48 Schriften aus der Fakultät Wirtschaftsinformatik und Angewandte Informatik der Otto-Friedrich-Universität Bamberg

Efficient machine learning for sensor systems with an application to computational spectrometers

Julio Wissing

University of Bamberg Press

48 Schriften aus der Fakultät Wirtschaftsinformatik und Angewandte Informatik der Otto-Friedrich-Universität Bamberg

Contributions of the Faculty Information Systems and Applied Computer Sciences of the Otto-Friedrich-University Bamberg

Schriften aus der Fakultät Wirtschaftsinformatik und Angewandte Informatik der Otto-Friedrich-Universität Bamberg

Contributions of the Faculty Information Systems and Applied Computer Sciences of the Otto-Friedrich-University Bamberg

Band 48

University of Bamberg Press
2026

Efficient machine learning for sensor systems with an application to computational spectrometers

Julio Wissing

University of Bamberg Press
2026

Bibliografische Informationen der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.dnb.de/ abrufbar.

Diese Arbeit hat der Fakultät Wirtschaftsinformatik und Angewandte Informatik der Otto-Friedrich-Universität Bamberg als Dissertation vorgelegen.
1. Gutachterin: Prof. Dr. Ute Schmid
2. Gutachterin: Prof. Dr. Dorothea Kolossa
Tag der mündlichen Prüfung: 12.12.2025

Herstellung und Druck: docupoint, Magdeburg
Umschlaggestaltung: University of Bamberg Press
Umschlaggraphik: Pexels, Merlin Lightpainting

University of Bamberg Press, ubp@uni-bamberg.de, Bamberg 2026
ROR https://ror.org/004fa0x02

ISSN: 1867-6197 (Print)
ISBN: 978-3-98989-108-1 (Print)
eISSN: 2750-8560 (Online)
eISBN: 978-3-98989-109-8 (Online)

URN: urn:nbn:de:bvb:473-irb-114288x
DOI: https://doi.org/10.20378/irb-114288

To my beloved Father
Klaus Wissing
*March 4th 1951 †February 22nd 2020
If anything exists beyond all this, I will find you there.

Acknowledgments

First and foremost, I would like to express my deepest gratitude to my supervisor, Ute Schmid, for her unwavering support, open-minded guidance, and invaluable advice—ranging from technical discussions to personal encouragement. This thesis would not have been possible without her continuous trust and mentorship, even during challenging times. I am also immensely grateful to Stephan Scheele, who has been not only a mentor but also a co-author on most of my publications over the past four and a half years. His insights, patience, and expertise have profoundly shaped my research journey, and I owe him my sincerest thanks and utmost respect. Additionally, I extend my appreciation to Dorothea Kolossa for co-supervising this thesis and to Michael Engel for chairing my dissertation committee.

Beyond the academic sphere, I am deeply indebted to my family for their constant support, understanding, and patience. They have stood by my side throughout this journey, even when my research commitments meant missing family gatherings. In particular, I want to thank my mother, Kim Wissing, her partner, Jörg Brinkmann, and my partner, Chantale Klostermann, for providing me with unwavering emotional support and encouragement during the toughest moments.

I would also like to extend my heartfelt thanks to my colleagues and friends at Fraunhofer IIS, whose collaboration and camaraderie have made this journey both productive and enjoyable. Special thanks to Stephan Junger for his trust in my work and for securing the necessary resources, Wladimir Tschekalinskij for his relentless support in setting up new measurement systems, and Daniel Cichon for his mentorship. I am equally grateful to Teresa Scholz for her technical insights and willingness to discuss any question, Stefan Saloman for his contributions to our research projects, Florian Stieglitz for his help in integrating the models from this thesis in the sensor firmware, and Christian Kohlbrenner and Julian Fronek for our many stimulating discussions. To all the other colleagues whose names may not appear here but whose support has been invaluable—thank you.

This work has also greatly benefited from the dedication and enthusiasm of the students I had the privilege to supervise over the years. I am especially grateful to Aliya Mohammed and Devesh Vashishth for their contributions to hierarchical machine learning, Apurwa Jagtap, Nils Englert, Martin Rottler, and Julian Hernandez for their efforts in collecting the datasets, and Lidia Fargueta for supportive work on spectral reconstruction. Their hard work not only directly influenced this thesis but also provided invaluable support in project-related tasks, allowing me to focus more freely on this research.

It is evident from the many people acknowledged here that this thesis has been a collective effort. Without the contributions, guidance, and encouragement of all those mentioned—and likely many more—this work would not have become a

reality. To honor this shared achievement, I have chosen to use ”we” throughout the thesis, recognizing the collaborative nature of this endeavor.

To all of you—thank you. I hope you enjoy reading through this thesis as much as I have enjoyed the journey of creating it.

Abstract

This thesis explores two distinct approaches to optimizing machine learning for resource-constrained environments: hierarchical machine learning (Hierarchical Machine Learning (HiML)) for energy-efficient classification and neural network-based spectral reconstruction for cost-effective computational spectrometers. Both topics address different challenges but share the common goal of improving computational efficiency for embedded systems.

The first part investigates hierarchical machine learning as a means to reduce energy consumption in classification tasks. Unlike traditional flat classifiers, which apply equal computational effort to all inputs, hierarchical machine learning structures classification as a hierarchy of decisions, allowing for early exits when classification confidence is high. Experimental results demonstrate that this approach reduces energy consumption by up to 47.63% while maintaining competitive accuracy. Additionally, reinforcement learning is introduced as an optimization strategy for classifier selection, significantly accelerating the search process compared to exhaustive methods.

The second part focuses on spectral reconstruction, where neural networks recover full spectral information from complex sensor outputs. A physics-informed data augmentation method is introduced, leveraging the optical properties of transmission spectra to expand the dataset from 214 to over 10,000 samples. This augmentation improves model generalization without requiring additional physical measurements. Furthermore, layer-wise relevance propagation is applied to optimize the sensor hardware, enabling a 94% reduction in input dimensions while preserving high reconstruction accuracy. Benchmarking results reveal that smaller, well-optimized models can match or even surpass the performance of larger networks, demonstrating the potential for efficient, compact spectral reconstruction systems.

The findings of this thesis contribute to the development of energy-efficient and computationally lightweight machine learning solutions. The proposed approaches enable more effective classification pipelines and high-fidelity spectral reconstruction on embedded systems, paving the way for real-world applications in industrial monitoring, portable spectral sensing, and other edge AI scenarios.

Zusammenfassung

Diese Arbeit untersucht zwei unterschiedliche Ansätze zur Optimierung von maschinellem Lernen für ressourcenbeschränkte Geräte: Hierarchisches maschinelles Lernen für energieeffiziente Klassifikation und effiziente neurale Netzwerke zur spektrale Rekonstruktion. Beide Themen behandeln unterschiedliche Herausforderungen, verfolgen jedoch das gemeinsame Ziel, die Recheneffizienz für eingebettete Systeme zu verbessern.

Der erste Teil untersucht hierarchisches maschinelles Lernen als Methode zur Reduktion des Energieverbrauchs bei Klassifikationsaufgaben. Im Gegensatz zu traditionellen sog. flachen Klassifikatoren, die für alle Eingaben denselben Rechenaufwand aufbringen, strukturiert hierarchisches maschinelles Lernen die Klassifikation als eine Abfolge von Entscheidungen. Dadurch können frühe Ausstiege erfolgen, sobald eine Klassifikation mit hoher Sicherheit getroffen werden kann. Experimentelle Ergebnisse zeigen, dass dieser Ansatz den Energieverbrauch um bis zu 47,63% reduziert, während die Klassifikationsgenauigkeit auf einem akzeptablen Niveau bleibt. Darüber hinaus setzen wir Reinforcement Learning als Optimierungsstrategie für die Klassifikatorauswahl ein, wodurch der Suchprozess im Vergleich zur Exhaustive Search deutlich beschleunigt wird.

Der zweite Teil konzentriert sich auf die spektrale Rekonstruktion, bei der neuronale Netze vollständige Spektren aus komplexen Sensorausgaben rekonstruieren. Eine physikalisch fundierte Datenaugmentierungsmethode wird eingeführt, die die optischen Eigenschaften von Transmissionsspektren nutzt, um den Datensatz von 214 auf über 10.000 Messwerte zu erweitern. Diese Augmentierung verbessert die Generalisierungsfähigkeit der Modelle, ohne dass zusätzliche physikalische Messungen erforderlich sind. Zudem setzen wir Layer-Wise Relevance Propagation ein, um die Sensorhardware zu optimieren, wodurch eine Reduktion der Eingangsmerkmale um 94% erzielt wird, ohne die Rekonstruktionsgenauigkeit zu beeinträchtigen. Benchmarking-Ergebnisse zeigen, dass kleinere, gut optimierte Modelle die Leistung größerer Netzwerke erreichen oder sogar übertreffen können, was das Potenzial für effiziente und kompakte spektrale Rekonstruktionssysteme verdeutlicht.

Die Ergebnisse dieser Arbeit tragen zur Entwicklung energieeffizienter und rechenoptimierter maschineller Lernlösungen bei. Die vorgeschlagenen Ansätze ermöglichen effektivere Klassifikationspipelines und hochpräzise spektrale Rekonstruktion auf eingebetteten Systemen und ebnen den Weg für reale Anwendungen in der industriellen Überwachung, der tragbaren Spektralanalyse und anderen Edge-AI-Szenarien.

Contents

List of Figures

List of Tables

List of Algorithms

List of Acronyms

CHAPTER 1

Introduction

The demand for efficient and intelligent sensing solutions has grown significantly in recent years, driven by advancements in the Internet of Things, edge computing, and embedded artificial intelligence. As modern applications increasingly require real-time decision-making and data processing at the edge, there is a pressing need for sensor systems that are not only accurate but also computationally efficient.

Traditionally, many sensing applications have relied on cloud-based processing, where sensor data is transmitted to centralized servers for analysis. While this approach offers access to high-performance computing resources, it introduces several drawbacks, including increased latency, dependency on network connectivity, and high energy consumption due to continuous data transmission. In contrast, edge computing enables localized data processing, allowing smart sensors to analyze information in real time while minimizing communication overhead. This paradigm shift necessitates the development of intelligent, energy-efficient sensor systems that integrate data-driven algorithms with optimized hardware to maximize performance within constrained environments.

This thesis addresses two key challenges in this domain: First, we explore HiML as a method to enhance the energy efficiency of classification systems for embedded platforms. By structuring classification tasks into hierarchical decision processes, we aim to reduce unnecessary computations while maintaining high classification accuracy. Second, we investigate the potential of computational spectrometers based on a low-cost Chip-Size Spectrometer (CSS), leveraging neural networks to reconstruct spectral information efficiently. These two research directions, while distinct, share a common goal: enabling intelligent and resource-efficient sensing through the integration of advanced machine learning techniques.

1.1 Smart Sensor Systems

Smart sensor systems represent a fusion of sensing, computation, and communication, enabling automated and intelligent decision-making. These systems consist of three primary components: (i) a sensing element that detects physical changes in the environment, (ii) an embedded processing unit that analyzes and interprets the sensor data, and (iii) a communication module that transmits processed information to higher-level systems or user interfaces. By integrating these components, smart sensors are capable of real-time data processing, reducing the

need for external computing resources and improving overall system responsiveness [124].

The importance of smart sensor systems extends across various domains, including industrial automation [44], healthcare [105], multi-modality fusion [133], and precision agriculture [120]. In manufacturing, predictive maintenance systems rely on sensor data to detect early signs of equipment failure, minimizing downtime and optimizing efficiency. Similarly, in healthcare, wearable smart sensors provide continuous patient monitoring, enabling early detection of medical conditions. Smart sensing system can also combine multiple sources of data or sensors to make prediction more robust. In precision agriculture, smart sensors facilitate real-time assessment of soil and crop conditions, improving resource utilization and increasing yield.

However, the development of smart sensors faces several challenges, particularly in the context of embedded AI. These devices often operate under strict power and memory constraints, limiting the complexity of the models that can be deployed. Furthermore, optimizing both the hardware and software of these systems is critical to achieving an effective trade-off between computational efficiency and sensing accuracy. This thesis follows a co-design approach, optimizing machine learning algorithms while considering the hardware constraints of the deployed sensor system. In addition, explainable AI techniques are leveraged to improve sensor design, ensuring that the resulting system is both interpretable and efficient.

1.1.1 Machine Learning Made Small—TinyML

The emergence of Tiny Machine Learning (TinyML) has significantly expanded the possibilities for deploying machine learning models on resource-limited embedded systems. Unlike traditional AI models, which are often designed for high-performance GPUs and cloud environments, TinyML focuses on optimizing neural networks for low-power microcontrollers and edge devices [15]. This enables on-device intelligence, allowing systems to process data locally with minimal latency and power consumption.

A variety of techniques have been developed to optimize machine learning models for TinyML applications. Model compression strategies such as pruning, quantization, and knowledge distillation reduce memory footprints while preserving accuracy [41]; [52]; [134]. Furthermore, hardware-aware neural architecture search (NAS) enables the automatic discovery of efficient model structures tailored to specific edge platforms [1].

The TinyML community actively contributes to the advancement of this field by developing benchmarks, datasets, and toolchains that facilitate model deployment on low-power devices. One such tool, TensorFlow Lite, provides a platform-agnostic framework for optimizing and running inference on embedded processors. De-

spite these advancements, deploying neural networks on microcontrollers remains challenging, as limited compute resources often restrict model complexity. This thesis addresses these challenges by developing tailored machine learning pipelines that optimize efficiency while maintaining performance in real-world applications.

1.1.2 Optical Sensors

Optical sensors play a crucial role in numerous applications, ranging from industrial inspection to biomedical diagnostics. These sensors detect and measure light-related phenomena, converting optical signals into electrical outputs that can be analyzed and interpreted [111].

Several types of optical sensors exist, each designed for different use cases. Simple photodiodes measure light intensity, while more complex imaging sensors, such as RGB cameras and hyperspectral cameras, capture spatial and spectral information. Among these, the CSS presents an approach to cost-effective spectral sensing [11]. These sensors incorporate fixed optical filters directly on the sensor chip, allowing them to capture spectral information without requiring expensive dispersive elements, such as gratings or prisms.

While the CSS significantly reduces the cost of spectral sensing, they introduce new challenges in data interpretation. Unlike traditional spectrometers, which provide high-resolution spectral data, a CSS produces mixed and overlapping spectral signals that must be computationally reconstructed. The accuracy of this reconstruction process directly influences the sensor's applicability. This thesis develops a machine learning-based approach to spectral reconstruction, leveraging neural networks to infer the spectral information and enhance measurement precision.

1.1.3 Possible Applications

The low-cost CSS developed in this thesis enables a wide range of applications where conventional spectrometers are impractical due to cost and size constraints. The following domains particularly benefit from this technology:

Smart Farming: Precision agriculture relies on spectral sensors to monitor plant health, detect nutrient deficiencies, and assess soil conditions [30]. Deploying cost-efficient spectral sensors allows farmers to implement high-resolution monitoring systems without the high costs associated with traditional spectrometers.

Environmental Monitoring: Spectral sensors are used to detect air and water pollutants, providing real-time environmental data for pollution control and urban planning [63]; [119]. By making spectral sensing more affordable, large-scale environmental monitoring networks become feasible.

Healthcare Diagnostics: Non-invasive medical diagnostics, such as oxygen saturation monitoring and tissue characterization, benefit from compact spectral sensors [35]. With the increasing demand for point-of-care testing, low-cost spectral sensors can facilitate early disease detection and remote diagnostics.

TinyML enables numerous edge AI applications. Wearable devices leverage TinyML for real-time health monitoring, while industrial IoT systems use embedded machine learning for predictive maintenance [1]; [41]; [135]. The techniques developed in this thesis contribute to these domains by enhancing the efficiency and deployability of machine learning models in constrained environments.

1.2 Summary of Contributions

This thesis explores two interconnected research directions: hierarchical machine learning for energy-efficient classification and computational spectrometry for resource-efficient spectral sensing. The first part of the thesis investigates HiML as a method to reduce computational costs in classification tasks, optimizing machine learning models for embedded platforms. The second part focuses on developing a cost-efficient spectral sensing system, integrating neural network-based spectral reconstruction with optimized hardware.

In the domain of TinyML, this thesis presents a framework for optimizing hierarchical classification taxonomies, demonstrating significant energy savings through structured decision-making processes. Reinforcement learning is introduced as an optimization technique, enabling efficient selection of classifier hierarchies while reducing search time.

For computational spectrometry, this work introduces a physics-informed data augmentation method, expanding spectral datasets by leveraging optical properties. Explainable AI techniques are applied to refine sensor hardware, leading to a 94% reduction in input dimensions while preserving spectral reconstruction accuracy. Finally, TinyML methodologies are employed to compress and optimize neural networks for deployment on microcontrollers, achieving a fully integrated spectral sensing system.

By bridging domain-specific expertise with modern machine learning techniques, this thesis contributes to the advancement of energy-efficient AI and computational sensing, enabling intelligent and resource-aware sensor systems for real-world applications.

1.2.1 Publications

The research presented in this thesis has led to several publications, each contributing to the overarching goal of developing efficient smart sensor systems. Table 1.1

Citation	Venue/Jornal	Chapter
[133]	ICASSP 2021	Chap. 1
[135]	ARTIIS 2022	Chap. 3
[134]	SMSI 2022	Chap. 2
[74]	TM 2024, Volume 91, Issue 12	Chap. 5
[41]	EEAI Conference 2024	Chap. 2
[136]	IEEE Sensors 2024	Chap. 5

Table 1.1: Listing of most essential publications contributing to this thesis. The chapters in which these publications contribute most are in the chapter column.

provides an overview of these publications and their respective contributions to different chapters of this work. While all relevant findings are cited in the corresponding sections, this summary highlights their key contributions and how they fit into the broader context of this thesis.

Our initial contribution, presented in [133], explores a multimodal smart sensing system that fuses visual and aural data to enhance speaker tracking in meeting environments. While this work introduces an innovative approach to speaker sensing, it does not directly contribute to the core topics of this thesis. However, it is referenced in the introduction as an example of smart sensor systems integrating multiple data modalities for improved performance.

The research presented in [135] marks our first exploration of TinyML, focusing on hierarchical machine learning as a meta-approach for orchestrating multiple models to achieve efficiency gains. The findings from this work form the foundation of Chap. 3, where hierarchical learning is discussed in detail, demonstrating its potential for energy-efficient classification.

Building upon our work in efficient machine learning, [134] and [41] review recent advancements in TinyML. While [134] provides a concise guide on optimization techniques applicable at different stages of the machine learning model lifecycle, [41] offers a more comprehensive survey of current TinyML methods and their practical implications. As both publications focus on literature analysis, their primary contributions are incorporated into Chap. 2, where fundamental approaches for optimizing sensing systems are introduced.

In the domain of spectral sensing, [74] serves as our first proof-of-concept for leveraging neural networks in spectral reconstruction with our custom-built sensing devices. This work evaluates multiple neural network architectures and their robustness against noise using the first sensor generation. Expanding upon these findings, [136] revisits and refines our approach with the second sensor generation, incorporating a physics-informed data augmentation technique to significantly improve data availability. Both of these publications form the core of Chap. 5, which presents our advancements in spectral reconstruction. While both publications

establish the foundation for our findings in regards to spectral reconstruction, Chap. 6 introduces novel, unpublished findings that integrate prior research with TinyML techniques. This final chapter demonstrates how the combination of optimized neural networks and hardware-aware design leads to a fully realized, efficient spectral sensing system.

1.2.2 Thesis Overview

As already introduced, this thesis is structured around two complementary research themes: the design of energy-efficient classification systems through hierarchical machine learning, and the development of resource-aware spectral sensing using computational spectrometry. These themes give rise to the following research questions:

RQ1: How can hierarchical machine learning (HiML) be leveraged to reduce computational costs in classification tasks on resource-constrained embedded systems?

RQ2: What optimization strategies, such as reinforcement learning, are effective in selecting energy-efficient classifier hierarchies within hierarchical classification frameworks?

RQ3: How can physics-informed data augmentation improve the diversity and robustness of spectral datasets for computational spectrometry?

RQ4: To what extent can explainable AI techniques guide the reduction of input dimensions in spectral sensors without compromising reconstruction accuracy?

RQ5: How can TinyML methods be applied to compress and optimize neural networks for real-time deployment on microcontrollers in spectral sensing applications?

RQ6: What are the benefits and trade-offs of integrating neural spectral reconstruction and hardware optimization into a fully deployable, resource-efficient sensing system?

Following this introduction, Chapter 2 presents the fundamental methods and tools required to optimize machine learning models for efficient execution on edge devices. Many of these techniques, such as quantization, pruning, and neural architecture optimization, play a crucial role throughout the thesis and contribute to answering all research questions, building on foundational knowledge.

Chapter 3 introduces hierarchical machine learning as a meta-approach for constructing cascaded classification systems. It explores how structured decision-making and classifier hierarchies can reduce energy consumption, thereby addressing **RQ1** and **RQ2**.

The focus then shifts to spectral reconstruction, beginning with Chapter 4, which provides the necessary background in physics and signal processing to contextualize the subsequent contributions. This chapter forms the basis for understanding the domain-specific challenges tackled in **RQ3** and **RQ4**.

Building on this foundation, Chapter 5 demonstrates the applicability of neural networks for spectral reconstruction, leveraging two sensor generations and applying physics-informed data augmentation. It directly addresses **RQ3** and contributes insights relevant to **RQ4** and **RQ5**.

Finally, Chapter 6 integrates the optimization techniques introduced in Chapter 2 to refine both the neural networks and the latest hardware generation. This chapter completes the development of a fully deployable, resource-efficient spectral reconstruction system, providing concrete answers to **RQ4**, **RQ5**, and **RQ6**.

The thesis concludes with Chapter 7, which summarizes the key findings in relation to each research question and outlines directions for future work.

CHAPTER 2

Efficient Machine-Learning Methods

A key requirement for enabling smart sensor systems that integrate machine learning into their computational pipeline is achieving high efficiency—both in terms of energy consumption and resource utilization. Many target platforms for TinyML are highly constrained, featuring limited memory capacities, restricted computational power, and, in many cases, operating under stringent power budgets. These limitations make deploying large-scale machine learning models impractical, as they require substantial processing power, storage, and energy, which exceed the capabilities of typical embedded systems.

To address these challenges, model optimization techniques are essential to ensure that machine learning algorithms can run efficiently while maintaining high accuracy. Optimization strategies must cover multiple dimensions, including reducing memory footprint, minimizing inference latency, and improving energy efficiency. Techniques such as model quantization, pruning, knowledge distillation, and hardware-aware neural architecture search have been developed to systematically compress models while retaining performance. Furthermore, specialized hardware acceleration, such as utilizing Digital Signal Processors (DSPs) and Neural Processing Units (NPUs), can further improve efficiency by offloading computationally intensive tasks from general-purpose microcontrollers [134].

In addition to optimizing individual models, efficiency can also be improved through algorithmic innovations, such as hierarchical processing pipelines. By structuring computations in a cascaded manner, systems can dynamically allocate computational resources, processing only the necessary information at each stage. This approach is particularly relevant in edge computing scenarios, where reducing redundant computations leads to substantial energy savings.

The following chapter introduces a range of optimization techniques tailored to improving the efficiency of machine learning models for embedded applications. It provides an overview of key methodologies, from reducing energy consumption and memory usage to accelerating inference through hardware-aware optimizations. These strategies play a crucial role in making intelligent sensing systems feasible, ensuring that advanced machine learning capabilities can be deployed even in highly resource-constrained environments.

2.1 Scalable Deep Neural Network Architectures

Scaling Deep Neural Networks (DNNs) is a fundamental approach to balancing computational efficiency and model performance, allowing neural networks to be adapted for different deployment scenarios. The capacity of a DNN can be increased or decreased using various strategies, each affecting accuracy, memory consumption, and inference speed in different ways. Common scaling techniques include increasing network depth by adding more layers, adjusting input resolution to influence the level of detail the network processes, or modifying network width by changing the number of filters in convolutional layers. Each of these approaches impacts the trade-off between computational cost and predictive accuracy, making them crucial for designing models that operate efficiently on constrained hardware [95].

For edge deployment, where memory, power consumption, and processing capabilities are limited, selecting an appropriately scaled model is essential. While deeper and wider networks often achieve higher accuracy, they also demand significantly more computational resources. Conversely, reducing model size too aggressively may lead to performance degradation, making it necessary to strike a balance between efficiency and effectiveness. Modern scalable architectures address this challenge by enabling systematic adjustments to model parameters, allowing developers to tailor networks to specific hardware constraints without compromising accuracy.

In its simplest form, scaling a DNN for edge deployment can be as simple as reducing the number of layers or filters in the architecture. Nonetheless, there are additional topologies specifically tailored towards scalability, which we will highlight in this section. By leveraging these architectures, it becomes possible to optimize neural networks for embedded systems, ensuring that high-performance deep learning can be achieved even in resource-constrained environments.

2.1.1 MobileNet

MobileNet [7] is specifically designed for edge devices, prioritizing both model size and runtime efficiency. It is a specialized form of the classic Residual network (ResNet) [79] that are introduced to address challenges in training very deep Neural Networks (NNs) like vanishing/exploding gradients or degradation due to depth [138]. A ResNet mitigates these issues by reformulating the learning process as residual mapping rather than direct function approximation. The authors hypothesize that optimizing residual functions is easier than learning the full transformation directly. To achieve this, ResNet employs shortcut connections that bypass one or more layers, enabling identity mappings that are simply added to the transformed features. This design introduces minimal computational overhead while improving gradient flow, making it possible to train significantly deeper net-

works. The standard residual block consists of two convolutional layers and a skip connection. Such blocks are one possibility to scale a standard ResNet by adding or removing residual blocks based on computational and accuracy constraints.

MobileNets heavily rely on depthwise-separable convolutions, which factorize standard convolution into a depthwise convolution followed by a 1×1 pointwise convolution. This separation significantly reduces both computational complexity and model size compared to traditional convolutions. To introduce scalability, MobileNet defines a width multiplier α that uniformly scales the number of input and output channels at each layer. Additionally, a resolution multiplier ρ scales the spatial dimensions of input images and activation tensors, providing further control over the trade-off between accuracy and efficiency.

Subsequent versions, MobileNetV2 [95] and MobileNetV3 [6], introduce further optimizations. MobileNetV2 replaces traditional residual blocks with inverted residual blocks that use linear bottleneck layers, improving memory efficiency. MobileNetV3 incorporates squeeze-and-excitation attention modules [71] and an optimized activation function (h-swish) to enhance quantization-friendliness and computational efficiency.

2.1.2 EfficientNet

EfficientNet [127] introduces a novel compound scaling approach that systematically balances depth, width, and input resolution to achieve an optimal trade-off between accuracy and efficiency. Unlike traditional scaling methods, which often modify these dimensions independently, compound scaling uniformly adjusts them using a single compound coefficient ϕ. This holistic approach allows the model to scale in a more balanced and efficient manner, ensuring that resources are allocated effectively across the network.

The authors propose the following scaling equations:

$$\begin{aligned} \text{depth scale: } & d = \alpha^{\phi} \\ \text{width scale: } & w = \beta^{\phi} \\ \text{resolution scale: } & r = \gamma^{\phi} \\ \text{s.t. } & \alpha * \beta^2 * \gamma^2 \approx 2 \\ & \alpha \geqslant 1, \beta \geqslant 1, \gamma \geqslant 1 \end{aligned} \tag{2.1}$$

Here, α, β, and γ determine the relative scaling factors for network depth, width, and input resolution, which are multiplied by the respective original network dimension. ϕ serves as a user-defined parameter that controls the overall model size. The constraint $\alpha * \beta^2 * \gamma^2 \approx 2$ ensures that computational cost increases approximately by a factor of 2^{ϕ}, maintaining efficiency in scaling.

EfficientNet models, named B0-B7, are derived by first performing a grid search to find the optimal values for α, β, and γ on a baseline model (B0). Once these parameters are determined, the compound scaling rule is applied to obtain larger variants (B1-B7), each progressively increasing in size while maintaining efficiency. This approach ensures that the scaled models retain a well-balanced structure, unlike conventional scaling methods, which often lead to over-parameterized or inefficient architectures.

The key advantage of EfficientNet's compound scaling lies in its ability to achieve superior accuracy with fewer parameters and lower computational cost compared to traditional architectures. Empirical results demonstrate that EfficientNet models outperform previous state-of-the-art CNNs on image classification tasks while using significantly fewer FLOPs and memory. This efficiency makes EfficientNet particularly well-suited for deployment in resource-constrained environments such as mobile devices and edge computing platforms, where both accuracy and computational efficiency are critical [127].

2.2 Neural Architecture Search

Finding an optimal DNN architecture that satisfies resource constraints on an edge device is typically framed as a multi-objective Hyper Parameter Optimization (HPO) problem [20]. This involves optimizing a black-box function that maps architecture-specific hyperparameters to performance metrics such as accuracy, memory usage, latency, or throughput. In this context, evaluating the black-box function corresponds to training a DNN with a specific configuration on a given dataset.

Neural Architecture Search (NAS) is a process that automates the discovery of efficient DNN architectures. Traditionally focused on maximizing accuracy, NAS methods have been extended to include resource-aware, multi-objective optimization, making them suitable for edge deployments [55]; [94]; [126].

There are three primary NAS approaches discussed in the literature: black-box HPO [17]; [23]; [113]; [132], differentiable NAS [21]; [59], and zero-cost NAS [141]. Black-box HPO is effective but computationally expensive, requiring extensive training and evaluation of different architectures. Differentiable NAS, on the other hand, optimizes architectures during regular training but suffers from stability issues and poor generalization [9]. Zero-cost NAS circumvents direct training by using empirical surrogate models, providing a highly efficient but approximate solution [2]; [55]; [89]; [92]; [97]; [128]; [129]. In the following, we discuss each of these approaches in detail.

2.2.1 Black-Box Multi-Objective Optimization

Black-box optimization is the most common approach for solving NAS problems. Evolutionary algorithms, such as NSGA-II [38], have been widely used, with studies exploring their integration with compression techniques like pruning and quantization [131]. However, evolutionary algorithms are sample-inefficient due to their population-based nature, making them costly in the NAS setting where each evaluation involves training a full DNN.

To address this, Bayesian optimization [81] provides a more sample-efficient alternative. It builds a probabilistic model (typically Gaussian processes) to approximate the black-box function and iteratively refines the search space by selecting new candidate architectures based on an acquisition function. This method significantly reduces the number of required evaluations and has been successfully applied to NAS [94]; [132].

Reinforcement learning (RL) has also been explored for NAS. [17] proposes an recurrent neural network-based controller that generates DNN architectures as variable-length strings, optimizing its search strategy via reinforcement learning. Similarly, [23] frames architecture selection as a Markov decision process, where a Q-learning agent sequentially constructs Convolutional Neural Network (CNN) topologies. These methods effectively refine architecture choices over time but remain computationally expensive, as they require a large number of network evaluations.

Despite its advantages, black-box optimization suffers from scalability issues in high-dimensional search spaces. As the number of architecture parameters grows, the complexity of the search increases exponentially, commonly referred to as the "curse of dimensionality" [18]. This remains a key challenge for black-box NAS methods.

2.2.2 Differentiable Neural Architecture Search

Differentiable NAS [21]; [59] reformulates NAS as a continuous optimization problem, allowing architectures to be optimized via gradient descent rather than exhaustive search. Instead of selecting from a discrete set of architectures, differentiable NAS defines an Directed Acyclic Graph (DAG) called supernet that holds all predefined architectures. Each node in the graph represents a feature representation (e.g., an activation map in CNNs), while the edges correspond to possible operations such as convolutions, pooling, or identity mappings. The differentiability comes from a relaxation where each edge holds a weighted sum of candidate operations. These weights, known as architecture parameters, are adjusted during training to determine the most effective operations for each layer. This continuous relaxation enables the use of gradient-based optimization methods to search for the best-performing architecture. During training, both the supernet's weights and its

architecture parameters are optimized. Once training is complete, the final architecture is derived by selecting the operations with the highest weights, effectively "discretizing" the supernet into a specific neural network configuration [143].

Mathematically, this optimization process is described in [59]. Let $\mathcal{O}$ be a set of candidate operations where each operation represents some function $o(\cdot)$ to be applied to $x^{(i)}$. To make the search space continuous, the categorical choice of a particular operation is relaxed to a softmax over all possible operations. The operation mixing weights for a pair of nodes (i, j) are parameterized by a vector $\alpha^{(i,j)}$ of dimension $|\mathcal{O}|$. The task of architecture search then reduces to learning a set of continuous variables $\alpha = \{\alpha^{(i,j)}\}$. At the end of search, a discrete architecture can be obtained by replacing each mixed operation $\overline{o}^{(i,j)}(x)$ with the most likely operation.

This approach eliminates the need to train and evaluate multiple architectures separately, making it computationally efficient. However, differentiable NAS has been shown to suffer from instability and poor generalization [9]. Moreover, it does not generate a Pareto-optimal set of architectures but rather a single best-performing model, limiting its flexibility in resource-aware optimization.

2.2.3 Zero-Cost Neural Architecture Search

Zero-cost NAS significantly reduces search overhead by using pre-trained supernets from which subnets can be sampled without retraining. The performance of candidate architectures is estimated using zero-cost proxy metrics $\mathcal{P}_{score}$, allowing rapid evaluation. Several proxy metrics exist, including activation overlaps [97], expressiveness measures (Zen-Score) [92], and gradient-based heuristics [2]; [89]; [128]. These metrics allow for efficient architecture ranking without full training and evaluation. Recent advances in zero-cost NAS have facilitated network specialization for diverse hardware platforms. Once-for-All networks [55] train a supernet via progressive shrinking, enabling efficient subnetwork selection post-training. Similarly, PreNAS [129] refines black-box search by pre-filtering architectures with zero-cost ranking, improving search efficiency while maintaining strong performance.

2.3 Quantization

Quantization is a widely used technique for reducing the energy consumption of DNNs by minimizing their size and computational complexity [84]. It applies to parameters such as weights, biases, and scaling factors as well as activations, leading to several benefits. It reduces the model size and therefore storage requirements, which is particularly important for deployment on edge devices. Additionally, quantization simplifies operations, in particular Multiply-Accumulate (MAC)

computations, resulting in lower power consumption. By decreasing the bit-width of parameters and activations, memory access and buffer requirements are also optimized, further improving efficiency.

2.3.1 Quantizers

Quantization maps input values to discrete output values and is generally categorized into uniform and non-uniform types. Non-uniform quantization, often implemented via lookup tables, is mainly used for model compression rather than computational efficiency. In contrast, affine uniform quantization

$$x_Q = \text{clamp}\left(\frac{\text{round}(x)}{s} + z, a, b\right) \tag{2.2}$$

is the standard approach for DNNs, where round() is a rounding function, s the scaling factor, z the zero-point, and a, b the range of the clamping function

$$\text{clamp}(x, a, b) = \min(\max(x, a), b) \tag{2.3}$$

that limits the maximum and minimum values of the quantizer. The transformation in (2.2) preserves the relative spacing between values but introduces an offset. Therefore, affine uniform quantization is also often called asymmetric Quantization. In contrast, symmetric quantization sets $z = 0$, meaning the mapping remains centered around zero without an explicit offset.

The choice of the rounding technique uniquely differs between quantization approaches. Common functions range from rounding to the nearest integer $\text{round}(x) = \lfloor x \rceil$, to rounding up $\lceil x \rceil$ or down $\lfloor x \rfloor$. [54] introduces randomness to the process with stochastic rounding

$$\text{round}(x) = \lfloor x \rfloor + (p > x - \lfloor x \rfloor), \tag{2.4}$$

where p is a random variable sampled from a uniform distribution. A trainable rounding method is introduced in [102], where the authors select the rounding function for each individual value by minimizing the overall network loss. The quantizer in (2.2) can be limited further to integers with bit-widths n by choosing

$$a = 0 \text{ and } b = 2^n - 1. \tag{2.5}$$

This limit not only reduces the memory footprint by lowering the number of bits for each quantized value, but makes the MAC operations more efficient due to the abundance of floating point operations [62]; [64]; [73]. For further efficiency gains,

the quantization can be parameterized with $a = -2^{n-1}$, $b = 2^{n-1} - 1$ and $z = 0$ to replace any multiplication or division with bit-shifts [41].

2.3.2 Quantization Granularity

Neural network inference relies on MAC operations

$$\mathbf{O} = \mathbf{b} + \mathbf{W}\mathbf{x}, \tag{2.6}$$

where $\mathbf{b}$ represents the bias vector, $\mathbf{W}$ the weight matrix, and $\mathbf{X}$ the activations from the previous layer or inputs to the network. In the quantized version

$$\mathbf{O} = \mathbf{b} + \mathbf{W}_q\mathbf{x}_q. \tag{2.7}$$

the weight matrix and activations are replaced with their respective quantized representation. In granularity the user can decide to either quantize the weights only or also include the activations. While weight-only quantization simplifies the implementation [46]; [146], full quantization offers greater efficiency but often requires datasets for calibration [70]. The biggest downside of weight-only quantization is the need for floating point operations during MAC. As only $\mathbf{W}_q$ is in integers, the multiplication $\mathbf{W}_q\mathbf{x}$ still needs to use a floating point unit, mitigating the positive effects of quantization in terms of efficiency. Most publications do not quantize the bias value $\mathbf{b}$, as reducing its precision heavily degrades the network's accuracy [41]. For example, in [70] the weights and activations are quantized to 8 bit integers, while the bias remains at 32 bit floating point. Also, the addition of the bias to the result of $\mathbf{W}_q\mathbf{x}_q$ is much less complex as no multiplications are needed.

Apart from the choice of weight-only or full quantization, the granularity also determines how quantization parameters are applied across the network. Per-layer quantization assigns a single set of parameters per layer, while channel-wise quantization refines this by applying distinct quantizers per output channel or kernel [88]; [112]. Some approaches further divide parameters into groups [37]; [104], balancing accuracy and computational overhead. In total, the quantization granularity describes a trade-off between simplicity and overhead for extra parameters. While a more coarse quantization can be implemented with nearly no extra cost, substantial efficiency gains can only be achieved with a finer quantization, adding the need for calibration datasets and extra memory for the additional parameter sets.

2.3.3 Quantization Methods

For quantization mainly two methods exist: Post-Training Quantization (PTQ), and Quantization-Aware Training (QAT). PTQ describes the process of adding quanti-

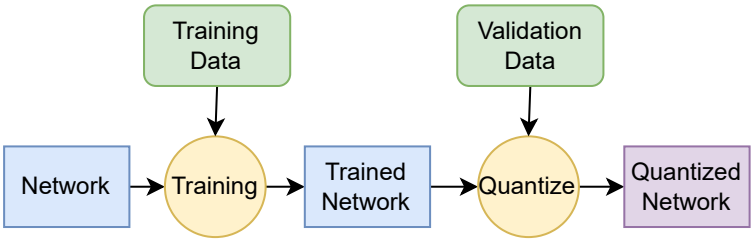


Figure 2.1: Illustration of post-training quantization, illustrated after [41]. The quantization is applied after training with an optional validation data step to tune the quantization range.

zation to an already trained network. In contrast, QAT includes the quantization in the training process, which is currently in a research state [41].

As depicted in Fig. 2.1 PTQ applies quantization to a fully trained network without requiring retraining. This makes it an efficient and widely adopted technique, especially for edge devices, where computational resources are limited. It is commonly used for converting networks to 8-bit representations [88]; [139], and in some cases, even to 4-bit [16]; [46]; [104]; [130]. While weight quantization is relatively straightforward, activation quantization introduces additional challenges. To mitigate accuracy loss, methods such as clipping [16] or dataset distillation [26] are employed. Clipping limits extreme activation values to ensure numerical stability, while dataset distillation generates synthetic data to estimate appropriate activation ranges. Despite its simplicity, PTQ can lead to significant performance degradation, particularly when reducing precision below 8-bit. Edge-deployment focused Deep learning frameworks such as TensorFlow-lite already support PTQ, with mostly all of the above mentioned quantization approaches already implemented.

QAT integrates quantization directly into the training process [70]; [103]; [109], allowing the model to adapt to low-bit-width constraints. Unlike PTQ, QAT simulates quantization effects during forward and backward passes (cf. Fig. 2.2), enabling the network to learn and compensate for quantization noise. This approach is particularly beneficial for aggressive quantization, such as sub-8-bit precision, where direct post-training quantization would cause unacceptable accuracy drops.

A key challenge in QAT is handling non-differentiable rounding operations within backpropagation. To address this, methods such as the Straight-Through Estimator [19] approximate the gradient of the rounding function, allowing gradients to

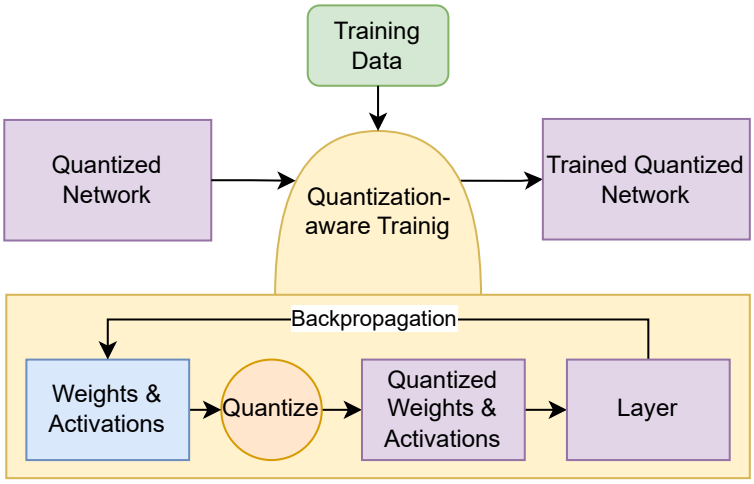


Figure 2.2: Illustration of quantization-aware training, illustrated after [41]. The weights and activations are quantized during training using the gradients calculated during backpropagation

flow through discrete operations. Additionally, QAT often employs per-layer or per-channel quantization to maintain accuracy while optimizing computational efficiency. Although more computationally intensive than PTQ, QAT generally produces more robust low-precision models suitable for deployment on energy-constrained hardware.

2.4 Pruning

An effective method to decrease a model's size is pruning. In general terms, pruning describes the process of removing unnecessary neurons, filters or even complete layers. This approach is based on the idea that due to the vast amount of parameters especially deep NNs have, models are often over parameterized, leading to redundancies [40]. Therefore, if done correctly, pruning can lead to a highly compressed model without decreasing the accuracy of it and can, in some cases, even improve the accuracy. However, determining which part of a neural network is not contributing enough to the performance is a complex task that has been studied thoroughly in the past [41]; [58]; [61]; [87].

2.4.1 Pruning granularity

Before pruning, it is necessary to decide to which extent the network should be pruned. The literature usually categorizes pruning granularity into two categories: Structured and unstructured pruning. In Structured pruning, complete structures

such as CNN filters [90] or channels [93] from the network. In some publications, structured pruning can also be extended to complete rows or columns of dense layers. In contrast, element pruning focuses on removing or setting specific weights to zero [61]; [87]. In a similar fashion as Dropout [47] this approach has first been used to tackle network generalization and overfitting rather than efficiency. However, more recent studies exploit the introduced sparsity to also save energy during inference [91].

Structural pruning is often times seen advantageous in comparison to element pruning for neural network compression, as higher compression rates can be achieved with the removal of complete structures. Nonetheless, this operation is also more invasive, potentially hurting the resulting accuracy. This is because the overall structure and shape of a neural network is often connected to the inherent data structure, leading also to changing shapes in output feature maps. This is especially problematic with branched NNs like res-nets [79], where the overall shape of a feature map needs to match to enable residual connections. Therefore, controlling the balance between accuracy and compression can be much harder with Structural pruning.

2.4.2 Pruning Heuristics

As already mentioned, it is essential to select suitable candidates for pruning to achieve a good trade-off between accuracy and compression. The brute-force method to this problem would be to remove each element or structure one-by-one to assess their respective relevance to the performance. However, this so called oracle criterion [99] is computationally demanding and not applicable for most deep NNs. Therefore, a complete research field exists that investigates computationally suitable heuristics for pruning, often called sensitivity analysis [57]. For a good tradeoff between structural and element pruning, the authors in [140] first apply some structural pruning before using element pruning as a refinement method. With their approach they combine the best of both worlds with a higher compression rate than pure element pruning while maintaining its selectiveness.

Heuristic Approaches for Parameter Pruning

Early pruning methods rely on second-order derivative-based heuristics to approximate parameter importance. [87] and later [61] introduce saliency scores to quantify relevance, removing parameters deemed least significant. In [87] the authors term this method *optimal brain damage*, using a Taylor series expansion to approximate the network's objective function without requiring an oracle. However, this approach introduces additional computational overhead.

To address this, more recent work favors simpler heuristics based on magnitude, thresholds, or gradient information [56]; [57]; [99]. Threshold-based pruning removes parameters with absolute values below a predefined threshold, assuming that higher-magnitude values contribute more significantly to activations and the layer's output. However, manually selecting thresholds is unintuitive and requires extensive network analysis. To mitigate this, weight distributions in neural networks are often approximated using zero-mean Gaussian distributions, with the standard deviation serving as an adaptive pruning threshold. This reduces the number of unknowns to a single scaling factor. The authors in [57] determine these factors empirically by gradually increasing sparsity in an unpruned baseline network and tracking accuracy degradation. However, this process remains costly, as it necessitates training the baseline network beforehand.

Gradient-based heuristics extend beyond magnitude-based approaches by incorporating sensitivity analysis [99]. Similar to optimal brain damage, these heuristics estimate parameter significance using a Taylor series expansion, measuring the impact of parameter removal on the loss function. Specifically, they accumulate the product of activation tensors and their gradients with respect to the cost function. Since these gradients are readily available via backpropagation, the method is computationally efficient. Structures with small parameter values and flat gradients are considered less significant, as they contribute minimally to network optimization.

Structure Pruning Heuristics

Heuristic methods extend beyond individual parameters to entire structures within neural networks. A common approach ranks parameter structures using the l_1 norm [90]. This norm estimates the overall magnitude of filters in convolutional layers and serves as a proxy for the expected scale of output feature maps. In some cases, the l_2 norm is also used to evaluate structure importance.

Another approach leverages activation sparsity to approximate the significance of parameter structures [65]. This method monitors the average percentage of zero values in activation tensors during training, particularly in rectified linear unit Rectified Linear Unit (ReLU) activations. Structures generating high percentage are considered less significant and thus more suitable for pruning. [99] also references this approach but caution against its application in early layers, which typically capture low-level features and produce denser activation tensors. In contrast, deeper layers, responsible for more refined feature extraction, exhibit greater sparsity, making zero value-based pruning more effective in later stages.

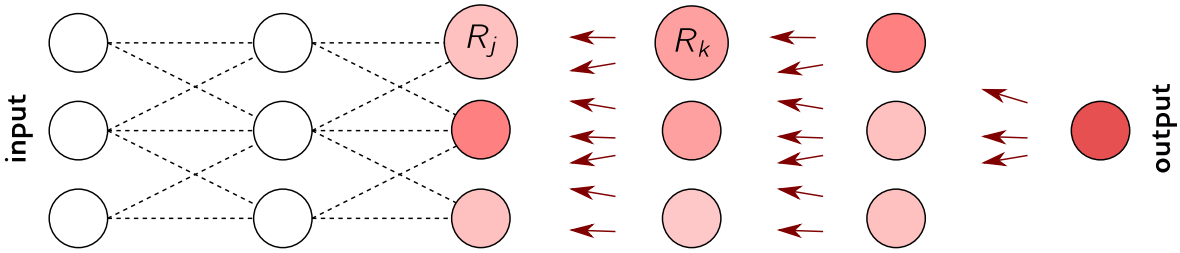


Figure 2.3: Illustration of LRP dataflow. Figure taken from [100] with permission. The relevance score is propagated backwards through the network until it reaches the input neurons

Relevance-based Heuristics

An additional pruning heuristic can be obtain by computing the relevance of a neuron with Layerwise Relevance Propagation (LRP) [100]. In its simplest form the relevance

$$R_i^{(l)} = \sum_j \frac{z_{ij}}{z_j} R_j^{l+1} \tag{2.8}$$

for a neuron i in layer l based on the proportional decomposition of the relevance scores of the neurons R_j connected to i in an upper layer $l+1$ is calculated wrt. to the localized pre activations z_{ij} and their respective aggregations z_j. When propagating the relevance from an individual output neuron back to the input space, it is possible to calculate the relevance of each input feature to the decision. To grasp the score calculation visually Fig. 2.3 shows the overall dataflow through the network when calculating the LRP scores.

Besides the basic formulation in Eq. 2.8, also called LRP_z, multiple extensions called LRP-rules exist in the literature. The authors in [83] summarize the best practices and give guidance on where to apply which deviation of LRP. With LRP_ϵ, a small constant ϵ is added to the denominator in Eq. 2.8, avoiding divisions by zero. $LRP_{\alpha\beta}$ splits the fraction in LRP_z into two addends

$$\alpha \frac{z_{ij}^+}{z_j^+} + \beta \frac{z_{ij}^-}{z_j^-}$$

one for positive (activatory) and one for negative activations (inhibitory). The resulting addends are then weighted with α for the activatory and β for the inhibitory part. Normally, β is given implicitly with $\alpha + \beta = 1$ and $\alpha \in \{1, 2\}$. Further LRP-rules can be found in [100], but are not as popular as the above mentioned variants.

The LRP approach originates from the field of explainable AI, where it is used to explain the relevance of inputs w.r.t. its outputs, leading to a heatmap representation

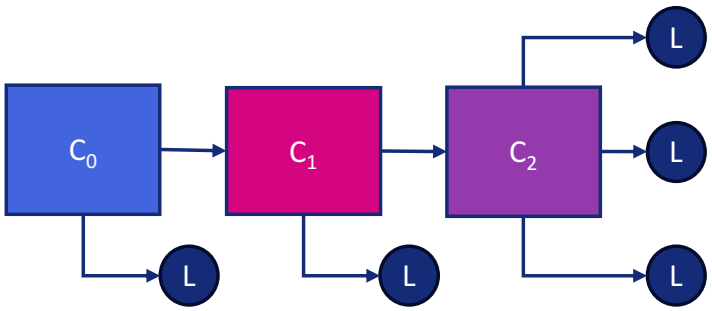


Figure 2.4: Simple cascade without any branching. Multiple classifiers (C_0 to C_2) process a data sample one after the other with some early labels (L) classifiable in earlier stages.

for each individual output neuron. As each output is often connected to a class, LRP can explain the influence of each input pixel to the given decision. As shown in the later chapters of this thesis, we can exploit this behavior by computing the relevance of each input channel w.r.t each reconstructed wavelength. Our output neurons represent one wavelength in the reconstructed spectrum each, while the inputs map directly to a channel of the sensor. Therefore, we can use LRP to not only prune neurons in the network, but also optimize the physical sensor by selecting and manufacturing only the most relevant channels.

2.5 Cascaded Processing

In addition to previously discussed methods, reducing the energy footprint of smart sensor systems can also be achieved through cascaded processing. This approach replaces a single large model with a sequence of smaller models, executed in a step-by-step manner. By integrating exit points between these models, the system can selectively activate subsequent stages, consuming only as much energy as needed to reach a decision (cf. Fig. 2.4). Each model in the sequence refines the previous results, allowing the process to halt as soon as sufficient confidence is achieved. This section outlines key advancements in cascaded processing, starting with hierarchical models and their relevance to distributed sensor networks, followed by Early-Exit NNs., which extend this concept to neural architectures.

2.5.1 Hierarchical Systems

Hierarchical models provide a natural framework for cascaded processing, with a prominent example being condition monitoring. Consider a system that classifies a machine's status into fault types and severities. Initially, a lightweight anomaly detection model operates continuously, distinguishing between normal and abnormal behavior. If an anomaly is detected, the next model identifies the fault type. Subsequent models focus on the severity of specific faults, stopping processing if

no further classification is required. This hierarchical design optimizes resource use by tailoring each model to its specific task.

The foundational work in [27] normalizes hierarchical classification using a class taxonomy modeled as a partially ordered set (poset) $(C, \prec)$, where C represents class concepts and $\prec$ defines a partial order that is asymmetric, anti-reflexive, and transitive. While this structure corresponds to a DAG, it is often simplified to a tree structure with single-parent nodes for practical implementations. They categorize hierarchical classification models into four main types:

- **Local Classifier per Node (LCN)**: Binary classifiers at each node
- **Local Classifier per Parent Node (LCPN)**: Multi-class classifiers for parent nodes
- **Local Classifier per Level (LCL)**: A single multi-class classifier for each level
- **Global Classifier (GC)**: A single comprehensive model for all classes, also known as flat classifier.

To formalize hierarchical systems, [27] defines hierarchical problems as a 3-tuple

$$(\Upsilon, \Psi, \Phi), \tag{2.9}$$

where Υ specifies the graph type: Tree or DAG, Ψ determines whether classes follow a single path (SPL) or multiple paths (MPL), and Φ indicates label depth: Full Depth (FD), or Partial Depth (PD). Algorithms are described analogously using a 4-tuple

$$(\Delta, \Xi, \Omega, \Theta). \tag{2.10}$$

Here, Δ describes the path prediction type: Single-path prediction (SPP) or Multi-path prediction (MPP), linked to Ψ, Ξ the leaf-node prediction type: mandatory (MLNP) or non-mandatory (NMLNP) leaf-node prediction, related to Φ. Ω the graph compatibility (Tree or DAG), tied to Υ, and Θ the classifier type (LCN, LCPN, LCL, or GC). For energy-efficient designs, the hierarchical configuration is often simplified to:

$$(\Delta, \Xi, \Omega, \Theta) = (\text{MPP, MLNP, Tree, LCPN}).$$

This structure is well-suited for applications like condition monitoring, where the *no-fault* label is often determined early and expected with a high frequency. MLNP ensures computation halts once a confident decision is reached, while tree structures reduce complexity. LCPN classifiers provide the flexibility needed to handle asymmetric taxonomies with multi-class predictions. Such a hierarchical model can be seen in Fig. 2.5

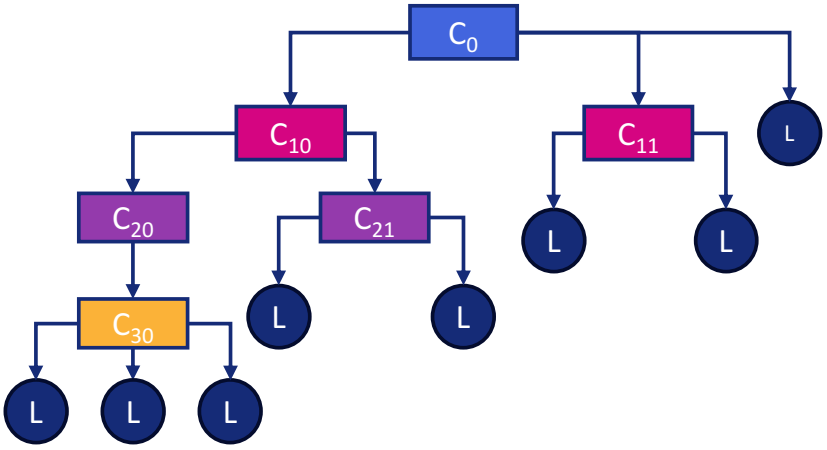


Figure 2.5: Complex hierarchical tree model. Multiple classifiers in multiple levels (C_0 to C_{30}) process a data sample one after the other with some early labels (L) classifiable in earlier stages.

Even though the benefits of cascaded processing is clear, hierarchical models face challenges with partially correct predictions, as standard metrics typically penalize such cases. For instance, a model might correctly classify a fault type but misjudge its severity. To address this, [80] proposes hierarchical precision, recall, and F-measure metrics:

$$hP = \frac{\sum_i |\hat{P}_i \cap \hat{T}_i|}{\sum_i |\hat{P}_i|}, \quad hR = \frac{\sum_i |\hat{P}_i \cap \hat{T}_i|}{\sum_i |\hat{T}_i|}, \quad hF = \frac{2 \cdot hP \cdot hR}{hP + hR}, \tag{2.11}$$

where $\hat{P}_i$ includes the predicted class and its parents for sample i, and $\hat{T}_i$ contains the true class and its parents. By accounting for hierarchical relationships, these metrics better reflect the model's performance in handling partially correct predictions.

The energy-saving potential of hierarchical models for small sensor systems is further explored in [82], where the authors introduce cascaded LCL models with lazy-trigger mechanisms. They propose a pass-on label to defer uncertain decisions to the next stage, impacting the hierarchical metrics defined in (2.11). The pass-on rate Pass On Rate (POR) quantifies this deferral, with a POR of 1 indicating that all decisions are deferred to the final classifier, yielding standard precision and recall metrics. Additionally, the authors assign cost measures to each stage, balancing memory and computation. They show that optimizing the number of stages can significantly reduce energy costs, especially in skewed class distributions where early-stage predictions dominate. For instance, in condition monitoring, the high frequency of *no-fault* labels ensures that early classifiers suffice, leading to substantial energy savings.

The choice of model type for each stage also impacts efficiency. Adams et al. [3]; [121] optimize this selection using reinforcement learning, where an agent bal-

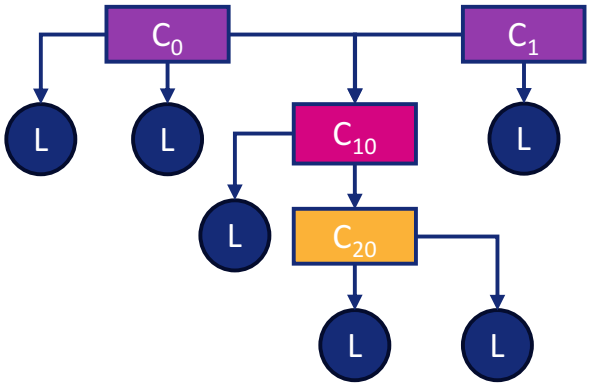


Figure 2.6: Hierarchical model for distributed computing. The overall structure is flipped in comparison to a standard tree, as some nodes need information from multiple parent nodes (C_0 to C_{20}) some labels (L) can be classified on-device without the need to connect to a more complex classifier.

ances accuracy and computational cost. We extend this approach with real hardware measurements, highlighting differences in model selection influenced by implementation factors like compiler optimizations. For further details on our approach to model selection, refer to Sec. 3.1

2.5.2 Distributed Computing

Another promising application of hierarchical models lies in distributed computing for IoT devices. In [8], the authors present a distributed hierarchical inference approach where parts of the hierarchy are executed locally on IoT devices, while others run in the cloud. This setup reduces unnecessary communication overhead, as only the essential data is transmitted. For instance, in a system of interconnected IoT appliances used for home automation, a traditional approach would send all data to a central hub or cloud infrastructure at every time step, resulting in significant communication costs. However, many decisions can be made locally without requiring the context of other devices. In a hierarchical framework, lightweight models running directly on IoT devices handle these decisions. If the local model cannot confidently reach a conclusion, the task is escalated to the cloud, where the next layer of the hierarchy is triggered.

The authors test this method on three datasets from distinct application areas—urban energy demand, human activity, and server performance. The results show a reduction in system energy consumption by 62%, achieved with a taxonomy of Multi-Layer Perceptron (MLP) models. Notably, the authors employed a DAG structure instead of a traditional tree, as multiple parent nodes often share the same child in IoT applications. This modification accommodates the nature of IoT systems, where decisions often require input from several sensors, making the hierarchy resemble an inverted tree structure better represented by a DAG (cf. Fig. 2.6).

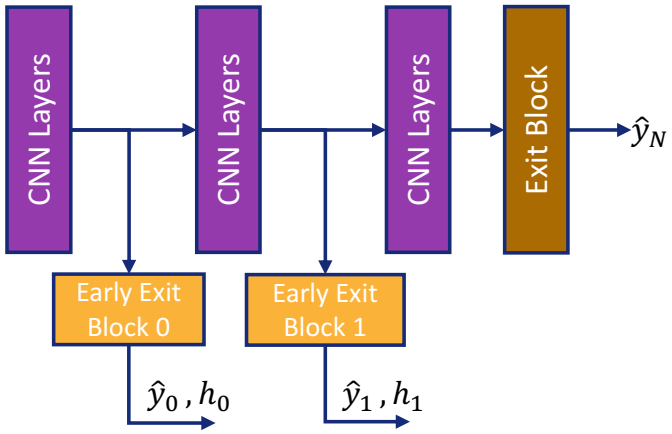


Figure 2.7: Structure of an Early-Exit Neural Network. After a CNN-block the computation can stop at early exists based on a confidence score h_i.

The practical benefits of hierarchical processing in IoT systems are further validated in [43], where the authors build a physical setup to measure the energy consumption of an LCL model for human activity recognition. In this model, a cascade of classifiers is partially executed directly on the device. A key innovation in this work is the introduction of an offload controller – an additional classifier trained to decide whether a task should remain on the device or be offloaded to a central hub. This controller operates with a simplified version of the problem, using only the features necessary for the next layer in the hierarchy. By performing a low-complexity pre-evaluation, the controller reduces the computational overhead associated with the full problem. The study demonstrates a reduction in energy consumption of three times compared to a fully offloaded system, including communication costs, while also achieving a slight improvement in accuracy.
Building on this concept, [78] proposes a dynamic split mechanism for computation between multiple IoT devices and a single NN. In their scenario, several IoT devices connect to a central hub, with the system dynamically deciding whether to process computations locally or offload them based on available network bandwidth. Additionally, they train a Q-learning agent to dynamically select the most efficient network type for transferring data between layers of the NN from WiFi, cellular, or Bluetooth. Their results show a 13.5% reduction in inference time compared to fully on-device execution, including communication delays. However, the study does not compare energy consumption metrics. While this approach is time-efficient, it may still consume more energy than purely on-device execution. Nonetheless, the method highlights valuable considerations when designing hierarchical models, particularly in scenarios where inference speed is critical.

2.5.3 Early-Exit Neural Networks

Most hierarchical approaches discussed so far rely on classical machine learning methods, such as Support Vector Machines or Random Forests. While some hier-

archies incorporate MLPs, the work in [125] extends the concept of lazily triggered computations to CNNs. Following the principle of NMLP in traditional hierarchies, the proposed BranchyNet includes exit points within the architecture. The network halts computation early when a decision is possible based on a confidence score (cf. Fig. 2.7). Similar to nodes in classical hierarchies, the authors define a branch as a non-overlapping subset of the network, characterized by a single entry point and multiple exit points. To train BranchyNet, the authors utilize a weighted loss function:

$$L_{branchynet}(\hat{y}, y, \theta) = \sum_{n=1}^{N} w_n L(\hat{y}_{exit_n}, y; \theta), \tag{2.12}$$

where w_n weights the loss for each exit n, $\hat{y}$ and y denote the predicted and true labels, and θ represents the parameters of the preceding branch. By adjusting w_n, the network learns to optimize either for earlier exits, prioritizing speed, or later exits, focusing on accuracy. Higher weights for earlier exits encourage discriminative feature learning in initial branches, while higher weights for later exits improve overall accuracy but reduce early exits.

During training, BranchyNet operates as a single network, without halting computation. For inference, it evaluates the entropy at each exit point, stopping computation when the entropy drops below a threshold T_n. This reduces inference time, with T_n serving as a hyperparameter that balances accuracy and runtime. The authors test this approach using three CNN architectures (LeNet, AlexNet, and ResNet), incorporating exit points into each. A sweep over T_n values enables a detailed trade-off analysis between accuracy and runtime. For all tested architectures, BranchyNet achieves a 2x-6x reduction in runtime while maintaining comparable or improved accuracy. Additionally, aggressive T_n values reduce cache misses, suggesting that thoughtful branch design can enhance system-level performance by effectively utilizing cache.

A similar concept is presented in [22], where the authors introduce exit points with a learnable decision function $\gamma_n(\sigma_n(x))$. Based on the output σ_n of the previous CNN layer, this function determines whether to halt execution. The authors also propose a method to populate branches with different architectures, allowing simpler networks like AlexNet to handle easier classes while reserving more complex models like ResNet for challenging cases. This adaptive strategy achieves a speed-up ranging from 1.5x to 7.53x, depending on the acceptable accuracy trade-off. For example, with ResNet50, a speed-up of approximately 2x is achieved while keeping accuracy degradation below 1%.

Another early-exit strategy, called MSDNet, is proposed in [48]. This approach introduces intermediate classifiers interconnected by small classification networks. The authors address two challenges with early exits: (1) early layers often lack coarse-level features, reducing classification accuracy for earlier exits, and (2) in-

termediate classifiers may interfere with the final exit, harming its accuracy. To resolve the first issue, they add parallel convolutional layers to compute coarse-scale features, which are concatenated with finer-scale features for intermediate classification. To address the second issue, they introduce connections between early and late convolutional layers, similar to dense networks. This design forwards information from earlier stages, ensuring the final exit remains independent of intermediate exits.

Optimizing Early-Exit Neural Networks for on-device execution is explored in [67], where the authors propose an on-device transfer learning approach for personalizing intermediate exits. They train a generic MSDNet with equidistantly placed exits and fine-tune only the intermediate exits on the device. This process leverages the network's bypass connections, ensuring the final exit's performance remains unaffected. Even without labeled field data, the intermediate exits learn from the final prediction. The authors further refine the exit thresholds using a Pareto front approach with a small user-provided dataset, optimizing the trade-off between inference time and accuracy. This method results in significant improvements in both metrics.

Further advancements are demonstrated in [144], where the authors adapt BranchyNet to include energy-aware criteria. Instead of relying solely on entropy for branch decisions, their system factors in the device's battery level. If the remaining capacity falls below a threshold, only shallow branches are computed; otherwise, entropy governs the decision. The authors perform extensive energy profiling and hardware-aware optimizations, deploying their approach on resource-constrained hardware with just 24kB SRAM and 128kB flash memory. They also estimate the required battery size for their system, showcasing the importance of platform-specific optimization in energy-efficient inference.

2.5.4 Discussion

The reviewed literature highlights that cascaded systems significantly reduce latency compared to flat classifiers, often translating into lower energy consumption. Hierarchies in these systems range from simple cascades to more complex tree or DAG structures with multiple branches. A key factor enabling their efficiency is the uneven distribution of data in real-world scenarios, where certain labels are either more frequent or easier to predict. This unevenness allows the system to terminate computation early when specific conditions are met, a principle encapsulated by the NMLNP property of hierarchical models.

When nodes in a hierarchy are interpreted as subsets of a neural network, the NMLNP property aligns with the behavior of early-exit neural networks. These networks leverage coarse features computed by initial CNN layers, which, for certain labels, suffice to make accurate predictions. By introducing intermediate exits, the

network assesses a confidence score to determine whether computation can halt early. This design effectively turns the decision into a hyperparameter optimization problem focused on threshold settings. By adjusting these thresholds, the network achieves a balance between accuracy, which favors later exits, and energy efficiency, which encourages frequent use of earlier exits.

2.6 Target Platforms

Deploying machine learning models on embedded systems presents unique challenges due to constraints in processing power, memory, and energy availability. TinyML aims to address these limitations by optimizing models for execution on low-power microcontrollers and edge devices. Choosing an appropriate hardware platform is crucial for balancing inference speed, power consumption, and model accuracy. This section provides an overview of the key target platforms for TinyML, covering microcontrollers, edge Tensor Processing Units (TPUs), and specialized neuromorphic processors.

2.6.1 Microcontrollers and Low-Power CPUs

Microcontroller Units (MCUs) are among the most widely used platforms for TinyML applications due to their ultra-low power consumption and widespread availability. These devices typically feature ARM Cortex-M series cores, such as the Cortex-M4 and Cortex-M7, which support fixed-point operations and are optimized for real-time applications [10]. Despite their limited computational resources, often operating at frequencies below 200 MHz with less than 1 MB of RAM, MCUs can run efficient neural networks utilizing the techniques mentioned above and specialized libraries.

Popular microcontroller-based platforms for TinyML provide efficient environments for running machine learning models on resource-constrained devices. In general, most MCUs can utilize the plain implementation of TensorFlow Lite for microcontrollers [145], which allows deep learning models to run on them with as little as 32 KB of RAM. This framework enables lightweight neural networks to perform real-time inference on low-power embedded systems. Another widely used platform is the STM32 series [123], which incorporates dedicated AI accelerators to enhance inference performance while maintaining energy efficiency. These microcontrollers are particularly suited for applications requiring real-time processing with minimal power consumption. They utilize STM's specialized X-Cube AI framework to optimize and run inference of multiple model formats including TensorFlow Lite and ONNX. Additionally, the ESP32 and ESP-CAM [42] are frequently employed in IoT applications, as they feature built-in wireless connectivity, enabling seamless integration into smart sensor networks and edge computing

systems. These platforms collectively demonstrate the feasibility of deploying intelligent models to already existing MCUs, expanding the range of applications for embedded AI [1].

2.6.2 AI Accelerators and Tensor Processing Units

For more demanding workloads, dedicated AI accelerators provide a balance between efficiency and performance. Google's Edge TPU is a well-known example, designed to execute quantized deep learning models with low power consumption while delivering high inference speed [51]. Other similar accelerators include the NVIDIA Jetson Nano [108], which supports full-precision floating-point computations but requires significantly more power than MCUs.

Edge AI accelerators are ideal for applications requiring real-time processing of sensor data, such as image recognition in drones or real-time speech processing. Unlike cloud-based models, these accelerators enable on-device inference, improving response times and reducing the need for data transmission [15].

2.6.3 Neuromorphic Processors and Specialized Hardware

Emerging neuromorphic processors, such as Loihi [49] and TrueNorth [39], take inspiration from biological neural networks to perform energy-efficient spiking neural network computations. These chips consume orders of magnitude less power than conventional CPUs and GPUs while achieving competitive performance on classification tasks. Similarly, analog and mixed-signal accelerators take the same approach skipping the digital computation of models by calculating the MAC operations in an analog circuit [101]. However, while these platforms show superior energy-efficiency, they heavily restrict the model topologies by either only supporting a selection of layers and operations, or by limiting the number of parameters a layer can have to completely fit into the accelerator [29].

Additionally, reconfigurable architectures such as Field-Programmable Gate Arrayss (FPGAs) (Field-Programmable Gate Arrays) allow for highly customized neural network implementations optimized for low latency and power efficiency [45]. However, FPGA programming remains complex, making these solutions less accessible for general TinyML applications.

CHAPTER 3

Hierarchical Machine Learning

A central aspect of energy-efficient machine learning is HiML. As discussed in Sec. 2.5, HiML can be seen as a subset of cascaded processing that enables far more complex topologies than a simple cascade. Therefore, building an optimized hierarchy can be one of the most essential tools on a meta-level to execute a machine learning model efficiently. Especially in cases where energy-intensive operations like executing a large machine learning model or communicating wirelessly should be reduced to a minimum, HiML can be helpful to delay such operations. A prominent example of this approach are smart assistants that, even running continuously, do not need vast amounts of battery. This is due to a cascaded system first performing keyword spotting before executing further levels in the hierarchy, like speaker recognition or communicating with a server to trigger a large language model [135]. Therefore, HiML is a powerful tool to enable local computing in resource constraint devices for smart assistants and various applications such as wireless sensing or condition monitoring of machines [117].

3.1 Taxonomy Search

Despite its benefits, cascaded systems, especially hierarchies, have a considerable drawback linked to increased topological complexity. As discussed in Sec. 2.2, parameterizing and selecting one machine learning model is already a research field. Scaling this problem to a complete hierarchy requires even further optimization. This chapter looks at selecting classifiers and features for a hierarchical system. Based on a condition monitoring example, we first exhaustively search for the best possible taxonomy before investigating a selection strategy based on reinforcement learning. Afterward, we analyze the data dependency of hierarchical machine learning and its impact on battery life.

3.1.1 Problem Statement

As a proof of concept, we utilize a condition monitoring example based on the popular CWRU bearing dataset [28]. It uses vibration data recorded from a 2HP motor connected to a dynamometer via a torque transducer/encoder. The connection is established with bearings supporting the motor shaft. These bearings have been artificially damaged at different locations, with fault depths ranging from

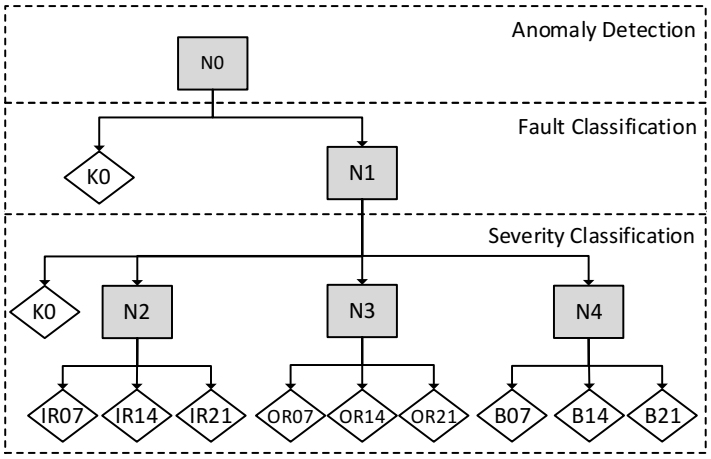


Figure 3.1: Hierarchy used for the CWRU bearing dataset. Illustration taken from [135] with permission. The problem is separated into three levels: Anomaly detection, fault classification, and severity detection. Each level has its own set of classifiers and potential labels.

0.007" to 0.021". The vibration signals are captured using single-axis accelerometers, sampled at 12kS/s (fan-end). Due to its inherent hierarchical structure, the CWRU-Dataset is a perfect example to study HiML on. We separate the problem into three levels: anomaly detection, fault classification, and severity detection (cf. Fig. 3.1). In the first level, a classifier (N0) differentiates faulty and non-faulty samples. Since this task is simple but executed for every incoming measurement, N0 needs to be lightweight. The faulty samples are then further processed in the fault classification step by N1 sorting the faulty sample into the three fault categories: inner ring (IR), outer ring (OR), or ball (B). Additionally, the fault classifier can correct for false positives from the anomaly detection by classifying a sample as non-faulty (K0). The last level classifies the severity of each sample with three distinct classifiers (N2-N4). Depending on the decision of N1, a different model is used to allow them to specifically train on a certain fault type. This design decision is based on the idea that some fault types might be easier to categorize in their severity class than others. After any classifier in the severity classification level, the inference process terminates with the decision for severity (07, 14, 21). Combined with the respective path through the hierarchy, the final label comes down to IR/OR/B-07/14/21.

The inherent hierarchy of the CWRU-bearing dataset is a prime example of straightforward hierarchical segmentation. However, not every problem can be broken down as quickly as our example, showing the dependency on expert knowledge in HiML. Identifying hierarchical structures can be a challenging task in many applications. However, at least a two-stage cascade with an initial anomaly detection should be applicable in nearly every battery-driven environment. In our case, the hierarchy has a good balance between complexity and potential efficiency gains. To measure the resulting efficiency, we must fill the topology in Fig. 3.1 with feature extraction and classification steps leading to the final taxonomy.

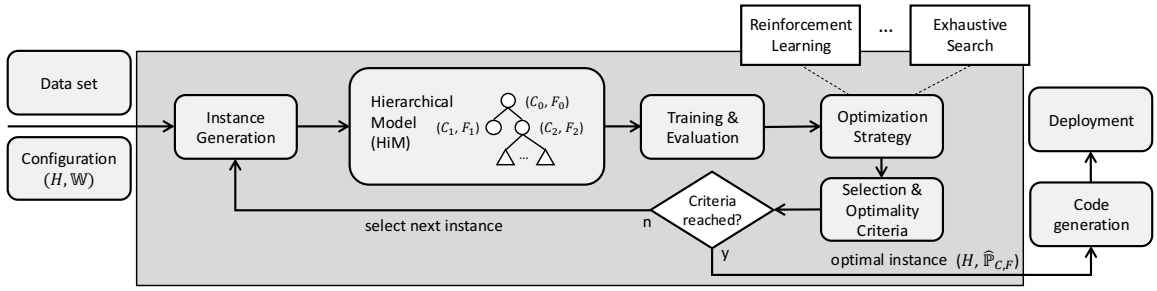


Figure 3.2: Overview of the HiMLEdge framework. The framework creates instances of hierarchical models by selecting combinations of classifiers and feature extractors (cf. Fig. 3.3). Given a model structure and a dataset, it iteratively generates and evaluates instances using an optimization strategy such as reinforcement learning or exhaustive search. Once optimization is complete, the framework can generate deployable C code for Arduino-based microcontrollers.

3.1.2 Model selection

To make model selection easier, we define a framework called HiML-Edge that streamlines the process. It leverages hierarchical machine learning to construct energy-efficient classification pipelines. By structuring the classification task into a hierarchical taxonomy, decisions are organized into a directed acyclic graph (DAG) or a partially ordered set (poset) $(N, \prec)$, where N represents a finite set of nodes, and $\prec$ denotes a partial order that is asymmetric, anti-reflexive, and transitive.

This hierarchical structure can be simplified into a tree-like format, as illustrated in Fig. 3.1, where each level or node is associated with a distinct classifier, comprising both feature extraction and model inference components. We depict the combination of feature extraction steps and classifier as a Hierarchical Model (HiM).

In contrast, flat classification employs a single, complex multilabel classifier responsible for distinguishing all possible labels simultaneously. As discussed in Sec. 2.5 Carlos N Silla and Alex A Freitas introduces a formalization of hierarchical classification approaches, among which we adopt the LCPN strategy. This approach enables modular construction, where each parent node operates as an independent classifier, offering the flexibility to select different classifiers for different nodes.

The HiMLEdge framework depicted in Fig. 3.2, provides a complete toolchain for generating, training, optimizing, and transpiling HiMs to C-code for deployment on embedded platforms. The framework automates the construction of HiMs using a JSON-like representation, enabling systematic exploration of feature and classifier permutations tailored to a given class taxonomy. Any HiM defined in this format can be trained, evaluated, and optimized within the framework. Training and inference in a HiM follow a recursive DAG-based algorithm that traverses the hierarchy from top to bottom. Each node maintains its classifier and feature

extraction methods, as well as references to subsequent nodes. During inference, a node invokes the prediction function of its child node(s) or terminates if a leaf node is reached.

The framework enables the creation of instances of hierarchical models, which are realizations of possible classifiers and feature extractors, as shown in e.g. Fig 3.3. These possible choices W in conjunction with the overall structure of the hierarchical model W form the input configuration to the framework. Additionally, the framework requires a data set for training, evaluating and testing the generated instances during the optimization process. The user may choose an optimization strategy, which also influences the selection and optimality criteria. In this work we test reinforcement learning and exhaustive search as possible optimization strategies, but additional strategies are possible. After the initial instance generation, the framework continuously chooses new instances based on the optimization strategy until the selection and/or optimality criteria has been reached. It should be noted that not every possible search strategy ensures optimality. While the exhaustive search tests every possible combination, the training process of each classifier might still introduce some uncertainty regarding the optimality of the found solution e.g. by converging to a local minimum. Nonetheless, since all possible combinations of classifiers and feature extractors are tried, the exhaustive search should find near optimal solution from the search space. In contrast, the reinforcement learning does not guarantee an optimal solution, but should still converge to an optimal policy that is adequate, which we will show in Sec. 3.3. After the optimization has converged, the framework can be used to generate deployable C code for an arduino-based micro controller.

For architecture search and parameter tuning, users can select different optimization strategies. The framework supports energy-aware optimization by incorporating real power consumption measurements or approximations when direct measurements are unavailable. Currently, HiMLEdge provides exhaustive search and reinforcement learning-based optimization strategies, as demonstrated in [3].

Further, the framework formalizes the selection process into an optimization problem. Let $\mathcal{C} = \{c_1, \ldots, c_j\}$ and $\mathcal{F} = \{F_1 \ldots F_k\}$ be finite sets of classifier and feature labels, respectively. The architecture search problem is formulated as an optimization problem to maximize the reward function

$$R(\hat{\mathbb{P}}_{C,F}) = \lambda \mathcal{A}(\hat{\mathbb{P}}_{C,F}) + \frac{1}{\lambda}\hat{E}(\hat{\mathbb{P}}_{C,F}), \tag{3.1}$$

where $\hat{\mathbb{P}}_{C,F}$ is a finite set indexed by N that assigns to each node $n \in N$ an optimal classifier-feature pair $(\hat{C}_n, \hat{F}_n)$, combining a classifier $\hat{C}_n \in \mathcal{C}$ with a set of feature labels $\hat{F}_n \in \mathcal{P}(\mathcal{F})$. The reward function balances classification accuracy $\mathcal{A}$ and estimated energy consumption $\hat{E}$, which can be replaced by real measurements. The weighting factor λ determines the trade-off between accuracy and energy ef-

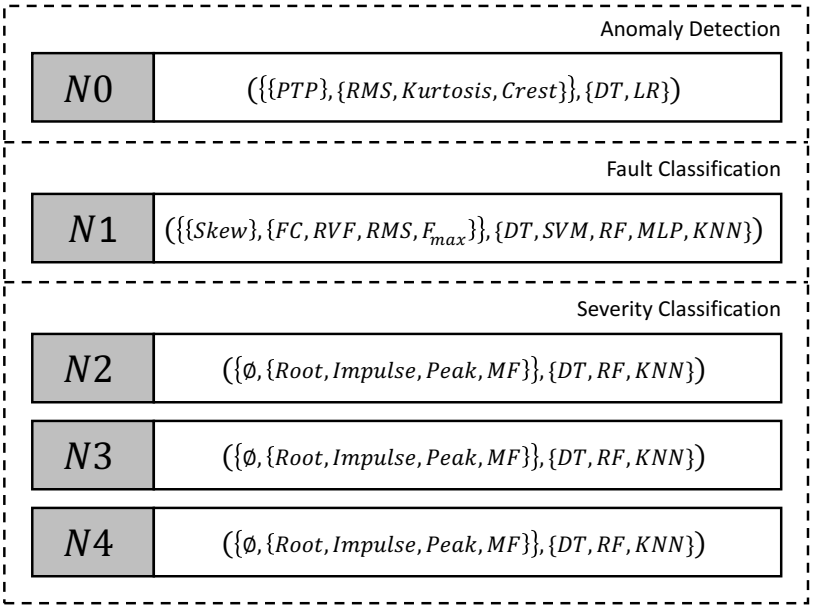


Figure 3.3: Possible classifiers and features for the CWRU bearing problem. Illustration taken from [135] with permission. Values represent the tuple $(\mathcal{F}, \mathcal{C})$ for the respective model in the hierarchy from which the optimization can choose from to find a feature set $\mathbb{f}_n$ and classifier $\mathbb{c}_n$. The abbreviations stand for: DT=Decision Tree, LR=Logistic Regression, SVM=Support Vector Machine, RF=Random Forest, MLP=Mulit-Layer-Perceptron, KNN=k-Nearest Neighbours, PTP=Peak-To-Peak, RMS=Root Mean Square, FC=Frequency Centroid, RVF=Root Variance Frequency, F_{max}=Maximum Frequency, and MF=Mean Frequency.

ficiency. Empirical tests indicate that $\lambda = 2$ provides a balanced trade-off. Lower values result in trivial classifiers, while higher values lead to inefficient models, as extreme λ values prioritize accuracy or energy efficiency exclusively. The optimization process selects configurations

$$\mathbb{W} = \{(\mathbb{c}_0, \mathbb{f}_0), (\mathbb{c}_1, \mathbb{f}_1), \ldots, (\mathbb{c}_N, \mathbb{f}_N)\}, \tag{3.2}$$

where each $\mathbb{c}_n \in \mathcal{P}(\mathcal{C}) \setminus \emptyset$ and $\mathbb{f}_n \subseteq \mathcal{P}(\mathcal{F})$. We should note that we utilize a reward rather than a loss function to make the optimization process more adaptable to the reinforcement learning below.

3.2 Exhaustive Search

To select models and features at every node in the graph of Fig. 3.1, we first want to find a lower bound by exhaustively searching the search space. This approach is, of course, not feasible in most scenarios as the number of possible combinations quickly exceeds a reasonable amount. Therefore, we utilize the knowledge of the given problem by cutting the potential combinations taken into account to a manageable selection. For example, we only allow lightweight classifiers for N0

and refrain from using any frequency-based features to avoid calculating a Fourier transform in every inference (cf. Fig. 3.3). Nonetheless, the search space is ample, forcing us to approximate the energy consumption

$$\hat{E}(\hat{\mathbb{P}}_{C,F}) = \left(1 - \frac{\tau(\hat{\mathbb{P}}_{C,F})}{\tau_{max}}\right), \tag{3.3}$$

using the mean latency per inference τ normalized the maximum recorded latency. Afterward, we transpile the overall best performing HiM into plain C and deploy it on an embedded device. We also transpile and deploy the best scoring HiM with an SVM for N_1 to have a fair comparison to the flat SVM baseline. In addition to the SVM baseline, we also include a Random Forest classifier to provide a lightweight baseline classifier. For the hierarchical model shown in Fig. 3.1, the optimization selects from the sets in Fig. 3.3, using the configuration:

$$\begin{aligned}
&\mathbb{c}_0 = \{\text{DT, LR}\},\\
&\mathbb{f}_0 = \{\{\text{PTP}\}, \{\text{RMS, Kurtosis, Crest}\}\},\\
&\mathbb{c}_1 = \{\text{DT, SVM, RF, MLP, KNN}\},\\
&\mathbb{f}_1 = \{\{\text{Skew}\}, \{\text{FC, RVF, RMS, F}_{\text{max}}\}\},\\
&\mathbb{c}_2 = \mathbb{c}_3 = \mathbb{c}_4 = \{\text{DT, RF, KNN}\},\\
&\mathbb{f}_2 = \mathbb{f}_3 = \mathbb{f}_4 = \{\emptyset, \{\text{Root, Impulse, Peak, MF}\}\},
\end{aligned}$$

leading to a total of 12,964 permutations. It should be noted that the subscripted classifier and feature sets refer to the concrete realizations of the set definitions in (3.2).

The results indicate that both HiMs outperform their respective baseline model in energy efficiency (cf. Fig. 3.4). A significant factor influencing the hierarchical model's performance is fault detection, which accounts for most of the computational workload. This is primarily due to the inclusion of the FFT in feature extraction and the complexity of the classification task. Consequently, further evaluations focus on comparing the hierarchical model with an RF core to the RF baseline and the SVM core to the SVM baseline.

The hierarchical approach for the RF-based models reduces mean energy consumption and latency by 24.96% while improving accuracy by 0.1%. This improvement is likely influenced by the dataset distribution, where 25% of cases represent non-fault conditions. In these cases, a DT using only time-domain features suffices for classification, allowing the system to terminate computation early. The hierarchi-

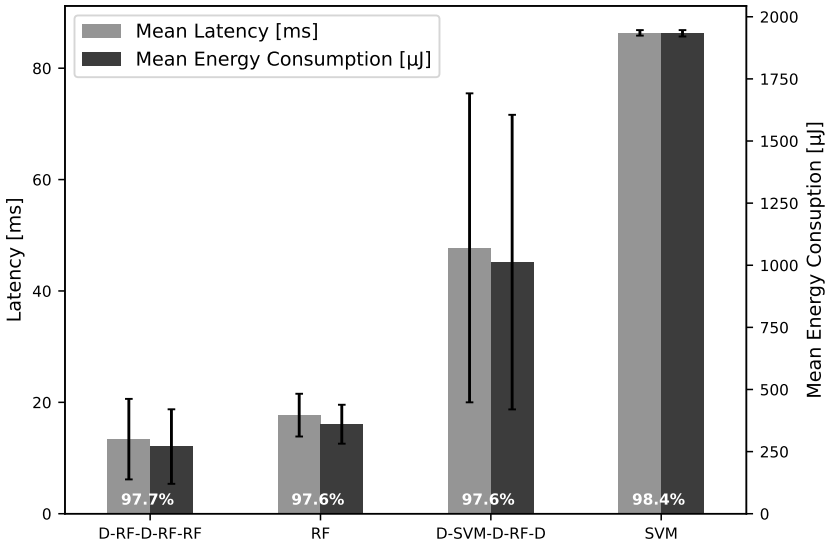


Figure 3.4: Results from the exhaustive taxonomy search. The abbreviations translate to D=Decision Tree Classifier, RF=Random Forest Classifier, SVM=Support Vector Machine Classifier, and refer to the selected classifier taxonomies from the search. Illustration taken from [135] with permission.

cal structure introduces additional overhead, which is mitigated during severity detection, where certain fault classes require no extra feature extraction.

For the SVM-based models, the hierarchical approach achieves an even more significant reduction in energy consumption (47.63%), though with a slight drop in accuracy to 97.6% (compared to 98.4% for the SVM baseline). The improved efficiency can be attributed to the computational complexity of SVMs, which in the worst case scales as $O(n^3)$ [107]. With increased input features and potential classes, this cubic complexity becomes a bottleneck. The slight decrease in accuracy is likely due to error propagation within the hierarchy—misclassifications at any node propagate through subsequent layers, affecting the final prediction.

3.3 Reinforcement Learning

As previously shown by Adams, Stephen and Meekins, Ryan and Beling, Peter A. and Farinholt, Kevin and Brown, Nathan and Polter, Sherwood and Dong, Qing in [3], Reinforcement Learning (RL) can be an effective tool to select classifiers for HiML. RL is a machine learning approach that learns an optimal sequence of actions, referred to as a *policy*, to achieve a specific goal. It consists of an *environment* and an *agent* that transitions between *states*. The agent interacts with the environment by selecting actions and receives feedback in the form of a *reward*, which it aims to maximize over time.

We extend the RL framework from [3] by testing it on an embedded system with real-world energy measurements. Here, the states correspond to different levels

Algorithm 1: Reinforcement Learning Algorithm for Hierarchical Classification

Input : $\epsilon,\ \alpha,\ \gamma,\ \lambda,\ X,\ Y$
Output: Optimal policy $\pi*$

```
1  Initialize Q-table
2  for episode = 0 ... Episodes do
3      for i = 0 ... I do
4          Select X_k and Y_k for hierarchy level k
5          Select testing instance x_i, y_i
6          Construct training set: X_k \ x_i, Y_k \ y_i
7          Select an action a ∈ A
8          Train classifier of type a using training set
9          Predict label for x_i using trained classifier
10         Move to next state s' based on the prediction
11         Update Q (s,a)
12         if s' is a leaf node then
13             Break;
14         else
15             Return to line 3;
16         end
17         for s ∈ S do
18             π*(s) = arg max_a Q (s,a)
19         end
20     end
21 end
```

in the hierarchy, and the actions represent classifier selections. In each episode, the agent selects a classifier, trains it, and performs inference on a random test instance. The agent transitions to the next state or collects a reward based on the predicted output. The episode concludes when the agent reaches a terminal state, which, in our case, is represented by a leaf node.

The reward function is defined similarly to (3.1), with $\hat{E}(\hat{\mathbb{P}}_{C,F})$ replaced by a direct measurement

$$R(\hat{\mathbb{P}}_{C,F}) = \lambda \mathcal{A}(\hat{\mathbb{P}}_{C,F}) - (1-\lambda)\hat{E}(\hat{\mathbb{P}}_{C,F}). \tag{3.4}$$

Here, λ is a weighting factor between 0 and 1 to maintain comparability with [3]. The RL training process follows a Monte Carlo on-policy strategy, where the Q-table is updated after each episode, as detailed in Algorithm 1.

In this setup, X and Y represent data samples and their corresponding labels. The parameter ϵ regulates exploration vs. exploitation in RL, α is the learning rate, and γ is the discount factor.

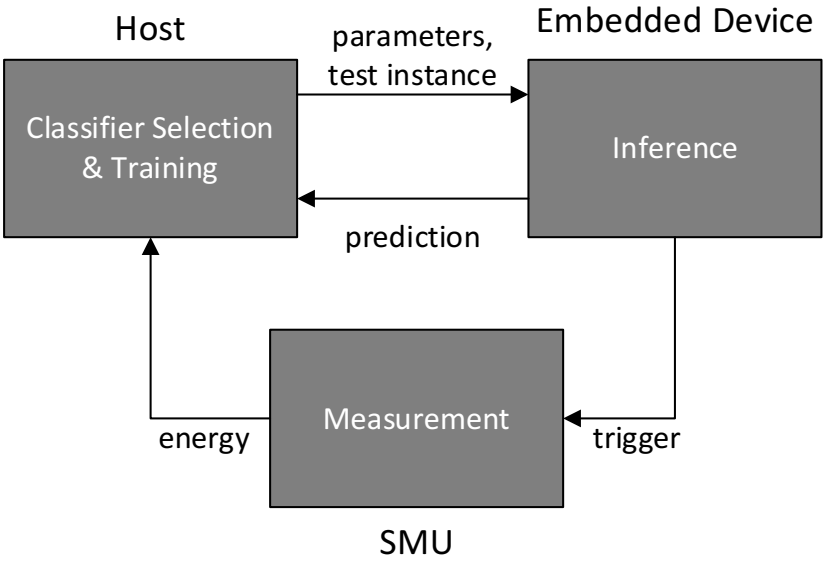


Figure 3.5: Measurement setup for Reinforcement Learning.

We implement an interface for communication between the host system and the embedded device to integrate real-time energy measurements, as shown in Fig. 3.5. The host selects and trains the classifier before sending its parameters and a test instance to the embedded device. The embedded system then performs inference using the received parameters while simultaneously measuring energy consumption. Once inference is complete, the embedded device returns the prediction, and the system retrieves the energy consumption from a source measurement unit (SMU). This feedback updates the Q-table and initiates a new episode with another test instance. Over multiple episodes, the agent learns an optimal policy that defines the best classifier type for each hierarchy level.

In [3], the authors train an RL-agent that decides which classifier to put in a node in every step. Afterward, it collects a reward when it reaches a leaf node in the hierarchy based on the prediction result of the chosen classifier and their respective energy consumption. To evaluate the effectiveness of the RL-based selection approach, we constrain the search space to allow for direct comparison with optimizing only the classifier selection. For all nodes $n \in \{0, 1, \ldots, 4\}$, the agent selects classifiers from the set $\mathbb{c}_n = \{\text{DT, RF, MLP, SVM}\}$. Logistic regression and KNN classifiers are excluded from this experiment, as they performed inadequately during the exhaustive search. Additionally, RF models are restricted to a maximum of ten estimators due to increased memory constraints from background runtime.

With real-time measurement feedback, the experiment yields the optimal policy {DT, MLP, RF, MLP, RF}, differing from the policy found using Adams, Stephen and Meekins, Ryan and Beling, Peter A. and Farinholt, Kevin and Brown, Nathan and Polter, Sherwood and Dong, Qing approximation {DT, RF, DT, DT, DT}. Testing reveals that although an MLP exhibits a higher complexity measure than an RF, energy measurements indicate only a marginal increase in consumption. Given

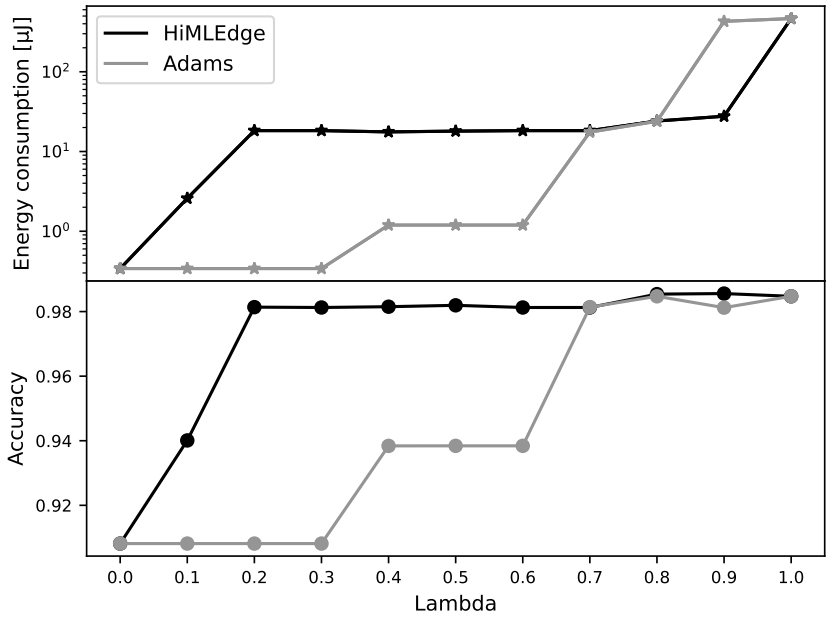


Figure 3.6: Influence of λ on energy consumption and accuracy

the MLP's higher accuracy, the agent appears to prioritize accuracy over the minimal energy gains of RF. Furthermore, training converges within a few hours on a single desktop PC, whereas the exhaustive search requires multiple days. This highlights the significance of efficient algorithm selection techniques, which reduce energy consumption during inference and minimize power usage during optimization and training. Without a suitable approach, algorithm selection becomes infeasible for hierarchical classification as search spaces expand, undermining its potential for sustainability.

The model selection process is highly sensitive to the trade-off parameter λ, which controls the balance between energy efficiency ($\lambda = 0$) and accuracy maximization ($\lambda = 1$). As expected, both Adams, Stephen and Meekins, Ryan and Beling, Peter A. and Farinholt, Kevin and Brown, Nathan and Polter, Sherwood and Dong, Qing's surrogate metric and our approach demonstrate increasing energy consumption and accuracy as λ rises (cf. Fig. 3.6). However, our method reaches near-peak accuracy at a significantly lower λ value ($\lambda = 0.2$), while maintaining a modest energy consumption of 18.28 µJ/inference. This trend is also evident in classifier selection, as shown in Fig. 3.7. The optimal policies for fault detection (N1) correlate strongly with energy consumption, reinforcing the observation from the exhaustive search that fault classification is the most influential factor in the overall HiM energy footprint.

Overall, our approach tends to favor MLP classifiers outside of extreme λ values, whereas the agent using a surrogate metric primarily selects tree-based models (RF, DT). This discrepancy likely results from an imperfect energy approximation for MLPs. While MLPs theoretically require higher computational resources than RF, their matrix operations are well-optimized by modern compilers, leading to

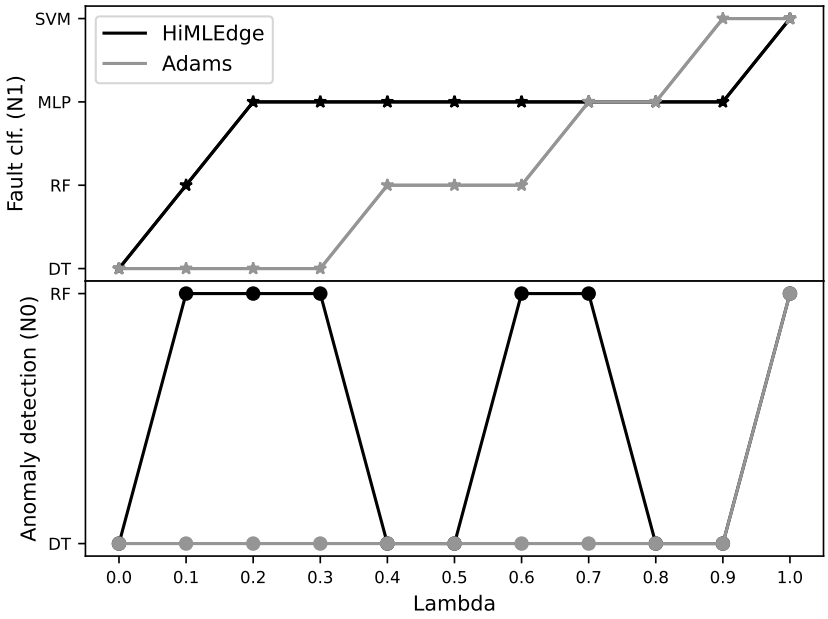


Figure 3.7: Influence of λ on energy model selection.

only a modest increase in energy consumption while significantly improving accuracy. Additionally, the Cortex M4 processor benefits from DSP functionalities that optimize matrix execution. Consequently, search processes should incorporate not only theoretical, computational complexity but also real-world measurements or hardware-aware surrogate models.

The RL-based tests further emphasize dataset dependency and the influence of λ on classifier selection. The exhaustive search requires several days on a CPU cluster, whereas the RL agent completed training within hours on a single machine, including measurement feedback. Using Adams, Stephen and Meekins, Ryan and Beling, Peter A. and Farinholt, Kevin and Brown, Nathan and Polter, Sherwood and Dong, Qing's surrogate metric, further reduces training time. A combination of heuristics, an appropriate search strategy, and hardware-aware surrogate models is essential for developing a practical algorithm selection framework for HiML.

3.4 Realistic system-wide energy requirements

Due to the nature of hierarchical classification, energy consumption strongly depends on the dataset. This is particularly relevant when the hierarchy includes a lightweight anomaly detection stage, as in our case. The ratio between faulty and non-faulty events

$$\alpha = \frac{\#\text{Fault Events}}{\#\text{Normal Events}} = \frac{\alpha_f}{\alpha_n},$$

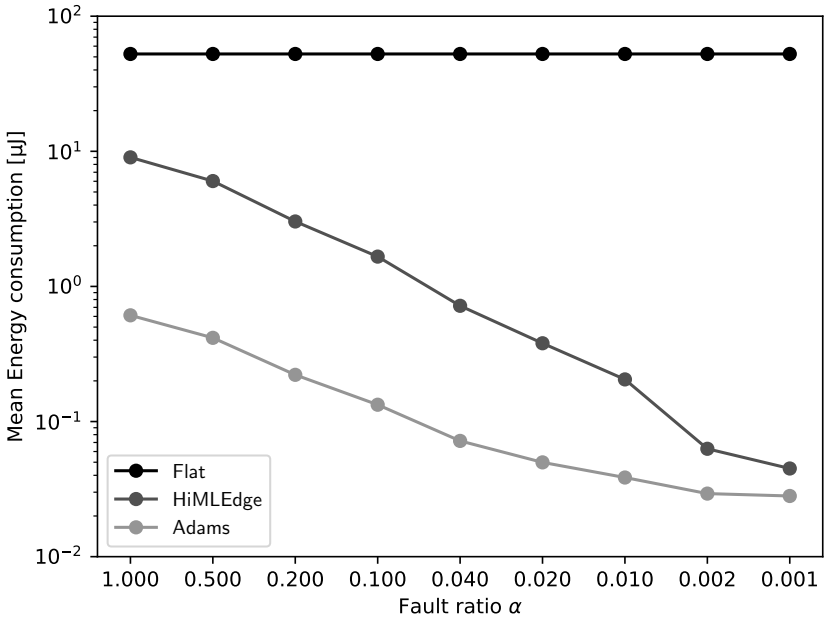


Figure 3.8: Mean energy consumptions for different fault ratios α. Illustration taken from [135] with permission. Our approach converges to the same energy consumption as previous approaches with rising α

plays a crucial role in determining energy requirements. As discussed earlier, with $\lambda = 0.5$, our approach requires more energy than Adams, Stephen and Meekins, Ryan and Beling, Peter A. and Farinholt, Kevin and Brown, Nathan and Polter, Sherwood and Dong, Qing's method but achieves higher classification performance. While this holds for the standard dataset distribution ($\alpha \approx 4$), real-world conditions often exhibit a much smaller α. In such cases, both approaches converge to the same mean energy consumption

$$\overline{E} = \frac{\alpha_n E_{N0} + \alpha_f \overline{E_f}}{\alpha_n + \alpha_f} = \frac{\alpha_n E_{N0} + \alpha\alpha_n \overline{E_f}}{\alpha_n + \alpha\alpha_n} = \frac{E_{N0} + \alpha \overline{E_f}}{1 + \alpha} \underset{\alpha \to 0}{=} E_{N0},$$

assuming that the energy required for anomaly detection E_{N0} remains identical across both approaches (i.e., the same policy for $N0$, cf. Fig. 3.7). As α approaches zero, the mean energy consumption of subsequent nodes ($N1 - N4$) $\overline{E_f}$ becomes negligible, which aligns with observations in Fig. 3.8.

These results highlight that the energy efficiency of HiML is highly dataset-dependent. A hierarchical approach is advantageous when a classification problem can be decomposed into multiple stages of increasing complexity, and low-complexity nodes are executed more frequently. However, in scenarios where complex computations are consistently required, a HiM may introduce additional overhead, leading to higher energy consumption.

We consider a system-level perspective to analyze power consumption in an application scenario as a final evaluation. Previous experiments focused solely on

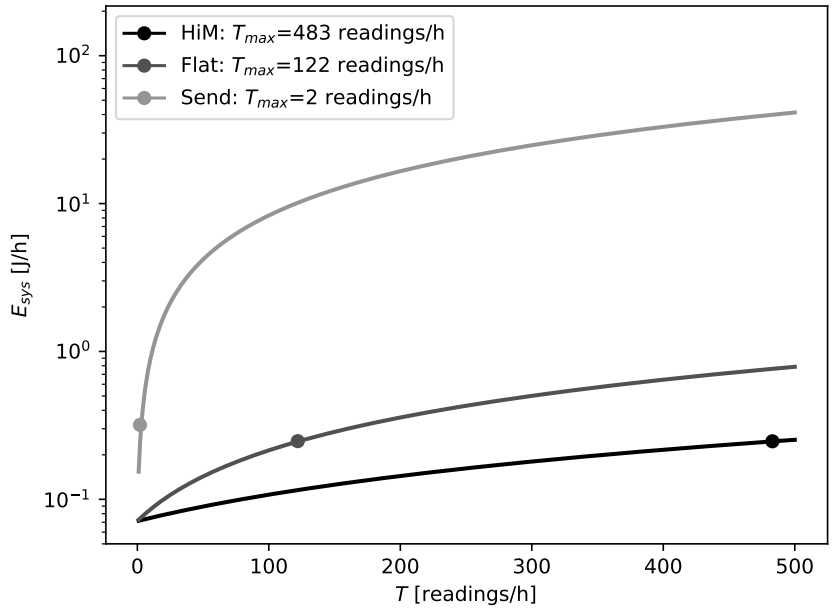


Figure 3.9: System-level energy consumption for hierarchical systems. Illustration taken from [135] with permission. With a HiM, the system can achieve nearly 500 readings per hour with the same battery life as the baseline approaches (122 readings per hour for the flat classifier and 2 readings per hour for sending the whole data).

optimizing the energy consumption of the machine learning pipeline, independent of other system components and communication overhead. Here, we introduce a system-level energy score per hour

$$E_{sys} = T[\alpha\overline{E_f} + (1-\alpha)E_{N0}] + E_{off}, \tag{3.5}$$

where T represents the number of readings per hour, and E_{off} denotes the system-off power consumption per hour. To simulate a realistic deployment scenario, we modify the dataset distribution to $\alpha_f = 1$ and $\alpha_n = 20$, meaning faults occur in only 5% of measurements. Using this setup, we compare three approaches: A flat classifier (RF), the best-performing HiM from Sec. 3.2, and a baseline scenario where all data is transmitted without local classification. In both classification cases, results are sent via BLE only when a fault is detected, whereas, in the data transmission case, each window of 512 values is sent to the host regardless of classification outcome. The power-off energy is derived from the NRF58240 datasheet (71.28mJ/h). The objective of this test is to determine the maximum achievable duty cycle T that ensures a five-year battery lifespan using a single CR2477 coin cell battery (1000mAh, 10800J @ 3V) for each method, thereby assessing the practical energy benefits of hierarchical classification.

The results in Fig. 3.9 clearly indicate that the optimized HiM achieves the highest duty cycle, enabling 3.95 times more classifications per hour than the flat model.

This significantly improves the feasibility of near real-time machine monitoring. Furthermore, the energy gap between classification-based approaches and direct data transmission underscores the importance of local data processing whenever possible. In more extreme cases where BLE is unavailable due to its limited range, alternative communication protocols may consume even more energy, reinforcing the necessity of efficient on-device filtering methods.

From a battery life perspective, these efficiency gains can either extend operational lifespan or reduce battery capacity requirements instead of increasing the duty cycle. Such optimizations contribute to sustainability in industrial IoT applications, aligning with Industry 4.0 objectives and enabling new deployment scenarios.

3.5 Conclusion

This chapter has explored hierarchical machine learning as a meta-approach to structuring machine learning models for energy-efficient classification in resource-constrained environments. Traditional flat classifiers apply the same computational effort to all inputs, even when an early classification is possible. In contrast, HiML organizes the classification process into a hierarchy of decisions, allowing the system to make early exits when confidence is high. This structure reduces unnecessary computations and optimizes resource usage.

One of the main challenges in HiML is designing an optimal taxonomy of classifiers that balances energy efficiency and classification accuracy. This chapter has first examined an exhaustive search approach to find optimal hierarchies. While this method provides the best possible efficiency gains, it requires extensive computational resources, making it impractical for real-world applications. To overcome this limitation, we have introduced reinforcement learning (RL) as an adaptive strategy that automatically optimizes the hierarchical structure. The results show that RL quickly finds efficient taxonomies, reducing the optimization time from several days to a few hours while maintaining high classification performance.

The experimental results demonstrate energy savings of up to 47.63% compared to conventional flat classifiers, while still achieving competitive accuracy. These findings highlight HiML as a promising solution for edge computing applications, where power efficiency has been a critical requirement.

We have also shown that the effectiveness of HiML has depends on the dataset distribution. When a dataset follows a hierarchy, early classification exits can occur more frequently, maximizing energy savings. However, if most inputs require deeper stages of the hierarchy, the added complexity of hierarchical processing reduces the efficiency.

Beyond classification efficiency, this chapter has also evaluated system-wide energy consumption, considering real-world deployment constraints. While HiML shows

reduced computational load at the model level, overall energy efficiency depends on additional factors such as sensor power consumption, data transmission costs, and application-specific requirements. Nonetheless, the evaluations including system level energy consumption show that the battery life of a system running HiML significantly increases and the savings are not negligible in comparison to the overall energy draw.

CHAPTER 4

Physics and Signal Processing for Computational Spectrometers

To combine the results from the previous chapters with a real-world application, this thesis continues by developing a small-scale optical sensing system with embedded machine learning, providing an excellent example of TinyML in a smart sensor system. As a basis for this system, we utilize multiple iterations of an optical sensor developed in our group. The optical sensors group at Fraunhofer IIS has been researching optical nanostructures for over ten years [76]. With holes and islands introduced into the upper metal layers of the Complementary Metal-Oxide-Semiconductor (CMOS) process, it is possible to filter the light in a way that each photodiode beneath these structures is only sensitive to a specific portion of the light spectrum. Ordered in an array and with enough variation, the combination of photodiodes and filters forms an integrated sensing device that can measure the spectrum of light in range based on the included filters. Using a spectral reconstruction algorithm, the sensor output can then be transformed into a humanly interpretable discrete spectrum. The following section will provide additional details about the technology used in filter-array-based spectral sensing devices, including various light filtering techniques and the challenges associated with spectral reconstruction.

4.1 Light Filtering Techniques

As discussed earlier, spectral sensing is the key to various application scenarios that benefit from an extended range of optical resolution. While traditional spectrometers offer the capability to measure the spectrum of light very precisely by relying on specialized gratings or prisms, computational spectrometers encode the spectral information with various techniques to reduce sensor cost [142]. In the past, a magnitude of approaches have been proposed to encode spectral information in a form that a reconstruction algorithm can combine the encoded information to estimate the original spectrum of light. Fig. 4.1 gives a rough overview of the history of computational miniature spectrometers, with the first approach dating back to 1984 [69]. It is clearly visible that an in-depth overview of sensing techniques is out of scope for this thesis. However, this section briefly highlights the general working principle behind computational spectrometers and the light-filtering technique for our sensing devices.

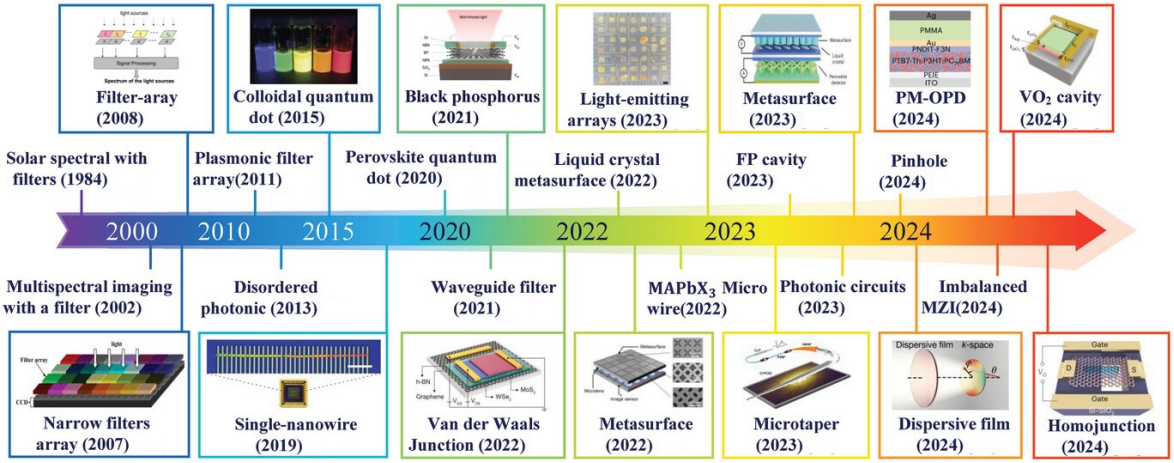


Figure 4.1: Timeline of small-scale computational spectrometers. Figure taken from [142] with permission. The illustration aims to provide an overview of the broad field of miniature spectrometers without the need for completeness or excessive detail.

In general, a computational spectrometer utilizes a three-stage processing pipeline to estimate a spectrum: pre-calibration, spectral information detection, and spectral reconstruction [142]. The pre-calibration step describes the initial evaluation of a sensor to characterize the encoder's spectral response. In a typical machine-learning approach this step is reflected by the training phase, while classic reconstruction methods capture a characterization matrix. The last two stages are also often called encoding and decoding [53]. The encoder utilizes a light-interaction technique to physically encode the spectral information into a representation that the decoder can then process into a spectral estimate. The encoder should generally be easier to miniaturize and/or be more cost-effective to offer an advantage over traditional spectrometers. The decoder interprets the encoded signals with a spectral reconstruction algorithm, usually realized in software. This last step is often transferable to different encoding techniques and represents this thesis's core. Designing the encoder and decoder in tandem is essential for achieving accurate estimation results, making computational spectrometers an excellent example of hardware-software co-design.

The sensors used in this thesis use a plasmonic filter array as the encoder, which have emerged as a powerful tool for miniaturizing optical components such as spectrometers [142]. By leveraging the interaction of light with free electrons at the surface of metallic nanostructures, these structures lead to the excitation of surface plasmon resonances, which can be tailored to manipulate light at subwavelength scales. The ability of plasmonic structures to confine and enhance electromagnetic fields enables highly compact and efficient spectral filtering mechanisms, making them particularly suitable for computational spectrometers integrated in a CMOS sensor.

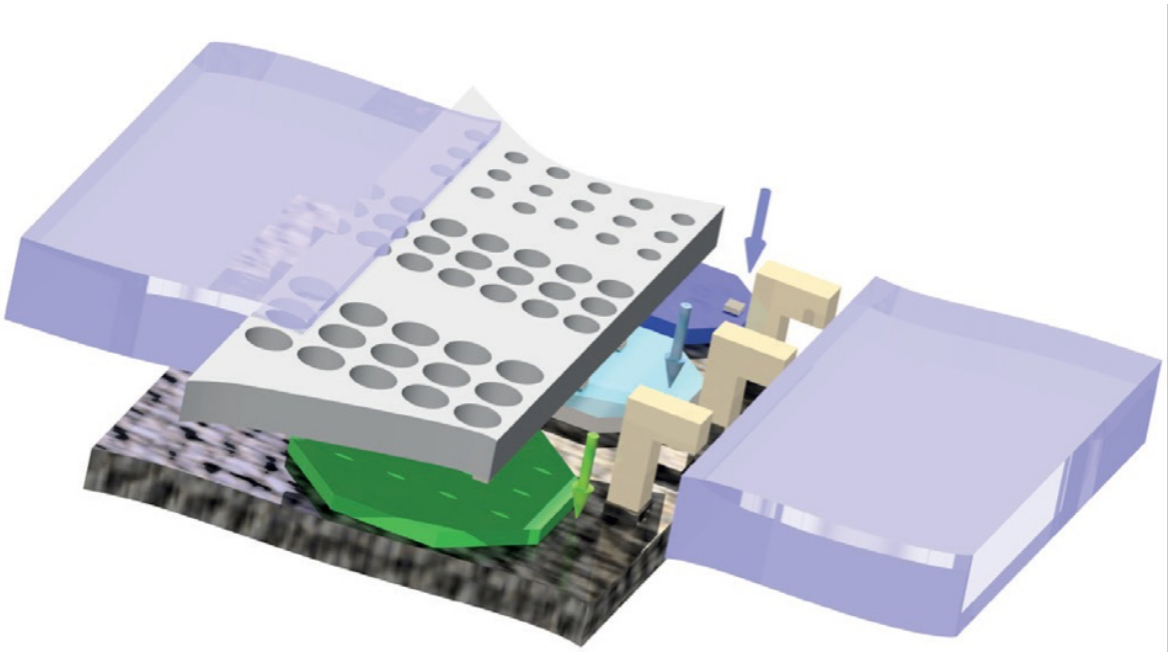
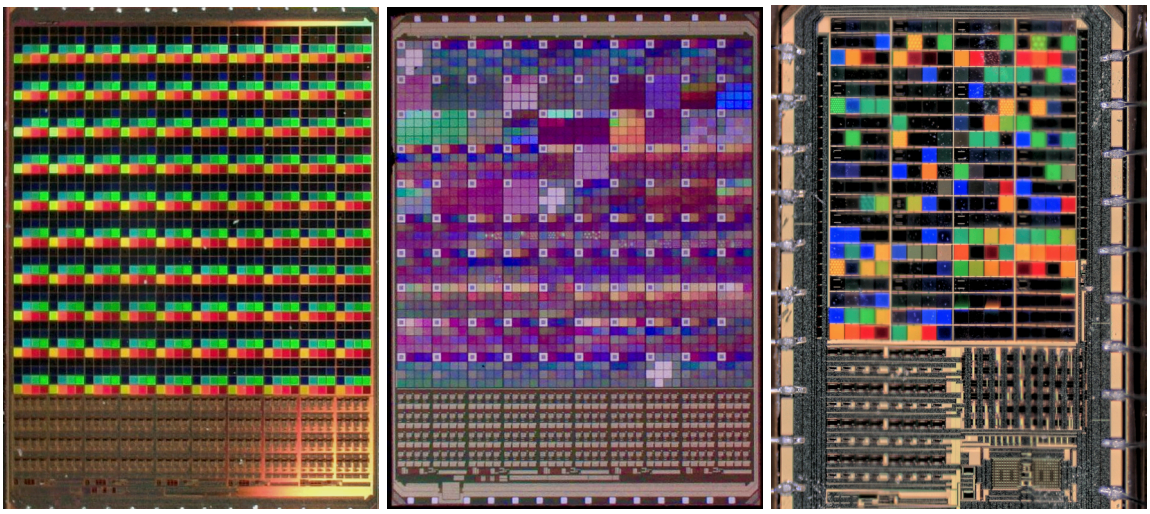


Figure 4.2: Illustration of optical nanostructures used for the CSS with photodiodes underneath. The plasmonic filter array interacts with the incoming light, so the photodiodes are only sensitive to a certain part of the light spectrum. © 2024 Fraunhofer IIS.

In plasmonic spectrometers, metallic nanostructures, such as nanohole arrays, nano-island filters, or nanoparticle-in-cavity designs, function as wavelength-selective elements [98]; [148]. When incident light interacts with these structures, specific wavelengths couple to the localized or propagating plasmonic modes, leading to strong absorption or transmission peaks. By designing a plasmonic filter array with a range of different spectral responses, it becomes possible to encode spectral information into a spatially varying intensity pattern, which can then be computationally reconstructed a spectral reconstruction algorithm [25].

Sensing devices with plasmonic filters often differ in the manufacturing of such an array. In [148] the authors integrate a plasmonic nanoparticles-in-cavity microfilter array onto a CMOS sensor, enabling high-resolution spectral measurements without the need for bulky diffraction gratings or prisms. The design in [66] utilizes a broadband gradient plasmonic nano-island filter, where the spectral response varies across the filter, allowing for angle-insensitive spectral encoding.

Further research has explored plasmonic filters in THz spectrometry, demonstrating improved sensitivity compared to traditional benchtop THz spectrometers. In this design, a plasmonic filter array modulates the response of a blocked-impurity-band detector, while an adaptive deep-learning algorithm reconstructs the spectral information [98]. Similarly, the authors in [25] have developed neural network-based reconstruction frameworks for plasmonic encoder chips fabricated using scalable imprint lithography, enabling rapid spectral inference with high accuracy.



(a) (b) (c)

Figure 4.3: Chip photos of the three sensor generations: (a) First generation (Infimedar), (b) second generation (Medusa), (c) third generation (NanoSpectral). © 2024 Fraunhofer IIS.

It is evident that various devices already use plasmonic filter arrays in combination with reconstruction algorithms to form a computational spectrometer. However, all the above-shown devices need a two-step manufacturing process that first processes a CMOS light sensor, followed by the plasmonic array in a different specialized process. Our sensing devices skip the second step with a novel filter design that can be realized directly in CMOS (cf. Fig. 4.2). We utilize the upper metal layers of the CMOS process to form islands and holes that also induce the plasmonic effect [76]. In this way, we can skip the second manufacturing step, driving costs further down and enabling additional miniaturization. Nonetheless, this cost reduction comes with the price of more complex filter functions that are harder to interpret. Therefore, we must emphasize the spectral reconstruction algorithm to facilitate a usable spectrometer, which is the focus of this thesis.

4.2 Sensing devices

Starting from February 2020, three generations of CSS have been developed in our group, contributing to this thesis. To avoid confusion, we will refer to the first generation as *Infimedar*, the second as *Medusa*, and the third as *NanoSpectral*. Figure 4.3 shows real chip photos taken under a microscope of these sensors. The Infimedar sensor was the first to integrate many optical channels, namely 1600, on a singular die, stemming from the name-giving research project Infimedar [75]. The optical nanostructures used for this sensor emerged from years of previous research [77], making plasmonic filters realizable in CMOS. The Infimedar project focused on smart farming and spectral sensing, with applications in crop classifi-

cation, nutritional content analysis, and plant health assessment. In addition to achieving spectral resolution beyond RGB, the project's primary sensor featured spatial resolution by repeating 16 hand-picked filter structures in both the x- and y-directions. In contrast, the first CSS was a by-product of this project by sacrificing the spatial resolution for a higher spectral one with 1600 distinct nanostructures. However, no high-resolution chip photos of the first CSS exist, so the Infimedar sensor shown in Fig. 4.3 shows the spatial sensor. This can be easily seen by the same color pattern repeated throughout the die.

The Medusa sensor (second generation) was developed to prepare the sensing devices for series production. The Infimedar sensor was a proof-of-concept showing that a high number of spectral channels can lead to a complete spectral sensor but was not manufacturable in large quantities. Therefore, in the follow-up internal project called Medusa, the technology was transferred to a different process, paving the way for a commercial product. Unfortunately, not all previously selected nanostructures were realizable with this new process, forcing us to replace some well-functioning structures and slightly degrading the potential performance. Besides these structures, the remaining filters were untouched for the second-generation sensor.

As before, the Medusa project yielded two versions: One sensor with less filter variation but with spatial resolution and one without spatial resolution but with 1600 different spectral channels. This changed for the most recent generation of CSS, NanoSpectral. With this version, we tailored the sensor specifically for precise spectral point measurements without the need for spatial resolution. To achieve this goal, we analyzed the grade and contribution to the output spectrum of all filter structures realized with the Medusa chip. Afterward, we selected only the most essential filters to be placed on the NanoSpectral chip. The exact selection process is highlighted in Sec. 6.1. Additionally, NanoSpectral includes a digital interface that simplifies sensor reading, enhancing its accessibility as an Application Specific Standard Product (ASSP).

4.3 Terminology and Definitions

As this thesis connects optics with machine learning and optimization, some terminology might not be commonly known to a reader unfamiliar with spectral sensing. This section covers the most widely used terminology and principles connected to our sensing devices as a short listing:

Metapixel: A grouping of 4 by 4 channels that form one pixel in the spatial variants of Medusa and Infimedar. During the sensor design, the filter placement is done on a per-metapixel basis, making the metapixels also visible on the CSS versions. The metapixels are best seen in Fig. 4.3 (b), where the different light patterns make the grouping quite easily distinguishable. Apart from the placement, the readout

circuit also reads the values on a per-metapixel basis, making the grouping still crucial for the non-spatial variants used in this thesis.

Reference diode: Upper left channel in each metapixel. The reference diode can measure the light intensity throughout the die. This channel has no filter structure, so the signal changes only based on light intensity and has no wavelength dependency. However, it is partially blocked to avoid saturation when other channels operate normally. The blocking structure has a transmittance of 25%.

Orientation Metapixel: Special metapixels that are used to infer the correct chip orientation. The Medusa sensor is a research-focused chip, so we must find the chip orientation in the measurements during the initial evaluation. For this purpose, we include three asymmetrical structures placed on the diagonal of the chip that are not wavelength-dependent. As NanoSpectral is targeted as a commercial product, these metapixels do not exist to save chip area.

Saturation: Maximum value a photodiode can read, marking the point where no additional light can be measured. For all versions, the saturation point is 3600 counts.

Integration time: The duration for which the sensor collects data, similar to a camera's exposure time. The sensor measures the photons that hit each channel throughout the acquisition phase. Thus, the integration time refers to the interval during which the sensor actively counts.

Polarization filter: Filter structure that does not filter the incoming light based on wavelength but on the polarization angle. This means it can measure the angle of light hitting the sensor. As the angle is irrelevant for measuring the spectrum, the channels with polarization filters can be excluded from the evaluation. For Infimedar and Medusa, the last row is made of polarization filters, while the final two metapixels in NanoSpectral hold polarization information.

Offset: Number of counts that the readout electronics measure without light being present. Due to currents flowing even without a light source, each channel does not read zero counts at the start of the integration time. For our sensors, the dark value lies around 2000 counts, but it varies for each channel and environmental conditions. Therefore, we need to capture a dark measurement that is subtracted from the actual measurement before the start of a measurement series. This leads to a signal swing of around 1600 counts after the offset correction.

Array size: Number of channels the sensing devices hold. For Infimedar and Medusa, the number of channels is 1600, sorted in a 40 by 40 array. NanoSpectral has only 320 channels, leading to an array size of 16 by 20. It should be noted that the number of channels is not equal to the number of spectrally active channels, as the number of channels also includes reference diodes and polarization filters. However, for simplicity and to maintain the data shape, nearly all models shown in this thesis include all channels in the evaluation. The only exceptions are classic

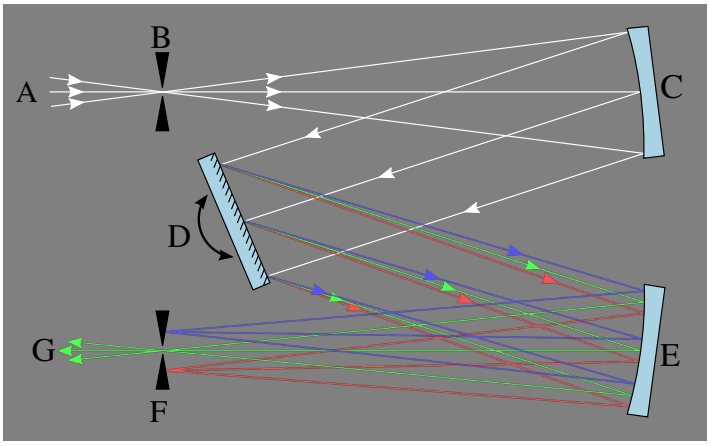


Figure 4.4: Mirror arrangement of a Czerny-Turner monochromator. Illustration taken from [114] with permission.

baseline models that do not rely on a 2D array representation. In these cases, we exclude the reference diodes after flattening the array.

Monochromator: Device that splits the polychromatic light from a light source into monochromatic light consisting of an adjustable singular wavelength. In the classic Czerny-Turner monochromator [36] this is done by an arrangement of mirrors (cf. Fig. 4.4). The polychromatic light (A) enters the device via the slit B and is reflected by the collimator C onto an diffracting grating or prisma D that splits the light into its different spectral components. Afterward, the mirror E refocuses the light back onto slit F, from where the resulting monochromatic light G can be collected. Depending on the angle of D, only certain wavelengths can pass F due to each color arriving at separate points around the slit.

4.4 Classic Reconstruction

As introduced before, the CSS consists of an array of filters representing multiple spectral sampling points. This leads to the assumption that the information in the sensor output is enough to represent the spectrum of light that reaches the die. However, the real discrete spectrum needs to be reconstructed from the output to make the chip usable in various applications. As the idea of measuring a spectrum using an array is well-known, multiple approaches already exist in the literature for spectral reconstruction. Most of these are built upon the assumption of a linear relationship

$$\begin{pmatrix} r_1 \\ \vdots \\ r_N \end{pmatrix} = \begin{pmatrix} H_1(\lambda_1) & \cdots & H_1(\lambda_M) \\ \vdots & & \vdots \\ H_N(\lambda_1) & \cdots & H_N(\lambda_M) \end{pmatrix} \begin{pmatrix} s(\lambda_1) \\ \vdots \\ s_(\lambda_M) \end{pmatrix}, \tag{4.1}$$

where $\mathbf{r} \in \mathbb{R}^N$ denotes the sensor response, $\mathbf{s} \in \mathbb{R}^M$ the sought-after discrete spectrum, and $\mathbf{H} \in \mathbb{R}^{N \times M}$ the filter characteristic for N spectral channels and M discrete wavelengths [32]. Normally, the filter characteristics $\mathbf{H}$ can be measured directly during the initial evaluation of the physical sensor or extracted from simulations during the filter design. In our case, we measure H to eliminate errors introduced by divergences from manufacturing using a monochromator as described in Sec. 4.3. Using this setup, we evaluate each channel n with a wavelength sweep, measuring its sensitivity dependent on the wavelength λ_m and thereby forming $\mathbf{H} \in \mathbb{R}^{N \times M}$. Other authors may use a similar setup so that $\mathbf{H}$ can be seen as a-priori information.

With this assumption, the authors in [32] form the Moore–Penrose Pseudoinverse [31] using Non-Negative Least Squares (NNLS)

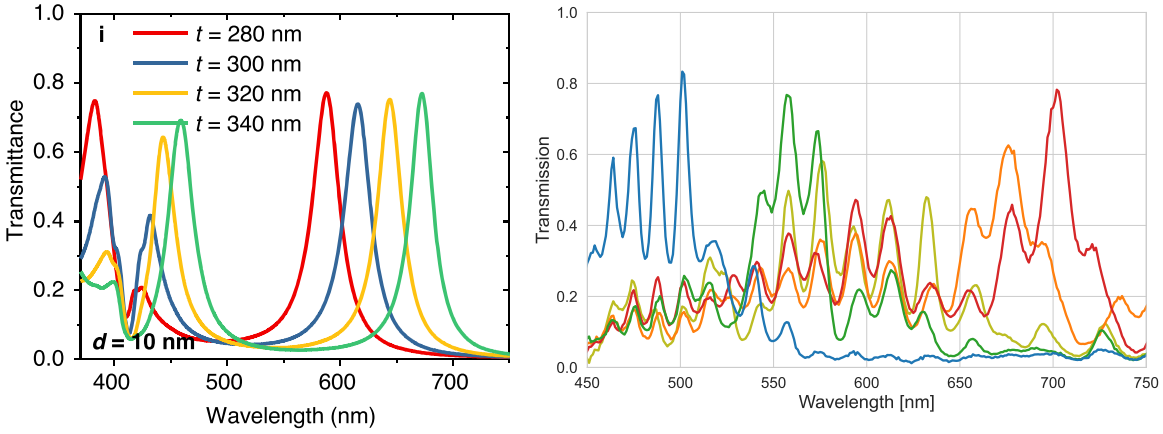
$$\min_s \|\mathbf{H}\mathbf{s} - \mathbf{r}\|_2^2, \text{ s.t. } \mathbf{s} \in \mathbb{R}_{>0} \tag{4.2}$$

to estimate the input spectrum $\hat{\mathbf{s}}$. It should be noted that Cheng-Chun Chang and Heung-No Lee assumes

$$H_n(\lambda_m) = f_n(\lambda_m) d(\lambda_m) \tag{4.3}$$

to be the product of the actual filter function f_n and the diode sensitivity d. The authors do not disclose whether they measure the filter function or use simulated values from the sensor design. However, with the measurement setup described above, we can measure $\mathbf{H}$ directly so that the diode sensitivity is already included in the measurement in our system. However, when utilizing simulated values, one needs to include d in the equation, which should be equal for each channel n.

While this simple approach might work with trivial filter characteristics, our characteristic is far more complex than a traditional miniature spectrometer. Ideally, the light-filtering structures should yield well-defined, narrow filters that assign a confined wavelength band to each channel. While this is true for other manufacturing techniques, the plasmonic filters realized in CMOS for our sensors produce far more complicated filtering characteristics (cf. Fig. 4.5). Instead of Gaussian-like curves, our sensors have non-confined characteristics that often show sensitivity in various spectral regions. Additionally, we notice an interference pattern on top of the actual filters, which might be caused by the different layers in the CMOS process. When etching multiple layers, each consecutive layer might not be 100% aligned with the previous one, causing uneven edges. This roughness can then lead to interferences that we measure in addition to our filter function. In addition to the complex filter characteristics, we expect additional perturbations caused by manufacturing deviations on a per-sensor basis and light angle dependencies in relation to the filter array. All these reasons, paired with the ill-posed nature of the



(a) (b)

Figure 4.5: Exemplary filter characteristics from the sensing device of [148] (a) and from the Medusa Sensor (b). With permission, the illustration in (a) has been taken from [148].

underlying problem due to the high dimensionality of the filter array, make the classic approach shown above unusable.

While previous research did not have these problems to an extent, we see with our sensors, certain aspects like the ill-posed nature of the inverse or manufacturing deviations have already been problematic before. Therefore, multiple approaches already exist to either stabilize the reconstruction or circumvent the matrix inverse. In [85], the authors apply Tikhonov regularization, turning the least square problem from (4.2) into the damped form

$$\min_s \|\mathbf{Hs} - \mathbf{r}\|_2^2 + \alpha^2 \|\mathbf{Ls}\|_2^2, \tag{4.4}$$

where α is the regularization parameter and $\mathbf{L}$ the Tikhonov Matrix. The latter is often set to the identity matrix $\mathbf{I}$ (zeroth-order), but can also equal

$$\mathbf{L} = \begin{bmatrix} 1 & -1 & & \\ & 1 & -1 & \\ & & \ddots & \\ & & & 1 \end{bmatrix}, \mathbf{L} = \begin{bmatrix} 2 & -1 & & \\ -1 & 2 & -1 & \\ & & \ddots & \\ & & -1 & 2 \end{bmatrix} \tag{4.5}$$

for first- or second-order regularization, respectively. The authors of [85] note that α and $\mathbf{L}$ are hyperparameters the user should optimize. Intuitively, the choice of $\mathbf{L}$ minimizes either the norm of the solution (zeroth-order) or the smoothness of features (second/third-order). Additionally, α should be set adaptively, changing the influence of the regularization term based on the input. The authors test multiple selection methods, namely adaptive L-curve and Generalized cross-validation

selection [13], as well as a brute-force search. As a baseline, they compare their approach with the unregularized version shown in (4.2), making two key findings: First, the regularized version with adaptive parameter selection outperforms the ground truth for every sample. Second, the reconstruction performance with and without regularization is nearly equal to a narrow-band target spectrum. The latter is a bit unintuitive, as one would expect smaller peaks to be harder to reconstruct. Small-band spectra need precise filter design to accurately estimate the peak position and Full Width at Half Maximum (FWHM). However, the exemplary reconstructions in [85] do not only show broader spectra for the broad-band samples but also unsymmetrical and multi-peak variants. Therefore, the distinction between broad-band and narrow-band peaks might be biased. The authors recommend using Tikhonov with L-curve parameter selection in a live environment, as this yields satisfactory results while not being as computationally expensive as brute-force parameter search.

In addition to Tikhonov regularization, [110] adds Singular Value Decomposition (SVD) to find solutions more quickly. The authors use zeroth-order regularization and decompose $\mathbf{H}$ into

$$\mathbf{H} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T, \tag{4.6}$$

with $\mathbf{\Sigma} \in \mathbb{R}^{N\times M}$ being a diagonal matrix holding the singular values σ_i in descending order, and $\mathbf{U}, \mathbf{V} \in \mathbb{R}^{M\times M}$ holding the left and right singular $m \times 1$ vectors $\mathbf{u}_i$ and $\mathbf{v}_i$ in their columns, respectively. In this form, the authors claim that solutions to (4.4) can be formulated as applying filter factors to the solution vector

$$\hat{\mathbf{s}}_\alpha = \sum_{i=1}^{M} \phi_i^{[\alpha]} \frac{\mathbf{u}_i^T \mathbf{v}_i}{\sigma_i} \mathbf{r}, \tag{4.7}$$

with the filter factors being

$$\phi_i^{[\alpha]} = \frac{\sigma_i^2}{\sigma_i^2 + \alpha^2}. \tag{4.8}$$

As in [85], the authors use L-curve optimization to find fitting values for α. Additionally, they explain why the correct setting of the regularization parameter is important by utilizing the relation between the SVD and a Fourier analysis. For higher values of i, $\mathbf{v}_i$ and $\mathbf{u}_i$ represent the high-frequency components, while lower index values reflect lower frequencies. Therefore, too-low values of α represent noise-dominated solutions with the high-frequency components not being suppressed enough. In contrast, too high values of α favor the low-frequency components of the solution, leading to a distorted spectrum. In total the approach in [110] shows interesting insights into the parameterization of the Tikhonov reg-

ularization and enables a quicker solution finding. However, it does not introduce an improved result over [85], making it not applicable to our scenario.

Another interesting and common approach to solve the inverse problem from (4.1) is sparse recovery [68]. The authors transform the sought-after spectrum $\mathbf{s}$ into a sparse representation

$$\mathbf{s} = \mathbf{\Phi c}, \tag{4.9}$$

with $\mathbf{\Phi}$ being a sparsifying basis, and $\mathbf{c}$ the corresponding sparse vector. Instead of using the ℓ_2-norm they formulate the optimization as an ℓ_1-norm problem

$$\min_c \|\mathbf{r} - \mathbf{H\Phi c}\|_2^2 + \alpha\|\mathbf{c}\|_1 \text{ s.t. } \mathbf{c} \in \mathbb{R}_{>0}. \tag{4.10}$$

If $\mathbf{s}$ is direct sparse $\mathbf{\Phi}$ can be equal to $\mathbf{I}$. However, usually this is not the case, so that the choice of $\mathbf{\Phi} \in \mathbb{R}^{M\times N}$ heavily influences the reconstruction performance. In most publications, a Gaussian kernel

$$\phi_n(\lambda) = \exp \frac{-(\lambda - \lambda_{0n})^2}{2\sigma_n^2} \tag{4.11}$$

is used for $\mathbf{\Phi}$ for its simplicity and preservation of smooth features. A Gaussian filter characteristic is also seen as the optimal case, making the Gaussian kernel the intuitive choice. For every column of Φ, one kernel function needs to be generated by varying the Gaussian peak location λ_{0n} and width σ_n. To make modeling easier, the width can also be described by the FWHM

$$w_0 = 2\sqrt{2\ln 2}\sigma_n. \tag{4.12}$$

Setting this approach into perspective is challenging, as the authors in [68] do not include a comparison to the NNLS with Tikhonov regularization approach. Also, their experimental validation is limited as they only show reconstruction results and metrics for one sample. Nonetheless, the authors of [147] describe the sparse recovery approach as more accurate regarding reconstruction error while delivering a higher spectral resolution than ℓ_2-norm minimization approaches. However, as mentioned before, the performance heavily relies on an optimal set of kernels and their combination with the filter characteristics $\mathbf{H}$. Therefore, a general statement on a superior approach can not be justified in our opinion. For our sensors, we could not produce a usable result with sparse recovery, which can be due to multiple reasons, such as the fact that the Gaussian filter characteristics are far from Gaussian or that Gaussian kernels are not the right choice for our system. Due to time limitations, we also did not investigate this approach further.

4.5 Data-Driven Reconstruction

To further increase reconstruction performance, newer literature has started to incorporate data-driven methods, which we categorize into Hybrid and one-shot learning. Hybrid learning includes data-driven corrections in the evaluation pipeline, combined with classic reconstruction approaches. In contrast, one-shot learning utilizes only one step to directly reconstruct a usable spectrum from the sensor output. In the following, we will review the most prominent data-driven approaches for spectral reconstruction, starting with hybrid approaches and finishing with one-shot approaches, as they are closest to the approach we take in this thesis.

4.5.1 Hybrid Reconstruction

In [86] the authors use the Tikhonov regularized NNLS algorithm as shown in (4.4) without the use of SVD. As a novelty, they add a learnable transformation $\mathbf{T} \in \mathbb{R}^{N \times N}$ to the sensor response $\mathbf{r}$ turning the model formulation of (4.1) into

$$\mathbf{Tr} = \mathbf{Hs}. \tag{4.13}$$

They learn $\mathbf{R}$ in a least-square sense minimizing

$$\|\mathbf{Hs} - \mathbf{Tr}\|_2. \tag{4.14}$$

The results from [86] utilize the Medusa sensor, making this approach a good hybrid-learning baseline in Chap. 6. The resulting reconstruction performance with the transformation shows the applicability of classic reconstruction methods to our sensing devices with a hybrid approach. While the standard NNLS with Tikhonov regularization can not reconstruct a usable spectrum, the addition of the transformation matrix shows promising results in both correlation and reconstruction errors. For concrete performance metrics please refer to Sec. 6.3, where we compare the transformation approach to our one-shot learning model.

In addition to performance metrics, the authors discuss the reason behind the vast improvements over the standard NNLS with Tikhonov regularization. They compare the angle between the left and right-hand side of (4.13) with a learned $\mathbf{T}$ and $\mathbf{T} = \mathbf{I}$ using the cosine similarity. Ideally, the similarity should be close to 1 to show that the model is valid for the given sensor. However, without the transformation, the cosine similarity is scattered around 0.9, with many outliers going down as low as 0.75. In contrast, with the transformation, the similarity is nearly entirely at 1, with outliers reaching 0.98. They reason that due to changes in light angle and between-sensor divergences, $\mathbf{H}$ does not correctly reflect the filter characteristics in the given measurement setup. Therefore, $\mathbf{T}$ corrects these discrepancies, training the model to incorporate divergences based on a different

measurement setup in a data-driven manner. The authors also show the data efficiency of their approach by obtaining the improvements in cosine similarity with only 10% of their theoretically available training data.

A deep-learning approach to hybrid reconstruction is taken by Zhang, Jinhui and Zhu, Xueyu and Bao, Jie in [147] and [72]. Both approaches are from the same group of authors and add neural networks to the reconstruction pipeline to improve its performance compared to a classic reconstruction. In the first publication [147], the authors create a pipeline that combines an Autoencoder (AE) with the sparse recovery approach from (4.10). They use the same Gaussian kernel as in (4.11) and vary the function for FWHMs between 5 and 100nm and peak locations evenly distributed between 450nm and 750nm. In total, the authors create 497 kernel functions with these parameters. As with other approaches, they also use l-curve optimization to set the regularization parameter α adaptively. In contrast to previous works, they denoise the raw signals with an AE before reconstructing them with sparse recovery. The idea behind this approach are the denoising properties of AEs previously demonstrated by other applications such as image denoising [50]. The network consists of 32 hidden neurons, with the input and output layer sizes not mentioned. It probably equals the number of Gaussian kernels used to achieve the mathematically correct shape. The authors test their theory by conducting two tests: First, they add Gaussian white noise with rising Signal-to-Noise-Ratio (SNR)s of 30dB, 35dB, and 40dB to simulated sensor readings and compare the reconstruction result to the standard sparse recovery approach. Second, they test real measurements without adding Gaussian noise to investigate probable performance gains in a real-world scenario. With the synthetic data, the authors can measure an average reduction in reconstruction error of 38% when denoising the synthetic data. As expected, the uplift in performance rises with a lower SNR, and the gap between raw and denoised widens. Additionally, the denoised version has a relatively constant error of 0.0781, 0.0775, and 0.0902 for 40dB, 35dB, and 30dB, respectively, while the error rises drastically from 0.0945 over 0.1358 to 0.2031 for the standard sparse recovery approach. For real measurements, the gap between raw and denoised widens even more with a reconstruction error of 0.4335 for the standard approach and 0.1004 for the AE. The last test shows that with this approach, the hybrid pipeline can correct Gaussian noise *and* systematic deviations. Similar to the transformation approach in [86], the system learns to adapt to data-driven deviations. However, instead of a transformation matrix, the authors in [147] learn an AE. This can be reflected by a non-linear function applied to the left-hand side of (4.1) turning it into

$$f_{AE}(\mathbf{r}) = \mathbf{H}\mathbf{s}. \tag{4.15}$$

Even though both hybrid approaches are pretty similar, it is hard to compare their results. First, the classic reconstruction approach connected to the learnable transformation/function differs for both approaches. Also, the choice of hyperparame-

ters heavily influences the performance, especially for the AE. The user must tune the choice of kernel functions, setting of α, and the type and parameterization of the AE. However, regarding data efficiency, the transformation is again far superior. For the AE, the authors need to use over 400 training samples in their second test, while the transformation approach can be trained with around 10 samples.

In their second work, Jinhui Zhang and Xueyu Zhu and Jie Bao go more into depth with an additional hybrid approach and multiple baselines [72]. They use the same sensing device but test its performance with regularized NNLS and sparse recovery as a baseline. In addition, they also learn two one-shot reconstruction networks, namely an MLP and a CNN. However, the authors use these one-shot approaches as an additional baseline for their proposed hybrid system rather than a novelty. The latter combines the regularized NNLS or sparse recovery and an MLP. In contrast to their previous publication, the authors now apply the neural network *after* the reconstruction with a classic approach. They argue that with the previously seen performance gains of one-shot approaches in the literature, combining the classic reconstruction with a neural network as a second mean of reconstruction could lead to an even better result. They call their hybrid approach solver-informed neural networks.

To prove their idea, the authors conduct four tests: First, they thoroughly set a baseline with the synthetic dataset with added noise as previously seen in [147]. They compare the two classic approaches to a plain one-shot MLP. Second, the authors use the same corrupted synthetic dataset to collect performance metrics for their proposed solver-informed approach. They also add a one-shot CNN as a third baseline. For the third and fourth tests, the authors use the same models/systems as in the first and second tests, but now with real measurement data. In summary, the thorough testing demonstrates that compared to the classic reconstruction approaches, the one-shot approaches are far superior. Even the simple one-shot MLP outperforms all classic approaches and the denoising AE system from [147] with 0.0377, 0.0496, and 0.0626 average reconstruction errors for 40dB, 35dB, and 30dB respectively. These numbers are on average 39% better than the previously best-performing denoising AE approach in the same test. In the second test, the solver-informed approaches take the lead over the one-shot approaches, but only by 13% average improvement in reconstruction error. Interestingly, it does not matter if the MLP or CNN is used as the one-shot approach. Both network architectures show nearly the same reconstruction performance. The same is true for the choice of classic reconstruction approach in the solver-informed networks. Both tested solvers, regularized NNLS and sparse recovery, show nearly identical performance when paired with a neural network. The last two tests with real data depict the same overall picture. The one-shot approaches heavily outperform the classic reconstruction methods, and the solver-informed neural networks beat the one-shot approaches slightly.

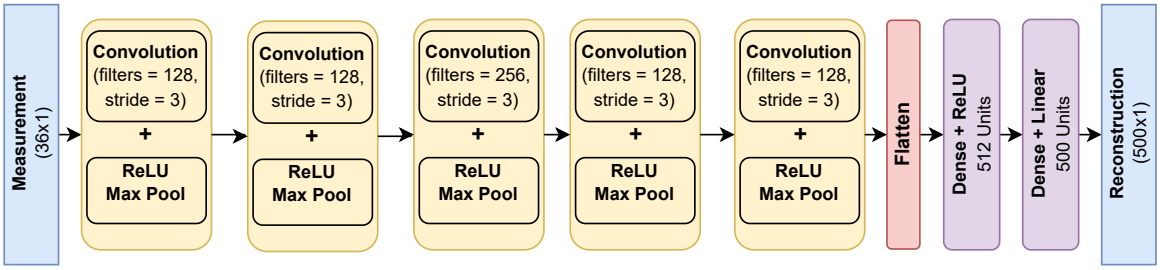


Figure 4.6: Overview of the CNN-based one-shot architecture. Illustrated after [33].

In total, both publications of Jinhui Zhang and Xueyu Zhu and Jie Bao show the substantial benefits of using neural networks to reconstruct spectral information in a data-driven manner. However, especially when looking at the exemplary reconstructions shown in both publications, it is visible that there is still open work in this field. While the general shape of the ground truth spectra is always reconstructed well for the solver-informed neural networks, the reconstruction error is still clearly visible and, in some samples, quite drastic. Additionally, even though the filter characteristics of the sensor used by the authors look considerably worse than the ones previously seen for the classic approaches, they are still smoother than our filters depicted in Fig. 4.5. Like our filters, the quantum dot array from [72]; [147] does not have confined wavelength bands but shows no interferences on top of the filter function. Also, every filter has the same overall shape in contrast to our drastically varying shapes. Therefore, we suspect the approaches shown by Jinhui Zhang and Xueyu Zhu and Jie Bao to perform not as well with our sensing devices. This is why we chose to investigate one-shot approaches further rather than hybrid ones to eliminate possible errors introduced by our chaotic filter characteristics.

4.5.2 One-shot learning

With the rise in computational capabilities and the popularity of recent Neural Network architectures computational spectrometers have also adopted deep learning as an alternative for the linear model in (4.1). One-shot learning uses a singular learned model that substitutes a classic reconstruction approach in one pass. Unlike hybrid methods, neither the filter characteristics $\mathbf{H}$ nor an inverse solver is needed to utilize such an approach. However, the novel approaches need data to train the reconstruction on target spectra and, therefore, are not as generalizable to other sensors as previous classic approaches.

The first appearance of one-shot learning for computational spectrometers can be found in the works of Cheolsun Kim and Dongju Park and Heung-No Lee. In two consecutive publications [33]; [34], the authors make foundational work for the one-shot learning approach. In [33] the authors propose a simple 1-D-CNN with 5 convolutional layers and a two-layer MLP-head as illustrated in Fig. 4.6. As with other one-shot approaches, the authors feed the neural network with the raw

sensor readings and train it to reconstruct a target spectrum directly. In terms of the previous notation, the authors learn a function represented by the CNN

$$f_{\text{CNN}}(\mathbf{r}) = \hat{\mathbf{s}} \tag{4.16}$$

that estimates the sough-after target spectrum $\mathbf{s}$. They compare the CNN's reconstruction performance to the sparse-recovery algorithm from (4.10). Instead of Gaussian kernels, they use a Discrete Cosine Transform (DCT) matrix, most probably to not give an unfair advantage to the sparse-recovery approach. This is because the training and test set comprises synthetic data constructed using a combination of Gaussian kernels. Unfortunately, the authors do not mention how they parameterize the DCT. However, the standard formulation

$$\phi_n = \sum_{m=1}^{M} \lambda_m \cos\left[\frac{\pi}{M}(m + \frac{1}{2})n\right] \tag{4.17}$$

originally proposed by [4] should also apply here. The authors can measure an improvement of 71.7% in reconstruction performance over the sparse recovery approach, showing the substantial lead of one-shot learning approaches for spectral reconstruction. Looking at exemplary reconstruction results, the uplift in performance is also visually noticeable. The CNN matches nearly perfectly with the target spectra, while the DCT struggles in more complex multi-peak scenarios. However, these results should be taken with a grain of salt. As mentioned, the authors use synthetic data to train their models, making it hard to draw conclusions for real data. Miniature spectrometers are susceptible to noise, changing measurement setups, and manufacturing uncertainties. All these factors are eliminated with synthetic data produced by calculating the ideal sensor response for a superposition of Gaussian kernels as the target spectrum.

In their subsequent publication [34], the same group of authors directly address the problem with synthetic data by training a novel architecture with real measurement data. They collect a dataset using colored foils and other filtering media to collect broad- and narrow-band spectra (cf. Fig. 4.7). Additionally, they varied the light intensity to collect 3223 samples, split into train, test, and validation sets with ratios of 80%, 10%, and 10%. Instead of a CNN backbone, the authors use a U-Net architecture with three initial convolutional layers, four down-scaling layers, four up-scaling deconvolutional layers, and four final convolutional layers. Like other U-Nets, the down- and upscaling layers are connected via skip connections, as is the input with the output layer. The authors do not include an MLP head after the U-Net, so the final reconstruction is handled using the four convolutional layers at the end. This architecture is much more complex than Cheolsun Kim and Dongju Park and Jioh Lee and Heung-No Lee's first approach, which could be connected to the more complicated training data with real-world measurements. During

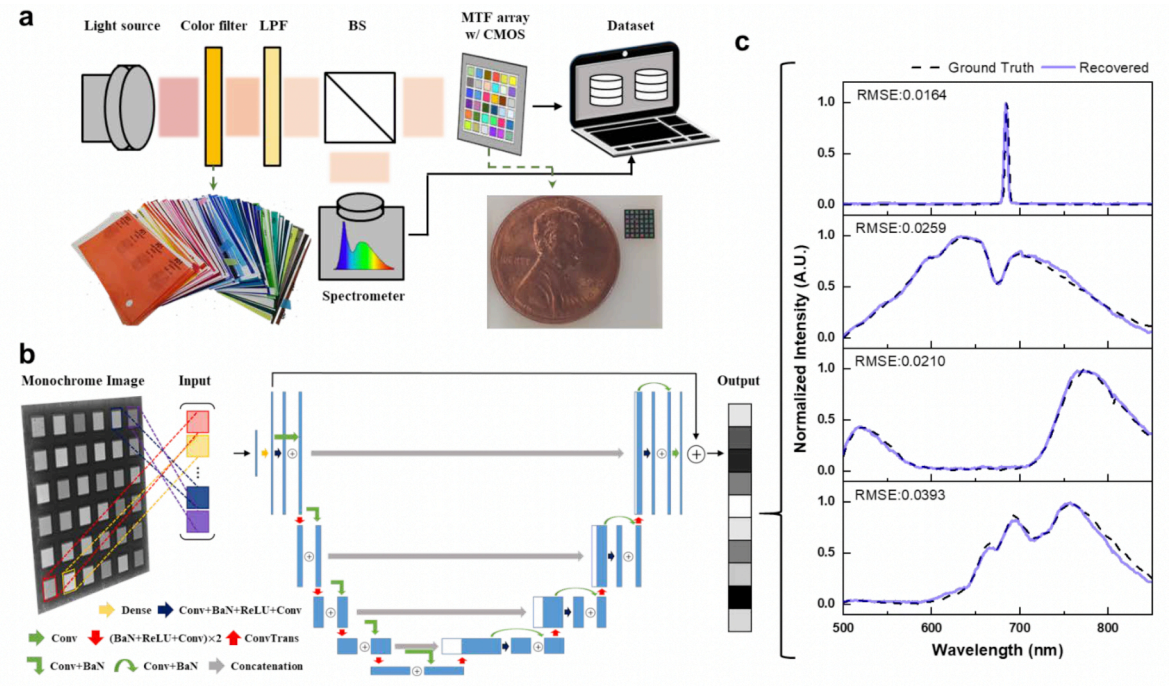


Figure 4.7: Overview of the U-net-based one-shot architecture. Taken from [34] with permission. (a) shows a Schematic overview of the measurement setup with a light source shining on colored filters, followed by a long pass filter (LPF). Afterward, a beam splitter (BS) is used to split the light equally between a reference spectrometer for ground truth and the used multi-layer thin film (MTF) filter array. In (b) an overview of the network architecture is given, with (c) showing exemplary reconstruction results in different liquids.

testing, they first benchmark the reconstruction performance based on the Root Mean Squared Error (RMSE). For the 323 test samples, the average RMSE lies at 0.0288, with the narrow-band spectra averaging at 0.0158 (33 samples) and the broad-band spectra at 0.0303 (280 samples). These numbers show two things. First, as seen before, narrow-band spectra can be reconstructed more easily in this setup. However, as with the original Tikhonov regularization approach in (4.4), the broad-band spectra are broader than the narrow-band peaks and more complex in shape. Nonetheless, seeing how precisely the narrow-band peaks can be reconstructed with the shown U-Net approach is impressive. Second, the overall RMSE is in comparison to the author's previous publication [33] similar, with 0.0288 (U-Net) compared to 0.0279 (CNN). Therefore, even though the training data is more complex, the reconstruction performance is similar to using the more complex U-Net. Unfortunately, the authors do not include a direct comparison between the two architectures for their new real-world dataset. However, we show a comparison of various architectures, including a U-Net and CNN, in our testing in Sec. 5.4.1.

As a final test, Cheolsun Kim and Dongju Park and Jioh Lee and Heung-No Lee reconstruct the transmission spectra of five colored liquids without retraining the network. This test reflects an application scenario, where, e.g., the color-changing

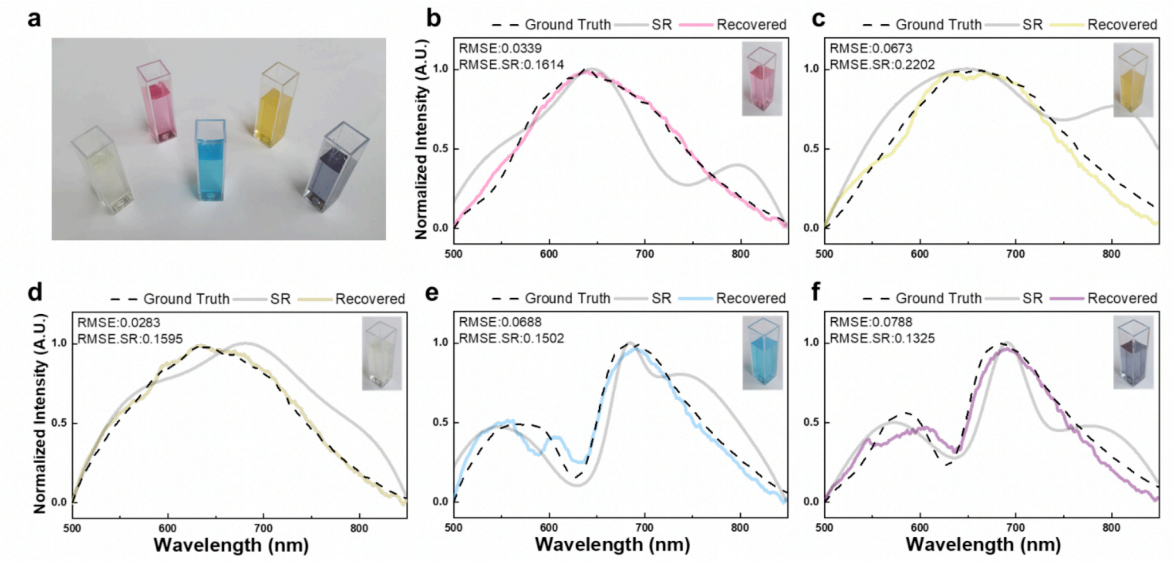


Figure 4.8: Results with colored liquids taken from [34] with permission. (a) shows the different liquids, while (b) to (c) depicts the corresponding reconstruction results with the ground truth as dashed lines, sparse recovery (SR) as solid gray lines and the reconstruction result from the proposed one-shot learning approach (recovered) as solid colored lines

properties of chemicals are used to analyze water quality. In contrast to the benchmark before, they also include the sparse-recovery approach, but without mentioning the used kernel function. However, as this publication is subsequent to [33], the authors probably use the same DCT kernel. Even though the number of samples is limited, some trends are visible (cf. 4.8). First, when reconstructing simpler shapes, the result is nearly perfect ((b) to (d)). Three of the five liquids have such a simple shaping, meaning there is only one very broad peak above the measured area. In contrast, the sparse recovery approach cannot locate the maximum, and the overall shape is more a guess than a clear indication. In terms of metrics, the one-shot network has an RMSE of 0.0432, while the sparse recovery approach scores 0.1804, showing an improvement of 76%. The remaining two samples, (e) to (f), with multiple peaks in the target spectrum, are not perfectly reconstructed by any approach. Nonetheless, the one-shot model again shows a substantial lead with 0.0738 RMSE compared to 0.1413 for the sparse recovery. Interestingly, the lead of the novel approach is much smaller in the multi-peak scenario, with only 47%. This change is also visually noticeable because the sparse recovery approach includes multiple peaks in both cases. This behavior is beneficial in the multi-peak scenario but degrades the performance if only one peak exists in the target spectrum. However, the authors do not reason why the baseline approach shows this behavior. It could be due to the parameterization not being transferable to spectra stemming from liquid transmission media. The degradation in reconstruction performance for the one-shot model in the multi-peak scenario is mostly linked to errors between 500nm and 600nm. In this range, the reconstructed spectrum

diverges drastically from the target spectrum. As this error is reproducible, it might be linked to biased training data, where this wavelength range is underrepresented. Another point of failure might be the filters not producing an optimal spectral representation for this wavelength range.

4.6 Conclusion and key takeaways

We have discussed the most relevant literature, techniques, and physics connected to computational spectrometers. Combining expert knowledge, physics, and efficient machine learning makes the approach proposed in the following chapters unique. Therefore, a clear understanding of the background is essential to grasping its benefits completely. The key takeaways for the next chapters are: (i) Computational spectrometers combine an array of light filtering media with a detector array that measures the different components of the light reaching the sensor. In signal processing terms, this sensing principle is similar to a Fourier transform that splits a signal into its frequency components. Analogously, the sensor splits the light into different wavelengths that reach it. (ii) The differentiating factor between sensing devices comes from the light filtering techniques used to create the filter array. Depending on the method, different sensors drastically vary in size, cost, and data quality. (iii) Our sensing devices use plasmonic filters manufactured in CMOS together with the rest of the chip, leading to the most cost-effective technology compared to other publications. Nevertheless, while this cost-effective solution enables cost-sensitive applications, its chaotic filter characteristics complicate the reconstruction of the desired outcome spectrum. (iv) Nonetheless, we can build up on the ideas of other reconstruction methods due to the similarities of computational spectrometers. Despite the reconstruction method, the task can always be traced back to the linear model in (4.1), albeit with solving the inverse explicitly with methods such as NNLS or sparse recovery, or implicitly with neural networks in terms of (4.16). (v) To circumvent our varying filter characteristics and the mathematically unstable inverse of (4.1), we have decided to investigate one-shot learning approaches further. This allows us to learn the behavior of the chip rather than relying on a characterization that changes between sensors and measurement setups.

CHAPTER 5

Spectral Reconstruction with Neural Networks

In the last chapter, we have discussed the background to solve the spectral reconstruction problem implicitly by learning a transformation from the 2D sensor output to the sought-after spectrum of light hitting the sensing device. This transformation can be formulated as a non-linear function applied to the sensor output, as described in (4.16). To illustrate this, Fig. 5.1 shows an exemplary sensor output and its respective ground-truth spectrum being transformed by a neural network f_{NN}. In this chapter we introduce all the technical details necessary for the key contributions of this thesis: (i) We benchmark multiple neural network architectures for our sensing devices in terms of reconstruction performance and stability against noise. (ii) We introduce our publicly available dataset for spectral reconstruction. (iii) Utilizing a physics-informed data augmentation, we expand this dataset to make it big enough to train deeper and more complex network architectures.

We first briefly summarize the challenges connected with classic spectral reconstruction to motivate the usage of one-shot learning for computational spectrometers. Afterward, we cover the dataset and data-augmentation that builds the basis for training the neural networks in Sec. 5.3. Then, we benchmark these networks with the Infimedar and Medusa sensor to find suitable architectures for spectral reconstruction. Concluding this chapter, we prune unnecessary filters from the sensor, which forms the basis for the more efficient NanoSpectral sensor utilized in Chap. 6.

5.1 Challenges with Classic Reconstruction

As discussed in Chap. 4, classic approaches for computational spectrometers solve the inverse of the linear model

$$\mathbf{r} = \mathbf{H}\mathbf{s}, \tag{5.1}$$

with $\mathbf{r} \in \mathbb{R}^N$ representing the sensor response and $\mathbf{s} \in \mathbb{R}^M$ the sought-after spectrum modified by the sensor's filter characteristic $\mathbf{H} \in \mathbb{R}^{N \times M}$. It is commonly known that ill-posed inverse problems have the tendency to be numerically unstable, especially in absence of proper regularization techniques. In our case, the linear system is heavily over determined with $N \gg M$ (1600 channels vs 240 wave-

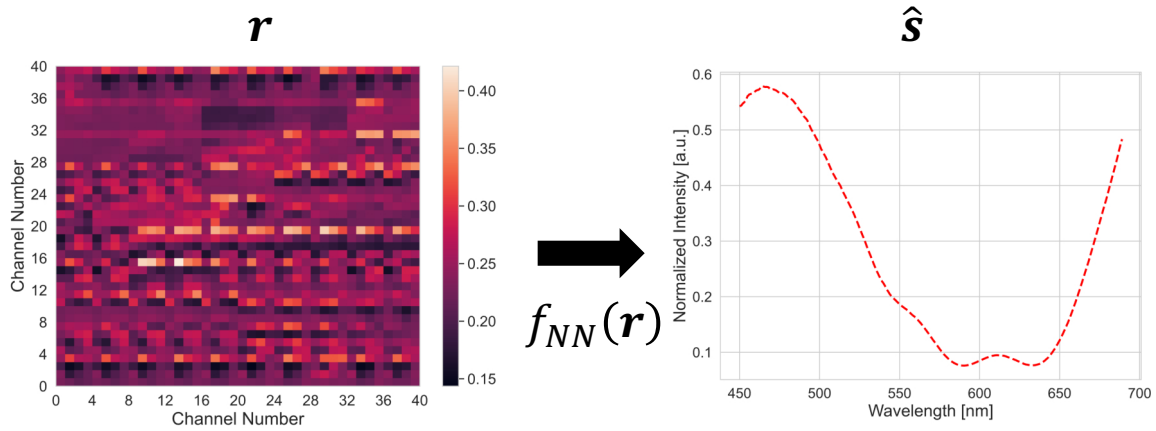


Figure 5.1: Illustration of the transformation in spectral reconstruction with neural networks. The neural network f_{NN} transforms the sensor output $\mathbf{r}$ into the estimated spectrum $\hat{\mathbf{s}}$.

lengths), leading to the assumption that classic reconstruction approaches will also be heavily unstable. This is especially problematic considering the different sources of data deviations introduced by the underlying light-filtering technique. As previously shown, $\mathbf{H}$ is far more complex than the characteristics from other sensing devices. Also, we currently cannot rule out in-between device deviations from manufacturing. Due to our sensors not being in series production yet, only a limited number of sensing devices have been produced. Therefore, no statistic on manufacturing uncertainties exist. Furthermore, $\mathbf{H}$ changes depending on the angle of light hitting the sensor, which is a characteristic of plasmonic filters. Apart from these systematic deviations, standard noise sources like gaussian sensor noise and quantization noise also introduce further sources of uncertainties. All these reasons demonstrate why classic reconstruction approaches like NNLS with Tikhonov regularization or sparse recovery do not produce a usable result, as we will demonstrate further into this chapter.

Even though hybrid approaches such as the rotation transformation in (4.13) might be an applicable alternative, we will further investigate one-shot learning approaches based on neural networks to completely circumvent the inverse problem mentioned above. Apart from this benefit stemming from the methodology itself, we can also rely on the great improvements introduced by the TinyML community, making the integration of a NN-based approach into the sensor system easier. The techniques mentioned in Chap. 2 such as scalable architectures or quantization make finding and optimizing a model for deployment more approachable. Additionally, the well integrated toolchain for embedded inference based on TensorFlow light and libraries from µC vendors let us deploy a new model to the sensor system quickly.

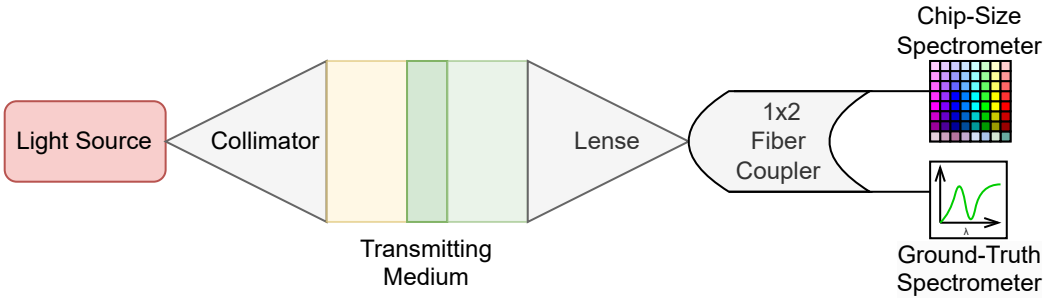


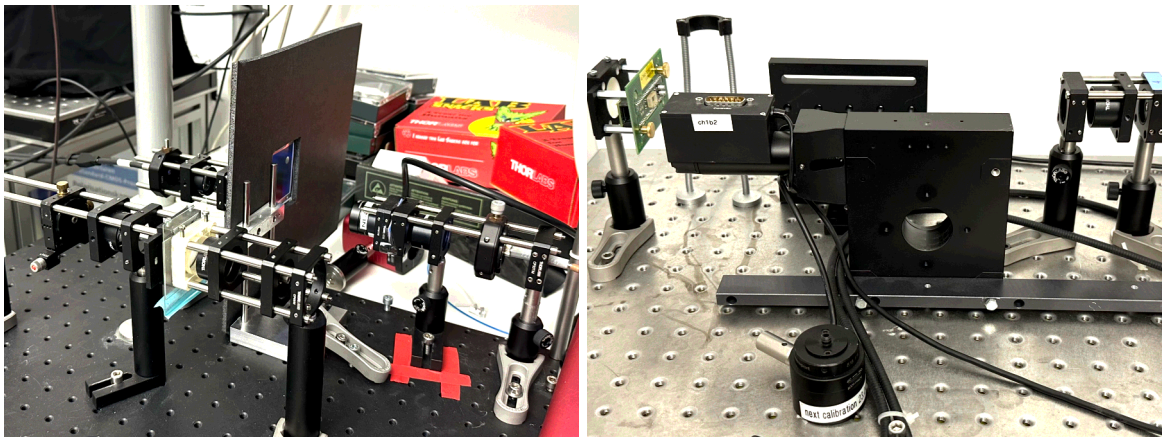
Figure 5.2: Schematic measurement setup for collecting the datasets. A halogen light source is connected to a collimator creating a beam of collimated light. In this beam transmitting media are placed that create the measurement samples. The resulting light is then focused into a 1x2 fiber coupler that splits the light equally between the ground-truth spectrometer and the CSS.

5.2 Dataset

Even though it is clear by now that learning-based approaches are favorable over classic methods for our sensing devices, we need to address the downside of data availability. An essential factor in training an effective machine-learning model for spectral reconstruction is ensuring that the dataset is diverse enough to encompass a wide range of complex spectra while maintaining sufficient redundancy to account for variations caused by manufacturing inconsistencies and sensor noise. However, since we work with custom sensors that are not commercially available right now, we can not rely on existing data to train models. Additionally, there is no dataset available for computational spectrometers making transfer learning approaches [149] also not feasible. Therefore, we have decided to collect our own dataset, which we also have published together with our work in [136]. The following section highlights the considerations about the measurement setup, complete acquisition process, and expansion of dataset using a physics informed augmentation. The general process highlighted here is valid for all three sensor generations and has been utilized every time a new generation was available. Nonetheless, over the course of the last years the measurement setup has been steadily improved with minor modifications. We will of course also discuss these modifications to provide a complete impression of every iteration.

5.2.1 Measurement Setup

For the dataset, we simultaneously measure the ground-truth spectrum $\mathbf{s}$ and the sensor response $\mathbf{r}$. To achieve this, we design a measurement setup as shown in Fig. 5.2. A light source is connected to a collimator via an optical fiber, producing a collimated light beam. Various transmitting media are placed within this beam to serve as measurement samples. For the most part, we use colored filter foils



(a) (b)

Figure 5.3: Photographs of light paths for the Infimedar Sensor. The left photograph (a) shows two paths with collimated light beams that can be used to place measurement species inside. The right photograph (b) shows the collimated light beam hitting the Infimedar sensor directly.

from Roscolux [116], similar to those employed in [34]. These foils, commonly used in theatrical lighting to create different color hues, provide a broad range of non-trivial transmission spectra, making them well-suited for our dataset.

However, Roscolux foils are transparent to infrared light, making them less effective for measurements beyond 700nm. To ensure a well-defined spectral range, we limit the reconstruction in this thesis to the visible spectrum between 450nm and 690nm. This restriction allows us to place an infrared filter in front of the light source, enhancing the signal dynamic range within the visible spectrum. The filter's effect is noticeable at wavelengths above 690nm, slightly reducing the measurement range below 700nm. Additionally, with a silicon-based substrate like the wafers used for CMOS, we are not able to measure light reliably below 450nm, creating our lower reconstruction bound.

We focus the resulting light into a 1x2 fiber coupler that splits it equally between our CSS and a ground-truth spectrometer from Broadcom. Particularly, we use the Broadcom QMini Wide-VIS [24] spectrometer with a measurement range from 225nm-1000nm. We cut this range down to the wavelengths mentioned above to cater to our designated reconstruction range.

It should be mentioned that the QMini has a fiber input allowing us to directly connect the fiber coupler to the spectrometer. However, with the CSS we only have a bare die to capture the incoming light. Therefore, we add a second collimator after the fiber coupler on the CSS end and shine this portion of light directly onto the sensor die. The collimator ensures that we have controllable and uniform light angles hitting the sensor to compensate for the angle dependency of the CSS.

Figure 5.3(a) shows a photograph of the implemented measurement setup for the Infimedar sensor. The image highlights two possible light paths available for dataset acquisition. The front path, designed as depicted in Fig. 5.2, is primarily used for measuring the colored filter foils and constitutes the majority of the dataset. The secondary path in the background includes a dampening filter and a protective window, intended for higher-intensity light sources to prevent oversaturation. However, since the dataset is recorded using a halogen light source, this alternative path remains unused.

The fiber coupler, currently connected to the background light path in Fig. 5.3(a), is linked to the two sensors located beneath the black cloth at the back of the setup. The cloth serves to block stray light from interfering with the exposed light path of the bare-die CSS sensor. A closer view of this measurement setup is provided in Fig. 5.3(b). Here, the sensor chip and its circuit board are mounted on the left side, directly illuminated by the collimated light beam exiting from the collimator on the right. The black boxes and surrounding installations are non-functional; they merely support the black cloth, ensuring it does not obstruct the light path.

5.2.2 Acquisition Process and Iterations

We measure a total of 214 Roscolux filter foils, with the ground truth spectrometer set to 1nm resolution between 450nm and 690nm. This leads to the dataset shape of (214, 240) for the ground truth and (214, 40, 40) for the CSS measurements. To measure the transmission spectrum of the measurement samples without the influence of the light source, we need to normalize the measurements before training. The normalized transmission spectrum for each channel n

$$\tilde{r_n} = \frac{r_n - r_n^d}{r_n^w - r_n^d}, \tag{5.2}$$

equals the original sensor response r_n subtracted by a dark measurement r_n^d and divided by a white measurement r_n^w also corrected by the dark measurement. We collect r_n^w without any transmitting medium, capturing the pure spectrum of the light source. In contrast, we measure the dark sensor response r_n^d without any light being present. By dividing by the white measurement we eliminate the influence of the light source, leaving only the spectrum of the actual transmitting medium. The subtraction of the dark measurement corrects for any remaining dark current that might be present from the readout electronics. Naturally, we apply this process also to the ground truth spectrum, but in this case for each wavelength m instead of each channel. In addition, this process ensures values between 0 and 1, because the colored foils all have a transmittance smaller than 1. This means that the white measurement will always show higher or equal values in comparison to a measurement with a transmitting medium. This has the advantage that we do not

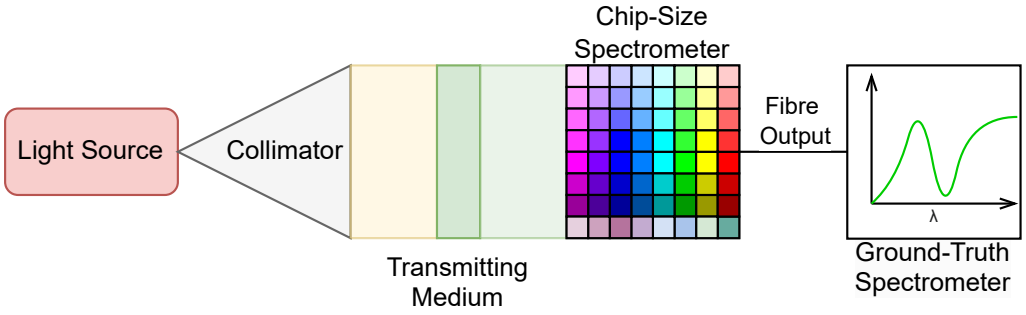


Figure 5.4: Schematic measurement setup with the Medusa sensor. Due to a change in the circuit board design, we can directly shine the light beam onto the sensor die and a reference fibre going to the ground truth spectrometer at the same time. Therefore, the fiber coupler is not needed anymore, simplifying the measurement setup.

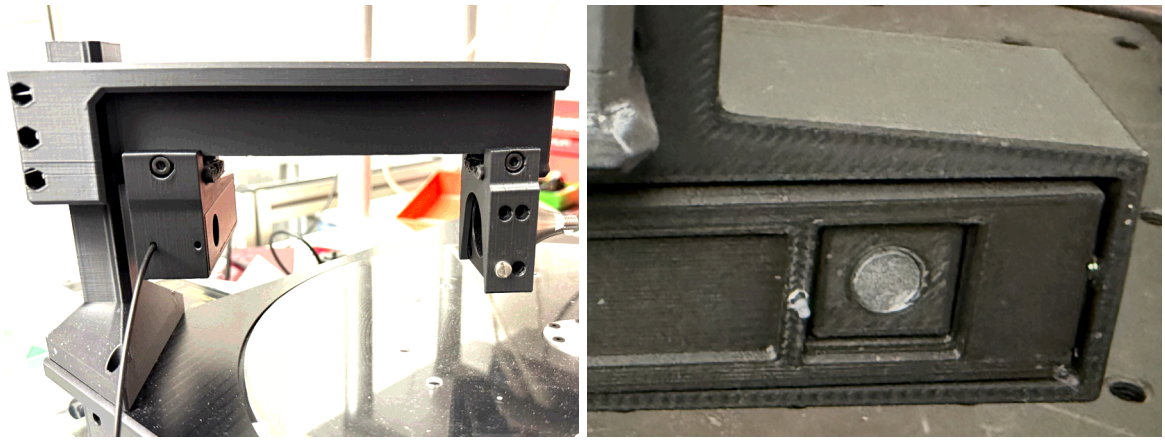
need to do an additional normalization step in preparation for the NN training, as the samples have already a good numerical range.

With the second generation CSS, we can simplify the measurement setup drastically. While the overall schematic shown in Fig. 5.2 remains intact, the new circuit board now has a place to attach a polymer optic fiber right next to the sensor die. This allows us to defer the fiber coupler and directly shine the light beam onto the die as well as the fiber for the reference spectrometer simultaneously. This simplifies the setup drastically, as we do not need a second optical setup to split the light between CSS and ground truth spectrometer (cf. Fig. 5.4).

This can also be seen in the Photographs of the Medusa setup in Fig. 5.5(a). The overall footprint is much smaller, leading to fewer error sources during the acquisition. For the last iteration, NanoSpectral, we only make minor adjustments. The fiber connection has been moved to the outside of the housing to allow for a diffusor (cf. Fig. 5.5(b)). The diffusor helps to correct the light angle hitting the sensor, which should lead to a more stable reconstruction.

5.2.3 Augmentation

The data collection process described above produces a base dataset of only 214 samples, which is insufficient for training complex neural networks. Expanding the dataset through additional measurements would require extensive manual effort to obtain thousands of new samples. To address this limitation, we introduce a data augmentation technique based on the physics of transmission spectra. The luminous flux $\Phi_{12}(\lambda)$ after passing through two optical filters with transmittances τ_1 and τ_2 is given by



(a) (b)

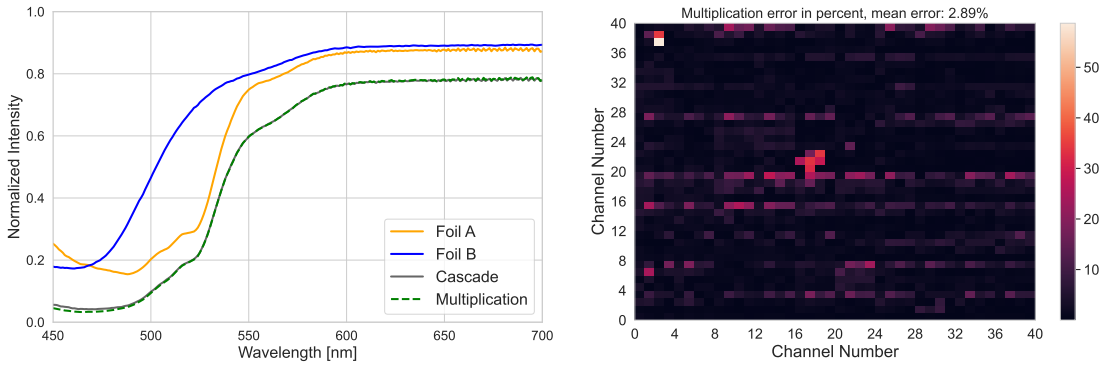
Figure 5.5: Photographs of setup improvements for Medusa (a) and NanoSpectral (b). We can shrink the overall setup size for both generations due to the addition of an optical fiber output. Nanospectral moves the fiber input to the front of the housing and adds a diffusor to correct light angles.

$$\Phi_{12}(\lambda) = \tau_1(\lambda)\tau_2(\lambda)\Phi_0(\lambda), \tag{5.3}$$

where Φ_0 represents the initial luminous flux entering the first filter [106]. This relation allows us to generate physically consistent data by multiplying two spectra from the base dataset, effectively simulating the cascading of two filter foils. To ensure the validity of the generated spectra, we remove any combinations where the average signal falls below 10%. In cases with low signal strength, the senors may not react linearly, making (5.3) invalid. This typically occurs when combining foils with low transmittance or from distinct spectral regions (e.g., blue and red). Afterwards the total dataset size increases to 10,117 samples. In theory the total number of samples should equal the original number of measurements squared. However, the order of applying two optical filters does not affect the resulting transmitted luminous flux. Given two filters with transmittance $\tau_A(\lambda)$ and $\tau_B(\lambda)$, the total transmitted luminous flux after passing through both filters is:

$$\Phi_{AB}(\lambda) = \tau_A(\lambda)\tau_B(\lambda)\Phi_0(\lambda) = \tau_B(\lambda)\tau_A(\lambda)\Phi_0(\lambda). \tag{5.4}$$

Thus, $\Phi_{AB}(\lambda) = \Phi_{BA}(\lambda)$, meaning that swapping the order of the filters does not change the outcome. Consequently, including both combinations in the dataset would introduce redundant entries. To avoid this, we only store one of the two possible combinations.



(a) (b)

Figure 5.6: Verification of the augmentation process. Illustration taken from [136] with permission © 2024 IEEE. The measured cascade matches nearly perfectly with the calculated cascade (b). For the CSS (b), some outliers are measurable, but the mean absolute error is still below 3%

To validate the proposed data augmentation method, we physically measure the transmission of two cascaded filter foils and compare the result to the simulated spectra obtained by multiplying the individual transmittance functions, as shown in Fig. 5.6(a). The close alignment between the measured and simulated spectra confirms the validity of (5.3) for the ground-truth spectrometer. We also verify the multiplication for the same cascade with the CSS. In Fig. 5.6(b), the relative error between the measured and multiplied cascade is plotted per channel. Unfortunately, some outliers show a high deviation from the measured cascade. However, these outliers stem from channels that are not spectral active (top left and middle meta-pixel) as they are orientation metapixel. Additionally, the mean error is below 3%, showing the validity of the augmentation approach also for the CSS.

To further assess the applicability of (5.3) within our setup, we evaluate the linearity of the CSS. For this test, we incrementally reduce the light intensity using different neutral-density filters and record the average signal response from both the CSS and ground-truth spectrometer (cf. Fig. 5.7). The ground-truth measurement defines the target intensity on the x-axis, while the CSS response is plotted as the average intensity over all channels, normalized to a reference measurement without a neutral-density filter. Although a slight offset from the bisector is observed, the CSS exhibits a linear response to variations in light intensity. This indicates that (5.3) holds in good approximation for the CSS as well.

5.3 Network Architectures

We include multiple network architectures in our testing to find the most suitable topology for spectral reconstruction. The architectures range from dense networks

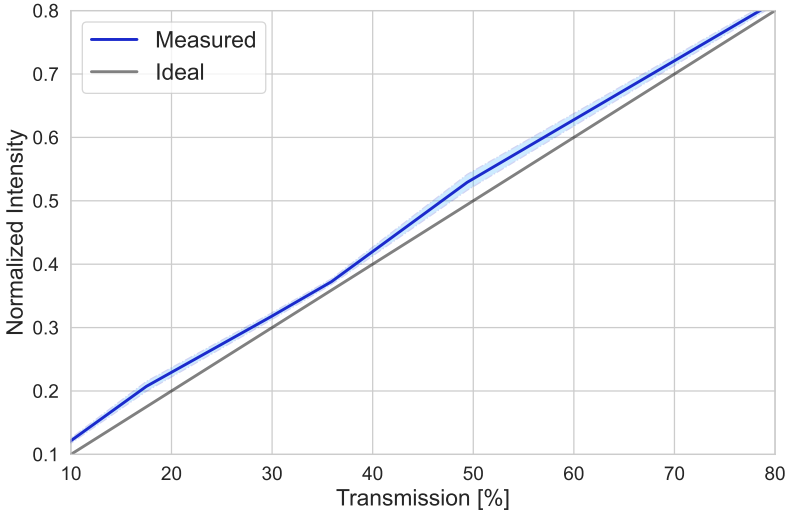


Figure 5.7: Linearity test for the Medusa sensor. Illustration taken from [136] with permission © 2024 IEEE. In the signal range outside of extreme transmission values (below 10% or above 80%), the sensor reacts linearly to luminance changes.

over CNNs and U-Nets to vision transformers. The following section highlights all the used architectures and explains the ideas behind using such topologies.

5.3.1 Dense and CNN

The simplest models in our testing are the dense and CNN networks illustrated in Fig. 5.8. The dense network features five hidden layers with [2560, 1024, 512, 256, 256] neurons respectively and ReLU activations for each layer. The output layer holds 240 neurons to oblige to the ground-truth dimensionality of 240 wavelength steps.

We hypothesize that incorporating convolutional layers may enhance spectral reconstruction by capturing spatial patterns within the sensor data. This idea stems from a loose spatial relationship within a meta pixel ordering the theoretical maximum intensity of each filter in ascending order. To test this assumption, we extend the dense network by introducing multiple convolutional layers before the fully connected layers (cf. Fig. 5.8). The first convolutional layer is configured with a 4×4 kernel and a stride of 4×4, ensuring that it processes the input on a per-meta-pixel basis. Between the convolutional layers, we add batch norm, ReLU activations, and max pooling.

5.3.2 EELSpecNet

In addition to the straightforward convolutional and dense architecture, we incorporate EELSpecNet [118], a network originally designed for denoising in elec-

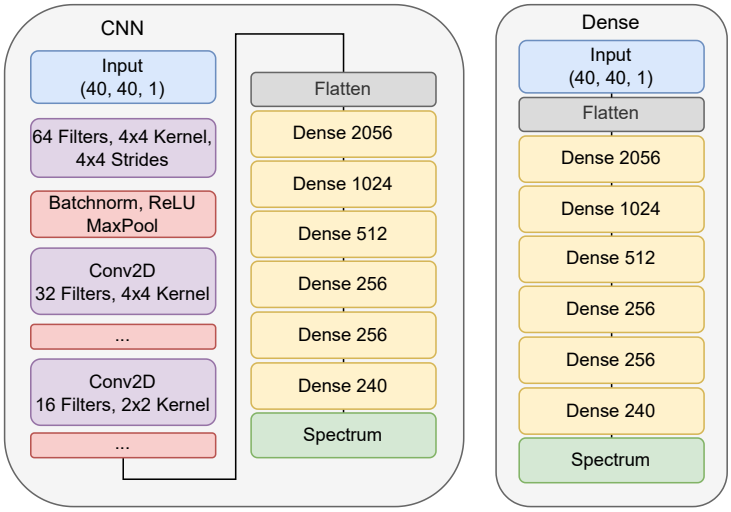


Figure 5.8: Topology of dense and CNN architectures, illustrated after [74]

tron energy loss spectroscopy. EELSpecNet follows a U-Net-like architecture [115], which has demonstrated exceptional performance in spectral reconstruction for the sensor used in [34]. Given its success in the related task, we include EELSpecNet in our benchmark as a U-Net-based model tailored to conventional spectrometers. However, to adapt the architecture to our setup, we flatten the input to match EELSpecNet's expected $n \times 1$ input dimension. Additionally, due to our smaller input size compared to the original network, we reduce the number of layers, limiting the architecture to six convolution and deconvolution layers instead of ten. After the down- and upscaling process, we also add an MLP head to shape the U-Net's output to our output dimension. The exact network topology can be found in Fig. 5.9

5.3.3 Transformer-Based Networks

The Vision Transformer (ViT) has recently gained significant attention as an alternative to CNNs for visual recognition tasks. In many image processing applications, they achieve improved performance over CNNs by leveraging self-attention mechanisms. We briefly highlight their underlying architecture to understand their differences from CNNs and why they might be advantageous for spectral reconstruction.

Google introduced the transformer in 2017 as a neural network model based on the attention mechanism for Natural Language Processing (NLP). Before this, NLP tasks were primarily dominated by encoder-decoder architectures and recurrent neural networks, which process sentences sequentially. However, sequential word dependencies are not always strictly linear, as some words rely more on distant parts of a sentence rather than on adjacent ones. To address this, the authors

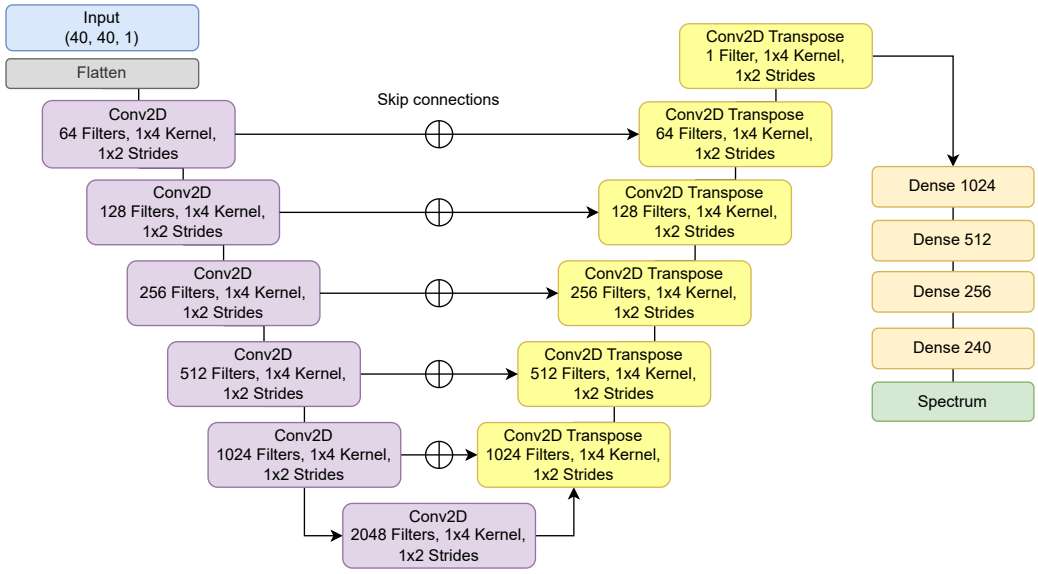


Figure 5.9: Topology of the EELSpecNet adaption, the backbone of the network is illustrated after [118].

introduce self-attention layers, allowing the network to assign different weights to various parts of the input sentence based on their relevance [12].

Building on the success of transformers in NLP, Google adapted the architecture for image processing, leading to the development of the ViT [5]. Similar to how transformers process text, ViTs split an image into smaller patches and compute relationships between them (cf. Fig. 5.10(a)). We apply the ViT as the first transformer-based architecture for spectral reconstruction. The patch size is set equal to the meta-pixel size, which we expect to be beneficial for reconstruction as it enables the network to identify relationships between meta-pixels.

While CNNs can also leverage the spatial structure of the data, ViTs and CNNs differ in several fundamental aspects: (i) CNNs are more sensitive to spatial variations, whereas transformers capture both local and global dependencies more effectively. (ii) CNNs excel at processing grid-structured data due to the convolution operation, whereas ViTs are optimized for sequential data by converting images into a sequence of non-overlapping patches. (iii) ViTs generally have larger model sizes and higher memory requirements compared to CNNs. (iv) ViTs typically require larger training datasets and more epochs to achieve optimal performance. [14]; [60]; [96].

In addition to standard ViT, hybrid models combining CNNs and transformers have been proposed to balance the strengths of both architectures. Such models aim to retain the spatial inductive biases of CNNs while benefiting from the global attention mechanisms of transformers. These hybrid approaches may of-

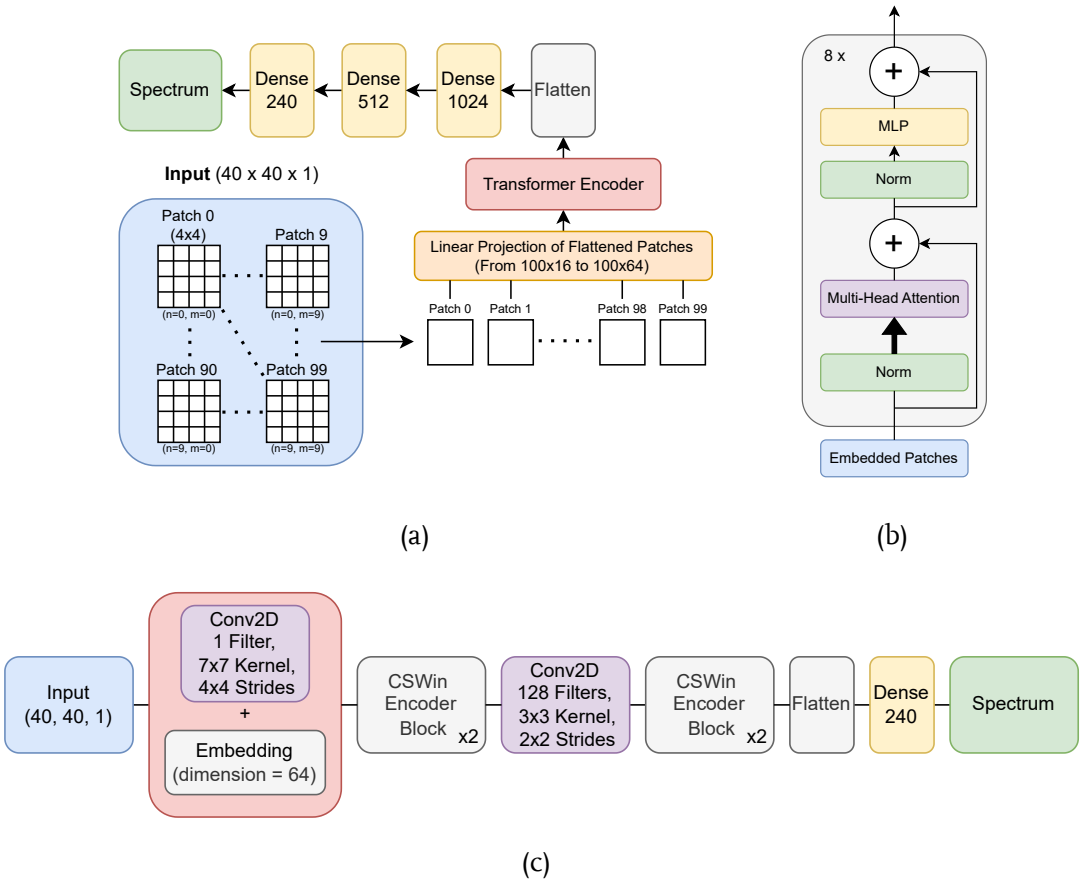


Figure 5.10: Overview of transformer-based architectures. Illustrations taken from [74] with permission. The ViT architecture based on [5] including patching is shown in (a), while the inner encoder structure is illustrated in (b). We repeat the inner encoder structure eight times before feeding the output into an MLP head to adapt the architecture to our application. The Cross-Shaped Window Transformer (CSWin) architecture based on [137] is shown in (c), which uses the same encoder block as the ViT except for the usage of cross-shaped window self-attention instead of traditional multi-head attention.

fer a promising alternative for spectral reconstruction, particularly in scenarios with limited data availability. Therefore, we also test the Cross-Shaped Window Transformer (CSWin) [137] as depicted in Fig. 5.10(c). Compared to the ViT, the most prominent differentiating factor is the strip-shaped window used to perform the attention mechanism. There are two main types of strips: horizontally striped and vertically striped windows. We alternate between both types to include vertical and horizontal relations in our input frame. Apart from that, the encoder block is implemented as depicted in 5.10(b). The convolutional part of this hybrid architecture is added after the first two encoder blocks with a singular 2D convolutional layer

5.4 Benchmark

In a broad benchmark, we test the applicability of the aforementioned NNs to spectral reconstruction. Please note that due to the different sensor generations, extended time periods lie between the different tests highlighted in this section. We start with a proof-of-concept using the Infimedar sensor without augmentation. This benchmark sets the foundation of every other test we conduct by showing the applicability of one-shot learning to our CSS. In addition to the optimal case, we also test the stability of each network against deviations introduced by noisy sensor data. Afterward, we conduct the following tests using the Medusa Sensor. Here, we re-evaluate the performance benchmark from before with the new sensor and also show the performance gains obtained by using the augmentation mentioned in (5.3).

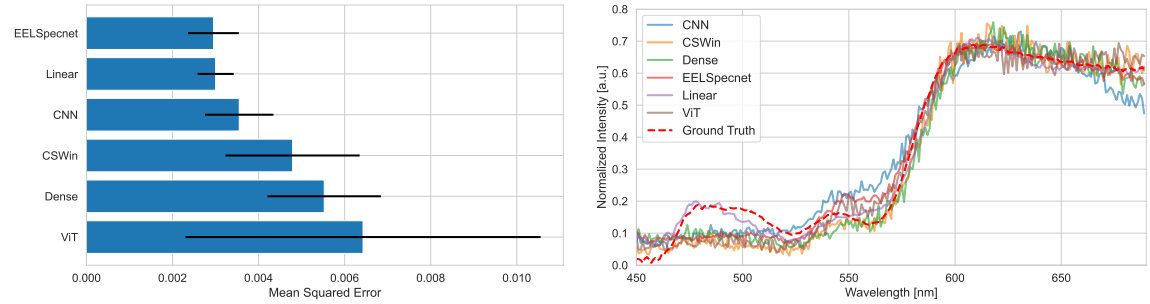
All networks are trained with consistent parameters in TensorFlow 2.15: Adam optimizer, MSE loss, and Early Stopping. To ensure comparability, we use 5-fold cross-validation with a fixed seed (42) and a 60/20/20 train, validation, and test split across all networks. The models' performance is evaluated using MSE as the reconstruction error, where smaller values indicate better performance. In the plots, error bars and shaded areas represent the standard deviations across the five folds, measuring the models' reconstruction capabilities across the entire dataset.

We add Gaussian noise to the CSS values for the stability test. This scenario assesses the models' robustness to sensor data deviations. The added noise has zero mean, with a gradually increasing standard deviation ranging from 0.0 to 0.2 in each run. Due to the normalization process, the standard deviation is expressed as a percentage of the expected signal. For clarity, a standard deviation of 0.08 will be referred to as an 8% noise level.

Given the suboptimal filter characteristics, classic methods, even with additional regularization, could not reconstruct the spectrum. As a result, comparisons with these methods are omitted in the following analysis. However, the performance of the neural networks is compared to that of a simple linear model, fitted using Linear Regression from the scikit-learn package in Python, serving as the baseline.

5.4.1 Proof-of-Concept

All models produce qualitatively usable spectral reconstructions from the sensor data. Figure 5.11(b) shows an example reconstruction for each approach. It is evident that all models generally follow the target spectrum's shape. However, only the linear model accurately reconstructs the spectral feature between 470 and 520 nm. This trend is also reflected in the benchmark results presented in Fig. 5.11(a). The linear model and EELSpecNet achieve the best performance, with the linear model exhibiting slightly lower standard deviation. The CNN ranks



(a) (b)

Figure 5.11: Benchmark for the Infimedar sensor without added noise. The left plot (a) shows the MSE over all folds, with their standard deviation as error bars. On the right (b) one exemplary reconstruction result is plotted with ground truth indicated as a dashed red line.

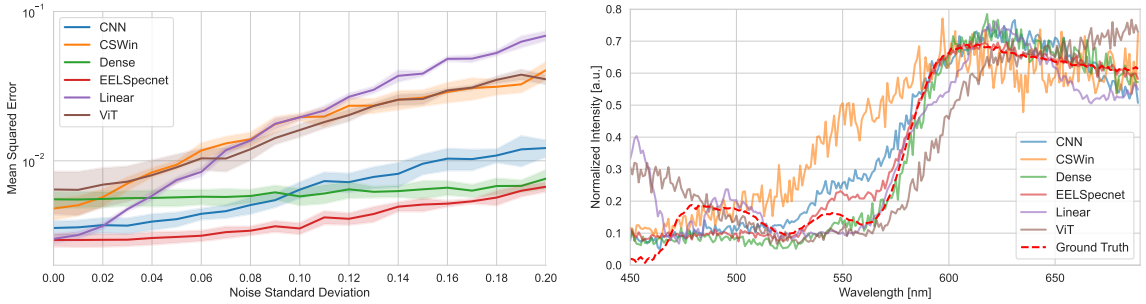
third in terms of mean squared error, while CSWin and Dense follow in fourth and fifth place, with CSWin demonstrating a marginal advantage. Surprisingly, the ViT ranks last, showing both higher average error and greater standard deviation compared to the other models.

As discussed earlier, the ViT's suboptimal performance is likely due to the limited dataset size. Either the data does not provide sufficient diversity to establish useful relationships between meta-pixels, or these relationships contribute less to reconstruction than initially assumed. The high standard deviation further underscores the data dependency of the problem. While local information within a meta-pixel can be crucial for certain target spectra, for others, a global perspective of the entire sensor array is more important. Consequently, reconstruction performance varies significantly across different data folds. This behavior is most pronounced in transformer-based models, which primarily process global dependencies between patches.

In theory, CSWin should bridge the gap between local and global information processing by integrating convolutional and attention mechanisms. However, the limited dataset remains a bottleneck, preventing it from fully leveraging this capability. Conversely, EELSpecNet benefits from the U-Net architecture's intrinsic data efficiency. The downsampling and upsampling operations within the network act as an implicit form of data augmentation, mitigating the negative effects of data scarcity. This characteristic was already observed in the original U-Net study [115], where the model achieved strong results despite being trained on only 30 samples.

Added Sensor Noise

Adding noise significantly degrades the reconstruction quality (cf. Fig. 5.12(b)). The linear model deviates drastically from the ground truth, displaying an almost



(a) (b)

Figure 5.12: Benchmark for the Infimedar with added noise. The left plot (a) shows the MSE over all folds depending on the noise level, with their standard deviation as shaded areas. On the right (b) one exemplary reconstruction result is plotted with added noise ($\sigma = 0.08$). The ground truth is indicated as a dashed red line.

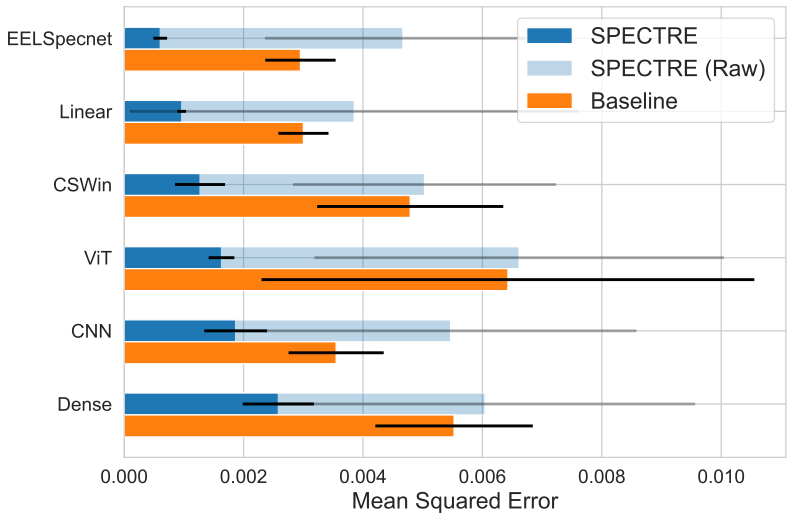
oscillatory behavior. The transformer-based models also struggle, introducing noticeable deviations. The reconstruction errors in Fig. 5.12(a) confirm this trend. While the linear model remains highly sensitive to data quality, the transformer-based networks also show increasing errors as noise levels rise. Interestingly, the dense network maintains a stable performance across different noise levels, surpassing the CNN when data deviations increase.

EELSpecNet outperforms all other models in this test. Although its reconstruction error increases slightly with added noise, its mean squared error stays consistently lower than any other approach. This strong robustness stems from its U-Net architecture. Originally designed for denoising electron energy loss spectroscopy data, U-Nets excel at filtering out noise while preserving essential features. This ability allows EELSpecNet to handle noisy input data effectively, which is clearly visible in the results.

In contrast, the linear model struggles with noise, likely due to the regularization applied during fitting. The model appears not to generalize well, leading to a sharp decline in performance when input data deviates.

5.4.2 Improvements with Augmentation

We evaluate the reconstruction a second time with the new sensor generation, Medusa. In addition to re-evaluation we also want to handle the data sparsity with this generation by including the physics-informed augmentation technique introduced in Sec. 5.2.3. For this test we benchmark the new sensor using the same models as before with and without augmentation. As a baseline, we compare the reconstruction performance using the Infimedar with the novel approach and sensor.



(a)

Figure 5.13: Benchmark for the Medusa sensor with augmentation. The plot shows the MSE over all folds, with the error bars depicting the standard deviation. The shaded area depicts the result without augmentation and the orange bars the performance from 5.11(a).

We refrain from testing the robustness against noise, as the model architectures stay the same, and this section focuses on the augmentation technique.

Without augmentation, the networks perform on par with or even worse than the previous generation. The high standard deviation highlights the strong dependency on the dataset, indicating unstable generalization. These results are expected, as we had to remove some well-functioning filter structures when designing Medusa. The linear model achieves the best results in this setting, outperforming all other architectures.

However, augmentation significantly improves reconstruction performance compared to both Infimedar and the non-augmented case. This improvement is evident in the lower mean reconstruction errors across all folds and the reduced standard deviations. As before, EELSpecNet delivers the best performance, now achieving a significant lead over the linear model. The transformer-based architectures also benefit from augmentation, with CSWin and ViT ranking third and fourth, respectively. Notably, their reconstruction performance improves by a factor of 4.08 compared to their previous results.

The simpler neural network architectures, CNN and Dense, place fifth and sixth, yet both outperform the best-performing model from last generation. The substantial dataset expansion from augmentation enables more complex models, such as CSWin and ViT, to train more effectively for spectral reconstruction. Their improved results demonstrate how larger datasets help transformer-based architec-

tures leverage their strengths. Nonetheless, also the more data-efficient approaches benefit from the increased dataset size further increasing their lead in this test.

CHAPTER 6

Efficiency Improvements and Final System

In the previous chapter, we demonstrated that NNs can successfully reconstruct spectra from our sensors in a one-shot learning manner. However, many of the networks introduced in Sec. 5.3 come with high memory and computational demands, making them impractical for a cost-effective solution. The CSS is intended for applications where traditional spectrometers are not viable due to their high cost. As a result, the accompanying neural network must run on hardware that remains within a reasonable price range, excluding solutions that require computing hardware costing tens or even hundreds of euros.

To address this constraint, we reduce the computational requirements of the reconstruction process so that it can be deployed on a microcontroller integrated into the sensor's PCB in the final iteration of the CSS. Specifically, we select the STM32F423 [122] for its balance between price, energy efficiency, and computational capabilities. Additionally, the X-Cube AI library from STMicroelectronics provides a well-established and accessible framework for deploying NNs on supported microcontrollers. However, the limited flash memory of 1.5MB imposes a strict upper bound on the model's size, necessitating careful optimization. It should be noted that our limit on the pure model size is even higher, as the flash memory also needs to hold the firmware code. Therefore, our models must not exceed 1MB in size.

This chapter details the efficiency improvements applied to achieve a deployable model for the final system, which is paired with the latest generation of CSS, NanoSpectral. We begin by discussing the filter selection process, which optimizes the sensor's filter array with spectral reconstruction in mind. We then leverage the scalability of select network architectures from the previous chapter, as outlined in Sec. 2.1, to reduce their size while preserving performance. Finally, we apply quantization techniques and deploy the models to the microcontroller, benchmarking their reconstruction accuracy in the final hardware setup.

6.1 Filter Selection

Both the Medusa and Infimedar sensors feature a total of 1600 channels arranged in a 40×40 array. While some of these channels are not spectrally active—such as the 100 reference channels, the three orientation detection metapixels, and the polarization measurement channels—the number of channels contributing

to spectral reconstruction still exceeds 1000. This suggests a high degree of redundancy in the data, likely including non-optimal filters that should be pruned. Therefore, selecting the most relevant filters is essential to designing an efficient, hardware/software co-optimized final iteration of the CSS.

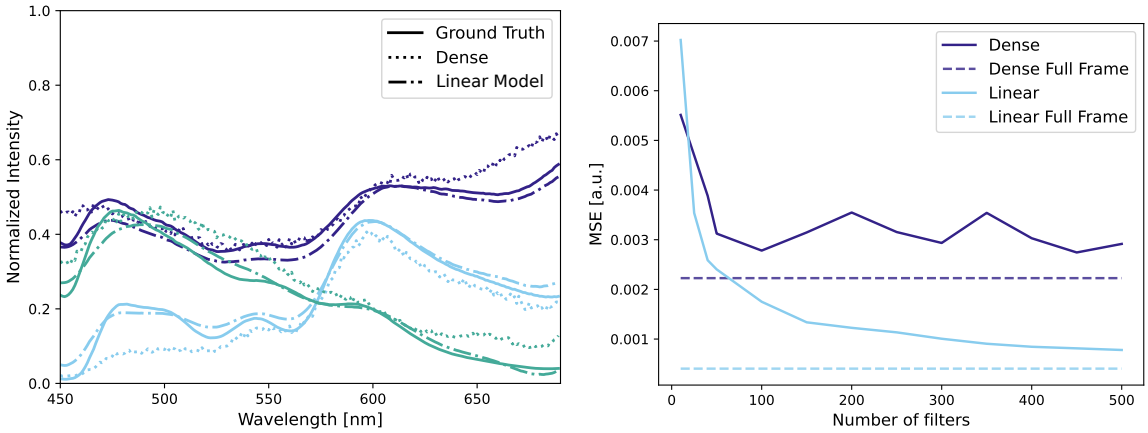
To achieve this, we employ LRP as described in Sec. 2.4. We assess each input feature's relevance when predicting the transmission spectrum of the transparent foil. This ensures that the measured light covers the full wavelength range of interest, forcing the network to rely on multiple filters from different spectral regions simultaneously. We conduct this analysis using the dense model for several reasons: First, as shown in Sec. 5.4.1, it demonstrates the highest robustness to noise, suggesting that it effectively integrates information from various filters for reconstruction. Second, the model operates on flattened input data, which is crucial for selecting relevant filters from the sensor's 2D array. Since this selection process discards spatial relationships between pixels, CNN- and ViT-based models would require significant modifications to remain applicable. Additionally, EELSpecNet incorporates down- and upsampling operations, requiring a different network architecture for each reduced filter set. This prevents a straightforward application of LRP to that model. Consequently, we limit our relevance analysis to the dense network.

It is important to note that the results presented in this section were obtained before introducing the augmentation process. As a result, they should be compared to the benchmarks in Sec. 5.4.1 rather than the results in Sec. 5.4.2. Since the selected filters have already been implemented in hardware, repeating the analysis with augmented data would not provide additional valuable insights. Therefore, we refrain from rerunning these experiments.

Following the recommendations in [83], we apply the LRP_ϵ rule, which is specifically suited for dense networks. To obtain input relevance values across the entire prediction range, we average the relevance scores over all output neurons, resulting in a final relevance matrix of size 40×40. From this matrix, we select between 10 and 500 filters based on the highest absolute relevance scores and retrain the network using only the reduced set. As a baseline, we also refit the linear model using the same optimized filter selection and compare both approaches (cf. Fig. 6.1(b)).

The results indicate that both the dense model and the linear model gradually regain their original reconstruction performance as the number of selected filters increases. Interestingly, the linear model remains capable of reconstructing spectra even though the input filters were selected based on the dense model's relevance analysis. This suggests that the neural network-based model captures a generalizable filter selection strategy, supporting the applicability of the method.

Another notable observation is the slightly irregular convergence behavior of the dense model. When more than 100 filters are selected, its MSE shows minor degradation. This could be attributed to the inherent randomness in NN training.



(a) (b)

Figure 6.1: Filter pruning results with LRP. The left plot (a) shows the benchmark results for different number of input filters with the reconstruction performances with all filter indicated as dashed lines. In the right plot (b) exemplary predictions using 100 filters for the dense and linear model are plotted. Different colors indicate different samples, with solid lines representing the ground truth, dotted lines the dense model, and dash-dotted lines the linear model.

Throughout all experiments, we maintain a fixed random seed and early stopping patience, which may result in convergence to local minima when training with a reduced filter set. The network may not escape these local minima if the additional filters beyond 100 do not provide sufficiently new information. The already near-optimal performance at 100 filters, compared to the original MSE with 1600 filters, further supports this hypothesis.

The exemplary predictions in Fig. 6.1(a) confirm the conclusions drawn from the numerical results. Both models are capable of reconstructing highly usable spectra with only 100 selected input features, representing a 94% reduction in feature space. While the linear model continues to achieve the best overall reconstruction performance, the dense model still delivers competitive results.

While the results suggest that a filter matrix with only 100 channels would be sufficient for spectral reconstruction, the final sensor design for NanoSpectral incorporates a larger number of filters. This decision is based on two main factors: First, in addition to our LRP-based analysis, a collaborating research group conducted an independent evaluation of the filter array using alternative methods. Since their findings remain unpublished and were not part of our testing, we cannot present their results here. However, the final filter selection for NanoSpectral integrates insights from both analyses, with duplicate filters naturally removed. Second, beyond spectral reconstruction in the visible range, NanoSpectral is designed to support additional applications. The sensor includes dedicated channels for infrared wavelengths up to 1000 nm, as well as specialized filters for crop analysis and polarization measurements. Considering these extended requirements,

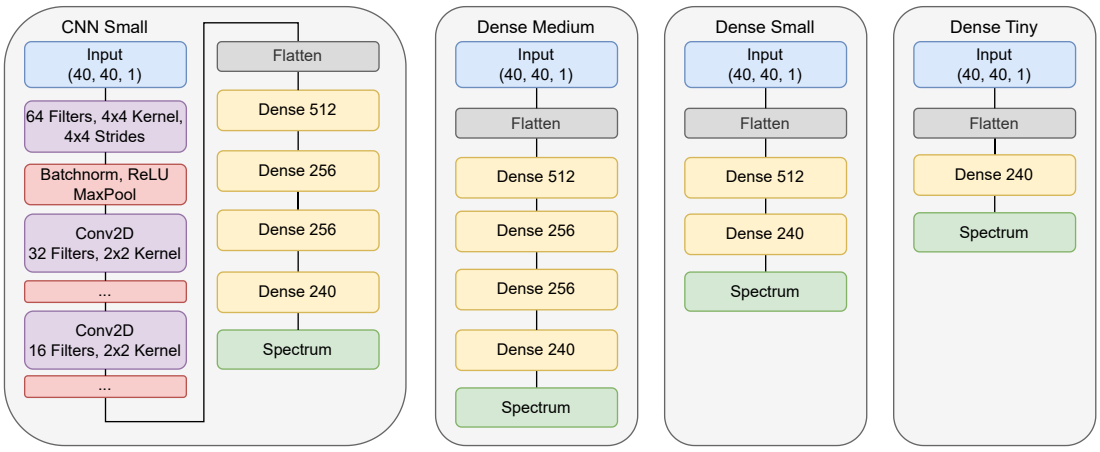


Figure 6.2: Architectures of scaled-down networks. The CNN architecture is similiar to the one from 5.8, but with less dense layers and neurons. The medium dense network matches the MLP head of the small CNN. The other dense architectures (small and tiny) are scaled-down even further to have minimal memory requirements.

the final filter array consists of 20×16 channels, totaling 320 filters. This still represents a substantial reduction from the original 1600 channels of Medusa and Infimedar, demonstrating the efficiency improvements achieved with this latest sensor generation.

6.2 Downscaling the Spectral Reconstruction

With the downscaled and optimized sensor, we further improve the efficiency of the final system by optimizing the models used for spectral reconstruction. Due to the smaller feature size, we expect smaller models to still be able to produce adequate results. Therefore, we use the scalability of the dense and CNN models to create smaller variations of the original topologies (cf. Fig. 6.2). We mostly downscale the dense layers, leaving the convolutional backbone in the small CNN version intact. In addition to the scaled-down CNN, we include three variations of purely dense networks. The biggest one, Dense Medium, features the same number of neurons and layers as the MLP head in the small CNN network, while the small dense topology only features one hidden layer with 512, and the tiny dense network has no hidden layer.

Unfortunately, the transformer-based models do not scale well, as discussed in Sec. 2.1, so that we were not able to find a topology that meets the computational requirements mentioned above. Also, EELSpecNet features Conv2DTranspose layers, which are not supported by XCube AI in its current state, hindering us from finding a deployable U-Net.

	After optimiztaion	Before optimization
Dense Tiny	82 KB	945 KB
Dense Small	301 KB	3.5 MB
CNN Small	346 KB	3.9 MB
Dense Medium	445 KB	5.1 MB
CNN	3.7 MB	44.1 MB
EELSpecNet	26.3 MB	314.4 MB

Table 6.1: Model memory usage after and before optimization.

For the deployment process, we optimize all trained models for size by applying dynamic range quantization. This process has virtually no impact on the model accuracy as it only quantizes the weights to int8 precision without quantizing the activations and outputs of the model [52]. In addition to the accuracy advantage, dynamic range quantization does not need a calibration dataset, making the deployment process more straightforward. The resulting model sizes can be found in Tab. 6.1. Interestingly, only dense tiny would fit our memory requirement without the optimization process. This shows the high impact of quantization on the deployability of networks in memory-constrained environments. Nonetheless, all scaled-down models are deployable when applying dynamic range quantization. For comparability, we also include the original CNN model size before scaling the architecture down, as well as EELSpecNet representing the best-performing model from the testing with the Medusa sensor. Both models would not fit into the flash memory of our microcontroller even with dynamic range quantization, which also shows the importance of scalable architectures in the realm of TinyML. It should be noted that further optimization or compression techniques such as weight sharing or full int8 quantization (cf. Chap. 2) could potentially enable even bigger models to be deployable in our case. However, as we will demonstrate in the next section, even the scaled down topologies can show similar or even improved reconstruction performance compared to bigger models. Therefore, we do not include further optimization steps that would potentially impact the reconstruction performance.

6.3 Benchmark results for complete System

For the final benchmark, we adjust our testing approach slightly to improve generalization and comparability to the approach shown in [86]. Similar to Laubmann, Jonathan and Saloman, Stefan and Wissing, Julio and Tschekalinskij, Wladimir and Hettenkofer, Sebastian and Stefani, Alessio and Scholz, Teresa, we perform a cluster-based split instead of five-fold cross-validation by splitting the ground-truth spectra into 20 clusters using k-means. We then select two heterogenous clusters to form the test set. The remaining clusters form the train set, to which we also apply data augmentation. This process leads to a harder training process as not

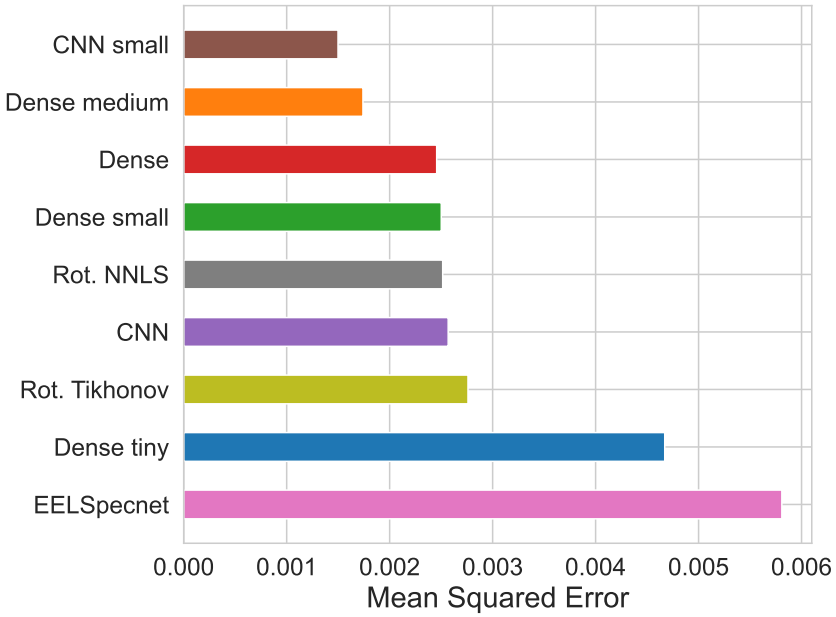


Figure 6.3: Benchmark results with NanoSpectral using the MSE. The colors of each bar matches with the colors used for the models in Fig. 6.4.

only unseen but also completely dissimilar samples are to be reconstructed with the test set. Therefore, we can benchmark for the best generalization capabilities and further assess the susceptibility to data deviations. In addition, we keep our testing method as close as possible to the hybrid learning approach from [86], which we also use as a baseline to compare our models against. Namely, we test the rotation correction with Tikhonov regularization (Rot. Tikhonov) and the rotation correction with standard NNLS (Rot. NNLS).

To further improve the training process, we train with a combined loss function

$$\mathcal{L}(\mathbf{s}, \hat{\mathbf{s}}) = l_1 \sum_{i=1}^{M} \frac{s_i - \hat{s}_i}{M} + l_2 \left(\frac{-\sum_{i=1}^{M} s_i \cdot \hat{s}_i}{\sqrt{\sum_{i=1}^{M} (s_i)^2} \cdot \sqrt{\sum_{i=1}^{M} (\hat{s}_i)^2}} + 1 \right). \tag{6.1}$$

In this formulation, we include the cosine similarity in addition to the MSE. Therefore, we optimize not only for the reconstruction error but also for the similarity between the ground truth and the reconstruction. We slightly alter the standard formulation of the cosine similarity to converge towards zero to stay in the same realm as the MSE. Using the two factors l_1 and l_2, we weigh the two components to be able to emphasize one metric stronger than the other. In our testing, $l_1 = 0.6$ and $l_2 = 0.4$ have proven to be an adequate tradeoff with emphasis on the reconstruction error.

Fig. 6.3 shows the benchmark results using the MSE of each model on the isolated test set. Nonetheless, we also include further metrics, namely the cosinus similarity

	MSE	CSIM	PCC
CNN small	0.00150	0.99896	0.98426
Dense medium	0.00174	0.99901	0.97798
Dense	0.00246	0.99816	0.96547
Dense small	0.00250	0.99803	0.96647
CNN	0.00257	0.99851	0.96981
Dense tiny	0.00467	0.99699	0.95443
EELSpecnet	0.00581	0.99893	0.97773

Table 6.2: All metrics for the NanoSpectral benchmark. The table is sorted by the MSE

and pearson correlation coefficient, which are shown in Tab. 6.2. However, as the metrics are highly correlated, we will focus on the MSE for the following analysis.

All tested models demonstrate strong reconstruction performance despite the increased difficulty of the new evaluation approach. Interestingly, larger models, such as EELSpecNet and the large CNN, perform slightly worse than their scaled-down counterparts. In particular, the strong performance of the small CNN highlights that a larger model footprint does not necessarily improve reconstruction quality. Since the task does not require a more complex parameter space, smaller networks that are easier to optimize have a distinct advantage.

Surprisingly, the trivial tiny dense network outperforms EELSpecNet in our tests. However, given the minimal performance differences across all models, this lead may result from the inherent non-determinism in neural network training.

Overall, these results illustrate how optimizing sensor design in combination with software improvements enables a highly efficient system. The reduced feature space of the optimized filter array allows even compact architectures, such as the tiny dense network to achieve effective spectral reconstruction. Additionally, the significantly lower memory footprint makes it possible to deploy these models on a microcontroller with only 1.5 MB of flash memory, demonstrating that NanoSpectral enables a truly low-cost spectral reconstruction system. With the introduction of noise, the ranking of the top-performing networks shifts (cf. Fig. 6.4). As observed previously, dense networks exhibit greater robustness against sensor noise, whereas CNN-based architectures rapidly accumulate errors. While the small CNN leads under optimal conditions, even a minor noise level of 2% causes its performance to drop, placing it second to last. Larger models, such as EELSpecNet and the original dense architecture, demonstrate the highest resilience to noise. This outcome aligns with the previous findings in Sec. 5.4.1, where these architectures also exhibit superior robustness. The fully connected dense network benefits from a global receptive field, allowing it to account for noise more effectively than other architectures. Similarly, EELSpecNet, designed initially to process noisy spectral data, maintains a distinct advantage in this evaluation.

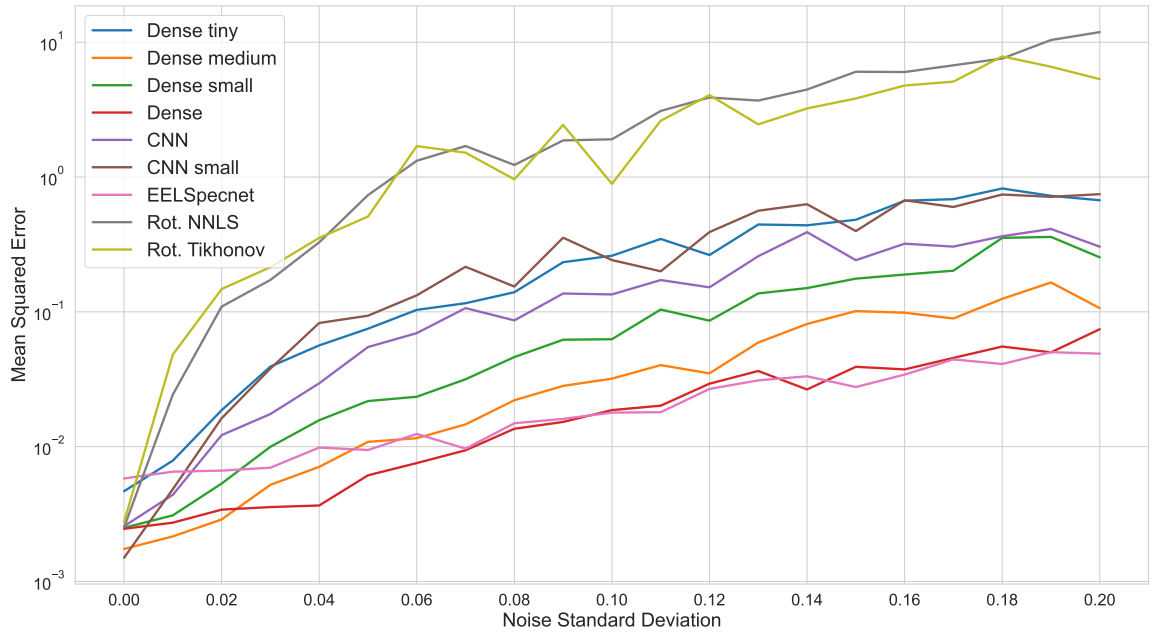


Figure 6.4: Benchmark results for NanoSpectral with added noise. The resulting reconstruction error is plotted against the standard deviation of the added noise.

Interestingly, the medium-dense network performs nearly the same as the larger models. While it has fewer parameters than the fully-fledged dense architecture, it still processes the entire input frame at once and retains enough capacity to model sensor noise effectively. The small-dense model follows this trend, performing slightly worse than the medium-dense variant in terms of noise robustness. In contrast, the tiny dense model struggles to handle noisy sensor data due to its highly constrained architecture, preventing it from effectively compensating for deviations in the input.

The qualitative results in Fig. 6.5 support the findings observed in the quantitative evaluation. Here, we reconstruct selected test samples using the five best-performing models. Notably, the medium dense network produces reconstructions that qualitatively outperform the small CNN, as its outputs more closely resemble the ground truth and maintain a more consistent spectral shape. This observation aligns with the quantitative results in Tab. 6.2, where the medium-dense model ranks first in cosine similarity despite a slightly different ordering in terms of MSE.

The performance gap between the medium dense network and the small CNN becomes particularly evident in the last example, where the small CNN exhibits significant deviations in the wavelength range below 580nm. Conversely, in the first two examples, the small dense network demonstrates strong qualitative performance, closely matching the ground truth in the challenging blue wavelength range.

The hybrid learning approach, Rot. NNLS, shows only moderate performance in this qualitative comparison. While it reconstructs the second example reasonably well, its performance in the blue wavelength range below 500nm remains suboptimal, leading to noticeable discrepancies. These results further emphasize the advantages of one-shot learning in achieving both accurate and consistent spectral reconstructions.

The NanoSpectral benchmark demonstrates that the optimized filter array allows smaller models to reconstruct accurate spectra, even outperforming larger models under ideal conditions. However, when dealing with non-optimal inputs, the larger architectures maintain an advantage due to their greater parameter capacity and more sophisticated structure.

Despite this, the medium dense model exhibits strong stability against noise, ranking just behind the large dense network and EELSpecNet. The qualitative results also indicate an overall better similarity between the ground truth and the reconstruction from the medium dense net. Given its balance between performance and efficiency, we select the medium dense model as the preferred candidate for deployment on NanoSpectral. It achieves results nearly on par with the small CNN under optimal conditions while maintaining sufficient robustness to noise and requiring only 445 KB of memory, making it well-suited for deployment.

6.4 Conclusion

In the last two chapters, we have demonstrated the feasibility and advantages of using neural networks for spectral reconstruction in computational spectrometers. Through a comprehensive analysis of various model architectures, we have shown that deep learning-based approaches outperform traditional reconstruction methods, particularly when dealing with the non-ideal characteristics of low-cost CSSs.

A major contribution is the introduction of a physics-informed data augmentation technique, which significantly increases the dataset size from 214 to over 10,000 samples. This augmentation method enables better generalization and stability without requiring additional physical measurements, a crucial improvement given the data scarcity typically found in spectral sensing applications. Experimental validation confirms that the synthesized spectra closely match real-world measurements, further supporting the effectiveness of this approach.

Furthermore, extensive benchmarking of neural network architectures has shown that while larger models achieve higher accuracy under optimal conditions, well-optimized compact networks can match or even surpass them when computational constraints are taken into account. The medium dense network, in particular,

emerges as the most balanced architecture, offering a strong trade-off between accuracy, noise robustness, and deployability.

Beyond algorithmic improvements, we have also explored hardware-aware optimizations. The application of layer-wise LRP provides insights into the importance of individual sensor channels. This analysis enables a theoretical reduction of 94% input dimensions while maintaining high reconstruction accuracy. Combined with further refinements to the filter array, these results directly influence the hardware design of the latest sensor generation, NanoSpectral.

Utilizing the novel sensor hardware, we have successfully integrated TinyML methodologies to refine and optimize spectral reconstruction models for deployment on embedded hardware. By systematically compressing and fine-tuning the neural networks, we have achieved a spectral reconstruction system that is both computationally efficient and suitable for real-world applications. The final system demonstrates that cost-effective computational spectrometers can achieve high reconstruction accuracy while operating on resource-constrained embedded platforms.

Together, these advancements contribute to the broader field of computational spectroscopy, opening new possibilities for low-cost, real-time spectral sensing in applications such as smart farming, environmental monitoring, and industrial quality control.

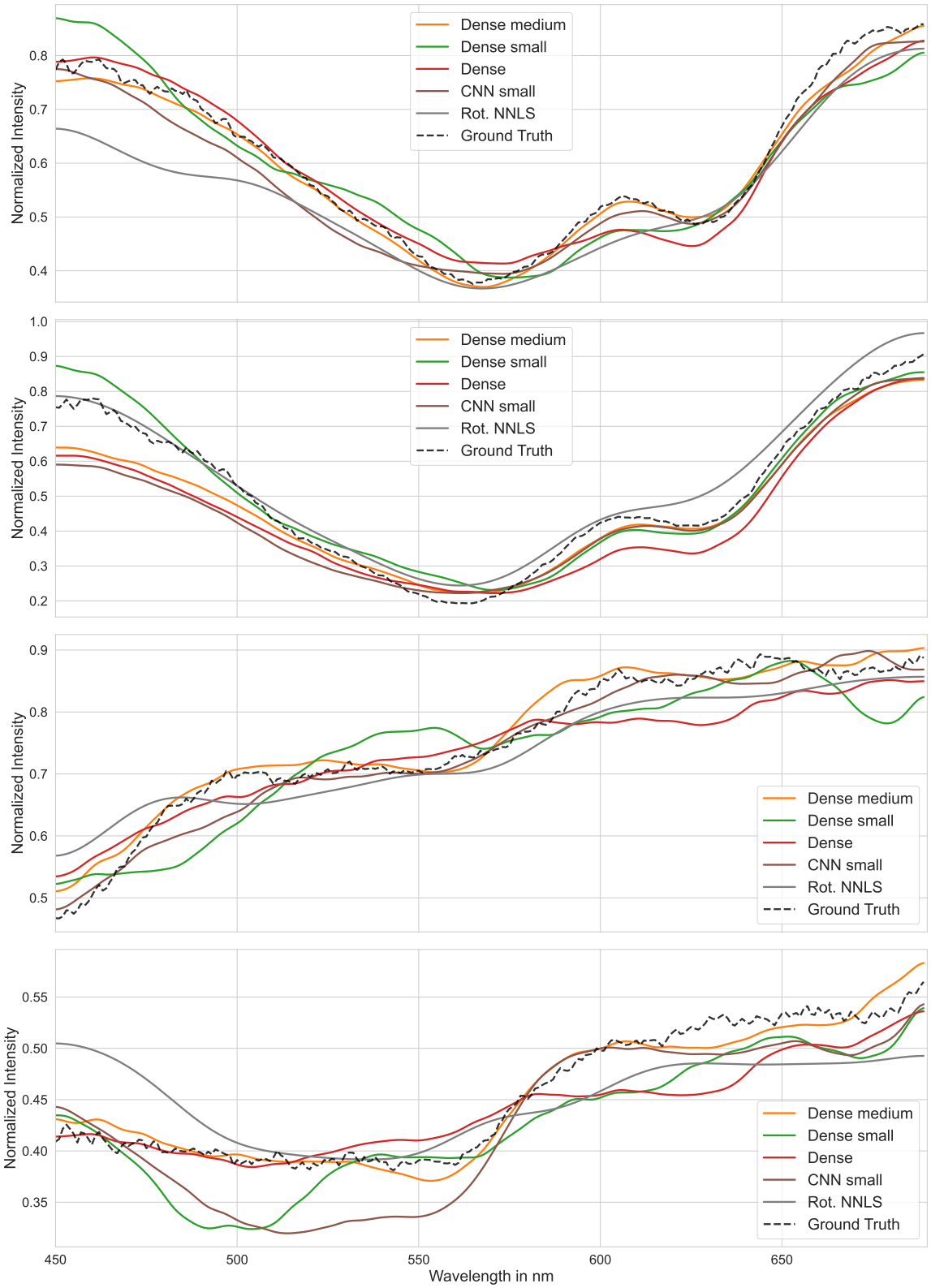


Figure 6.5: Example reconstructions for NanoSpectral. We only plot the top-performing 5 models to improve visibility. The ground truth is marked as dashed black line, while the reconstructions are plotted with the colors from the benchmark results in a solid line.

CHAPTER 7

Conclusion and Outlook

This thesis explores two research directions in the field of efficient machine learning for embedded systems. The first part focuses on efficient edge processing and hierarchical machine learning (HiML) for resource-constrained devices, while the second part investigates spectral reconstruction using neural networks for embedded deployment. Although both parts share the common goal of enabling efficient processing at the edge, they address two distinct challenges and contribute to separate domains of embedded AI. Therefore, we summarize the two contributions separately.

7.1 Hierarchical Machine Learning for Energy-Efficient Classification

The first part of this thesis investigates the potential of HiML to improve the energy efficiency of classification tasks on embedded platforms. Traditional flat classifiers process all inputs with the same computational effort, regardless of whether an early classification decision is possible. HiML overcomes this limitation by structuring the classification task as a hierarchy of decisions, allowing for early exits when a classification can be made confidently.

The results show that this approach significantly reduces energy consumption while maintaining competitive accuracy. Compared to conventional flat classifiers, our HiML-based models reduce energy usage by up to 47.63%. This efficiency gain is particularly relevant for edge computing applications, where power constraints are a key limiting factor. The hierarchical structure enables a more adaptive processing pipeline, ensuring that computational resources are allocated efficiently.

Additionally, this thesis explores RL as an optimization strategy for classifier selection within the hierarchy. By formulating classifier selection as an automated search task, RL effectively balances accuracy and energy consumption while significantly reducing the computational effort required for optimization. Unlike exhaustive search methods, which require extensive computational resources, the RL-based approach converges within hours on a single machine. This result highlights the feasibility of adaptive, energy-efficient classification models that can be dynamically optimized for different deployment scenarios.

7.1.1 Outlook

In a world with more and more AI systems finding their way into everyday tasks, simple flat models do not fit an ever-changing multi-modal environment. We think hierarchical systems will enable more complex and personalized topologies without the extensive energy footprint of recent AI models. Nonetheless, in its current state, HiML needs extensive domain knowledge and manual tuning, even with introducing our RL-based model selection. Therefore, future research should focus on combining manual hierarchical taxonomies with learned early-exit NNs, bringing personalization and cutting-edge efficiency together. Additionally, more sophisticated optimization techniques should be explored to cover bigger search spaces than the one we used. Some methods discussed in the NAS Section, such as differentiable NAS or Zero-Cost NAS, might be transferable early-exit NNs. In addition manual hierarchical taxonomies can also be optimized using approaches like particle swarm optimization or evolutionary algorithms.

7.2 Machine Learning for Spectral Reconstruction

The second part of this thesis focuses on spectral reconstruction, where neural networks are used to recover a full spectrum of visible light from an optical filter array. The challenge in this task lies in the complex sensor output stemming from a low-cost implementation of the filtering structures, which makes it necessary to infer missing spectral information from misleading learned patterns. Due to the instability of the sensor data, traditional methods fail to reconstruct a usable spectrum, making learning-based approaches necessary. However, since NNs often come with substantial computational efforts, TinyML approaches are needed to make them deployable to low-cost hardware.

We first conduct a proof-of-concept, showing that even with our complicated filter characteristics, NNs can reconstruct a usable spectrum where traditional methods fail. Nonetheless, many open points remain after the proof of concept such as data availability and model efficiency.

A key contribution of this work is the introduction of a physics-informed data augmentation method based on the properties of optical transmission spectra. Given the time-intensive nature of collecting real-world training data, we derive a method to generate additional training samples by simulating cascaded optical filters. This augmentation method increases the dataset size from 214 to over 10,000 samples, significantly improving model generalization without requiring additional physical measurements. Experimental validation confirms that the generated samples closely match real-world measurements, demonstrating the physical correctness of the approach.

To further improve efficiency, we apply layer-wise relevance propagation (LRP) to identify the most important spectral filters. This technique enables a 94% reduction in input features while preserving high reconstruction accuracy. The results indicate that not all sensor channels contribute equally to spectral reconstruction, and many can be removed without vastly impacting performance. This insight directly influences the hardware design of our most recent spectral sensor, NanoSpectral.

Beyond data augmentation, this thesis also explores how neural networks can be optimized for spectral reconstruction. Extensive benchmarking of different network architectures reveals that smaller, well-optimized networks can achieve comparable performance to larger models. This finding suggests that deep networks with excessive capacity are unnecessary for spectral reconstruction tasks and that efficiency gains can be realized by selecting appropriate model architectures.

The NanoSpectral benchmark results further validate the potential of an optimized filter array in combination with efficient neural networks. The findings show that smaller models perform well under optimal conditions and even surpass larger models in some cases. However, larger models with more parameters still exhibit superior robustness under non-optimal conditions, such as when introducing sensor noise. Among the tested architectures, the medium dense model shows the best balance between accuracy, noise robustness, and deployability, making it the optimal choice for real-world implementation.

These results have significant implications for spectral sensing applications. Combining physics-based augmentation, hardware/software co-design, and network optimization enables highly efficient spectral reconstruction with minimal computational resources. This approach paves the way for developing compact and low-power spectral sensors that can be deployed on embedded systems for real-time spectral analysis.

7.2.1 Outlook

For computational spectrometers, cost-effective light filtering solutions will stay to enable low-cost applications for spectral sensing. Therefore, future sensing devices apart from our sensors will trade sub-par filtering properties for cost-effectiveness, making the approaches shown in this thesis even more relevant. Future research should focus on further refining the network architecture by combining even more methodology from Chap. 2 with spectral reconstruction. Currently, we optimize the networks manually but approaches like NAS exist, allowing us to automatically find even more efficient networks. Additionally, the pre-processing pipeline can be further optimized. Currently, the neural networks shown here can only process data normalized to the spectral footprint of the light source. However, standard spectrometers work with non-normalized data, making them more usable in do-

mains apart from transmission spectra. Therefore, the approaches shown here should be transferred to work with raw-sensor data without normalization.

We observed another interesting behavior from the linear models in the benchmarks in Chap.5 and Chap. 6. In many tests, these models produce a similar or even better performing reconstruction in comparison to neural networks. Even though their lead diminishes with rising added noise, further investigations into shallow learning methods for spectral reconstruction could be interesting. In our case, the clear focus was on spectral reconstruction with neural networks, resulting in a tool chain for deployment tailored for deployment of neural networks to embedded platforms. Therefore, developing another deployment path for a linear model for a fair comparison was not feasible due to time constraints.

7.3 Final Remarks

While this thesis explores two distinct topics—hierarchical classification and spectral reconstruction, they both contribute to the broader goal of optimizing machine learning for constrained environments. The findings on HiML provide insights into structuring classification tasks for energy efficiency, while the results on spectral reconstruction demonstrate how domain-specific knowledge can enhance data efficiency and model performance. Both approaches emphasize the importance of balancing computational cost and accuracy to achieve practical and deployable AI solutions.

The results are particularly relevant as machine learning continues to move toward edge and embedded applications. In many real-world scenarios, local data processing with minimal energy consumption is critical, whether for industrial monitoring, portable spectral analysis, or autonomous sensing systems. The methods developed in this thesis demonstrate that both classification and spectral reconstruction can be optimized for deployment on resource-constrained hardware without sacrificing performance.

Moreover, this work highlights the value of integrating physics-based insights into machine learning workflows. The spectral reconstruction results show that augmenting datasets with physically meaningful transformations significantly improves model performance without additional data collection. Similarly, reinforcement learning enables automated model selection, reducing the need for extensive manual tuning. These interdisciplinary approaches demonstrate that combining domain knowledge with machine learning techniques leads to more efficient and effective models.

In conclusion, this thesis demonstrates that structured machine learning workflows, efficient feature selection, and adaptive optimization strategies enable both energy-efficient classification and high-fidelity spectral reconstruction on embedded systems. These findings contribute to the development of smarter, more effi-

cient AI-driven sensing technologies, providing a foundation for future advancements in energy-aware and resource-efficient machine learning.

Bibliography

[1] Abadade, Youssef and Temouden, Anas and Bamoumen, Hatim and Benamar, Nabil and Chtouki, Yousra and Hafid, Abdelhakim Senhaji, "A Comprehensive Survey on TinyML," *IEEE Access*, **vol. 11**, pp. 96 892–96 922, 2023 (cit. on pp. 2, 4, 30).

[2] Abdelfattah, Mohamed S and Mehrotra, Abhinav and Dudziak, Łukasz and Lane, Nicholas D, "Zero-cost proxies for lightweight nas," *International Conference on Learning Representations (ICLR)*, 2021 (cit. on pp. 12, 14).

[3] Adams, Stephen and Meekins, Ryan and Beling, Peter A. and Farinholt, Kevin and Brown, Nathan and Polter, Sherwood and Dong, Qing, "Hierarchical fault classification for resource constrained systems," *Mechanical Systems and Signal Processing*, **vol. 134**, p. 106 266, Dec. 2019 (cit. on pp. 24, 34, 37–42).

[4] Ahmed, N. and Natarajan, T. and Rao, K.R., "Discrete Cosine Transform," *IEEE Transactions on Computers*, **vol. C-23** (1), pp. 90–93, 1974 (cit. on p. 62).

[5] Alexey Dosovitskiy and Lucas Beyer and Alexander Kolesnikov and Dirk Weissenborn and Xiaohua Zhai and Thomas Unterthiner and Mostafa Dehghani and Matthias Minderer and Georg Heigold and Sylvain Gelly and Jakob Uszkoreit and Neil Houlsby, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale," in *International Conference on Learning Representations*, 2021 (cit. on pp. 77 f.).

[6] Andrew G. Howard and M. Sandler and Grace Ch and, Liang-Chieh Chen and Bo Chen and Mingxing Tan and Weijun Wang and Yukun Zhu and Ruoming Pang and Vijay Vasudevan and Quoc V. Le and Hartwig Adam, "Searching for MobileNetV3," *Conference on Computer Vision and Pattern Recognition*, 2019 (cit. on p. 11).

[7] Andrew G. Howard and Menglong Zhu and Bo Chen and Dmitry Kalenichenko and Weijun Wang and Tobias Weyand and Marco Andreetto and Hartwig Adam, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," *Conference on Computer Vision and Pattern Recognition*, 2017 (cit. on p. 10).

[8] Anthony Thomas and Yunhui Guo and Yeseong Kim and Baris Aksanli and Arun Kumar and Tajana S Rosing and Uc San Diego, "Hierarchical and Distributed Machine Learning Inference Beyond the Edge; Hierarchical and Distributed Machine Learning Inference Beyond the Edge," *2019 IEEE 16th International Conference on Networking, Sensing and Control (ICNSC)*, 2019 (cit. on p. 25).

[9] Arber Zela and Thomas Elsken and Tonmoy Saikia and Yassine Marrakchi and Thomas Brox and Frank Hutter, "Understanding and Robustifying

Differentiable Architecture Search," in *International Conference on Learning Representations*, 2020 (cit. on pp. 12, 14).

[10] ARM Ltd., *ARM Cortex-M Series: High-performance Microcontrollers for Embedded Applications*, 2021 (cit. on p. 29).

[11] Ashapure, Akash and Jung, Jinha and Chang, Anjin and Oh, Sungchan and Maeda, Murilo and Landivar, Juan, "A Comparative Study of RGB and Multispectral Sensor-Based Cotton Canopy Cover Modelling Using Multi-Temporal UAS Data," *Remote Sensing*, **vol. 11** (23), 2019 (cit. on p. 3).

[12] Ashish Vaswani and Noam Shazeer and Niki Parmar and Jakob Uszkoreit and Llion Jones and Aidan N. Gomez and Lukasz Kaiser and Illia Polosukhin, *Attention Is All You Need*, 2023 (cit. on p. 77).

[13] Aster, Richard and Borchers, Brian and Thurber, Clifford, "Parameter Estimation and Inverse Problems," *Recherche*, **vol. 67**, p. 02, Jan. 2012 (cit. on p. 56).

[14] Bai, Yutong and Mei, Jieru and Yuille, Alan L and Xie, Cihang, "Are Transformers more robust than CNNs?" In *Advances in Neural Information Processing Systems*, M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, Eds., vol. 34, Curran Associates, Inc., 2021, pp. 26 831–26 843 (cit. on p. 77).

[15] Baller, Stephan Patrick and Jindal, Anshul and Chadha, Mohak and Gerndt, Michael, "DeepEdgeBench: Benchmarking Deep Neural Networks on Edge Devices," in *2021 IEEE International Conference on Cloud Engineering (IC2E)*, 2021, pp. 20–30 (cit. on pp. 2, 30).

[16] Banner, Ron and Nahshan, Yury and Soudry, Daniel, "Post training 4-bit quantization of convolutional networks for rapid-deployment," *Advances in Neural Information Processing Systems*, **vol. 32**, 2019 (cit. on p. 17).

[17] Barret Zoph and Quoc Le, "Neural Architecture Search with Reinforcement Learning," in *International Conference on Learning Representations*, 2017 (cit. on pp. 12 f.).

[18] Bellman, Richard, "Dynamic programming," *Science*, **vol. 153** (3731), pp. 34–37, 1966 (cit. on p. 13).

[19] Bengio, Yoshua and Léonard, Nicholas and Courville, Aaron, "Estimating or propagating gradients through stochastic neurons for conditional computation," *arXiv preprint arXiv:1308.3432*, 2013 (cit. on p. 17).

[20] Bergstra, James and Yamins, Daniel and Cox, David, "Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures," in *International conference on Machine Learning*, 2013 (cit. on p. 12).

[21] Bichen Wu and Xiaoliang Dai and Peizhao Zhang and Yanghan Wang and Fei Sun and Yiming Wu and Yuandong Tian and Peter Vajda and Yangqing Jia and Kurt Keutzer, "FBNet: Hardware-Aware Efficient ConvNet Design via Differentiable Neural Architecture Search," in *IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, 2019 (cit. on pp. 12 f.).

[22] Bolukbasi, Tolga and Wang, Joseph and Dekel, Ofer and Saligrama, Venkatesh, "Adaptive Neural Networks for Efficient Inference," in *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ser. Icml'17, Sydney, NSW, Australia: JMLR.org, 2017, pp. 527–536 (cit. on p. 27).

[23] Bowen Baker and Otkrist Gupta and Nikhil Naik and Ramesh Raskar, "Designing Neural Network Architectures using Reinforcement Learning," in *International Conference on Learning Representations*, 2017 (cit. on pp. 12 f.).

[24] Broadcom Inc., *Broadcom QMini Wide VIS - AFBR-S20M2WV* (cit. on p. 70).

[25] Brown, Calvin and Goncharov, Artem and Ballard, Zachary S. and Fordham, Mason and Clemens, Ashley and Qiu, Yunzhe and Rivenson, Yair and Ozcan, Aydogan, "Neural Network-Based On-Chip Spectroscopy Using a Scalable Plasmonic Encoder," *ACS Nano*, **vol. 15** (4), pp. 6305–6315, Apr. 2021 (cit. on p. 49).

[26] Cai, Yaohui and Yao, Zhewei and Dong, Zhen and Gholami, Amir and Mahoney, Michael W and Keutzer, Kurt, "Zeroq: A novel zero shot quantization framework," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 13 169–13 178 (cit. on p. 17).

[27] Carlos N Silla and Alex A Freitas, "A survey of hierarchical classification across different application domains," *Data Min Knowl Disc*, **vol. 22** (1), pp. 31–72, Jan. 2011 (cit. on pp. 23, 33).

[28] "Case Western Reserve University Bearing Data Center Website." (), [Online]. Available: `https://csegroups.case.edu/bearingdatacenter/pages/welcome-case-western-reserve-university-bearing-data-center-website` (visited on 05/05/2021) (cit. on p. 31).

[29] Catherine D. Schuman and Thomas E. Potok and Robert M. Patton and J. Douglas Birdwell and Mark E. Dean and Garrett S. Rose and James S. Plank, *A Survey of Neuromorphic Computing and Neural Networks in Hardware*, 2017 (cit. on p. 30).

[30] Chander Prakash and Lakhwinder Pal Singh and Ajay Gupta and Shiv Kumar Lohan, "Advancements in smart farming: A comprehensive review of IoT, wireless communication, sensors, and hardware for agricultural automation," *Sensors and Actuators A: Physical*, **vol. 362**, p. 114 605, 2023 (cit. on p. 3).

[31] Charles L. Lawson and Richard J. Hanson, "Solving least squares problems," in *Classics in applied mathematics*, 1976 (cit. on p. 54).

[32] Cheng-Chun Chang and Heung-No Lee, "On the estimation of target spectrum for filter-array based spectrometers," *Opt. Express*, **vol. 16** (2), pp. 1056–1061, Jan. 2008 (cit. on p. 54).

[33] Cheolsun Kim and Dongju Park and Heung-No Lee, "Convolutional neural networks for the reconstruction of spectra in compressive sensing spectrometers," in *Optical Data Science II*, B. Jalali and K.-i. Kitayama, Eds.,

International Society for Optics and Photonics, vol. 10937, SPIE, 2019, p. 109370L (cit. on pp. 61, 63 f.).

[34] Cheolsun Kim and Dongju Park and Jioh Lee and Heung-No Lee, *Deep learning-based single-shot computational spectrometer using multilayer thin films*, 2022 (cit. on pp. 61–64, 70, 76).

[35] Cui, Rong and Yu, He and Xu, Tingfa and Xing, Xiaoxue and Cao, Xiaorui and Yan, Kang and Chen, Jiexi, "Deep Learning in Medical Hyperspectral Images: A Review," *Sensors*, **vol. 22** (24), 2022 (cit. on p. 4).

[36] Czerny, M. and Turner, A. F., "Über den Astigmatismus bei Spiegelspektrometern," *Zeitschrift für Physik*, **vol. 61** (11), pp. 792–797, Nov. 1930 (cit. on p. 53).

[37] Darvish Rouhani, Bita and Lo, Daniel and Zhao, Ritchie and Liu, Ming and Fowers, Jeremy and Ovtcharov, Kalin and Vinogradsky, Anna and Massengill, Sarah and Yang, Lita and Bittner, Ray and others, "Pushing the limits of narrow precision inferencing at cloud scale with microsoft floating point," *Advances in neural information processing systems*, **vol. 33**, pp. 10 271–10 281, 2020 (cit. on p. 16).

[38] Deb, K. and Pratap, A. and Agarwal, S. and Meyarivan, T., "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, 2002 (cit. on p. 13).

[39] DeBole, Michael V. and Taba, Brian and Amir, Arnon and Akopyan, Filipp and Andreopoulos, Alexander and Risk, William P. and Kusnitz, Jeff and Ortega Otero, Carlos and Nayak, Tapan K. and Appuswamy, Rathinakumar and Carlson, Peter J. and Cassidy, Andrew S. and Datta, Pallab and Esser, Steven K. and Garreau, Guillaume J. and Holland, Kevin L. and Lekuch, Scott and Mastro, Michael and McKinstry, Jeff and di Nolfo, Carmelo and Paulovicks, Brent and Sawada, Jun and Schleupen, Kai and Shaw, Benjamin G. and Klamo, Jennifer L. and Flickner, Myron D. and Arthur, John V. and Modha, Dharmendra S., "TrueNorth: Accelerating From Zero to 64 Million Neurons in 10 Years," *Computer*, **vol. 52** (5), pp. 20–29, 2019 (cit. on p. 30).

[40] Denil, Misha and Shakibi, Babak and Dinh, Laurent and Ranzato, Marc'Aurelio and de Freitas, Nando, "Predicting Parameters in Deep Learning," *Advances in Neural Information Processing Systems*, **vol. 26**, 2013 (cit. on p. 18).

[41] M. Deutel, M. Mallah, J. Wissing, and S. Scheele, "Recent trends in edge ai: Efficient design, training and deployment of machine learning models," *Charting the Intelligence Frontiers – Edge AI Systems Nexus*, pp. 181–220, Dec. 2025 (cit. on pp. 2, 4 f., 16 ff.).

[42] Espressif Systems, *ESP32 and TinyML: Deploying Machine Learning on Low-Power Edge Devices*, 2022 (cit. on p. 29).

[43] Farzad Samie and Lars Bauer and Jorg Henkel, "Hierarchical Classification for Constrained IoT Devices: A Case Study on Human Activity Recogni-

tion," *IEEE Internet of Things Journal*, **vol. 7**, pp. 8287–8295, 9 Sep. 2020 (cit. on p. 26).

[44] Fedullo, Tommaso and Morato, Alberto and Tramarin, Federico and Rovati, Luigi and Vitturi, Stefano, "A Comprehensive Review on Time Sensitive Networks with a Special Focus on Its Applicability to Industrial Smart and Distributed Measurement Systems," *Sensors*, **vol. 22** (4), 2022 (cit. on p. 2).

[45] Feng Yan and Andreas Koch and Oliver Sinnen, *A survey on FPGA-based accelerator for ML models*, 2024 (cit. on p. 30).

[46] Frantar, Elias and Ashkboos, Saleh and Hoefler, Torsten and Alistarh, Dan, "GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers," *arXiv preprint arXiv:2210.17323*, 2022 (cit. on pp. 16 f.).

[47] Gal, Yarin and Ghahramani, Zoubin, "Dropout as a bayesian approximation: Representing model uncertainty in deep learning," in *international conference on machine learning*, PMLR, 2016, pp. 1050–1059 (cit. on p. 19).

[48] Gao Huang and Danlu Chen and Tianhong Li and Felix Wu and Laurens van der Maaten and Kilian Q. Weinberger, "Multi-Scale Dense Networks for Resource Efficient Image Classification," in *International Conference on Learning Representations*, 2017 (cit. on p. 27).

[49] Garrick Orchard and E. Paxon Frady and Daniel Ben Dayan Rubin and Sophia Sanborn and Sumit Bam Shrestha and Friedrich T. Sommer and Mike Davies, *Efficient Neuromorphic Signal Processing with Loihi 2*, 2021 (cit. on p. 30).

[50] Gondara, Lovedeep, "Medical Image Denoising Using Convolutional Denoising Autoencoders," in *2016 IEEE 16th International Conference on Data Mining Workshops (ICDMW)*, 2016, pp. 241–246 (cit. on p. 59).

[51] Google, "Edge TPU: Energy-Efficient AI Processing for the Edge," 2019 (cit. on p. 30).

[52] Google AI, *Quantization and Deployment of Neural Networks with TensorFlow Lite*, 2021 (cit. on pp. 2, 89).

[53] Guan, Qingze and Lim, Zi Heng and Sun, Haoyang and Chew, Jeremy Xuan Yu and Zhou, Guangya, "Review of Miniaturized Computational Spectrometers," *Sensors*, **vol. 23** (21), 2023 (cit. on p. 48).

[54] Gupta, Suyog and Agrawal, Ankur and Gopalakrishnan, Kailash and Narayanan, Pritish, "Deep learning with limited numerical precision," in *International conference on machine learning*, PMLR, 2015, pp. 1737–1746 (cit. on p. 15).

[55] Han Cai and Chuang Gan and Tianzhe Wang and Zhekai Zhang and Song Han, "Once-for-All: Train One Network and Specialize it for Efficient Deployment," in *International Conference on Learning Representations*, 2020 (cit. on pp. 12, 14).

[56] Han, Song and Mao, Huizi and Dally, William J, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015 (cit. on p. 20).

[57] Han, Song and Pool, Jeff and Tran, John and Dally, William, "Learning both weights and connections for efficient neural network," *Advances in neural information processing systems*, **vol. 28**, 2015 (cit. on pp. 19 f.).

[58] Hanson, Stephen and Pratt, Lorien, "Comparing Biases for Minimal Network Construction with Back-Propagation," *Advances in Neural Information Processing Systems*, **vol. 1**, 1988 (cit. on p. 18).

[59] Hanxiao Liu and Karen Simonyan and Yiming Yang, "DARTS: Differentiable Architecture Search," in *International Conference on Learning Representations*, 2019 (cit. on pp. 12 ff.).

[60] Haoran Zhu and Boyuan Chen and Carter Yang, *Understanding Why ViT Trains Badly on Small Datasets: An Intuitive Perspective*, 2023 (cit. on p. 77).

[61] Hassibi, Babak and Stork, David, "Second order derivatives for network pruning: Optimal brain surgeon," *Advances in neural information processing systems*, **vol. 5**, 1992 (cit. on pp. 18 f.).

[62] Hettiarachchi, Don Lahiru Nirmal and Davuluru, Venkata Salini Priyamvada and Balster, Eric J, "Integer vs. floating-point processing on modern FPGA technology," in *2020 10th Annual Computing and Communication Workshop and Conference (CCWC)*, IEEE, 2020, pp. 0606–0612 (cit. on p. 15).

[63] Hodgkinson, Jane and Tatam, Ralph P, "Optical gas sensing: a review," *Measurement Science and Technology*, **vol. 24** (1), p. 012 004, Nov. 2012 (cit. on p. 3).

[64] Horowitz, Mark, "1.1 computing's energy problem (and what we can do about it)," in *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, IEEE, 2014, pp. 10–14 (cit. on p. 15).

[65] Hu, Hengyuan and Peng, Rui and Tai, Yu-Wing and Tang, Chi-Keung, "Network trimming: A data-driven neuron pruning approach towards efficient deep architectures," *arXiv preprint arXiv:1607.03250*, 2016 (cit. on p. 20).

[66] Huameng Li and Rubo Chen and Hongru Li and Chaoying Shi and Han Qi and Guoliang Deng and Hong Zhang and Hao Zhou, "Micro-spectrometer based on a broadband gradient plasmonic nano-islands filter," *Opt. Lett.*, **vol. 49** (23), pp. 6673–6676, Dec. 2024 (cit. on p. 49).

[67] Ilias Leontiadis and Stefanos Laskaridis and Stylianos I. Venieris and Nicholas D. Lane, "It's always personal: Using Early Exits for Efficient On-Device CNN Personalisation," Association for Computing Machinery, Inc, Feb. 2021, pp. 15–21 (cit. on p. 28).

[68] J. Oliver and Woongbi Lee and Sangjun Park and Heung-No Lee, "Improving resolution of miniature spectrometers by exploiting sparse nature of signals," *Opt. Express*, **vol. 20** (3), pp. 2613–2625, Jan. 2012 (cit. on p. 57).

[69] J.J. Michalsky and E.W. Kleckner, "Estimation of continuous solar spectral distributions from discrete filter measurements," *Solar Energy*, **vol. 33** (1), pp. 57–64, 1984 (cit. on p. 47).

[70] Jacob, Benoit and Kligys, Skirmantas and Chen, Bo and Zhu, Menglong and Tang, Matthew and Howard, Andrew and Adam, Hartwig and Kalenichenko, Dmitry, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 2704–2713 (cit. on pp. 16 f.).

[71] Jie Hu and Li Shen and Gang Sun, "Squeeze-and-excitation networks," in *Conference on Computer Vision and Pattern Recognition*, 2018 (cit. on p. 11).

[72] Jinhui Zhang and Xueyu Zhu and Jie Bao, "Solver-informed neural networks for spectrum reconstruction of colloidal quantum dot spectrometers," *Opt. Express*, **vol. 28** (22), pp. 33 656–33 672, Oct. 2020 (cit. on pp. 59 ff.).

[73] Johnson, Jeff, "Rethinking floating point for deep learning," *arXiv preprint arXiv:1811.01721*, 2018 (cit. on p. 15).

[74] Julio Wissing and Lidia Fargueta and Stephan Scheele, "Spectral reconstruction using neural networks in filter-array-based chip-size spectrometers," *tm - Technisches Messen*, **vol. 91** (12), pp. 649–657, 2025 (cit. on pp. 5, 76, 78).

[75] S. Junger, *Projekt infimedar*, Available at `https://www.iis.fraunhofer.de/de/ff/sse/sensor-solutions/forschung/infimedar.html`, Feb. 2020 (cit. on p. 50).

[76] Junger, Stephan and Tschekalinskij, Wladimir and Weber, Norbert, "Optical bandpass filter system, in particular for multichannel spectral-selective measurements," Nov. 2014 (cit. on pp. 47, 50).

[77] Junger, Stephan and Verwaal, Nanko and Nestler, Rico and Gäbler, Daniel, "Integrierte Multispektralsensoren in CMOS-Technologie Integrated multispectral sensors in CMOS technology," in *Mikrosystemtechnik Kongress München*, Oct. 2017 (cit. on p. 50).

[78] Jyotirmoy Karjee and Kartik Anand and Vanamala Narasimha Bhargav and Praveen S Naik and Ramesh Babu Venkat Dabbiru and N Srinidhi, "Split Computing: Dynamic Partitioning and Reliable Communications in IoT-Edge for 6G Vision," Ieee, Aug. 2021, pp. 233–240 (cit. on p. 26).

[79] Kaiming He and Xiangyu Zhang and Shaoqing Ren and Jian Sun, "Deep Residual Learning for Image Recognition," *Conference on Computer Vision and Pattern Recognition*, 2016 (cit. on pp. 10, 19).

[80] Kiritchenko, Svetlana and Famili, Fazel, "Functional Annotation of Genes Using Hierarchical Text Categorization," *Proceedings of BioLink SIG, ISMB*, Jan. 2005 (cit. on p. 24).

[81] Knowles, J., "ParEGO: a hybrid algorithm with on-line landscape approximation for expensive multiobjective optimization problems," *IEEE Transactions on Evolutionary Computation*, 2006 (cit. on p. 13).

[82] Koen Goetschalckx and Bert Moons and Steven Lauwereins and Martin Andraud and Marian Verhelst, "Optimized Hierarchical Cascaded Processing; Optimized Hierarchical Cascaded Processing," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, **vol. 8** (4), pp. 884–894, 4 Dec.

2018, Conference Name: IEEE Journal on Emerging and Selected Topics in Circuits and Systems (cit. on p. 24).

[83] Kohlbrenner, Maximilian and Bauer, Alexander and Nakajima, Shinichi and Binder, Alexander and Samek, Wojciech and Lapuschkin, Sebastian, "Towards Best Practice in Explaining Neural Network Decisions with LRP," in *2020 International Joint Conference on Neural Networks (IJCNN)*, 2020, pp. 1–7 (cit. on pp. 21, 86).

[84] Krishnamoorthi, Raghuraman, "Quantizing deep convolutional networks for efficient inference: A whitepaper," *arXiv preprint arXiv:1806.08342*, 2018 (cit. on p. 14).

[85] Kurokawa, Umpei and Choi, Byung Il and Chang, Cheng-Chun, "Filter-Based Miniature Spectrometers: Spectrum Reconstruction Using Adaptive Regularization," *IEEE Sensors Journal*, **vol. 11** (7), pp. 1556–1563, 2011 (cit. on pp. 55 ff.).

[86] Laubmann, Jonathan and Saloman, Stefan and Wissing, Julio and Tschekalinskij, Wladimir and Hettenkofer, Sebastian and Stefani, Alessio and Scholz, Teresa, "A Data Driven Correction Algorithm for Inverse Problems with Application to Spectral Reconstruction," in *2024 IEEE SENSORS*, 2024, pp. 1–4 (cit. on pp. 58 f., 89 f.).

[87] LeCun, Yann and Denker, John and Solla, Sara, "Optimal brain damage," *Advances in neural information processing systems*, **vol. 2**, 1989 (cit. on pp. 18 f.).

[88] Lee, Jun Haeng and Ha, Sangwon and Choi, Saerom and Lee, Won-Jo and Lee, Seungwon, "Quantization for rapid deployment of deep neural networks," *arXiv preprint arXiv:1810.05488*, 2018 (cit. on pp. 16 f.).

[89] Lee, Namhoon and Ajanthan, Thalaiyasingam and Torr, Philip HS, "Snip: Single-shot network pruning based on connection sensitivity," *International Conference on Learning Representations (ICLR)*, 2019 (cit. on pp. 12, 14).

[90] Li, Hao and Kadav, Asim and Durdanovic, Igor and Samet, Hanan and Graf, Hans Peter, "Pruning Filters for Efficient ConvNets," in *International Conference on Learning Representations*, 2016 (cit. on pp. 19 f.).

[91] Liberis, Edgar and Lane, Nicholas D., "Differentiable Neural Network Pruning to Enable Smart Applications on Microcontrollers," *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, **vol. 6** (4), Jan. 2023 (cit. on p. 19).

[92] Lin, Ming and Wang, Pichao and Sun, Zhenhong and Chen, Hesen and Sun, Xiuyu and Qian, Qi and Li, Hao and Jin, Rong, "Zen-nas: A zero-shot nas for high-performance image recognition," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 347–356 (cit. on pp. 12, 14).

[93] Lin, Mingbao and Ji, Rongrong and Zhang, Yuxin and Zhang, Baochang and Wu, Yongjian and Tian, Yonghong, "Channel pruning via automatic structure search," in *Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence*, 2021, pp. 673–679 (cit. on p. 19).

[94] Mark Deutel and Georgios Kontes and Christopher Mutschler and Jürgen Teich, *Augmented Random Search for Multi-Objective Bayesian Optimization of Neural Networks*, 2023 (cit. on pp. 12 f.).

[95] Mark Sandler and Andrew Howard and Menglong Zhu and Andrey Zhmoginov and Liang-Chieh Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," *Conference on Computer Vision and Pattern Recognition*, 2018 (cit. on pp. 10 f.).

[96] Maurício, José and Domingues, Inês and Bernardino, Jorge, "Comparing Vision Transformers and Convolutional Neural Networks for Image Classification: A Literature Review," *Applied Sciences*, **vol. 13** (9), 2023 (cit. on p. 77).

[97] Mellor, Joe and Turner, Jack and Storkey, Amos and Crowley, Elliot J, "Neural architecture search without training," in *International Conference on Machine Learning*, PMLR, 2021, pp. 7588–7598 (cit. on pp. 12, 14).

[98] Mengjuan Liu and Meichen Yang and Jiaqi Zhu and He Zhu and Yao Wang and Ziyang Ren and Yihui Zhai and Haiming Zhu and Yufeng Shan and Hongxing Qi and Junli Duan and Huizhen Wu and Ning Dai, "High-sensitivity computational miniaturized terahertz spectrometer using a plasmonic filter array and a modified multilayer residual CNN," *Nanophotonics*, **vol. 12** (23), pp. 4375–4385, 2023 (cit. on p. 49).

[99] Molchanov, P and Tyree, S and Karras, T and Aila, T and Kautz, J, "Pruning convolutional neural networks for resource efficient inference," in *5th International Conference on Learning Representations, ICLR 2017-Conference Track Proceedings*, 2019 (cit. on pp. 19 f.).

[100] Montavon, Grégoire and Binder, Alexander and Lapuschkin, Sebastian and Samek, Wojciech and Müller, Klaus-Robert, "Layer-Wise Relevance Propagation: An Overview," in *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*, W. Samek, G. Montavon, A. Vedaldi, L. K. Hansen, and K.-R. Müller, Eds. Cham: Springer International Publishing, 2019, pp. 193–209 (cit. on p. 21).

[101] Müller, Roland and Mateu, Loreto and Brederlow, Ralf, "Analog/Mixed-Signal Standard Cell Based Approach for Automated Circuit Generation of Neural Network Accelerators," in *2023 38th Conference on Design of Circuits and Integrated Systems (DCIS)*, 2023, pp. 1–6 (cit. on p. 30).

[102] Nagel, Markus and Amjad, Rana Ali and Van Baalen, Mart and Louizos, Christos and Blankevoort, Tijmen, "Up or down? adaptive rounding for post-training quantization," in *International Conference on Machine Learning*, PMLR, 2020, pp. 7197–7206 (cit. on p. 15).

[103] Nagel, Markus and Fournarakis, Marios and Amjad, Rana Ali and Bondarenko, Yelysei and Van Baalen, Mart and Blankevoort, Tijmen, "A white paper on neural network quantization," *arXiv preprint arXiv:2106.08295*, 2021 (cit. on p. 17).

[104] Nascimento, Marcelo Gennari do and Fawcett, Roger and Prisacariu, Victor Adrian, "DSConv: efficient convolution operator," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 5148–5157 (cit. on pp. 16 f.).

[105] Nasr, Mahmoud and Islam, Md. Milon and Shehata, Shady and Karray, Fakhri and Quintana, Yuri, "Smart Healthcare in the Age of AI: Recent Advances, Challenges, and Future Prospects," *IEEE Access*, **vol. 9**, pp. 145 248–145 270, 2021 (cit. on p. 2).

[106] Niedrig, Heinz and Bergmann, Ludwig and Schaefer, Clemens, "Optische Filter," in *Optik*, 9th ed. W. de Gruyter, 1993, vol. 3, pp. 675–676 (cit. on p. 73).

[107] Ns, Abdiansah, "Time complexity analysis of support vector machines (SVM) in LibSVM," *International Journal of Computer Applications*, **vol. 128**, Oct. 2015 (cit. on p. 37).

[108] NVIDIA, "Jetson Nano: NVIDIA's Embedded AI Platform for Edge Computing," 2021 (cit. on p. 30).

[109] Park, Eunhyeok and Yoo, Sungjoo and Vajda, Peter, "Value-aware quantization for training and inference of neural networks," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 580–595 (cit. on p. 17).

[110] Peng Wang and Rajesh Menon, "Computational spectroscopy via singular-value decomposition and regularization," *Opt. Express*, **vol. 22** (18), pp. 21 541–21 550, Sep. 2014 (cit. on p. 56).

[111] Pirzada, Muqsit and Altintas, Zeynep, "Recent Progress in Optical Sensors for Biomedical Diagnostics," *Micromachines*, **vol. 11** (4), 2020 (cit. on p. 3).

[112] Polino, Antonio and Pascanu, Razvan and Alistarh, Dan, "Model compression via distillation and quantization," *arXiv preprint arXiv:1802.05668*, 2018 (cit. on p. 16).

[113] Real, Esteban and Aggarwal, Alok and Huang, Yanping and Le, Quoc V, "Regularized evolution for image classifier architecture search," in *Proceedings of the aaai conference on artificial intelligence*, 2019 (cit. on p. 12).

[114] Renard, Feu, *Czerny-Turner Monochromator*, Jun. 2015 (cit. on p. 53).

[115] Ronneberger, Olaf and Fischer, Philipp and Brox, Thomas, "U-Net: Convolutional Networks for Biomedical Image Segmentation," in *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, N. Navab, J. Hornegger, W. M. Wells, and A. F. Frangi, Eds., Cham: Springer International Publishing, 2015, pp. 234–241 (cit. on pp. 76, 80).

[116] Roscolux, *Roscolux Filter Swatchbook*, `https://www.edmundoptics.com/f/roscolux-color-filter-swatchbook/12186`, Accessed: (April 2024) (cit. on p. 70).

[117] Schwartz, Daniel and Selman, Jonathan Michael Gomes and Wrege, Peter and Paepcke, Andreas, "Deployment of Embedded Edge-AI for Wildlife Monitoring in Remote Regions," in *2021 20th IEEE International Conference*

on Machine Learning and Applications (ICMLA), Dec. 2021, pp. 1035–1042 (cit. on p. 31).

[118] Shayan Mousavi M, S. and Pofelski, Alexandre and Botton, Gianluigi, "EELSpecNet: Deep Convolutional Neural Network Solution for Electron Energy Loss Spectroscopy Deconvolution," *Microscopy and Microanalysis*, **vol. 27** (S1), pp. 1626–1627, 2021 (cit. on pp. 75, 77).

[119] Shi, Zhining and Chow, Christopher W. K. and Fabris, Rolando and Liu, Jixue and Jin, Bo, "Applications of Online UV-Vis Spectrophotometer for Drinking Water Quality Monitoring and Process Control: A Review," *Sensors*, **vol. 22** (8), 2022 (cit. on p. 3).

[120] Soussi, Abdellatif and Zero, Enrico and Sacile, Roberto and Trinchero, Daniele and Fossa, Marco, "Smart Sensors and Smart Data for Precision Agriculture: A Review," *Sensors*, **vol. 24** (8), 2024 (cit. on p. 2).

[121] Stephen Adams and Tyler Cody and Peter A Beling and Sherwood Polter and Kevin Farinholt, "Hierarchical Classification for Unknown Faults; Hierarchical Classification for Unknown Faults," *2020 IEEE International Conference on Prognostics and Health Management (ICPHM)*, pp. 1–8, Jun. 2020 (cit. on p. 24).

[122] *STM32F423 web page* (cit. on p. 85).

[123] STMicroelectronics, *AI Acceleration on STM32 Microcontrollers*, 2022 (cit. on p. 29).

[124] Sun, Tianming and Feng, Bin and Huo, Jinpeng and Xiao, Yu and Wang, Wengan and Peng, Jin and Li, Zehua and Du, Chengjie and Wang, Wenxian and Zou, Guisheng and Liu, Lei, "Artificial Intelligence Meets Flexible Sensors: Emerging Smart Flexible Sensing Systems Driven by Machine Learning and Artificial Synapses," *Nano-Micro Letters*, **vol. 16** (1), p. 14, Nov. 2023 (cit. on p. 2).

[125] Surat Teerapittayanon and Bradley McDanel and H. T. Kung, "BranchyNet: Fast inference via early exiting from deep neural networks," vol. 0, Institute of Electrical and Electronics Engineers Inc., Jan. 2016, pp. 2464–2469 (cit. on p. 27).

[126] Tan, Mingxing and Chen, Bo and Pang, Ruoming and Vasudevan, Vijay and Sandler, Mark and Howard, Andrew and Le, Quoc V, "Mnasnet: Platform-aware neural architecture search for mobile," in *Conference on Computer Vision and Pattern Recognition*, 2019 (cit. on p. 12).

[127] Tan, Mingxing and Le, Quoc, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *International Conference on Machine Learning*, 2019 (cit. on pp. 11 f.).

[128] Wang, Chaoqi and Zhang, Guodong and Grosse, Roger, "Picking winning tickets before training by preserving gradient flow," *International Conference on Learning Representations (ICLR)*, 2020 (cit. on pp. 12, 14).

[129] Wang, Haibin and Ge, Ce and Chen, Hesen and Sun, Xiuyu, "PreNAS: Preferred One-Shot Learning Towards Efficient Neural Architecture Search," *arXiv preprint arXiv:2304.14636*, 2023 (cit. on pp. 12, 14).

[130] Wang, Peisong and Chen, Qiang and He, Xiangyu and Cheng, Jian, "Towards accurate post-training network quantization via bit-split and stitching," in *International Conference on Machine Learning*, PMLR, 2020, pp. 9847–9856 (cit. on p. 17).

[131] Wang, Zhehui and Luo, Tao and Li, Miqing and Zhou, Joey Tianyi and Goh, Rick Siow Mong and Zhen, Liangli, "Evolutionary Multi-Objective Model Compression for Deep Neural Networks," *IEEE Computational Intelligence Magazine*, 2021 (cit. on p. 13).

[132] White, Colin and Neiswanger, Willie and Savani, Yash, "Bananas: Bayesian optimization with neural architectures for neural architecture search," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2021 (cit. on pp. 12 f.).

[133] Wissing, Julio and Boenninghoff, Benedikt and Kolossa, Dorothea and Ochiai, Tsubasa and Delcroix, Marc and Kinoshita, Keisuke and Nakatani, Tomohiro and Araki, Shoko and Schymura, Christopher, "Data Fusion for Audiovisual Speaker Localization: Extending Dynamic Stream Weights to the Spatial Domain," in *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2021, pp. 4705–4709 (cit. on pp. 2, 5).

[134] Wissing, Julio and Scheele, Stephan, "Boosting Energy Efficient Machine Learning in Smart Sensor Systems," *Sensor and Measurement Science International (SMSI) 2023*, pp. 53–54, 2023 (cit. on pp. 2, 5, 9).

[135] Wissing, Julio and Scheele, Stephan and Mohammed, Aliya and Kolossa, Dorothea and Schmid, Ute, "HiMLEdge – Energy-Aware Optimization for Hierarchical Machine Learning," in *Advanced Research in Technologies, Information, Innovation and Sustainability*, T. Guarda, F. Portela, and M. F. Augusto, Eds., Cham: Springer Nature Switzerland, 2022, pp. 15–29 (cit. on pp. 4 f., 31 f., 35, 37, 42 f.).

[136] Wissing, Julio and Scholz, Teresa and Saloman, Stefan and Fargueta, Lidia and Junger, Stephan and Stefani, Alessio and Tschekalinskij, Wladimir and Scheele, Stephan and Schmid, Ute, "SPECTRE: A Dataset for Spectral Reconstruction on Chip-Size Spectrometers with a Physics-Informed Augmentation Method," in *2024 IEEE SENSORS*, 2024, pp. 1–4 (cit. on pp. 5, 69, 74 f.).

[137] X. Dong and J. Bao and D. Chen and W. Zhang and N. Yu and L. Yuan and D. Chen and B. Guo, "CSWin Transformer: A General Vision Transformer Backbone with Cross-Shaped Windows," in *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Los Alamitos, CA, USA: IEEE Computer Society, Jun. 2022, pp. 12 114–12 124 (cit. on p. 78).

[138] Xavier Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," *Journal of Machine Learning Research - Proceedings Track*, **vol. 9**, pp. 249–256, 2010 (cit. on p. 10).

[139] Xiao, Guangxuan and Lin, Ji and Seznec, Mickael and Demouth, Julien and Han, Song, "Smoothquant: Accurate and efficient post-training quantization for large language models," *arXiv preprint arXiv:2211.10438*, 2022 (cit. on p. 17).

[140] Xu, Xiaofan and Park, Mi Sun and Brick, Cormac, "Hybrid pruning: Thinner sparse networks for fast inference on edge devices," *arXiv preprint arXiv:1811.00482*, 2018 (cit. on p. 19).

[141] Xuan Shen and Yaohua Wang and Ming Lin and Yilun Huang and Hao Tang and Xiuyu Sun and Yanzhi Wang, "DeepMAD: Mathematical Architecture Design for Deep Convolutional Neural Network," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023 (cit. on p. 12).

[142] Xue, Qian and Yang, Yang and Ma, Wenkai and Zhang, Hanqiu and Zhang, Daoli and Lan, Xinzheng and Gao, Liang and Zhang, Jianbing and Tang, Jiang, "Advances in Miniaturized Computational Spectrometers," *Advanced Science*, **vol. 11** (47), p. 2 404 448, 2024 (cit. on pp. 47 f.).

[143] Xunyu Zhu and Jian Li and Yong Liu and Weiping Wang, "Improving Differentiable Architecture Search via self-distillation," *Neural Networks*, **vol. 167**, pp. 656–667, 2023 (cit. on p. 14).

[144] Yuyang Li and Yawen Wu and Xincheng Zhang and Jingtong Hu and Inhee Lee, "Energy-Aware Adaptive Multi-Exit Neural Network Inference Implementation for a Millimeter-Scale Sensing System," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, **vol. 30**, pp. 849–859, 7 Jul. 2022 (cit. on p. 28).

[145] Zaidi, Syed Ali Raza and Hayajneh, Ali M. and Hafeez, Maryam and Ahmed, Q. Z., "Unlocking Edge Intelligence Through Tiny Machine Learning (TinyML)," *IEEE Access*, **vol. 10**, pp. 100 867–100 877, 2022 (cit. on p. 29).

[146] Zhang, Dongqing and Yang, Jiaolong and Ye, Dongqiangzi and Hua, Gang, "Lq-nets: Learned quantization for highly accurate and compact deep neural networks," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 365–382 (cit. on p. 16).

[147] Zhang, Jinhui and Zhu, Xueyu and Bao, Jie, "Denoising Autoencoder Aided Spectrum Reconstruction for Colloidal Quantum Dot Spectrometers," *IEEE Sensors Journal*, **vol. 21** (5), pp. 6450–6458, 2021 (cit. on pp. 57, 59 ff.).

[148] Zhang, Yangxi and Zhang, Sheng and Wu, Hao and Wang, Jinhui and Lin, Guang and Zhang, A. Ping, "Miniature computational spectrometer with a plasmonic nanoparticles-in-cavity microfilter array," *Nature Communications*, **vol. 15** (1), p. 3807, Apr. 2024 (cit. on pp. 49, 55).

[149] Zhuang, Fuzhen and Qi, Zhiyuan and Duan, Keyu and Xi, Dongbo and Zhu, Yongchun and Zhu, Hengshu and Xiong, Hui and He, Qing, "A Compre-

hensive Survey on Transfer Learning," *Proceedings of the IEEE*, **vol. 109** (1), pp. 43–76, 2021 (cit. on p. 69).

48

University of Bamberg Press

This thesis explores two distinct approaches to optimizing machine learning for resource-constrained environments: hierarchical machine learning (HiML) for energy-efficient classification and neural network-based spectral reconstruction for cost-effective computational spectrometers. Both topics address different challenges but share the common goal of improving computational efficiency for embedded systems.

The first part investigates hierarchical machine learning as a means to reduce energy consumption in classification tasks. Unlike traditional flat classifiers, which apply equal computational effort to all inputs, hierarchical machine learning structures classification as a hierarchy of decisions, allowing for early exits when classification confidence is high. Experimental results demonstrate that this approach reduces energy consumption by up to 47.63% while maintaining competitive accuracy. Additionally, reinforcement learning is introduced as an optimization strategy for classifier selection, significantly accelerating the search process compared to exhaustive methods.

The second part focuses on spectral reconstruction, where neural networks recover full spectral information from complex sensor outputs. A physics-informed data augmentation method is introduced, leveraging the optical properties of transmission spectra to expand the dataset from 214 to over 10,000 samples. This augmentation improves model generalization without requiring additional physical measurements. Furthermore, layer-wise relevance propagation is applied to optimize the sensor hardware, enabling a 94% reduction in input dimensions while preserving high reconstruction accuracy. Benchmarking results reveal that smaller, well-optimized models can match or even surpass the performance of larger networks, demonstrating the potential for efficient, compact spectral reconstruction systems.

The findings of this thesis contribute to the development of energy-efficient and computationally lightweight machine learning solutions. The proposed approaches enable more effective classification pipelines and high-fidelity spectral reconstruction on embedded systems, paving the way for real-world applications in industrial monitoring, portable spectral sensing, and other edge AI scenarios.

ISBN: 978-3-98989-109-8

9 783989 891098

www.uni-bamberg.de/ubp