685.53	573.43	499.25	198.95	2001.61	956.15	9957.64	1446.70	5966.28	825.81	27485.17	12339.04
128	1581.63	426.94	603.81	172.02	1163.53	896.96	9574.26	1431.91	5846.80	795.26	27411.16	12060.26
256	1769.55	359.56	955.84	159.28	1069.10	859.49	9195.21	1401.60	5701.87	769.84	27238.91	12003.39
512	1679.18	325.16	1044.08	153.94	1053.60	837.34	10227.64	1322.10	6160.94	766.28	27089.40	12174.23
1024	1316.64	343.49	1141.85	191.87	1195.33	870.90	10314.02	1248.37	6054.20	731.22	27087.85	11815.95
2048	1245.94	362.98	1137.08	200.35	1112.04	874.40	10132.01	1260.64	6009.57	742.13		
4096	1227.58	368.26	1136.61	210.41	1098.60	906.01	9845.00	1303.47	5957.00	731.17		
8192	2111.39	496.70	2100.78	290.85	1061.15	908.55	9844.23	1220.52	5939.32	721.46		

Table 32 – Inference time for 18,450 samples on CPU across different models and platforms

Batch	P0 - 1D Default (ms)	P0 - 1D Reduced (ms)	P1 - 1D Default (ms)	P1 - 1D Reduced (ms)	P0 - 3D-CNN (ms)	P1 - 3D-CNN (ms)	J0 - 1D Default (ms)	J0 - 1D Reduced (ms)	J2 - 1D Default (ms)	J2 - 1D Reduced (ms)	J0 - 3D-CNN (ms)	J2 - 3D-CNN (ms)
1	86556.56	35970.94	28199.27	11737.97	90221.42	17268.59	142803.97	32230.89	80630.74	17839.91	252668.00	135040.54
2	51997.31	20543.80	16981.55	6708.64	45194.67	9034.99	104087.20	21950.57	57602.27	12506.85	224463.67	103899.00
4	32276.19	11244.85	10522.52	3708.65	27308.10	6869.26	91659.92	15642.20	52039.88	8893.94	189945.77	91650.66
8	20457.62	6188.92	6909.33	2021.79	18577.45	5691.52	71400.92	12042.56	40915.71	6589.33	178537.25	80778.87
16	13000.85	3786.76	4128.74	1262.60	14833.73	5240.28	64597.24	9786.14	35706.55	5366.12	172759.12	75148.58
32	12758.60	2398.69	2706.02	842.65	12537.45	5917.54	59438.92	8204.01	34243.48	4582.74	166756.24	72739.32
64	8836.68	2251.64	2109.99	634.51	10638.27	5241.51	55404.71	6895.92	32836.05	3935.48	164605.34	73559.84
128	8068.76	1811.25	2526.10	470.81	5460.21	4941.66	52599.05	6513.09	32131.28	3651.14	163741.37	74344.46
256	8882.21	1345.90	4439.54	373.20	5091.79	4729.83	50710.08	6104.36	31393.03	3482.22	163013.41	71650.90
512	6232.44	1080.88	4690.61	327.11	4823.11	4603.51	55773.63	5810.05	34485.76	3390.11	162709.59	71152.92
1024	4877.16	1101.60	4715.18	464.03	5709.63	4621.73	55132.92	5773.00	34018.19	3248.82	162726.54	72229.09
2048	4960.82	1205.24	4887.10	552.88	5369.08	4582.58	54535.30	6106.01	33777.79	3470.37		
4096	4908.36	933.32	4818.26	560.45	5241.62	4601.73	54194.07	6010.59	33626.42	3417.14		
8192	5748.10	822.35	5698.12	651.24	4903.70	4605.96	56724.57	6134.33	35743.09	3371.99		

input or for embedded systems that cannot rely on GPU acceleration consistently. Table 33 and Table 34 summarize and highlight these behaviors.

As the data volume increases, particularly in the test scenario with 18,450 samples, the behavior reverses. In this case, the GPU becomes highly advantageous, significantly reducing inference time. On the Jetson J2, for example, the *Reduced* model

Table 33 – Comparasion and Acceleration GPU and CPU - 3,075 samples

Platform	Model	Best Time GPU (ms)	Best Time CPU (ms)	Batch Size GPU	Batch Size CPU	Accelaration from CPU	Acceleration Reduced to 3D-CNN in GPU	Acceleration Reduced to 3D-CNN in CPU
	Default	1125.41	1227.58	64	4096	1.09		
P0	Reduced	517.13	325.16	4096	512	0.63	2.85	3.24
	3D-CNN HybriSN	1472.08	1053.60	32	512	0.72		
	Default	751.68	499.25	64	64	0.66		
P1	Reduced	343.84	153.94	512	512	0.45	3.03	5.44
	3D-CNN HybriSN	1042.57	837.34	16	512	0.80		
	Default	6862.61	9195.21	256	256	1.34		
J0	Reduced	1341.78	1220.52	128	8192	0.91	6.91	22.19
	3D-CNN HybriSN	9277.89	27087.85	16	1024	2.92		
	Default	3630.72	-	128		1.57		
J1	Reduced	883.34	-	256		0.82	5.65	16.38
	3D-CNN HybriSN	4993.83	-	16		2.37		
	Default	3394.12	5701.87	128	256	1.68		
J2	Reduced	802.91	721.46	128	8192	0.90	5.15	16.38
	3D-CNN HybriSN	4133.71	11815.95	16	1024	2.86		

took 1547.95 ms on the GPU, compared to 3248.82 ms on the CPU, with the same batch size (512). This represents a 2.1$\times$ speedup. Overall, the GPU demonstrates clear superiority with larger datasets, making it well-suited for batch processing pipelines or applications that require continuous analysis of large volumes of data.

Table 34 – Comparation and Acceleration GPU and CPU - 18,450 samples

Platform	Model	Best Time GPU (ms)	Best Time CPU (ms)	Batch Size GPU	Batch Size CPU	Accelaration from CPU	Acceleration Reduced to 3D-CNN in GPU	Acceleration Reduced to 3D-CNN in CPU
	Default	1765.16	4877.16	64	8192	2.76		
P0	Reduced	573.98	822.35	512	4096	1.43	4.07	5.87
	3D-CNN HybriSN	2336.45	4823.11	64	512	2.06		
	Default	1241.63	2109.99	64	64	1.70		
P1	Reduced	462.50	327.11	512	512	0.71	3.86	3.86
	3D-CNN HybriSN	1786.77	4582.58	64	2048	2.56		
	Default	26356.90	50710.08	512	256	1.92		
J0	Reduced	3332.14	5773.00	512	1024	1.73	7.45	7.45
	3D-CNN HybriSN	24838.96	162709.59	16	512	6.55		
	Default	11006.61	-	512	-	2.85		
J1	Reduced	1510.09	-	1024	-	2.15	7.25	7.25
	3D-CNN HybriSN	10947.36	-	64	-	6.50		
	Default	10705.80	31393.03	512	256	2.93		
J2	Reduced	1547.95	3248.82	512	1024	2.10	6.34	6.34
	3D-CNN HybriSN	9809.35	71152.92	32	512	7.25		

Despite its energy and computational limitations compared to a notebook, the Jetson device exhibited surprisingly robust performance with the Reduced model. This is particularly noteworthy given that it represents a realistic scenario for onboard applications, such as satellites. Even on the Jetson platform, the Reduced model remained significantly faster than the 3D-CNN. These results reinforce the feasibility of deploying the Reduced model in embedded systems that operate in real-time and require a balance among accuracy, energy efficiency, and processing time.

The time required to perform deep learning model inference varies not only with the architecture used but also with the way the data is processed. In general, the computational cost per sample can be significantly reduced by processing in larger batches, especially on devices capable of parallel execution, such as GPUs. This section analyzes this behavior using the average inference time per sample metric across different

platforms. When processing a single input sample at a time (batch size equal to 1), inference time per sample is significantly higher. This is primarily due to the overhead associated with executing the model, such as tensor initialization, kernel invocation, and data transfer, being amortized over only one input, resulting in low computational efficiency.

This elevated processing cost stems from the underutilization of the communication bus when dispatching the inference task to the GPU. As shown in Table 35, a large number of samples, combined with the exploitation of parallelism via larger batch sizes, enables a substantial reduction in inference time. With the Reduced model, it is possible to achieve a throughput of up to 12,500 samples/s on the J2 configuration and 33,333 samples/s on the P1 configuration. It is important to note that these values already account for the entire processing pipeline, including data transfer, GPU computation, and inference result retrieval, within an application running on the Linux Ubuntu operating system.

Table 35 – Time comparasion to a one sample processing

Platform	Model	Average GPU Sample Time (ms)	GPU Calculated Time - 3,075 samples (ms)	GPU Calculated Time - 18,450 samples (ms)	CPU Calculated Time - 3,075 samples (ms)	CPU Calculated Time - 18,450 samples (ms)
P0	Default	1085.98	0.37	0.10	0.40	0.26
	Reduced	537.66	0.11	0.03	0.11	0.03
	3D-CNN HybriSN	1147.86	0.48	0.13	0.34	0.26
P1	Default	671.51	0.24	0.07	0.16	0.11
	Reduced	336.93	0.11	0.03	0.05	0.02
	3D-CNN HybriSN	768.04	0.34	0.10	0.27	0.25
J0	Default	2888.02	2.23	1.43	2.99	2.75
	Reduced	901.37	0.44	0.18	0.40	0.31
	3D-CNN HybriSN	3770.59	3.02	1.35	8.81	8.82
J1	Default	2085.56	1.18	0.60	-	-
	Reduced	709.96	0.29	0.08	-	-
	3D-CNN HybriSN	2346.49	1.62	0.59	-	-
J2	Default	1902.44	1.10	0.58	1.85	1.70
	Reduced	618.96	0.26	0.08	0.23	0.18
	3D-CNN HybriSN	1842.38	1.34	0.53	3.84	3.86

It is important to emphasize that the presented data reinforces the fact that single-sample inference is highly inefficient on GPUs, especially for more complex architectures. For applications that require very low response times per sample, such as real-time classification with sporadic inputs, the choice of both architecture and hardware platform becomes a critical factor. Additionally, CPUs can only be competitive in low-load scenarios, particularly when the GPU initialization latency becomes a bottleneck. Although CPU-based inference proves efficient in some cases, this advantage was only observed with the Reduced architecture and was limited to configurations with high-performance processors and GPUs, such as the notebook used in the tests. As evidenced in the J0, J1, and J2 platforms, which rely on processors with lower computational capacity, the GPU becomes essential to meet performance demands.

Based on the results, the Reduced model is the most efficient in terms of pro-

cessing time, whether executed on the CPU or GPU, on both the notebook and the Jetson platform. The proposed Reduced architecture proves ideal for real-time applications, especially in embedded systems with resource constraints, without compromising predictive capabilities. Furthermore, the acceleration achieved relative to the 3D-CNN model reinforces that optimized Transformer-based architectures can offer an excellent computational cost-benefit ratio, provided they are tuned with a focus on efficiency. In summary, the Reduced model represents a high-performance, low-complexity, and highly versatile alternative that can be employed in a wide range of scenarios, from cloud-based analysis to embedded devices with limited computing capacity.

Another important point to highlight is that the Reduced architecture was the only one capable of performing inference across all tested batch sizes. In contrast, the Default architecture, despite having just over 34 thousand parameters, was unable to handle a batch size of 8,192, and the HybridSN model failed with 8,192 on the GPU and 2,048 on the CPU. This final observation further exposes the limitations of CPU-based inference compared to GPU-based inference, where models are unable to effectively exploit higher degrees of parallelism.

5.3.2 GPU and Memory utilization

The percentage usage of the GPU, in the case how much of the GPU's capacity was effectively utilized, exhibits behavior directly correlated with the increase in batch size, for both the smaller dataset (3,075 samples), on Table 36, and the larger one (18,450 samples), on Table 37. This behavior is expected, as a larger number of samples processed simultaneously implies a greater computational load that can be parallelized, thereby favoring full GPU utilization.

When analyzing the inference behavior with 3,075 samples, it is notable that for the 1D models (Default and Reduced), GPU utilization begins at modest levels, such as 31% for P0 Default and only 16% for P0 Reduced when using batch size equal 1. From a batch size of 64 onward, a significant increase is observed, with P0 Default reaching 77% and Reduced reaching 33%. On the NVIDIA Jetson Orin Nano, across the J0, J1, and J2 configurations, utilization is more intense. For instance, with a batch size as small as 8, the J0 Reduced configuration already jumps to 49% GPU usage and reaches 100% from batch size 32 onwards.

When analyzing the inference behavior with 18,450 samples, a similar pattern is observed; however, the increase in GPU utilization occurs more rapidly.

Notably, in configuration P1, the Reduced model reaches 100% GPU usage already with a batch size of 8192, whereas it did not with 3,075 samples. On the Jetson platforms, 100% utilization is also achieved with smaller batch sizes, such as 256 or 512. By analyzing both behaviors, we can conclude that increased data volume accelerates GPU saturation. With more samples, internal parallelization is better exploited, and

Table 36 – GPU Utilization in % with 3,075 samples

Batch	P0 1D Default (%)	P0 1D Reduced (%)	P1 1D Default (%)	P1 1D Reduced (%)	P0 3D-CNN (%)	P1 3D-CNN (%)	J0 1D Default (%)	J0 1D Reduced (%)	J1 1D Default (%)	J1 1D Reduced (%)	J2 1D Default (%)	J2 1D Reduced (%)	J0 3D-CNN (%)	J1 3D-CNN (%)	J2 3D-CNN (%)
1	31	16	62	62	30	63	33	22	28	21	25	19	75	63	72
2	31	16	62	62	30	63	46	27	35	24	32	21	80	63	73
4	32	16	64	63	35	65	54	35	48	27	46	26	85	70	81
8	37	17	67	50	38	67	90	49	61	27	79	34	90	82	84
16	47	20	71	57	45	73	93	61	79	36	91	48	93	85	89
32	61	26	77	72	52	73	100	84	90	50	94	72	100	91	94
64	77	33	81	63	68	74	100	91	99	62	99	88	100	92	91
128	90	40	93	72	78	83	100	95	100	78	100	93	100	100	100
256	95	50	96	73	88	88	100	100	100	89	100	97	100	100	100
512	98	73	98	75	84	91	100	100	100	95	100	100	100	100	100
1024	99	85	99	90	72	77	100	100	100	99	100	100	100	100	100
2048	99	91	99	93	67	79	100	100	100	100	100	100	100	100	100
4096	100	94	100	97	80	80	100	100	100	100	100	100	100	100	100
8192	100	97	100	95	80	100	100	100	100	100	100	100	100	100	100

Table 37 – GPU Utilization in % with 18,450 samples

Batch	P0 1D Default (%)	P0 1D Reduced (%)	P1 1D Default (%)	P1 1D Reduced (%)	P0 3D-CNN (%)	P1 3D-CNN (%)	J0 1D Default (%)	J0 1D Reduced (%)	J1 1D Default (%)	J1 1D Reduced (%)	J2 1D Default (%)	J2 1D Reduced (%)	J0 3D-CNN (%)	J1 3D-CNN (%)	J2 3D-CNN (%)
1	31	17	60	60	35	63	34	23	28	21	25	19	75	66	71
2	33	17	62	63	39	63	46	25	34	22	30	22	80	63	72
4	33	18	67	63	41	65	58	37	49	27	46	26	85	70	80
8	38	21	69	50	40	70	89	50	62	35	82	33	89	80	84
16	46	21	73	67	45	75	99	58	81	51	88	46	93	88	88
32	62	26	74	73	55	74	100	85	90	62	94	74	91	93	91
64	76	34	83	65	63	77	100	90	99	75	99	88	100	90	93
128	90	41	93	69	83	87	100	99	100	90	100	94	100	100	100
256	96	54	96	77	88	88	100	100	100	95	100	96	100	100	100
512	97	70	98	77	91	92	100	100	100	100	100	100	100	100	100
1024	99	84	99	89	64	80	100	100	100	100	100	100	100	100	100
2048	99	91	100	93	84	87	100	100	100	100	100	100	100	100	100
4096	100	93	100	97	100	100	100	100	100	100	100	100	100	100	100
8192	100	96	100	100				100		100		100			

GPU usage reaches its maximum more quickly. This effect is even more pronounced in the Reduced models, whose lightweight structure allows better scalability with lower per-sample overhead.

As with GPU utilization, memory demand also increases with batch size, but in a distinct manner: while GPU usage tends to saturate near 100%, memory consumption grows continuously and almost linearly with batch size. This reflects the need to maintain more data and tensors in parallel during inference.

For 3,075 samples (Table 38), memory usage starts at low values. For instance, the Default architecture consumes only 0.145 GiB with a batch size equal to 1 when executed on platform P0. However, it is noteworthy that the Reduced model is consistently lighter during execution, ranging from 0.145 to 0.207 GiB, even with a batch size equal to 512, demonstrating its suitability for memory-constrained systems. It is also important to highlight the memory footprint of the 3D-CNN architecture compared to the Reduced model. The 3D-CNN requires significantly more memory, even with small batches, for example, 0.643 GiB at batch size 1, increasing to 6.041 GiB at batch size 2048.

When analyzing memory demand for inference with the 18,450-sample dataset (Table 39), the difference between the models remains consistent with that observed in the 3,075-sample dataset. The Reduced model in the J2 configuration consumes only 0.141 GiB with a batch size of 512, while the 3D-CNN consumes 3.06 GiB. In the P1 configuration, even with a large batch size. For instance, 1024, the Reduced model

Table 38 – GPU Memory Utilization - 3,075 samples

Batch	P0 1D Default (GiB)	P0 1D Reduced (GiB)	P1 1D Default (GiB)	P1 1D Reduced (GiB)	P0 3D-CNN (GiB)	P1 3D-CNN (GiB)	J0 1D Default (GiB)	J0 1D Reduced (GiB)	J1 1D Default (GiB)	J1 1D Reduced (GiB)	J2 1D Default (GiB)	J2 1D Reduced (GiB)	J0 3D-CNN (GiB)	J1 3D-CNN (GiB)	J2 3D-CNN (GiB)
1	0.145	0.145	0.145	0.145	0.522	0.518	0.055	0.064	0.077	0.077	0.078	0.077	0.363	0.376	0.372
2	0.145	0.145	0.145	0.145	0.643	0.647	0.055	0.064	0.077	0.077	0.078	0.077	0.363	0.376	0.376
4	0.149	0.145	0.149	0.145	0.647	0.647	0.068	0.064	0.081	0.077	0.082	0.077	0.490	0.503	0.503
8	0.149	0.145	0.149	0.145	0.643	0.643	0.068	0.064	0.081	0.077	0.082	0.077	0.490	0.492	0.503
16	0.156	0.145	0.156	0.145	0.647	0.643	0.076	0.064	0.089	0.081	0.090	0.077	0.489	0.504	0.504
32	0.172	0.149	0.172	0.149	0.643	0.647	0.092	0.068	0.105	0.077	0.106	0.081	0.617	0.632	0.628
64	0.176	0.156	0.176	0.156	1.147	1.143	0.096	0.076	0.109	0.081	0.110	0.089	0.612	0.627	0.631
128	0.207	0.172	0.207	0.172	1.143	1.147	0.128	0.077	0.141	0.089	0.142	0.089	1.062	1.060	1.060
256	0.270	0.176	0.270	0.176	2.178	2.143	0.192	0.096	0.205	0.101	0.206	0.109	2.162	2.060	2.060
512	0.395	0.207	0.395	0.207	4.209	4.205	0.320	0.128	0.333	0.105	0.334	0.141	3.462	3.460	3.660
1024	0.645	0.270	0.645	0.270	5.736	5.736	0.576	0.192	0.589	0.137	0.590	0.205	3.362	3.660	3.660
2048	1.145	0.395	1.145	0.395	6.041	6.041	1.069	0.320	1.053	0.205	1.062	0.333	3.462	3.460	3.660
4096	2.145	0.645	2.145	0.645	6.041	6.041	2.069	0.320	2.053	0.333	2.062	0.333	3.562	3.560	3.560
8192	4.149	1.145	4.149	0.395	6.041	6.041	2.069	0.320	2.053	0.333	2.062	0.333	3.562	3.660	3.560

occupies only 0.270 GiB, compared to 5.74 GiB required by the 3D-CNN.

The Reduced architecture not only requires less memory per batch but also maintains this efficiency even as the number of samples and batch sizes increases. In contrast, the 3D-CNN architecture exhibits a steep increase in memory demand, which may become a critical limitation in embedded systems with limited RAM.

Table 39 – GPU Memory Utilization - 18,450 samples

Batch	P0 1D Default (GiB)	P0 1D Reduced (GiB)	P1 1D Default (GiB)	P1 1D Reduced (GiB)	P0 3D-CNN (GiB)	P1 3D-CNN (GiB)	J0 1D Default (GiB)	J0 1D Reduced (GiB)	J1 1D Default (GiB)	J1 1D Reduced (GiB)	J2 1D Default (GiB)	J2 1D Reduced (GiB)	J0 3D-CNN (GiB)	J1 3D-CNN (GiB)	J2 3D-CNN (GiB)
1	0.145	0.145	0.145	0.145	0.522	0.522	0.064	0.065	0.078	0.077	0.078	0.078	0.363	0.376	0.372
2	0.145	0.145	0.145	0.145	0.643	0.643	0.064	0.065	0.078	0.077	0.078	0.078	0.363	0.376	0.372
4	0.149	0.145	0.149	0.145	0.643	0.643	0.068	0.065	0.082	0.077	0.082	0.078	0.491	0.501	0.504
8	0.149	0.145	0.149	0.145	0.643	0.643	0.068	0.065	0.082	0.077	0.082	0.078	0.490	0.494	0.504
16	0.156	0.145	0.156	0.145	0.643	0.643	0.076	0.064	0.090	0.077	0.090	0.077	0.489	0.504	0.500
32	0.172	0.149	0.172	0.149	0.647	0.647	0.093	0.068	0.106	0.084	0.106	0.081	0.616	0.628	0.632
64	0.176	0.156	0.176	0.156	1.147	1.147	0.097	0.079	0.110	0.098	0.110	0.081	0.612	0.632	0.632
128	0.270	0.172	0.207	0.172	1.143	1.147	0.129	0.079	0.142	0.114	0.151	0.089	1.062	1.060	1.160
256	0.270	0.176	0.270	0.176	2.143	2.143	0.193	0.096	0.206	0.118	0.215	0.109	2.162	2.060	2.160
512	0.395	0.207	0.395	0.207	4.209	4.209	0.321	0.128	0.334	0.150	0.343	0.141	2.962	3.660	3.060
1024	0.645	0.270	0.645	0.270	5.740	5.740	1.069	0.192	0.590	0.208	0.590	0.205	3.462	3.560	3.360
2048	1.145	0.395	1.145	0.395	5.740	6.041	1.069	0.320	1.066	0.333	1.062	0.333	3.462	3.560	3.360
4096	2.145	0.645	2.145	0.645	6.041	6.041	2.069	0.576	2.066	0.589	2.062	0.589	3.562	3.460	3.460
8192	4.149	1.145	4.149	1.145				1.067		1.062		1.064			

Even under peak GPU usage conditions (100%), the Reduced model remains stable and lightweight in terms of memory consumption, consolidating its profile as a highly efficient architecture, as shown in Table 40. Furthermore, it can be concluded that GPU utilization increases with batch size until it reaches saturation (100%), especially with lightweight and optimized models. Memory usage also grows proportionally with batch size; however, the Reduced model demonstrates significantly lower memory demands compared to the Default model and, notably, the 3D-CNN. Finally, across all scenarios, the Reduced model achieves high GPU utilization with efficient memory usage, making it ideal for deployment in computationally constrained environments such as the Jetson platform.

5.3.3 Power and Current Comparasion

Based on the following tables, it is possible to analyze power and current consumption during model inference, enabling the evaluation of the proposed Reduced model and the comparison of performance across platforms P0, P1, J0, J1, and J2. It

Table 40 – GPU and memory Utilization to lowest inference time

Platform	Model	Best Time GPU 3,075 (ms)	Batch Size GPU 3,075	GPU Utilization 3,075 (%)	Memory Utilization 3,075 (GiB)	Best Time GPU 18,450 (ms)	Batch Size GPU 18,450	GPU Utilization 18,450 (%)	Memory Utilization 18,450 (GiB)
P0	Default	1125.41	64	77	0.176	1765.16	64	76	0.176
	Reduced	517.13	4096	73	0.207	573.98	512	70	0.207
	3D-CNN HybriSN	1472.08	32	52	0.643	2336.45	64	63	1.147
P1	Default	751.68	64	81	0.176	1241.63	64	83	0.176
	Reduced	343.84	512	75	0.207	462.50	512	77	0.207
	3D-CNN HybriSN	1042.57	16	73	0.643	1786.77	64	77	1.147
J0	Default	6862.61	256	100	0.192	26356.90	512	100	0.321
	Reduced	1341.78	128	95	0.077	3332.14	512	100	0.128
	3D-CNN HybriSN	9277.89	16	93	0.504	24838.96	16	93	0.489
J1	Default	3630.72	128	100	0.141	11006.61	512	100	0.334
	Reduced	883.34	256	89	0.101	1510.09	1024	100	0.208
	3D-CNN HybriSN	4993.83	16	85	0.504	10947.36	64	90	0.632
J2	Default	3394.12	128	100	0.142	10705.80	512	100	0.343
	Reduced	802.91	128	93	0.089	1547.95	512	100	0.141
	3D-CNN HybriSN	4133.71	16	89	0.504	9809.35	32	91	0.632

also allows for an examination of the behavior of the Reduced model in comparison to the 3D-CNN HybridSN model.

Regarding GPU power consumption, on platforms P0 and P1, the GPU power usage for the Reduced model reaches 81–87 W in the best inference scenarios (batch size equal to 512), while the *Default* model consumes up to 85 W, and the 3D-CNN HybridSN varies from 52 W to 98 W. On the Jetson Orin Nano (J0, J1, J2), GPU power for the Reduced model remains extremely low, ranging from 0.7 W to 3.8 W, even at large batch sizes such as 1024 or 512. Table 41 and Table 42 present the evolution of different models across various platforms as the batch size varies.

Table 41 – GPU power (in Watts) comparison between platforms to 3,075 samples.

Batch	P0 Default	P0 Reduced	P1 Default	P1 Reduced	P0 3D-CNN	P1 3D-CNN	J0 Default	J0 Reduced	J1 Default	J1 Reduced	J2 Default	J2 Reduced	J0 3D-CNN	J1 3D-CNN	J2 3D-CNN
1	17.0	14.0	26.0	24.0	18.0	38.0	0.8	0.7	0.9	0.9	1.8	1.7	1.4	1.6	3.1
2	17.0	14.0	29.0	25.0	18.0	40.0	0.9	0.8	1.0	0.9	1.9	1.8	1.5	2.0	3.4
4	19.0	14.0	33.0	27.0	23.0	48.0	1.1	0.8	1.1	0.9	2.3	1.8	1.5	2.6	3.8
8	22.0	15.0	41.0	29.0	31.0	59.0	1.3	0.9	1.6	0.9	3.0	2.0	1.6	3.1	4.0
16	38.0	16.0	51.0	35.0	52.0	80.0	1.4	1.2	2.5	1.0	3.3	2.5	1.7	3.4	4.3
32	54.0	18.0	63.0	46.0	64.0	91.0	1.4	1.4	2.7	1.2	3.4	3.2	1.7	3.5	4.4
64	79.0	24.0	85.0	49.0	78.0	95.0	1.4	1.4	2.8	1.7	3.5	3.5	1.8	3.6	4.5
128	79.0	42.0	81.0	63.0	81.0	93.0	1.4	1.4	2.9	2.6	3.5	3.6	1.8	3.7	4.6
256	80.0	62.0	81.0	78.0	83.0	93.0	1.4	1.4	2.9	2.9	3.5	3.7	1.8	3.9	4.7
512	81.0	81.0	81.0	85.0	88.0	95.0	1.4	1.4	2.9	3.0	3.6	3.7	1.8	3.9	4.7
1024	82.0	81.0	83.0	87.0	77.0	86.0	1.4	1.4	2.9	3.1	3.4	3.8	2.0	3.7	4.4
2048	82.0	83.0	82.0	85.0	73.0	87.0	1.4	1.5	2.9	3.1	3.5	3.8	2.0	4.1	4.5
4096	83.0	84.0	83.0	85.0	80.0	85.0	1.4	1.5	2.9	3.1	3.6	3.8	2.0	4.3	5.1
8192	82.0	84.0	83.0	86.0	83.0	84.0	1.4	1.5	2.9	3.2	3.6	3.8	2.0	4.3	5.2

Table 42 – GPU power (in Watts) comparison between platforms to 18,450 samples.

Batch	P0 Default	P0 Reduced	P1 Default	P1 Reduced	P0 3D-CNN	P1 3D-CNN	J0 Default	J0 Reduced	J1 Default	J1 Reduced	J2 Default	J2 Reduced	J0 3D-CNN	J1 3D-CNN	J2 3D-CNN
1	17.0	14.0	26.0	24.0	18.0	37.0	0.8	0.7	0.9	0.9	1.8	1.7	1.4	1.6	3.1
2	18.0	14.0	28.0	26.0	25.0	39.0	0.9	0.8	0.9	0.9	1.9	1.8	1.5	2.2	3.4
4	19.0	14.0	34.0	28.0	25.0	47.0	1.1	0.8	1.1	0.9	2.3	1.9	1.6	2.6	3.8
8	23.0	15.0	41.0	29.0	33.0	57.0	1.3	0.9	1.5	1.0	3.1	2.0	1.6	3.2	4.0
16	37.0	16.0	51.0	35.0	53.0	82.0	1.4	1.2	2.5	1.2	3.3	2.4	1.7	3.5	4.2
32	55.0	18.0	64.0	45.0	67.0	86.0	1.4	1.4	2.7	1.7	3.4	3.3	1.7	3.6	4.4
64	78.0	24.0	85.0	49.0	80.0	98.0	1.4	1.4	2.8	2.6	3.5	3.5	1.7	3.7	4.5
128	79.0	42.0	82.0	63.0	88.0	92.0	1.4	1.4	2.9	2.9	3.5	3.6	1.8	3.8	4.7
256	80.0	61.0	81.0	77.0	92.0	91.0	1.4	1.4	2.9	3.0	3.6	3.7	1.8	3.8	4.6
512	81.0	81.0	81.0	87.0	87.0	94.0	1.4	1.4	3.0	3.1	3.6	3.8	1.8	3.8	4.7
1024	83.0	81.0	81.0	86.0	74.0	85.0	1.4	1.4	2.9	3.1	3.5	3.8	2.1	3.8	4.7
2048	83.0	81.0	83.0	86.0	83.0	85.0	1.4	1.4	2.9	3.2	3.5	3.8	2.1	3.9	5.1
4096	84.0	84.0	83.0	87.0	88.0	95.0	1.4	1.5	3.0	3.2	3.6	3.8	2.0	5.1	5.5
8192	84.0	86.0	83.0	87.0	-	-	-	1.5	-	3.1	-	3.8	-	-	-

Figure 31 and Figure 32 visually illustrate the trend of increasing GPU power consumption as the batch size increases. It is noteworthy that this behavior remains consistent when comparing the inference of 3,075 and 18,450 samples.

Figure 31 – Comparison of GPU power consumption during inference on the Test Notebook with (a) 3,075 samples and (b) 18,450 samples.

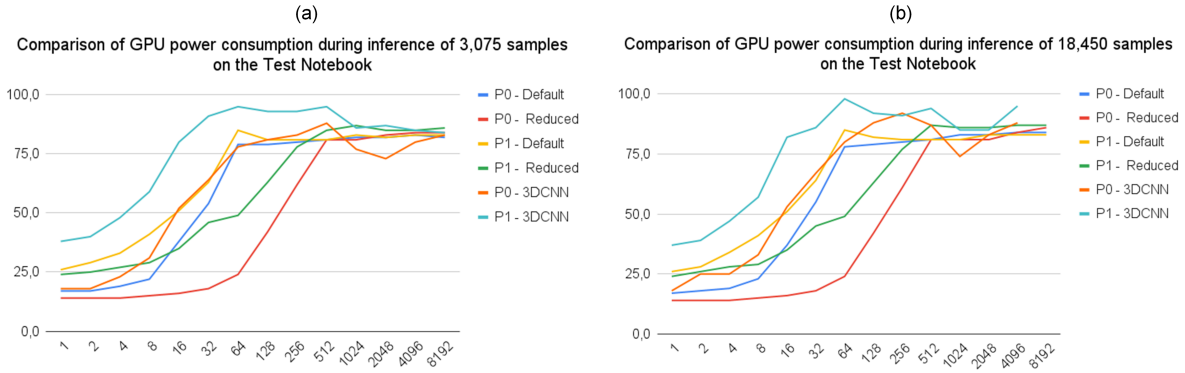
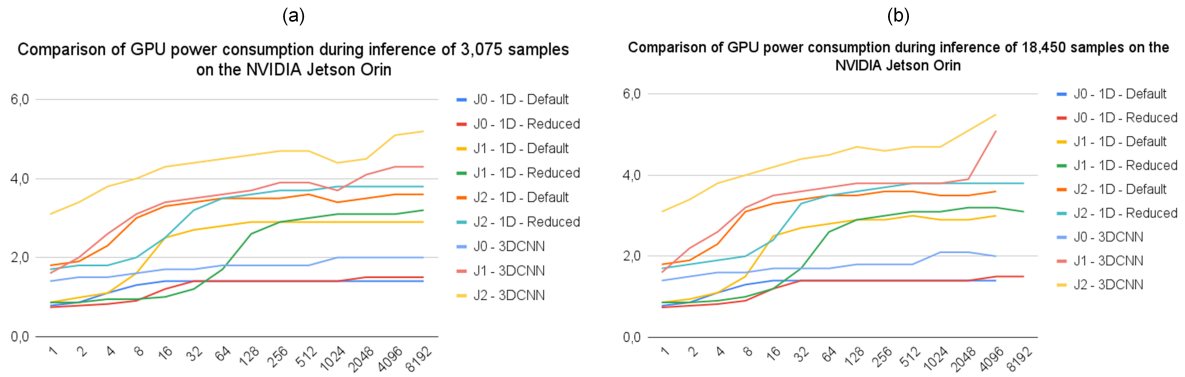


Figure 32 – Comparison of GPU power consumption during inference on the NVIDIA Jetson Orin Nano with (a) 3,075 samples and (b) 18,450 samples.



This indicates a stable performance pattern, suggesting that the power consumption scaling is more closely related to batch size than to the absolute volume of data being processed. This consistent behavior can also be observed when comparing the performance and power consumption of models with lower computational demands to those with higher power and performance requirements. Figure 33 and Figure 34 illustrate this behavior, highlighting the scalability pattern across both energy-efficient and high-performance model configurations.

In tests, the NVIDIA Jetson Orin Nano stood out for very low, stable electrical current levels, even during relatively demanding tasks such as the inference of 18,450 samples using neural networks. For the Reduced model, the total system current remained between 1.4 A and 2.1 A, even with large data batches (batch sizes of 512 or 1024). These values can be observed in Table 43 and Table 44.

Figure 33 – Comparison of GPU power consumption during inference in the power-saving versions of the platforms with (a) 3,075 and (b) 18,450 samples.

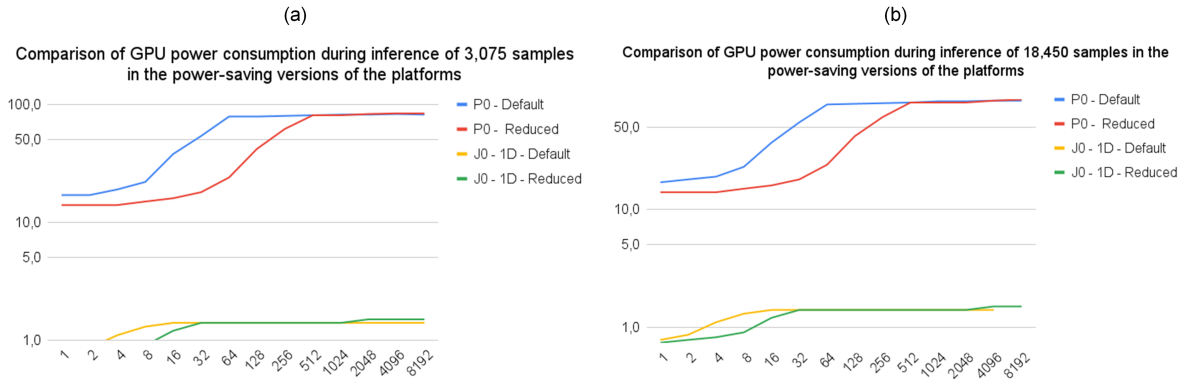


Figure 34 – Comparison of GPU power consumption during inference in the max performance versions of the platforms with (a) 3,075 and (b) 18,450 samples.

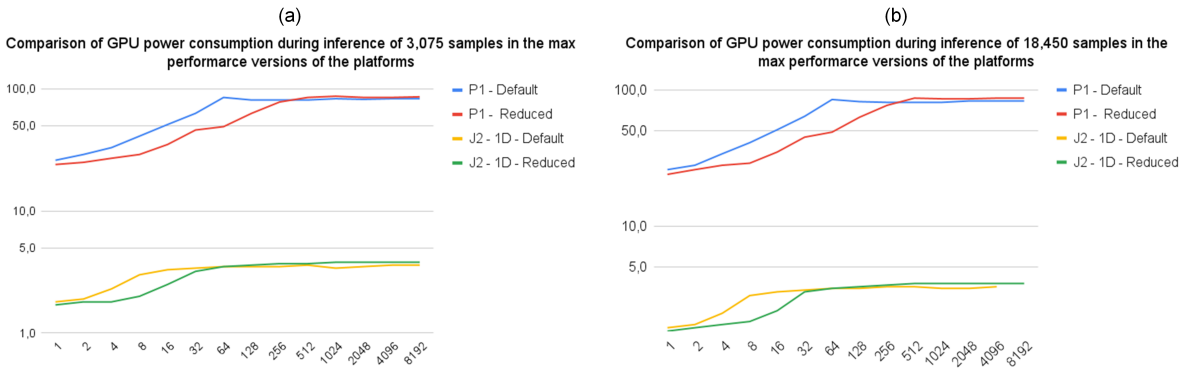


Table 43 – Electrical Power and Current to the lowest inference time.

Plataform	Model	Best Time GPU 3,075 (ms)	Batch Size GPU 3,075	GPU Power 3,075 (W)	GPU Current 3,075 (A)	Best Time GPU 18,450 (ms)	Batch Size 18,450 GPU	GPU Power 18,450 (W)	GPU Current 18,450 (A)
P0	Default	1125.41	64	79.0	-	1765.16	64	78.0	-
	Reduced	343.84	512	81.0	-	573.98	512	81.0	-
	3D-CNN HybriSN	1472.08	32	52.0	-	2336.45	64	80.0	-
P1	Default	751.68	64	85.0	-	1241.63	64	85.0	-
	Reduced	343.84	512	85.0	-	462.50	512	87.0	-
	3D-CNN HybriSN	1042.57	16	80.0	-	1786.77	64	98.0	-
J0	Default	6862.61	256	1.4	0.264	26356.90	512	1.4	0.272
	Reduced	1341.78	128	1.4	0.280	3332.14	512	1.4	0.280
	3D-CNN HybriSN	9277.89	16	1.7	0.336	24838.96	16	1.7	0.328
J1	Default	3630.72	128	2.9	0.568	11006.61	512	3.0	0.584
	Reduced	883.34	256	2.9	0.560	1510.09	1024	3.1	0.616
	3D-CNN HybriSN	4993.83	16	3.4	0.656	10947.36	64	3.7	0.344
J2	Default	3394.12	128	3.5	0.688	10705.80	512	3.6	0.696
	Reduced	802.91	128	3.6	0.704	1547.95	512	3.8	0.736
	3D-CNN HybriSN	4133.71	16	4.3	0.832	9809.35	32	4.4	0.864

The current consumed by the GPU alone ranged from 0.560 to 0.736 A, depending on the model and network complexity. Even in the most demanding scenario (Reduced model with batch size 1024), the current did not exceed 0.74 A, a value considered very low for inference tasks involving deep neural networks.

This behavior indicates that the NVIDIA Jetson can perform inference on opti-

Table 44 – NVIDIA Jetson input electrical power and current comparison to the lowest inference time.

Plataform	Model	Best Time GPU 3,075 (ms)	Batch Size GPU 3,075	Input Power 3,075 (W)	Input Current 3,075 (A)	Best Time GPU 18,450 (ms)	Batch Size GPU 18,450	Input Power 18,450 (W)	Input Current 18,450 (A)
J0	Default	6862.61	256	7.2	1.4	26356.90	512	7.2	1.4
	Reduced	1341.78	128	7.1	1.4	3332.14	512	7.3	1.4
	3D-CNN HybriSN	9277.89	16	7.6	1.5	24838.96	16	7.5	1.5
J1	Default	3630.72	128	10.1	1.9	11006.61	512	10.2	2.0
	Reduced	883.34	256	9.6	1.9	1510.09	1024	10.3	2.0
	3D-CNN HybriSN	4993.83	16	10.1	2.0	10947.36	64	11.0	2.1
J2	Default	3394.12	128	10.7	2.1	10705.80	512	10.9	2.1
	Reduced	802.91	128	10.5	2.0	1547.95	512	11.0	2.1
	3D-CNN HybriSN	4133.71	16	11.2	2.2	9809.35	32	11.5	2.3

mized models with low energy consumption, which is ideal for embedded applications such as satellite systems.

5.3.4 General comparison

The following analysis compares the performance of the Default, Reduced, and 3D-CNN HybriSN models in terms of inference time, acceleration, computational resource utilization (GPU, memory), power consumption, and electric current across two datasets: a small one (3,075 samples) and a large one (18,450 samples). The focus is on the best GPU inference times for each model and platform, with emphasis on the energy efficiency and scalability of the Reduced model. This analysis is supported by the summary data presented in Table 45 and Table 46.

Regarding GPU processing time, the best absolute time was achieved by the Reduced model on the P0/P1 notebook with a dedicated GPU, with a batch size of 512, registering 343.84 ms. On the Jetson Orin Nano (J2), the inference time of the same model was 802.91 ms, approximately 2.34$\times$ slower, which, given the embedded nature of the hardware, represents excellent relative performance.

When compared to the 3D-CNN, on the Jetson J2, the Reduced model was 5.15$\times$ faster than the 3D-CNN (802.91 ms vs. 4,133.71 ms). On CPU, the difference is even more pronounced, with the Reduced model being 16.38$\times$ faster, highlighting the significant advantage of the optimized model in resource-constrained environments.

In terms of energy efficiency, GPU consumption for the Reduced model on Jetson ranged from 3.6 W (J2) down to 1.4 W (7 W power mode, J0), with current levels between 0.280 A and 0.704 A. GPU memory usage was extremely low, with values below 0.21 GiB, indicating a compact, resource-efficient architecture. GPU utilization on Jetson reached 100% (J0 and J1), confirming efficient usage of the available computational capacity.

Finally, in a comparison between the NVIDIA Jetson and the notebook (P1), although the notebook achieved a lower inference time (343.84 ms), its energy consumption was substantially higher, with the GPU drawing 85 W. Compared to the NVIDIA Jetson (3.6 W), the notebook consumed over 23 times more power for only a 2.34$\times$

reduction in inference time.

For the 18,450-sample dataset, the lowest inference time was again achieved by the Reduced model on the P1 notebook, with 462.50 ms. On the NVIDIA Jetson J2, the same model reached 1,510.09 ms, maintaining good scalability with only a 3.27× increase in time, despite consuming nearly 8 times less total power. Compared to the 3D-CNN, the Reduced model showed a significant advantage on Jetson, being 6.5× faster on GPU and 21.9× faster on CPU.

Even compared to its most power-constrained version (J0 at 7 W), the Reduced model still achieved a 3.08× speedup over the 3D-CNN while consuming less than half the energy. GPU power consumption on Jetson, while processing 18,450 samples with the Reduced model, ranged from 3.1 W to 1.4 W, with current levels varying from 0.280 A to 0.616 A.

Table 45 – Comparison between platforms to lowest inference time to 3,075 samples.

Platform	Model	Best Time GPU (ms)	Best Time CPU (ms)	Batch Size GPU	Batch Size CPU	Accelaration from CPU	Acceleration Reduced to 3D-CNN in GPU	Acceleration Reduced to 3D-CNN in CPU	GPU Utilization (%)	Memory Utilization (GiB)	GPU Power (W)	GPU Current (A)	Input Power (W)	Input Current (A)
P0	Default	1125.41	1227.58	64	4096	1.09	-	-	77	0.176	79.0	-	-	-
	Reduced	343.84	325.16	512	512	0.95	4.28	3.24	73	0.207	81.0	-	-	-
	3D-CNN													
	HybriSN	1472.08	1053.60	32	512	0.72	-	-	52	0.643	52.0	-	-	-
P1	Default	751.68	499.25	64	64	0.66	-	-	81	0.176	85.0	-	-	-
	Reduced	343.84	153.94	512	512	0.45	3.03	5.44	75	0.207	85.0	-	-	-
	3D-CNN													
	HybriSN	1042.57	837.34	16	512	0.80	-	-	73	0.643	80.0	-	-	-
J0	Default	6862.61	9195.21	256	256	1.34	-	-	100	0.192	1.4	0.264	7.2	1.4
	Reduced	1341.78	1220.52	128	8192	0.91	6.91	22.19	95	0.077	1.4	0.280	7.1	1.4
	3D-CNN													
	HybriSN	9277.89	27087.85	16	1024	2.92	-	-	93	0.504	1.7	0.336	7.6	1.5
J1	Default	3630.72	-	128	-	1.57	-	-	100	0.141	2.9	0.568	10.1	1.9
	Reduced	883.34	-	256	-	0.82	5.65	16.38	89	0.101	2.9	0.560	9.6	1.9
	3D-CNN													
	HybriSN	4993.83	-	16	-	2.37	-	-	85	0.504	3.4	0.656	10.1	2.0
J2	Default	3394.12	5701.87	128	256	1.68	-	-	100	0.142	3.5	0.688	10.7	2.1
	Reduced	802.91	721.46	128	8192	0.90	5.15	16.38	93	0.089	3.6	0.704	10.5	2.0
	3D-CNN													
	HybriSN	4133.71	11815.95	16	1024	2.86	-	-	89	0.504	4.3	0.832	11.2	2.2
Best value All (ms)		343.84	153.94	-	-	-	-	-	75	0.207	85.0			
Best value Jetson (ms)		802.91	721.46	-	-	-	-	-	93	0.089	3.6			
Difference		2.34	4.69	-	-	-	-	-	1.24	2.33	23.61			
Best value Reduced Jetson (ms)		802.91	721.46	-	-	-	-	-	93.00	0.09	3.60	0.70	10.50	2.00
Best value 3D-CNN Jetson (ms)		4133.71	11815.95	-	-	-	-	-	89.00	0.50	4.30	0.83	11.20	2.20
Difference		5.15	16.38	-	-	-	-	-	1.04	5.66	1.19	1.18	1.07	1.10
Best value Reduced Jetson (ms)		802.91	721.46	-	-	-	-	-	93	0.089	3.6	0.704	10.5	2.0
Best value Reduced Jetson 7W (ms)		1341.78	1220.52	-	-	-	-	-	95	0.077	1.4	0.280	7.1	1.4
Difference		1.67	1.69	-	-	-	-	-	1.02	1.16	2.57	2.51	1.48	1.43
Best value Reduced Jetson 7W (ms)		1341.78	1220.52	-	-	-	-	-	95	0.077	1.4	0.280	7.1	1.4
Best value 3D-CNN Jetson (ms)		4133.71	11815.95	-	-	-	-	-	89	0.504	4.3	0.832	11.2	2.2
Difference		3.08	9.68	-	-	-	-	-	0.94	6.55	3.07	2.97	1.58	1.57

In contrast, the 3D-CNN model on Jetson consumed up to 4.4 W with 0.864 A of current, and inference times around 9,809 ms, which is extremely inefficient in both time and energy. Finally, GPU memory usage by the Reduced model remained below 0.21 GiB on Jetson, compared to up to 0.632 GiB for the 3D-CNN model. GPU utilization for the Reduced model reached 100% on the Jetson, demonstrating that the model fully leverages the device's computational capacity even in low-power scenarios.

One notable point regarding the 7 W mode on the NVIDIA Jetson is that, even under energy constraints (7 W mode on J0), the Reduced model was able to process 18,450 samples in 3,332.14 ms on the GPU, with a power consumption of only 1.4 W and a current of 0.280 A. Despite this low resource demand, it was still 3× faster than the 3D-CNN, which consumed more than twice the energy. This result is highly signif-

Table 46 – Comparison between platforms to lowest inference time to 18,450 samples.

Platform	Model	Best Time GPU (ms)	Best Time CPU (ms)	Batch Size GPU	Batch Size CPU	Accelaration from CPU	Acceleration Reduced to 3D-CNN in GPU	Acceleration Reduced to 3D-CNN in CPU	GPU Utilization (%)	Memory Utilization (GiB)	GPU Power (W)	GPU Current (A)	Input Power (W)	Input Current (A)
P0	Default	1765,16	4877,16	64	1024	2,76	-	-	76	0,176	78,0	-	-	-
	Reduced	573,98	517,13	512	4096	0,90	4,07	9,33	70	0,207	81,0	-	-	-
	3D-CNN HybriSN	2336,45	4823,11	64	512	2,06	-	-	63	1,147	80,0	-	-	-
P1	Default	1241,63	2109,99	64	64	1,70	-	-	83	0,176	85,0	-	-	-
	Reduced	462,50	327,11	512	512	0,71	3,86	14,01	77	0,207	87,0	-	-	-
	3D-CNN HybriSN	1786,77	4582,58	64	2048	2,56	-	-	77	1,147	98,0	-	-	-
J0	Default	26356,90	50710,08	512	256	1,92	-	-	100	0,321	1,4	0,272	7,2	1,4
	Reduced	3332,14	5773,00	512	1024	1,73	7,45	7,45	100	0,128	1,4	0,280	7,3	1,4
	3D-CNN HybriSN	24838,96	162709,59	16	512	6,55	-	-	93	0,489	1,7	0,328	7,5	1,5
J1	Default	11006,61	-	512	-	2,85	-	-	100	0,334	3,0	0,584	10,2	2,0
	Reduced	1510,09	-	1024	-	2,15	7,25	7,25	100	0,208	3,1	0,616	10,3	2,0
	3D-CNN HybriSN	10947,36	-	64	-	6,50	-	-	90	0,632	3,7	0,344	11,0	2,1
J2	Default	10705,80	31393,03	512	256	2,93	-	-	100	0,343	3,6	0,696	10,9	2,1
	Reduced	1547,95	3248,82	512	1024	2,10	6,34	6,34	100	0,141	3,8	0,736	11,0	2,1
	3D-CNN HybriSN	9809,35	71152,92	32	512	7,25	-	-	91	0,632	4,4	0,864	11,5	2,3
Best value All (ms)		462,50	327,11	-	-	-	-	-	77	0,207	87,0			
Best value Jetson (ms)		1510,09	3248,82	-	-	-	-	-	100	0,141	3,1			
Difference		3,27	9,93	-	-	-	-	-	1,30	1,47	28,06			
Best value Reduced Jetson (ms)		1510,09	3248,82	-	-	-	-	-	100,00	0,21	3,10	0,62	10,30	2,00
Best value 3D-CNN Jetson (ms)		9809,35	71152,92	-	-	-	-	-	91,00	0,63	4,40	0,86	11,50	2,30
Difference		6,50	21,90	-	-	-	-	-	1,10	3,04	1,42	1,40	1,12	1,15
Best value Reduced Jetson (ms)		1510,09	3248,82	-	-	-	-	-	100	0,208	3,1	0,616	10,3	2,0
Best value Reduced Jetson 7W (ms)		3332,14	5773,00	-	-	-	-	-	100	0,128	1,4	0,280	7,3	1,4
Difference		2,21	1,78	-	-	-	-	-	1,00	1,63	2,21	2,20	1,41	1,43
Best value Reduced Jetson 7W (ms)		1510,09	5773,00	-	-	-	-	-	100	0,128	1,4	0,280	7,3	1,4
Best value 3D-CNN Jetson (ms)		9809,35	71152,92	-	-	-	-	-	91	0,632	4,4	0,864	11,5	2,3
Difference		6,50	12,33	-	-	-	-	-	0,91	4,94	3,14	3,09	1,58	1,64

icant, as it demonstrates that the Reduced model maintains competitive performance even under severe energy constraints.

In conclusion, the Reduced model exhibits an exceptional balance between performance and energy efficiency. Even under strict energy limitations (Jetson operating in 7 W mode), it maintains competitive inference times while drastically reducing power and current consumption compared to the Default and 3D-CNN models.

Furthermore, its ability to handle large batch sizes with minimal memory usage reinforces its profile as an ideal solution for embedded systems, mobile applications, and scenarios with energy or computational constraints, such as satellite applications.

The efficiency gap widens with increasing sample volumes, confirming that the Reduced model is not only more energy-efficient but also more scalable and sustainable for large-scale applications.

6 CONCLUSIONS

The demand for computational power in space systems, such as remote sensing platforms, is steadily increasing. This growth drives advances in processor technologies that accelerate complex, computationally intensive tasks in hardware. In this context, hardware accelerators are employed to handle high computational demands, such as those found in artificial ANNs and the large volumes of data inherent in HSIs. Applications involving ANNs and HSIs leverage hardware acceleration to provide autonomy and intelligence for space systems intended for remote sensing. Additionally, hardware acceleration enhances performance and reduces energy consumption.

In line with this, the focus of this thesis was to investigate ANN architectures designed for hardware acceleration, offering trade-offs among accuracy, processing time, and memory usage.

From both experimental results and an extensive literature review, it was found that the Transformer architecture combined with component reduction and redundancy techniques can achieve the desired balance of accuracy and resource efficiency using only spectral information from HSIs. While Transformer architectures are widely adopted for spectral-spatial classification, the literature review revealed a lack of studies exploring reduced Transformer models focused on spectral classification and hardware acceleration. It is worth noting that the Transformer adopted in this work represents one possible variation among many. Architectural modifications are commonly employed in the literature to improve specific performance metrics.

A comparative analysis involving 1D-, 2D-, and 3D-CNNs, LSTM, and Transformer-based architectures enabled the characterization of each model's performance in terms of accuracy, inference time, memory usage, and energy consumption. The results allowed for clear answers to the research questions. The evaluation showed that models based solely on spectral data, such as 1D-CNN and Transformer, can achieve comparable or even superior accuracy to spectral-spatial models while significantly reducing inference time and computational cost. The proposed architecture, the Reduced Transformer (Transformer-R), proved to be the most efficient among the explored options, offering an excellent balance between classification accuracy, structural simplicity, and processing time. Despite not incorporating spatial information, it matched the accuracy of the reference 3D-CNN and outperformed it in inference time on both CPU and GPU, especially when processing large data volumes. This level of performance was achieved under favorable training conditions, with a sufficiently large and balanced dataset.

The evaluation on an embedded platform (NVIDIA Jetson Orin Nano embedded GPU) further confirmed the reduced architecture's ability to deliver fast inference, low power consumption, and efficient memory utilization. In experiments with 18,450 samples, the model maintained high performance without being constrained by batch size,

unlike reference architectures such as 3D-CNN.

Moreover, this thesis introduced a novel Transformer-based architecture for purely spectral classification, validated across multiple computational environments. The analysis demonstrated that on CPUs, the Reduced architecture is more suitable for small datasets, achieving significantly faster inference than heavier models such as 3D-CNNs. On GPUs, it exhibited excellent scalability, maintaining consistent performance even as the input volume increased substantially. Energy-consumption analyses on both notebook and embedded platforms supported the model's suitability for low-power embedded AI systems.

Although the proposed architecture has not yet been implemented as an IP core for FPGA deployment, the GPU-based evaluation demonstrated its practical viability. This work paves the way for future implementations on dedicated hardware, expanding its applicability to real-world satellite-based remote sensing systems.

In summary, this thesis contributes to the field by demonstrating that spectral classifiers based on Transformer models are viable, efficient, and reliable; by proposing a reduced and optimized architecture suitable for embedded deployment; and by validating the model under various computational scenarios (CPU, GPU, embedded), with detailed assessments of accuracy, runtime, memory usage, and energy consumption. These findings highlight the growing feasibility of using embedded Deep Learning with Transformer models in HSI applications, particularly in energy- and time-constrained environments such as space missions.

As future work, the proposed architecture will be explored as a hardware accelerator for FPGA deployment. Furthermore, this work also opens up several possibilities for new research and approaches that can be investigated:

- Exploration of fixed-point, integer, and FP16 libraries alongside NVIDIA architecture, primarily to improve latency;
- Exploring new implementation approaches for Multi-head Attention layers, aiming to reduce mathematical complexity to investigate their impact on hardware acceleration and the trade-off between accuracy, inference time, memory demand, and processing energy;
- Investigation of parallelism and approximate computation within neural network layers to minimize the impact of computation operations that have and do not have data dependency;
- Proposing hardware-friendly approaches for implementing ANN architectures for FPGAs and embedded GPUs;
- Investigating the use of synthesis tools using high-level languages for ANN acceleration in FPGAs;

- Exploring fault tolerance techniques in ANN architectures to protect inference in a spatial environment; and
- Investigate the use of devices, such as microcontrollers and DSPs, for the execution of ANN architectures in applications with hard constraints on power, memory, and space.

REFERENCES

ABADI, Martín et al. **TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems**. [S.l.: s.n.], 2015. Software available from tensorflow.org. Available from: `https://www.tensorflow.org/`.

AHMAD, Muhammad et al. Hyperspectral Image Classification—Traditional to Deep Models: A Survey for Future Prospects. **IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing**, v. 15, p. 968–999, 2022.

ALCOLEA, Adrián; PAOLETTI, Mercedes E; HAUT, Juan M; RESANO, Javier; PLAZA, Antonio. Inference in supervised spectral classifiers for on-board hyperspectral imaging: An overview. **Remote Sensing**, MDPI, v. 12, n. 3, p. 534, 2020.

ALCOLEA, Adrián; PAOLETTI, Mercedes E.; HAUT, Juan M.; RESANO, Javier; PLAZA, Antonio. Inference in Supervised Spectral Classifiers for On-Board Hyperspectral Imaging: An Overview. **Remote Sensing**, v. 12, n. 3, 2020. ISSN 2072-4292.

ARM. **Mali-G710**. 2022. Available from: `https://developer.arm.com/Processors/Mali-G710`.

ASHENDEN, Peter J. **Digital design (VHDL): An embedded systems approach using VHDL**. [S.l.]: Morgan Kaufmann, 2007.

ASSUDANI, Purshottam J; KRISHNA, V Rama. Multimodal Deep Learning Framework for Proactive Plant Disease Diagnosis. **Metallurgical and Materials Engineering**, v. 31, n. 4, p. 153–161, 2025.

AUDEBERT, Nicolas; LE SAUX, Bertrand; LEFEVRE, Sebastien. Deep Learning for Classification of Hyperspectral Data: A Comparative Review. **IEEE Geoscience and Remote Sensing Magazine**, v. 7, n. 2, p. 159–173, 2019.

BAHDANAU, Dzmitry; CHO, Kyunghyun; BENGIO, Yoshua. Neural machine translation by jointly learning to align and translate. **arXiv preprint arXiv:1409.0473**, 2014.

BAHL, Gaetan; DANIEL, Lionel; MORETTI, Matthieu; LAFARGE, Florent. Low-Power Neural Networks for Semantic Segmentation of Satellite Images. In: 2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW). [S.l.: s.n.], 2019. P. 2469–2476.

BISHOP, Christopher M et al. **Neural networks for pattern recognition**. [S.l.]: Oxford university press, 1995.

BLOTT, Michaela; PREUSSER, Thomas B.; FRASER, Nicholas J.; GAMBARDELLA, Giulio; O'BRIEN, Kenneth; UMUROGLU, Yaman. FINN-R: An

End-to-End Deep-Learning Framework for Fast Exploration of Quantized Neural Networks. **CoRR**, abs/1809.04570, 2018. arXiv: 1809.04570.

BOCK, Sebastian; GOPPOLD, Josef; WEISS, Martin. An improvement of the convergence proof of the ADAM-Optimizer. **arXiv preprint arXiv:1804.10587**, 2018.

BOCK, Sebastian; WEISS, Martin. A Proof of Local Convergence for the Adam Optimizer. In: 2019 International Joint Conference on Neural Networks (IJCNN). [S.l.: s.n.], 2019. P. 1–8.

BRADSKI, G. The OpenCV Library. **Dr. Dobb's Journal of Software Tools**, 2000.

CABA, Julián; DÍAZ, María; BARBA, Jesús; GUERRA, Raúl; TORRE, Jose A de la; LÓPEZ, Sebastián. FPGA-based on-board hyperspectral imaging compression: Benchmarking performance and energy efficiency against GPU implementations. **Remote Sensing**, MDPI, v. 12, n. 22, p. 3741, 2020.

CAO, Shijie; ZHANG, Chen; YAO, Zhuliang; XIAO, Wencong; NIE, Lanshun; ZHAN, Dechen; LIU, Yunxin; WU, Ming; ZHANG, Lintao. Efficient and effective sparse LSTM on FPGA with bank-balanced sparsity. In: PROCEEDINGS of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. [S.l.: s.n.], 2019. P. 63–72.

CARBONE, Andrea; SPILLER, Dario; AMICI, Stefania; THANGAVEL, Kathiravan; SABATINI, Roberto; LANEVE, Giovanni. Comparison of 1D and 3D Convolutional Neural Networks for Wildfire Detection Using PRISMA Hyperspectral Imagery and Domain Adaptation. In: 2023 IEEE International Conference on Metrology for eXtended Reality, Artificial Intelligence and Neural Engineering (MetroXRAINE). [S.l.: s.n.], 2023. P. 911–916.

CARDOSO, João Manuel Paiva; COUTINHO, José Gabriel de Figueiredo de; DINIZ, Pedro C. **Embedded computing for high performance: Efficient mapping of computations using customization, code transformations and compilation**. [S.l.]: Morgan Kaufmann, 2017.

CHEN, Wen-Tung; HUANG, Chun-Hsian. Area-Adaptive Air Quality Monitoring Based on FPGA with Edge AI and Hyperspectral Imaging. In: IECON 2023- 49th Annual Conference of the IEEE Industrial Electronics Society. [S.l.: s.n.], 2023. P. 1–8.

CHEN, Xiaofeng; JI, Jingyu; MEI, Shaohui; ZHANG, Yifan; HAN, Manli; DU, Qian. FPGA based implementation of convolutional neural network for hyperspectral classification. In: IEEE. IGARSS 2018-2018 IEEE International Geoscience and Remote Sensing Symposium. [S.l.: s.n.], 2018. P. 2451–2454.

CHOLLET, Francois. **Deep learning with python**. New York, United States of America: Manning Publication Co., 2018.

CIARDI, Roberto; GIUFFRIDA, Gianluca; BENELLI, Gionata; CARDENIO, Christian; MADERNA, Riccardo. GPU@SAT: A General-Purpose Programmable Accelerator for on Board Data Processing and Satellite Autonomy. In: IERACITANO, Cosimo; MAMMONE, Nadia; DI CLEMENTE, Marco; MAHMUD, Mufti; FURFARO, Roberto; MORABITO, Francesco Carlo (Eds.). **The Use of Artificial Intelligence for Space Applications**. Cham: Springer Nature Switzerland, 2023. P. 35–47.

DANG, Lanxue; LIU, Chongyang; DONG, Weichuan; HOU, Yane; GE, Qiang; LIU, Yang. An effective global learning framework for hyperspectral image classification based on encoder-decoder architecture. **International Journal of Digital Earth**, Taylor & Francis, v. 15, n. 1, p. 1350–1376, 2022.

DAVIDSON, Rebecca L; BRIDGES, Christopher P. Error resilient GPU accelerated image processing for space applications. **IEEE Transactions on Parallel and Distributed Systems**, IEEE, v. 29, n. 9, p. 1990–2003, 2018.

DE LUCIA, Gianluca; LAPEGNA, Marco; ROMANO, Diego. Unlocking the potential of edge computing for hyperspectral image classification: An efficient low-energy strategy. **Future Generation Computer Systems**, v. 147, p. 207–218, 2023. ISSN 0167-739X.

DING, Caiwen; REN, Ao; YUAN, Geng; MA, Xiaolong; LI, Jiayu; LIU, Ning; YUAN, Bo; WANG, Yanzhi. Structured weight matrices-based hardware accelerators in deep neural networks: Fpgas and asics. In: PROCEEDINGS of the 2018 on Great Lakes Symposium on VLSI. [S.l.: s.n.], 2018. P. 353–358.

DONG, Peiyan et al. HeatViT: Hardware-Efficient Adaptive Token Pruning for Vision Transformers. **arXiv preprint arXiv:2211.08110**, 2022.

ESA. **PhiSat-1**. June 2020. Available from: `https://directory.eoportal.org/web/%20eoportal/satellite-missions/p/phisat-1`.

FAN, Hongxiang; FERIANC, Martin; LUK, Wayne. Enabling fast uncertainty estimation: accelerating bayesian transformers via algorithmic and hardware optimizations. In: PROCEEDINGS of the 59th ACM/IEEE Design Automation Conference. [S.l.: s.n.], 2022. P. 325–330.

FARRELL, Michael D; MERSEREAU, Russell M. On the impact of PCA dimension reduction for hyperspectral detection of difficult targets. **IEEE Geoscience and Remote Sensing Letters**, IEEE, v. 2, n. 2, p. 192–195, 2005.

FELDMAN, SC; TARANIK, JV; MOUAT, DA. A first look at Airborne Imaging Spectrometer (AIS) data in an area of altered volcanic rocks and carbonate formations, Hot Creek Range, south central Nevada. In: JPL Proc. of the Airborne Imaging Spectrometer Data Anal. Workshop. [S.l.: s.n.], 1985.

FIRAT, Hüseyin; ASKER, Mehmet Emin; BAYINDIR, Mehmet İlyas; HANBAY, Davut. Hybrid 3D/2D complete inception module and convolutional neural network for hyperspectral remote sensing image classification. **Neural Processing Letters**, Springer, v. 55, n. 2, p. 1087–1130, 2023.

FOUDA, Mohamed Sobhy Abdelsalam. **Evaluation and implementation of an auto-encoder for compression of satellite images in the ScOSA project**. 2022. PhD thesis – TU-Berlin.

FURANO, Gianluca et al. Towards the Use of Artificial Intelligence on the Edge in Space Systems: Challenges and Opportunities. **IEEE Aerospace and Electronic Systems Magazine**, v. 35, n. 12, p. 44–56, 2020.

GAO, Qishuo; LIM, Samsung; JIA, Xiuping. Hyperspectral Image Classification Using Convolutional Neural Networks and Multiple Feature Learning. **Remote Sensing**, v. 10, n. 2, 2018. ISSN 2072-4292.

GEIST, Alessandro et al. SpaceCube v3. 0 NASA next-generation high-performance processor for science applications. In: PROCEEDINGS 33rd Annual AIAA/USU Conference of Small Satellite. [S.l.: s.n.], 2019.

GÉRON, Aurélien. **Hands-on machine learning with Scikit-Learn, Keras and TensorFlow: concepts, tools, and techniques to build intelligent systems**. Third edition. Beijing: O'Reilly Media, Inc, 2023. ISBN 978-1-0981-2597-4.

GIUFFRIDA, Gianluca; DIANA, Lorenzo; GIOIA, Francesco de; BENELLI, Gionata; MEONI, Gabriele; DONATI, Massimiliano; FANUCCI, Luca. Cloudscout: a deep neural network for on-board cloud detection on hyperspectral images. **Remote Sensing**, MDPI, v. 12, n. 14, p. 2205, 2020.

GIUFFRIDA, Gianluca et al. The Φ-Sat-1 mission: the first on-board deep neural network demonstrator for satellite earth observation. **IEEE Transactions on Geoscience and Remote Sensing**, IEEE, v. 60, p. 1–14, 2021.

GIUFFRIDA, Gianluca et al. The -Sat-1 Mission: The First On-Board Deep Neural Network Demonstrator for Satellite Earth Observation. **IEEE Transactions on Geoscience and Remote Sensing**, v. 60, p. 1–14, 2021.

GONZÁLEZ, Carlos; SÁNCHEZ, Sergio; PAZ, Abel; RESANO, Javier; MOZOS, Daniel; PLAZA, Antonio. Use of FPGA or GPU-based architectures for remotely sensed hyperspectral image processing. **Integration**, v. 46, n. 2, p. 89–103, 2013. ISSN 0167-9260.

GOODFELLOW, Ian; BENGIO, Yoshua; COURVILLE, Aaron. **Deep Learning**. [S.l.]: MIT Press, 2016. `http://www.deeplearningbook.org`.

GREEN, Robert. **Imaging Spectroscopy**. 2022. Available from: https://microdevices.jpl.nasa.gov/capabilities/imaging-spectroscopy/.

HU, Wei; XU, Dian; FAN, Zimeng; LIU, Fang; HE, Yanxiang. Vis-TOP: Visual Transformer Overlay Processor. **arXiv preprint arXiv:2110.10957**, 2021.

HUANG, Chun-Hsian; CHEN, Wen-Tung; CHANG, Yi-Chun; WU, Kuan-Ting. An Edge and Trustworthy AI UAV System With Self-Adaptivity and Hyperspectral Imaging for Air Quality Monitoring. **IEEE Internet of Things Journal**, v. 11, n. 20, p. 32572–32584, 2024.

HUANG, Shaoyi; LIU, Ning; LIANG, Yueying; PENG, Hongwu; LI, Hongjia; XU, Dongkuan; XIE, Mimi; DING, Caiwen. An automatic and efficient bert pruning for edge ai systems. In: IEEE. 2022 23rd International Symposium on Quality Electronic Design (ISQED). [S.l.: s.n.], 2022. P. 1–6.

INTEGRATED, Maxim. **MAX78000: Artificial Intelligence Microcontroller with UltraLow-Power Convolutional Neural Network Accelerator**. 2021. Available from: https://www.analog.com/media/en/technical-documentation/data-sheets/max78000.pdf.

JIANG, Min; WANG, Benkuan; LIU, Datong; PENG, Yu. On-board Sensor Data Monitoring System For Unmanned Aerial Vehicle PHM. In: IEEE. 2019 International Conference on Sensing, Diagnostics, Prognostics, and Control (SDPC). [S.l.: s.n.], 2019. P. 535–540.

JIAO, Qiang; HU, Wei; LIU, Fang; DONG, Yong. RISC-VTF: RISC-V Based Extended Instruction Set for Transformer. In: IEEE. 2021 IEEE International Conference on Systems, Man, and Cybernetics (SMC). [S.l.: s.n.], 2021. P. 1565–1570.

KINGMA, Diederik P.; BA, Jimmy. **Adam: A Method for Stochastic Optimization**. [S.l.: s.n.], 2017. arXiv: 1412.6980 [cs.LG]. Available from: https://arxiv.org/abs/1412.6980.

KONG, Zhenglun et al. SPViT: Enabling Faster Vision Transformers via Latency-Aware Soft Token Pruning. In: SPRINGER. EUROPEAN Conference on Computer Vision. [S.l.: s.n.], 2022. P. 620–640.

KONG, Zhenglun et al. SPViT: Enabling Faster Vision Transformers via Soft Token Pruning. **arXiv preprint arXiv:2112.13890**, 2021.

KOSMIDIS, Leonidas; RODRIGUEZ, Iván; JOVER, Álvaro; ALCAIDE, Sergi; LACHAIZE, Jérôme; ABELLA, Jaume; NOTEBAERT, Olivier; CAZORLA, Francisco J; STEENARI, David. GPU4S: Embedded GPUs in space-Latest project updates. **Microprocessors and Microsystems**, Elsevier, v. 77, p. 103143, 2020.

KOTHARI, Vivek; LIBERIS, Edgar; LANE, Nicholas D. The Final Frontier: Deep Learning in Space. In: (HotMobile '20), p. 45–49.

KOURIS, Alexandros; VENIERIS, Stylianos I; BOUGANIS, Christos-Savvas. Towards efficient on-board deployment of DNNS on intelligent autonomous systems. In: IEEE. 2019 IEEE Computer Society Annual Symposium on VLSI (ISVLSI). [S.l.: s.n.], 2019. P. 568–573.

LA SALVIA, Marco; TORTI, Emanuele; MARENZI, Elisa; DANESE, Giovanni; LEPORATI, Francesco. Edge and cloud computing approaches in the early diagnosis of skin cancer with attention-based vision transformer through hyperspectral imaging. **The Journal of Supercomputing**, Springer, v. 80, n. 11, p. 16368–16392, 2024.

LEE, Christine M.; CABLE, Morgan L.; HOOK, Simon J.; GREEN, Robert O.; USTIN, Susan L.; MANDL, Daniel J.; MIDDLETON, Elizabeth M. An introduction to the NASA Hyperspectral InfraRed Imager (HyspIRI) mission and preparatory activities. **Remote Sensing of Environment**, v. 167, p. 6–19, 2015. Special Issue on the Hyperspectral Infrared Imager (HyspIRI). ISSN 0034-4257.

LENTARIS, George; MARAGOS, Konstantinos; SOUDRIS, Dimitrios; ZABULIS, Xenophon; LOURAKIS, Manolis. Single-and multi-FPGA acceleration of dense stereo vision for planetary rovers. **ACM Transactions on Embedded Computing Systems (TECS)**, ACM New York, NY, USA, v. 18, n. 2, p. 1–27, 2019.

LI, Shutao; SONG, Weiwei; FANG, Leyuan; CHEN, Yushi; GHAMISI, Pedram; BENEDIKTSSON, Jón Atli. Deep Learning for Hyperspectral Image Classification: An Overview. **IEEE Transactions on Geoscience and Remote Sensing**, v. 57, n. 9, p. 6690–6709, 2019.

LI, Yangfan; HU, Yikun; WU, Fan; LI, Kenli. DiVIT: Algorithm and architecture co-design of differential attention in vision transformer. **Journal of Systems Architecture**, Elsevier, v. 128, p. 102520, 2022.

LI, Zhengang et al. Auto-ViT-Acc: An FPGA-Aware Automatic Acceleration Framework for Vision Transformer with Mixed-Scheme Quantization. **arXiv preprint arXiv:2208.05163**, 2022.

LIANG, Hanxue; FAN, Zhiwen; SARKAR, Rishov; JIANG, Ziyu; CHEN, Tianlong; ZOU, Kai; CHENG, Yu; HAO, Cong; WANG, Zhangyang. M^{3}3 ViT: Mixture-of-Experts Vision Transformer for Efficient Multi-task Learning with Model-Accelerator Co-design. **arXiv preprint arXiv:2210.14793**, 2022.

LIU, Peng; YANG, Zikai; KANG, Lin; WANG, Jian. A Heterogeneous Architecture for the Vision Processing Unit with a Hybrid Deep Neural Network Accelerator. **Micromachines**, MDPI, v. 13, n. 2, p. 268, 2022.

LIU, Ran; FANG, Shanting; ZHANG, Yutao; SUN, Jinping. Convolutional Neural Network-Based Accelerator for Hyperspectral Image Classification. In: PROCEEDINGS of the 2024 8th International Conference on Computer Science and Artificial Intelligence. [S.l.]: Association for Computing Machinery, 2025. (CSAI '24), p. 62–67.

LIU, Shuanglong; CHU, Ringo SW; WANG, Xiwei; LUK, Wayne. Optimizing cnn-based hyperspectral image classification on fpgas. In: SPRINGER. INTERNATIONAL Symposium on Applied Reconfigurable Computing. [S.l.: s.n.], 2019. P. 17–31.

LU, Feiyu; ZHANG, Boming; HOU, Yifei; XIONG, Xiao; DONG, Chaoran; LU, Wenbo; LI, Liangxue; LV, Chunli. A Spatiotemporal Attention-Guided Graph Neural Network for Precise Hyperspectral Estimation of Corn Nitrogen Content. **Agronomy**, v. 15, n. 5, 2025. ISSN 2073-4395.

LUONG, Minh-Thang; PHAM, Hieu; MANNING, Christopher D. Effective Approaches to Attention-based Neural Machine Translation. **CoRR**, abs/1508.04025, p. 1–11, 2015. arXiv: 1508.04025.

M GRAÑA MA VEGANZONS, B Ayerdi. **Hyperspectral Remote Sensing Scenes**. Mar. 2021. Available from: http://www.ehu.eus/ccwintco/index.php?title=Hyperspectral_Remote_Sensing_Scenes.

MARTINS, Lucas Amilton; VIEL, Felipe; SEMAN, Laio Oriel; BEZERRA, Eduardo Augusto; ZEFERINO, Cesar Albenes. A real-time SVM-based hardware accelerator for hyperspectral images classification in FPGA. **Microprocessors and Microsystems**, v. 104, p. 104998, 2024. ISSN 0141-9331.

MATEO-GARCIA, Gonzalo; VEITCH-MICHAELIS, Josh; PURCELL, Cormac; LONGEPE, Nicolas; REID, Simon; ANLIND, Alice; BRUHN, Fredrik; PARR, James; MATHIEU, Pierre Philippe. In-orbit demonstration of a re-trainable machine learning payload for processing optical imagery. **Scientific Reports**, Nature Publishing Group UK London, v. 13, n. 1, p. 10391, 2023.

MELO, Douglas R; ZEFERINO, Cesar A; DILILLO, Luigi; BEZERRA, Eduardo A. Maximizing the inner resilience of a network-on-chip through router controllers design. **Sensors**, MDPI, v. 19, n. 24, p. 5416, 2019.

MITCHELL, Tom M. **Machine learning**. [S.l.]: McGraw-hill New York, 1997. v. 1.

MURPHY, Kevin P. **Machine learning: a probabilistic perspective**. [S.l.]: MIT press, 2012.

NTAKOURIS, Theodoros. **Timeseries classification with a Transformer model**. June 2021. Available from: https://keras.io/examples/timeseries/%20timeseries_classification_transformer/.

NVIDIA. **NVIDIA ADA GPU ARCHITECTURE**. 2022. Available from: https://images.nvidia.com/aem-dam/Solutions/geforce/ada/nvidia-ada-gpu-architecture.pdf.

PAOLETTI, Mercedes Eugenia; TAO, Xuanwen; HAUT, Juan Mario; MORENO-ÁLVAREZ, Sergio; PLAZA, Antonio. Deep mixed precision for hyperspectral image classification. **The Journal of Supercomputing**, Springer, v. 77, p. 9190–9201, 2021.

PEREIRA, Lucas MV; SANTOS, Douglas A; ZEFERINO, Cesar A; MELO, Douglas R. A low-cost hardware accelerator for ccsds 123 predictor in fpga. In: IEEE. 2019 IEEE International Symposium on Circuits and Systems (ISCAS). [S.l.: s.n.], 2019. P. 1–5.

PÉREZ, Arturo; RODRÍGUEZ, Alfonso; OTERO, Andrés; ARJONA, David González; JIMÉNEZ-PERALO, Alvaro; VERDUGO, Miguel Ángel; DE LA TORRE, Eduardo. Run-time reconfigurable MPSoC-based on-board processor for vision-based space navigation. **IEEE Access**, IEEE, v. 8, p. 59891–59905, 2020.

QI, Panjie; SHA, Edwin Hsing-Mean; ZHUGE, Qingfeng; PENG, Hongwu; HUANG, Shaoyi; KONG, Zhenglun; SONG, Yuhong; LI, Bingbing. Accelerating framework of transformer by hardware design and model compression co-optimization. In: IEEE. 2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD). [S.l.: s.n.], 2021. P. 1–9.

QUE, Zhiqiang; LIU, Yanyang; GUO, Ce; NIU, Xinyu; ZHU, Yongxin; LUK, Wayne. Real-time anomaly detection for flight testing using AutoEncoder and LSTM. In: IEEE. 2019 International Conference on Field-Programmable Technology (ICFPT). [S.l.: s.n.], 2019. P. 379–382.

RAITA-HAKOLA, Anna-Maria; ANNALA, Leevi; LINDHOLM, Vivian; TROPS, Roberts; NÄSILÄ, Antti; SAARI, Heikki; RANKI, Annamari; PÖLÖNEN, Ilkka. FPI Based Hyperspectral Imager for the Complex Surfaces—Calibration, Illumination and Applications. **Sensors**, Multidisciplinary Digital Publishing Institute, v. 22, n. 9, p. 3420, 2022.

RAPUANO, Emilio; MEONI, Gabriele; PACINI, Tommaso; DINELLI, Gianmarco; FURANO, Gianluca; GIUFFRIDA, Gianluca; FANUCCI, Luca. An fpga-based hardware accelerator for cnns inference on board satellites: benchmarking with myriad 2-based solution for the cloudscout case study. **Remote Sensing**, MDPI, v. 13, n. 8, p. 1518, 2021.

REDDY, Dasari Venkat; RAO, MV Nageswara; SATYANARAYANA, TVV; BABU, T Aravinda; REDDY, Karna Vishnu Vardhana. A lightweight FPGA accelerator for onboard processing of hyperspectral anomaly detection based on optimized TinyYOLOv3 model. **Integration**, v. 104, p. 102443, 2025. ISSN 0167-9260.

RODRIGUEZ, Alfonso; SANTOS, Lucana; SARMIENTO, Roberto; DE LA TORRE, Eduardo. Scalable hardware-based on-board processing for run-time adaptive lossless hyperspectral compression. **IEEE Access**, IEEE, v. 7, p. 10644–10652, 2019.

ROJAS, Raúl. **Neural networks: a systematic introduction**. [S.l.]: Springer Science & Business Media, 2013.

ROY, Swalpa Kumar; KRISHNA, Gopal; DUBEY, Shiv Ram; CHAUDHURI, Bidyut B. HybridSN: Exploring 3D-2D CNN Feature Hierarchy for Hyperspectral Image Classification. **IEEE Geoscience and Remote Sensing Letters**, v. 17, 2020.

RŮŽIČKA, Vít; MARKHAM, Andrew. HyperspectralViTs: General Hyperspectral Models for On-Board Remote Sensing. **IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing**, v. 18, p. 10241–10253, 2025.

RŮŽIČKA, Vít; MARKHAM, Andrew. **HyperspectralViTs: General Hyperspectral Models for On-board Remote Sensing**. [S.l.: s.n.], 2024. arXiv: `2410.17248 [cs.AI]`. Available from: `https://arxiv.org/abs/2410.17248`.

RYBALKIN, Vladimir; NEY, Jonas; TEKLEYOHANNES, Menbere Kina; WEHN, Norbert. When massive GPU parallelism ain't enough: A novel hardware architecture of 2D-LSTM neural network. **ACM Transactions on Reconfigurable Technology and Systems (TRETS)**, ACM New York, NY, v. 15, n. 1, p. 1–35, 2021.

SABOGAL, Sebastian; GEORGE, Alan. A Methodology for Evaluating and Analyzing FPGA-Accelerated, Deep-Learning Applications for Onboard Space Processing. In: 2021 IEEE Space Computing Conference (SCC). [S.l.: s.n.], 2021. P. 143–154.

SABOGAL, Sebastian; GEORGE, Alan; CRUM, Gary. ReCoN: A Reconfigurable CNN Acceleration Framework for Hybrid Semantic Segmentation on Hybrid SoCs for Space Applications. In: 2019 IEEE Space Computing Conference (SCC). [S.l.: s.n.], 2019. P. 41–52.

SCHOWENGERDT, Robert A. **Remote sensing: models and methods for image processing**. [S.l.]: Elsevier, 2006.

SCHWENK, Kurt; ULMER, Moritz; PENG, Ting. ScOSA: application development for a high-performance space qualified onboard computing platform. In: SPIE. HIGH-PERFORMANCE Computing in Geoscience and Remote Sensing VIII. [S.l.: s.n.], 2018. v. 10792, p. 32–44.

SHAO, Yichuan; YANG, Jiapeng; ZHOU, Wen; SUN, Haijing; XING, Lei; ZHAO, Qian; ZHANG, Le. An Improvement of Adam Based on a Cyclic Exponential Decay Learning Rate and Gradient Norm Constraints. **Electronics**, v. 13, n. 9, 2024. ISSN 2079-9292.

SHINDE, Sandhya; PATIDAR, Hemant. Hyperspectral image classification using principle component analysis and deep convolutional neural network. **Journal of Ambient Intelligence and Humanized Computing**, Springer, v. 14, n. 12, p. 16491–16497, 2022.

SISODIYA, Neha; DUBE, Nitant; THAKKAR, Priyank. Next-generation artificial intelligence techniques for satellite data processing. In: ARTIFICIAL Intelligence Techniques for Satellite Image Analysis. [S.l.]: Springer, 2020. P. 235–254.

SKLIAROVA, Iouliia; SKLYAROV, Valery. **FPGA-BASED hardware accelerators**. [S.l.]: Springer, 2019.

SONG, Yifan; ZHAO, Shunpeng; CHEN, Song; KANG, Yi. SAUST: A Scheme for Acceleration of Unstructured Sparse Transformer. In: IEEE. 2022 IEEE International Conference on Integrated Circuits, Technologies and Applications (ICTA). [S.l.: s.n.], 2022. P. 56–57.

SPILLER, Dario; THANGAVEL, Kathiravan; SASIDHARAN, Sarathchandrakumar T; AMICI, Stefania; ANSALONE, Luigi; SABATINI, Roberto. Wildfire segmentation analysis from edge computing for on-board real-time alerts using hyperspectral imagery. In: IEEE. 2022 IEEE International Conference on Metrology for Extended Reality, Artificial Intelligence and Neural Engineering (MetroXRAINE). [S.l.: s.n.], 2022. P. 725–730.

SPILLER, Dario; THANGAVEL, Kathiravan; SASIDHARAN, Sarathchandrakumar T.; AMICI, Stefania; ANSALONE, Luigi; SABATINI, Roberto. Wildfire segmentation analysis from edge computing for on-board real-time alerts using hyperspectral imagery. In: 2022 IEEE International Conference on Metrology for Extended Reality, Artificial Intelligence and Neural Engineering (MetroXRAINE). [S.l.: s.n.], 2022. P. 725–730.

SUN, Chen; WANG, Xinyu; LIU, Zhenqi; WAN, Yuting; ZHANG, Liangpei; ZHONG, Yanfei. SiamOHOT: A Lightweight Dual Siamese Network for Onboard Hyperspectral Object Tracking via Joint Spatial-Spectral Knowledge Distillation. **IEEE Transactions on Geoscience and Remote Sensing**, v. 61, p. 1–12, 2023.

SUN, Mengshu; MA, Haoyu; KANG, Guoliang; JIANG, Yifan; CHEN, Tianlong; MA, Xiaolong; WANG, Zhangyang; WANG, Yanzhi. VAQF: Fully Automatic Software-hardware Co-design Framework for Low-bit Vision Transformer. **arXiv preprint arXiv:2201.06618**, 2022.

SUN, Yang; HU, Wei; LIU, Fang; JIANG, Min; HUANG, FeiHu; XU, Dian. Speformer: An Efficient Hardware-Software Cooperative Solution for Sparse Spectral Transformer. In: IEEE. 2022 IEEE 9th International Conference on Cyber Security and Cloud Computing (CSCloud)/2022 IEEE 8th International Conference on Edge Computing and Scalable Cloud (EdgeCom). [S.l.: s.n.], 2022. P. 180–185.

SUN, Ziheng et al. A review of Earth Artificial Intelligence. **Computers Geosciences**, v. 159, p. 105034, 2022. ISSN 0098-3004.

THANGAVEL, Kathiravan; SABATINI, Roberto; GARDI, Alessandro; RANASINGHE, Kavindu; HILTON, Samuel; SERVIDIA, Pablo; SPILLER, Dario. Artificial Intelligence for Trusted Autonomous Satellite Operations. **Progress in Aerospace Sciences**, v. 144, p. 100960, 2024. ISSN 0376-0421.

THANGAVEL, Kathiravan; SPILLER, Dario; SABATINI, Roberto; AMICI, Stefania; SASIDHARAN, Sarathchandrakumar Thottuchirayil; FAYEK, Haytham; MARZOCCA, Pier. Autonomous Satellite Wildfire Detection Using Hyperspectral Imagery and Neural Networks: A Case Study on Australian Wildfire. **Remote Sensing**, v. 15, n. 3, 2023. ISSN 2072-4292.

TIPALDI, Massimo; FERUGLIO, Lorenzo; DENIS, Pierre; D'ANGELO, Gianni. On applying AI-driven flight data analysis for operational spacecraft model-based diagnostics. **Annual Reviews in Control**, Elsevier, v. 49, p. 197–211, 2020.

TORTI, Emanuele; D'AMATO, Cristina; DANESE, Giovanni; LEPORATI, Francesco. A low power and real-time hardware recurrent neural network for time series analysis on wearable devices. **Microprocessors and Microsystems**, Elsevier, v. 87, p. 104374, 2021.

TORTI, Emanuele; D'AMATO, Cristina; DANESE, Giovanni; LEPORATI, Francesco. An hardware recurrent neural network for wearable devices. In: IEEE. 2020 23rd Euromicro Conference on Digital System Design (DSD). [S.l.: s.n.], 2020. P. 293–300.

TORTI, Emanuele; GAZZONI, Marco; MARENZI, Elisa; LEON, Raquel; CALLICÒ, Gustavo Marrero; DANESE, Giovanni; LEPORATI, Francesco. An Attention-Based Parallel Algorithm for Hyperspectral Skin Cancer Classification on Low-Power GPUs. In: 2023 26th Euromicro Conference on Digital System Design (DSD). [S.l.: s.n.], 2023. P. 111–116.

TZANOS, Georgios; KACHRIS, Christoforos; SOUDRIS, Dimitrios. Hardware acceleration on gaussian naive bayes machine learning algorithm. In: IEEE. 2019 8th International Conference on Modern Circuits and Systems Technologies (MOCAST). [S.l.: s.n.], 2019. P. 1–5.

VAHID, Frank; GIVARGIS, Tony D. **Embedded system design: a unified hardware/software introduction**. [S.l.]: John Wiley & Sons, 2001.

VAN DEN BERGH, Michael; BOIX, Xavier; ROIG, Gemma; CAPITANI, Benjamin de; VAN GOOL, Luc. SEEDS: Superpixels Extracted via Energy-Driven Sampling. In: FITZGIBBON, Andrew; LAZEBNIK, Svetlana; PERONA, Pietro; SATO, Yoichi; SCHMID, Cordelia (Eds.). **Computer Vision – ECCV 2012**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012. P. 13–26.

VARSHNEY, Pramod K; ARORA, Manoj K, et al. **Advanced image processing techniques for remotely sensed hyperspectral data**. [S.l.]: Springer Science & Business Media, 2004.

VASWANI, Ashish; SHAZEER, Noam; PARMAR, Niki; USZKOREIT, Jakob; JONES, Llion; GOMEZ, Aidan N; KAISER, Łukasz; POLOSUKHIN, Illia. Attention is all you need. **Advances in neural information processing systems**, v. 30, 2017.

VENKATESAN, Ragav; LI, Baoxin. **Convolutional neural networks in visual computing: a concise guide**. [S.l.]: CRC Press, 2017.

VIEL, Felipe; MACIEL, Renato Cotrim; SEMAN, Laio Oriel; ZEFERINO, Cesar Albenes; BEZERRA, Eduardo Augusto; LEITHARDT, Valderi Reis Quietinho. Hyperspectral Image Classification: An Analysis Employing CNN, LSTM, Transformer, and Attention Mechanism. **IEEE Access**, v. 11, p. 24835–24850, 2023.

VIEL, Felipe; PARREIRA, Wemerson Délcio; SUSIN, Altamiro Amadeu; ZEFERINO, Cesar Albenes. A Hardware Accelerator for Onboard Spatial Resolution Enhancement of Hyperspectral Images. **IEEE Geoscience and Remote Sensing Letters**, IEEE, v. 18, n. 10, p. 1796–1800, 2020.

WANG, Chao. **High Performance Computing for Big Data: Methodologies and Applications**. [S.l.]: CRC Press, 2017.

WANG, Hong-Yi; CHANG, Tian-Sheuan. Row-wise Accelerator for Vision Transformer. **arXiv preprint arXiv:2205.03998**, 2022.

WANG, Liguo; ZHAO, Chunhui. **Hyperspectral image processing**. [S.l.]: Springer, 2016.

WANG, Teng; GONG, Lei; WANG, Chao; YANG, Yang; GAO, Yingxue; ZHOU, Xuehai; CHEN, Huaping. Via: A novel vision-transformer accelerator based on fpga. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, IEEE, v. 41, n. 11, p. 4088–4099, 2022.

WANG, Wenxuan; LIU, Leiming; ZHANG, Tianxiang; SHEN, Jiachen; WANG, Jing; LI, Jiangyun. Hyper-ES2T: Efficient Spatial–Spectral Transformer for the classification of hyperspectral remote sensing images. **International Journal of Applied Earth Observation and Geoinformation**, v. 113, p. 103005, 2022. ISSN 1569-8432.

WOJCICKI, Filip; QUE, Zhiqiang; TAPPER, Alexander D; LUK, Wayne. Accelerating Transformer Neural Networks on FPGAs for High Energy Physics Experiments. In: IEEE. 2022 International Conference on Field-Programmable Technology (ICFPT). [S.l.: s.n.], 2022. P. 1–8.

WU, Yuancong; LIU, Y; CHEN, TP. Stock Price Prediction Based on an Energy-Efficient Spiking-LSTM Hardware Accelerator. In: IOP PUBLISHING, 1. JOURNAL of Physics: Conference Series. [S.l.: s.n.], 2021. v. 1828, p. 012050.

XILINX. **AI Engines and Their Applications (WP506)**. 2022. Available from: https://docs.xilinx.com/v/u/en-US/wp506-ai-engine.

XILINX. **Isolation Design Flow**. 2022. Available from: https://www.xilinx.com/applications/isolation-design-flow.html.

XILINX. **Versal ACAP Configurable Logic Block Architecture Manual (AM005)**. 2021. Available from: https://docs.xilinx.com/r/en-US/am005-versal-clb/Revision-History.

XILINX. **Vitis High-Level Synthesis User Guide (UG1399)**. 2022. Available from: https://docs.xilinx.com/r/en-US/ug1399-vitis-hls/Working-with-Sources.

XIONG, Ruibin et al. On layer normalization in the transformer architecture. In: PMLR. INTERNATIONAL Conference on Machine Learning. [S.l.: s.n.], 2020. P. 10524–10533.

YADAV, Dhirendra Prasad; KISHORE, Kamal; GAUR, Ashish; KUMAR, Ankit; SINGH, Kamred Udham; SINGH, Teekam; SWARUP, Chetan. A Novel Multi-Scale Feature Fusion-Based 3SCNet for Building Crack Detection. **Sustainability**, v. 14, n. 23, 2022. ISSN 2071-1050.

YAN, Lei; CAO, Suzhi; GONG, Yongsheng; HAN, Hao; WEI, Junyong; ZHAO, Yi; YANG, Shuling. SatEC: A 5G satellite edge computing framework based on microservice architecture. **Sensors**, MDPI, v. 19, n. 4, p. 831, 2019.

YIN, Jun; HAN, Jun; XIE, Ruiqi; WANG, Chenghao; DUAN, Xuyang; RONG, Yitong; ZENG, Xiaoyang; TAO, Jun. MC-LSTM: Real-time 3d human action detection system for intelligent healthcare applications. **IEEE Transactions on Biomedical Circuits and Systems**, IEEE, v. 15, n. 2, p. 259–269, 2021.

YIN, Junru; QI, Changsheng; CHEN, Qiqiang; QU, Jiantao. Spatial-Spectral Network for Hyperspectral Image Classification: A 3-D CNN and Bi-LSTM Framework. **Remote Sensing**, v. 13, n. 12, 2021. ISSN 2072-4292.

YOO, Hanju; JUNG, Taehun; DAI, Linglong; KIM, Songkuk; CHAE, Chan-Byoung. Real-Time Semantic Communications with a Vision Transformer. **arXiv preprint arXiv:2205.03886**, 2022.

YU, Yong; SI, Xiaosheng; HU, Changhua; ZHANG, Jianxun. A review of recurrent neural networks: LSTM cells and network architectures. **Neural computation**, MIT Press One Rogers Street, Cambridge, MA 02142-1209, USA journals-info . . ., v. 31, n. 7, p. 1235–1270, 2019.

YUAN, Ye; ZHU, Zhiliang; YU, Hai; ZHANG, Wei. Watershed-Based Superpixels With Global and Local Boundary Marching. **IEEE Transactions on Image Processing**, v. 29, p. 7375–7388, 2020.

ZANG, Zhenya; XIAO, Dong; WANG, Quan; JIAO, Ziao; YU, Chen; LI, David Day-Uei. Compact and Robust Deep Learning Architecture for Fluorescence Lifetime Imaging and FPGA Implementation. **arXiv preprint arXiv:2209.03272**, 2022.

ZHU, Botao; LIN, Siyuan; ZHU, Yifei; WANG, Xudong. Collaborative Hyperspectral Image Processing Using Satellite Edge Computing. **IEEE Transactions on Mobile Computing**, v. 23, n. 3, p. 2241–2253, 2024.

ZHU, Dehui; DU, Bo; ZHANG, Liangpei. Two-stream convolutional networks for hyperspectral target detection. **IEEE Transactions on Geoscience and Remote Sensing**, IEEE, v. 59, n. 8, p. 6907–6921, 2020.