POLITECNICO DI TORINO

Master's Degree Program
in Data Science and Engineering

Efficient Hand Detection with Low-resolution Infrared Sensors

Supervisors
Prof. Daniele JAHIER PAGLIARI
Matteo RISSO
Chen XIE
Alessio BURRELLO

Candidate
Donato LANZILLOTTI
316001

A.Y. 2023/2024

Summary

In the evolving landscape of Machine Learning (ML) and Deep Learning (DL), there is an increasing emphasis on developing models that efficiently balance performance and computational constraints, rather than focusing exclusively on accuracy. This work investigates efficient ML and DL models for detecting the presence of hands and other objects using low-resolution infrared (IR) sensors, which represent a cost-effective, low-power, and privacy preserving alternative to higher resolution cameras. To support this study, a specialized dataset was collected, containing images of hands, objects, hands with objects, and empty backgrounds. The dataset includes a variety of objects with different sizes and temperatures, providing a comprehensive foundation for robust model training and evaluation across diverse detection scenarios. The implications of this research study extend to various domains, including robotics, industries, healthcare, and sign language recognition. The study begins with a comprehensive analysis of traditional ML techniques, specifically Support Vector Machines (SVM) and Random Forest (RF), to establish baseline performance metrics. Subsequently, the research explores the capabilities of DL models, in particular Convolutional Neural Networks (CNNs), known for their ability to capture intricate patterns in the visual domain. This exploration follows a two-phase approach. The first phase deals with the design of a CNN with the aim of maximizing the sole detection accuracy. The second phase considers balancing accuracy with model complexity, addressing the need for models that can operate effectively in resource constrained scenarios, such as edge devices. To achieve these objectives, the study integrates Bayesian Optimization with Differentiable Neural Architecture Search (DNAS). Specifically, the OpTuna framework is combined with two main gradient-based techniques: Pruning In Time (PIT) and Mixed-Precision Search (MPS). This hybrid strategy allows for precise architectural tuning, ensuring the models achieve a balance of high detection accuracy and optimal resource usage, adaptable to various performance and computational needs. Experimental results indicate that traditional ML techniques, such as RF and SVM, achieve accuracy rates of 80% and 85%, respectively, while CNNs outperform them with an accuracy of 90%. These findings underscore the potential of CNNs in applications requiring higher precision, although traditional methods, particularly RF, offer advantages

in computational speed. The optimization phase demonstrates that it is possible to significantly reduce model size with limited impact on prediction capabilities. A reduction in model size of up to 99.5% was achieved, resulting in a 2% increase in test accuracy.

A mio fratello Angelo.
Al nostro legame silenzioso,
che non ha bisogno di parole per essere forte.

Acknowledgements

Desidero ringraziare il Politecnico di Torino per avermi offerto l'opportunità di formarmi in un ambiente stimolante, interagendo con docenti altamente qualificati. Un ringraziamento speciale va al Professore Daniele Jahier Pagliari e ai dottori Matteo Risso, Chen Xie e Alessio Burrello, che mi hanno accompagnato in questa fase finale del mio percorso, trasmettendomi la loro passione e guidandomi con competenza, disponibilità e preziosi consigli.

Ringrazio Marenza, che sin dall'inizio mi accompagna e mi supporta in ogni fase della mia vita, rappresentando per me un punto di riferimento fondamentale.

Ringrazio Mariachiara, non solo per essermi sempre stata accanto in questi anni e per aver condiviso ogni mia scelta, ma soprattutto per riuscire a vedere sempre il lato positivo in tutto ciò che faccio, apprezzando ogni mio impegno e sostenendomi con ammirazione.

Ringrazio la mia famiglia, che mi ha permesso di raggiungere questo risultato indicandomi la strada, senza farmi mai mancare nulla e fornendomi tutto il supporto di cui avevo bisogno. I valori che mi hanno trasmesso sono stati fondamentali per concludere al meglio questo percorso.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

As the influence of technology expands across nearly every aspect of daily life, the demand for intelligent systems that can interpret, respond to, and interact with the environment has grown significantly. From autonomous vehicles and smart homes to wearable devices and IoT applications, machines are increasingly expected to perform complex tasks. Central to these advancements is the rapid evolution of Machine Learning and Deep Learning [1], which enable machines to learn from data, identify patterns, and make informed decisions. However, while early advancements primarily focused on improving task performance, today's technology landscape poses a new set of challenges: the need for efficiency, scalability, and sustainability.

This new paradigm arises from a mix of different factors. Firstly, the rapid diffusion of sensors and IoT devices created an ecosystem of resource-constrained edge devices that need to process data in real-time while operating under strict limitations in terms of memory, computational power, and energy. These devices often lack the capacity required to support large and complex models, necessitating simpler alternatives. Secondly, not all organizations have the privilege of having vast computational resources. Many small to medium-sized companies and industries face the challenge of leveraging AI without access to large-scale, expensive training infrastructure. As a result, the need for lightweight, scalable models has become more critical. Finally, the theme of sustainability has become increasingly central across various sectors. Optimizing resources - whether computational, energy, or material - aligns with broader global efforts to reduce environmental impact and promote sustainable practices.

In this context, Computer Vision [2] stands out as one of the most critical application fields within the Artificial Intelligence sector, alongside Natural Language Processing. It aims to enabling machines to interpret and understand visual data, such as images and videos, including a wide range of tasks that are essential in today's technology landscape. Historically, computer vision has been recognized as one of the earliest applications of intelligent systems, addressing foundational

challenges in image analysis and pattern recognition. This pioneering role underscores its suitability for both Machine Learning and Deep Learning models. Initially, traditional machine learning techniques were employed to tackle computer vision tasks by training algorithms to recognize patterns and classify images based on handcrafted features. However, as the complexity and scale of visual data increased, the limitations of these methods became evident. The advent of Deep Learning, particularly with the introduction of Convolutional Neural Networks, revolutionized the field by enabling models to automatically learn hierarchical feature representations from raw pixel data. This capability has made deep learning models exceptionally effective for complex tasks such as image classification, object detection, and image segmentation. Thus, the evolution of computer vision from early intelligent systems to its current state illustrates its pivotal role in advancing both machine learning and deep learning technologies, demonstrating its ongoing relevance in the development of sophisticated applications across various domains.

In light of these advancements, this thesis project focuses on **Hand Detection with Low-Resolution Infrared Sensors**. The primary objective is to develop efficient machine learning and deep learning models capable of detecting the presence of a hand in IR images. The choice of using IR images, rather than traditional RGB images, is motivated by the energetic efficiency of IR sensors and the reduced computational complexity required to process 8x8 IR image data. Moreover, IR sensors offer distinct advantages in terms of privacy, as individuals are not identifiable in the captured images making them suitable for being used in public spaces. A notable application of this research lies in its potential use in industrial settings, where hand detection technology could be employed to enhance workplace safety. By integrating efficient hand detection systems with machinery, it becomes possible to automatically stop operations when a human hand is detected in potentially hazardous areas, thereby reducing the risk of accidents and injuries. The implementation of machine learning and deep learning models is crucial, as these models must accurately distinguish between hands and other objects that may be present in the environment. The system should halt operations only when a hand is detected, ensuring that safety measures are not activated in situations where other objects are also present. A simple temperature sensor, for instance, would not suffice in this context, as it cannot differentiate between a hand and other objects that may emit similar thermal signatures. This precise differentiation is essential to prevent false alarms.

This thesis can be structured into four main stages:

- **Dataset Collection**: The first phase of the thesis focuses on the collection of IR images, which will form the basis for the following stages of the research. This phase involves the use of an 8x8 Panasonic Grid-EYE IR sensor, integrated with a Raspberry Pi board for data acquisition.

- **Machine Learning Models**: The second phase focuses on developing traditional machine learning models, such as Random Forest and Support Vector Machines. This step is essential for establishing a baseline performance, which will be used in the subsequent phases to highlight the potential need for more advanced deep learning models.
- **Deep Learning Models and their Optimization**: The third phase involves the development of deep learning models, specifically Convolutional Neural Networks. The initial stage will focus exclusively on achieving high accuracy, without consideration for resource efficiency. In the subsequent phase, attention will shift to optimizing these models to enhance their efficiency in terms of resource utilization and computational demands. This optimization process will leverage several advanced state-of-the-art techniques and their combinations. In particular, Differentiable Neural Architecture Search will be employed to explore pruning and quantization techniques, with constraints applied to key metrics such as the number of parameters, computational complexity, and model size.
- **Results Analysis**: The fourth and final phase involves analyzing the results obtained, allowing for an assessment of model performance, the effectiveness of the optimization techniques, and the overall impact of resource constraints on the accuracy and efficiency of the deep learning models developed throughout the project. In addition, it highlights areas that could be further explored and improved in future works.

The rest of the thesis is structured as follows. Chapter 2 provides the necessary background and theoretical concepts essential for a comprehensive understanding of the project. Chapter 3 reviews related works on infrared images and the methodologies employed in those studies. Chapter 4 offers a detailed description of each phase of the project, emphasizing the methods utilized. Chapter 5 discusses the experimental settings and presents the results obtained. The findings are analyzed in terms of accuracy, computational efficiency, and resource utilization. Finally, Chapter 6 concludes the thesis with reflections on the overall project and suggestions for future research directions.

Chapter 2

Background

2.1 Machine Learning and Deep Learning Concepts

This chapter provides essential concepts required to fully understand the thesis. It introduces foundational topics in Machine Learning and Deep Learning, including key models such as Support Vector Machines, Random Forests, and Convolutional Neural Networks, as well as challenges within these fields. It then addresses efficient Machine Learning and Deep Learning techniques, outlining their importance, applications, and primary methods. Finally, the chapter covers infrared sensors, focusing on their types, characteristics, and limitations in low-resolution contexts.

2.1.1 Machine Learning Overview

Machine Learning (ML) is a pivotal area within the broader field of artificial intelligence (AI), where algorithms are designed to learn from and make predictions (so called inference) based on data. By using statistical techniques, ML enables computers to identify patterns and make decisions without being explicitly programmed for every input. This inference capability on never-seen-before inputs is what distinguishes ML from traditional programming, where rules and logic are manually coded.

The advent of ML has been significantly influenced by the exponential growth of data and the increase in computational power. The availability of big data and the development of more powerful computational infrastructures have been highlighted as major drivers of ML advancements [1].

Machine learning can be broadly categorized into three main types [3]:

- **Supervised Learning**: This approach involves training a model on a labeled dataset, meaning that to each input corresponds a known given output. The

objective is to learn a mapping from inputs to outputs, allowing the model to predict outputs for unseen data. Common algorithms include linear regression, decision trees, and support vector machines (SVM).

- **Unsupervised Learning**: In contrast to supervised learning, unsupervised learning deals with unlabeled data. The model tries to learn the underlying structure or distribution of the data. This category includes clustering algorithms and dimensionality reduction techniques.

- **Reinforcement Learning**: This approach is inspired by behavioral psychology and involves training an agent to make a series of decisions in an environment to maximize a reward. The agent learns from the consequences of its actions, iterating towards the optimal strategy through exploration and exploitation [4].

This thesis specifically leverages supervised learning algorithms, which are described in detail in the following sections.

2.1.2 Support Vector Machines

Support Vector Machines (SVM) are a class of supervised learning algorithms widely used for classification and regression tasks in data science. Developed by Vladimir Vapnik and colleagues in the 1990s [5], SVMs are particularly known for their ability to handle high-dimensional spaces and achieve high accuracy even with small datasets. The core idea behind SVMs is to find an optimal separating hyperplane that maximizes the margin between different classes in the training data.

Key Concepts of SVM

In order to fully understand how SVM works, the following basic concepts are needed:

- **Hyperplane**: In an n-dimensional space (where n is the number of features), a hyperplane is a flat affine subspace with n-1 dimensions. For classification, the goal of SVM is to find the optimal hyperplane that separates the data into different classes.

- **Margin**: The margin is the distance between the hyperplane and the nearest data points from each class. SVM tries to maximize this margin to ensure a robust classifier. The larger the margin, the better the generalization to unseen data.

- **Support Vectors**: Support vectors are the data points that lie closest to the hyperplane. These points are critical because they directly influence the position and orientation of the hyperplane. In SVM, only the support vectors are used to construct the hyperplane, while the rest of the data points are ignored.

Mathematical Formulation of SVM

Consider a binary classification problem where the training dataset consists of N samples $\{(x_i, y_i)\}_{i=1}^{N}$, where $x_i \in \mathbb{R}^n$ is the feature vector of the i-th sample and $y_i \in \{-1, +1\}$ is its class label. The objective of a linear SVM is to find the optimal hyperplane that separates the two classes with the maximum margin.

The hyperplane is represented by the equation:

$$w^T x + b = 0$$

where $w \in \mathbb{R}^n$ is the weight vector (normal to the hyperplane), and $b \in \mathbb{R}$ is the bias term.

For each sample x_i, we need to satisfy the following constraints:

$$y_i(w^T x_i + b) \geq 1 \quad \forall i = 1, \ldots, N$$

This constraint ensures that all data points are classified correctly, with the margin separating the positive and negative classes.
The goal of SVM is to maximize the margin, which is equivalent to minimizing the norm of the weight vector w. This leads to the following optimization problem:

$$\min_{w,b} \frac{1}{2}||w||^2$$

subject to:

$$y_i(w^T x_i + b) \geq 1 \quad \forall i$$

This is a convex quadratic optimization problem, which can be solved using methods like quadratic programming (QP) [5]. The factor $\frac{1}{2}$ is included to simplify derivatives during optimization.

Soft Margin for Non-Separable Data

In many real-world scenarios, data are not perfectly separable, meaning there may be some overlap between classes. To handle such cases, SVM introduces a *soft margin* by allowing for some misclassifications. This is achieved by introducing

slack variables $\xi_i \geq 0$, which measure the degree to which a data point violates the margin.

The modified optimization problem becomes:

$$\min_{w,b,\xi} \frac{1}{2}||w||^2 + C\sum_{i=1}^{N} \xi_i$$

subject to:

$$y_i(w^T x_i + b) \geq 1 - \xi_i \quad \forall i$$

where C is a regularization parameter that controls the trade-off between maximizing the margin and minimizing classification errors. If C is large, the SVM prioritizes classifying all training points correctly, while a small C allows for a wider margin with some classification errors.

Non-Linear SVM and the Kernel Trick

While the linear SVM works well when the data are linearly separable, many real-world datasets are non-linearly separable. To address this, SVM uses the so-called *kernel trick* to map the original data into a higher-dimensional feature space, where a linear hyperplane can effectively separate the data [6].

Instead of explicitly computing the mapping to the higher-dimensional space, SVM leverages kernel functions $K(x_i, x_j)$ to compute the dot product between two points in the transformed feature space, without ever computing the transformation directly.

The decision function of SVM in the transformed space can be written as:

$$f(x) = \sum_{i=1}^{N} \alpha_i y_i K(x_i, x) + b$$

Here, α_i are the Lagrange multipliers obtained from solving the dual optimization problem, and $K(x_i, x)$ is the kernel function that implicitly maps the data into the higher-dimensional space.

As follows, the most popular Kernel functions are reported:

- **Linear Kernel**: $K(x_i, x_j) = x_i^T x_j$. This is equivalent to performing linear classification.
- **Polynomial Kernel**: $K(x_i, x_j) = (x_i^T x_j + c)^d$. This kernel allows SVM to learn polynomial decision boundaries, where d is the degree of the polynomial.

- **Radial Basis Function (RBF) Kernel**: $K(x_i, x_j) = \exp(-\gamma||x_i - x_j||^2)$. The RBF kernel is widely used for non-linear classification, as it measures the similarity between points based on their distance.
- **Sigmoid Kernel**: $K(x_i, x_j) = \tanh(\kappa x_i^T x_j + c)$. This kernel resembles the activation function of a neural network.

The choice of the kernel and its parameters (e.g., C, γ) significantly influences the performance of the SVM model. *Grid search* and *cross-validation* are often used to find the optimal combination of these parameters.

Advantages and Disadvantages of SVM

SVM offers several advantages, including effectiveness in high-dimensional spaces, memory efficiency due to reliance on support vectors, and flexibility through the use of various kernel functions that allow modeling of both linear and non-linear relationships. However, it also presents some disadvantages, such as high computational costs for large datasets due to the complexity of solving quadratic optimization problems, sensitivity to the choice of kernel and hyperparameters requiring extensive tuning, and reduced interpretability, especially with non-linear kernels compared to simpler models like decision trees or linear regression. In addition, SVM is sensitive to the magnitude of the feature values. Therefore, it is essential to scale or normalize the input features before applying SVM. Moreover, SVM, as well as RF, which will be described in the next subsection, require a non-negligible feature extraction process for raw data such as images, which is often a manual step and necessitates domain knowledge.

2.1.3 Random Forest

Random Forests (RF) are an ensemble learning method primarily used for classification and regression tasks. Unlike SVM, which are based on maximizing the margin of a hyperplane, RF build multiple decision trees during training and aggregate their results to improve predictive performance and reduce overfitting [7].

Decision Trees as the Foundation

At the core of Random Forests are decision trees. A decision tree is a tree-like structure where each internal node represents a decision based on a feature, and each leaf node represents a predicted outcome or class label. Trees are constructed by recursively partitioning the feature space based on conditions that minimize a given loss function, typically Gini impurity or entropy for classification tasks, and Mean Squared Error (MSE) for regression tasks.

However, decision trees are prone to overfitting, especially when they grow deep, as they tend to capture noise and specific patterns in the training data. This is where Random Forests come into play to mitigate such overfitting tendencies by combining multiple trees in an ensemble.

Ensemble Learning and Bagging

Random Forests are based on an ensemble learning technique called ***bagging*** (Bootstrap Aggregating) [8]. In bagging, multiple models are trained independently, and their results are aggregated to improve overall prediction accuracy. Specifically, for Random Forests, the steps are as follows:

1. **Bootstrapped Sampling**: For each tree in the forest, a random subset of the training data is drawn with replacement, creating multiple bootstrap samples. This ensures that each tree is trained on a slightly different dataset, introducing diversity among the trees.

2. **Random Feature Selection**: During the construction of each tree, at each split, Random Forests randomly select a subset of features rather than considering all features. This further reduces correlation between the trees and prevents individual trees from dominating the ensemble by focusing on the same strong features.

3. **Aggregation**: For classification tasks, Random Forests take a majority vote from all trees in the forest to determine the final predicted class. In regression tasks, the predictions of all trees are averaged to produce the final result.

This combination of bagging and random feature selection results in models that are more robust, with lower variance than individual decision trees, and less prone to overfitting.

The Role of Hyperparameters

Like many machine learning algorithms, Random Forests include several hyperparameters that significantly influence their behavior and performance. Some of the most critical hyperparameters are as follows:

- **Number of Trees**: This parameter specifies the number of decision trees in the forest. Generally, increasing the number of trees enhances performance by reducing variance, although it also increases computational complexity.

- **Max Features**: This parameter controls the number of features considered when splitting a node. Lower values introduce greater randomness, which helps

reduce correlations among the trees, whereas higher values make individual trees more similar to a standard decision tree.

- **Maximum Tree Depth**: This parameter sets the maximum depth of each decision tree. Allowing deeper trees enables the model to capture more complex patterns, but it may lead to overfitting. Restricting the depth can help improve generalization on unseen data.

- **Minimum Samples to Split**: This parameter determines the minimum number of samples required to split an internal node. Increasing this threshold can prevent the model from creating overly specific splits, thus reducing the likelihood of overfitting.

Careful tuning of these hyperparameters is essential for optimizing the performance of Random Forests, as they directly impact the model's ability to balance variance, bias, and computational efficiency.

Out-of-Bag Error and Feature Importance

One of the advantages of Random Forests is the ability to estimate their generalization error without requiring a separate validation set, using a concept known as *Out-of-Bag (OOB) error*. Since each tree is trained on a bootstrapped sample, some data points are left out of the training set. These OOB samples can be used to evaluate the performance of the model on unseen data, providing an unbiased estimate of the test error.

Random Forests also offer insights into feature importance, which measures how useful each feature is in predicting the target variable. Feature importance is computed by analyzing how much each feature contributes to reducing the splitting criterion (such as Gini impurity or entropy) across all trees in the forest. Features with high importance scores have a greater impact on the model's predictions.

Random Forest for Classification

In classification tasks, Random Forests aim to minimize classification errors by aggregating the predictions of individual decision trees. Each tree in the forest predicts a class label, and the class that receives the majority vote is selected as the final output. This majority voting mechanism helps Random Forests outperform individual decision trees by reducing their tendency to overfit and increasing generalization. Random Forests are especially effective when dealing with noisy or imbalanced datasets. Since they build multiple trees on different subsets of data, they are less sensitive to noise and more resilient to class imbalances, making them a popular choice in real-world applications.

Random Forest for Regression

Random Forests can also be adapted for regression tasks, where the goal is to predict a continuous value rather than a discrete class label. In this case, the trees output a numerical prediction, and the final output is computed as the average of all predictions across the trees. This averaging process helps smooth out predictions and reduces the likelihood of overfitting.
One key advantage of using Random Forests for regression is their ability to handle complex, non-linear relationships between features and the target variable.

Practical Considerations

- **Feature Scaling**: Random Forests do not require feature scaling or normalization, as decision trees are invariant to the scaling of input features. This makes Random Forests easier to apply in practice compared to algorithms like SVM or logistic regression, which are sensitive to the scale of input features.
- **Handling Missing Data**: Random Forests are relatively robust to missing data. They can handle missing values in both the training and testing phases by employing strategies such as surrogate splits or by ignoring missing values during tree construction.
- **Interpretability**: While Random Forests provide insights into feature importance, the model itself is generally considered less interpretable than simpler models like decision trees or linear regression. The complexity of Random Forests, due to the large number of trees, makes it difficult to explain individual predictions.
- **Computational Complexity**: The computational complexity of Random Forests is typically higher than that of single decision trees due to the construction and evaluation of multiple trees. However, they can be parallelized efficiently, making them scalable for large datasets.

In summary, Random Forests are a powerful and flexible ensemble learning technique, well-suited for both classification and regression tasks. Their use of multiple decision trees to reduce overfitting, combined with techniques like bagging and random feature selection, makes them a robust algorithm for a wide range of applications. While Random Forests may have limitations in terms of interpretability and computational cost, their advantages in terms of accuracy, resilience to noise, and feature importance estimation make them a popular choice in practical scenarios.

2.1.4 Deep Learning Overview

Deep Learning (DL) is a specialized subset of machine learning that focuses on algorithms inspired by the structure and function of the brain, specifically Artificial Neural Networks (ANN). Unlike traditional ML algorithms, which often require manual feature extraction, DL automates this process, enabling the model to learn features directly from raw data [9]. In the case of image data, raw data refers to the pixel values in their original form. The feature extraction process involves reorganizing these pixel values or computing descriptors that highlight key characteristics and distinguishable features of the image. This automated extraction allows deep learning models to capture complex patterns and relevant information without the need for manual intervention. A typical ANN (2.1), that represents the core of DL, is composed by:

- **Input Layer**: It receives the initial data, which can be either raw or preprocessed, depending on the specific requirements of the neural network. Each neuron in this layer corresponds to a distinct feature of the input, providing the necessary structure for further processing in the network.
- **Hidden Layers**: These layers lie between the input and output layers and are responsible for processing and transforming the data. Each hidden layer consists of neurons that apply various activation functions to the inputs received from the previous layer, enabling the network to learn complex patterns and representations. The number of hidden layers and the number of neurons within each layer can significantly affect the model's performance and its ability to generalize from the training data.
- **Output Layer**: It produces the model's predictions or classifications based on the processed information from the hidden layers.

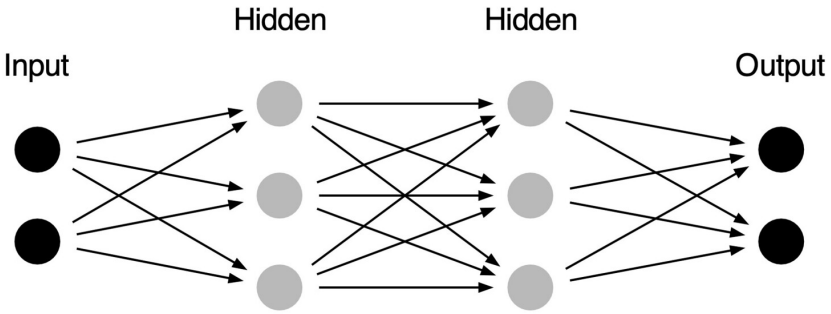


Figure 2.1: ANN.

Activation Functions

A crucial aspect in ANN is represented by the *Activations Functions*. They play a crucial role in deep learning by introducing non-linearity into the model, allowing it to learn complex patterns and relationships in the data. Without activation functions, a neural network would essentially behave like a linear model, limiting its capacity to capture intricate structures within the data. Among the most commonly used activation functions are:

- **Sigmoid Function**: It maps inputs to a range between 0 and 1 through the function:
$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.1}$$
The smooth gradient provided by its derivative makes it suitable for the *backpropagation*. However, it is susceptible to the *vanishing* gradient problem, particularly with extreme values, which can result in inefficiencies during weight updates in the training process.

- **Hyperbolic Tangent Function (tanh)**: It offers an alternative to the sigmoid by producing outputs in the range of -1 to 1, described by the equation:
$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{2.2}$$
This property results in a zero-centered output that can promote faster convergence during training, while its derivative retains a smooth gradient. Despite these advantages, tanh also suffers from the vanishing gradient problem for inputs far from zero.

- **Rectified Linear Unit Function (ReLU)**: It represents the "standard" for ANN and it is defined as follows:
$$f(x) = \max(0, x) \tag{2.3}$$
ReLU offers a straightforward computational advantage, producing zero for any negative input and maintaining a linear relationship for positive input. Its derivative allows for effective gradient propagation, thereby mitigating the vanishing gradient problem often encountered in deep networks [10]. However, a significant limitation of ReLU is that it stops learning from input examples that produces a negative output. In order to overcome this issue, some variants have been introduced, *Leaky ReLU* [11] and *Exponential Linear Unit (ELU)*.

- **Softmax Function**: It converts raw output scores (logits) into probabilities that sum to 1, making the outputs interpretable. Mathematically, it is expressed as:

$$f(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}} \tag{2.4}$$

for class i among K classes. While softmax provides a clear probabilistic framework, it can exhibit sensitivity to large input values, potentially leading to numerical instability.

In summary, the choice of activation function is critical in shaping the performance and capabilities of neural networks. Each function possesses distinct strengths and weaknesses, and a deep understanding of these characteristics allows researchers and practitioners to select the most appropriate activation functions tailored to the specific requirements of their models.

Training Neural Networks

In the context of supervised learning, training neural networks involves optimizing the model's parameters, weights and biases, using a labeled dataset. The training process typically follows these steps:

1. **Forward Propagation**: Input data is passed through the network layer by layer, and predictions are generated at the output layer. Each neuron in the network computes a weighted sum of its inputs followed by an activation function. For a neuron j, this can be expressed as:

$$z_j = \sum_i w_{ij} x_i + b_j \tag{2.5}$$

where w_{ij} are the weights, x_i are the inputs, and b_j is the bias term. The output a_j after applying an activation function f is given by:

$$a_j = f(z_j) \tag{2.6}$$

2. **Loss Calculation**: The difference between the predicted output and the actual label is calculated using a loss function. For classification tasks, a common choice is the cross-entropy loss L, defined as:

$$L(y, \hat{y}) = -\sum_i y_i \log(\hat{y}_i) \tag{2.7}$$

where y is the true label and $\hat{y}$ is the predicted probability distribution over classes [9].

For regression tasks, the mean squared error (MSE) is often used:

$$L(y, \hat{y}) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{2.8}$$

where n is the number of samples.

3. **Backpropagation**: The gradients of the loss with respect to each parameter (weights and biases) are computed using the chain rule. This process involves calculating how changes in each parameter affect the loss function, enabling the model to adjust its parameters to minimize the loss [12].

4. **Weight Update**: Parameters are updated using an optimization algorithm, such as Stochastic Gradient Descent (SGD) or Adam [13]. The update rule for a weight w can be expressed as:

$$w := w - \eta \frac{\partial L}{\partial w} \tag{2.9}$$

where η is the learning rate and $\frac{\partial L}{\partial w}$ is the gradient of the loss with respect to the weight.

This iterative process continues until the model converges, meaning the loss reaches a satisfactory level, or until a predefined number of epochs is completed. Each epoch consists of one complete pass through the training dataset, allowing the network to learn and adjust its weights progressively.

2.1.5 Convolutional Neural Network (CNN)

Convolutional Neural Networks (CNNs) are a class of deep learning models primarily used for image recognition, classification, and segmentation tasks. Unlike traditional machine learning methods that rely on handcrafted features, CNNs automatically learn hierarchical feature representations from raw input data, making them particularly effective for tasks involving spatial data, such as images and videos [14] [15].

Basic Principles of CNNs

The core idea behind CNNs is to use convolutional layers to extract local features from input data. A convolutional layer consists of a set of filters, also known as kernels, that slide over the input image and compute a dot product at each position, producing a feature map.

A typical CNN architecture consists of several types of layers:

- **Convolutional Layers**: These layers apply convolution operations to the input data using learned filters. Each filter is designed to detect specific features, such as edges or textures, in the input image. The output of this operation is a set of feature maps [16].

- **Activation Functions**: After each convolutional operation, an activation function is applied to introduce non-linearity into the model. The most commonly used activation function in CNNs is the Rectified Linear Unit (ReLU).

- **Pooling Layers**: These layers reduce the spatial dimensions of the feature maps, typically using operations such as max pooling or average pooling.

- **Fully Connected Layers**: Towards the end of the network, fully connected layers are used to combine features extracted by previous layers and produce the final output. The last fully connected layer typically applies a softmax activation function for classification tasks.

This hierarchical structure allows CNNs to learn increasingly complex features at each layer, enabling them to perform well on tasks such as image classification and object detection.

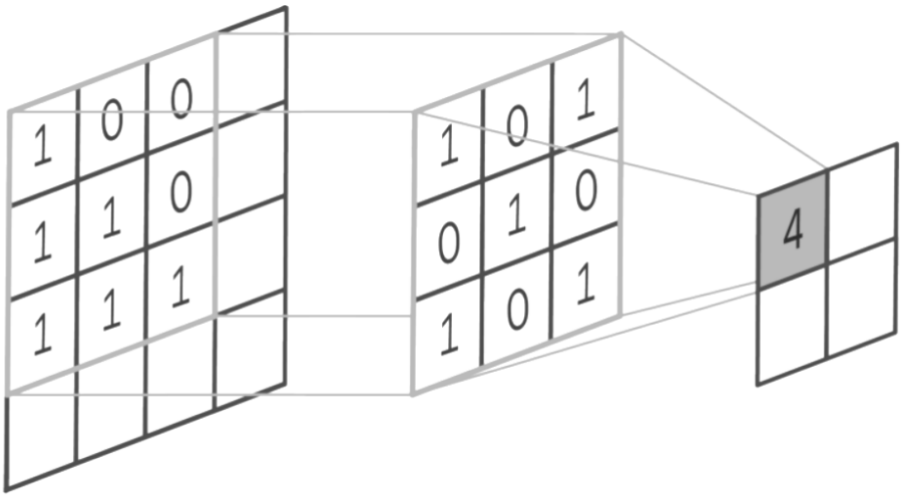


Figure 2.2: 2D Convolution [16] .

Advantages and Disadvantages of CNN

CNNs offer several advantages, including the ability to automatically learn feature representations, robustness to noise, and strong performance on image-related tasks. However, they also have drawbacks, such as a high computational cost, the need for large amounts of labeled data for effective training, and a tendency to overfit on small datasets [17].

Transfer Learning and Pretrained Models

One effective approach to leveraging CNNs is through transfer learning, where a model trained on a large dataset (e.g., ImageNet) is fine-tuned for a specific task with a smaller dataset. Pretrained models, such as VGGNet, ResNet, and Inception, provide a solid foundation, enabling faster training and improved performance on related tasks by leveraging the learned features [18].

In summary, Convolutional Neural Networks are a powerful tool in the field of deep learning, particularly for tasks involving image data. Their hierarchical architecture, which leverages convolutional layers to automatically learn features, allows them to outperform traditional methods significantly. While challenges such as interpretability and computational demands exist, CNNs have proven their effectiveness in various applications, making them a foundational technology in computer vision and beyond.

2.1.6 Challenges in ML and DL

Despite their transformative potential, ML and DL face several challenges:

- **Overfitting**: It occurs when a model learns not just the underlying patterns in the training data but also the noise and outliers, leading to poor generalization on new, unseen data. This happens when the model is too complex, such as having too many parameters or layers, and performs exceptionally well on the training set but poorly on the test set. Techniques to overcome overfitting include *dropout*, *regularization*, and *early stopping*. Dropout randomly deactivates neurons during training, preventing over-reliance on specific nodes. Regularization (like L1 or L2) adds a penalty for large weights, keeping the model simpler. Early stopping monitors the model's performance on a validation set and stops training when performance begins to decline.
- **Data Quality and Quantity**: The performance of ML/DL models is highly dependent on the quality and quantity of training data. Insufficient, noisy, or biased data can lead to inaccurate results. Common challenges include data imbalance, where the model favors the majority class in classification

tasks, and noisy data, which degrades performance. To address these issues, techniques like data augmentation, outlier detection, and noise filtering can improve data quality and enhance model reliability [19] [20].

- **Model Interpretability**: A significant challenge in ML/DL models is their lack of interpretability, particularly in complex models like deep neural networks, which often function as "black boxes." This poses issues in critical domains such as healthcare or finance, where understanding the rationale behind a model's decision is essential for trust and accountability. To address this, methods such as SHAP (Shapley Additive Explanations) and LIME (Local Interpretable Model-Agnostic Explanations) are commonly used to provide insights into how specific features influence model predictions [21] [22]. These methods are mentioned for completeness, but they are not discussed in detail as they are beyond the scope of this thesis.

- **Hyperparameter Tuning**: The performance of ML and DL models is sensitive to hyperparameters. Careful tuning is essential for optimal performance. Techniques for tuning hyperparameters include grid search, random search, and more advanced methods like Bayesian optimization.

2.2 Introduction to Efficient Machine Learning and Deep Learning

2.2.1 Definition and Overview

Traditionally, the primary goal of ML and DL has been to maximize the prediction capability of models, often through increasing their complexity and using large amounts of data. While this approach has successfully advanced the capabilities of AI, it has also resulted in models that are resource-intensive and frequently impractical for deployment outside of high-performance computing environments. In this context, *efficient* Machine Learning and Deep Learning represent a paradigm shift. The goal is developing algorithms and models that perform well in resource-constrained settings by reducing the model size, decreasing the computational complexity, and optimizing both memory and energy consumption, while still aiming to maintain high levels of accuracy and reliability.

2.2.2 Reasons that make efficient ML/DL crucial

The growing popularity of the optimization phase in developing ML/DL models is closely related to the increasing need to run these models on IoT devices, which are often constrained by limited resources.

But why is it important to deploy ML/DL models on IoT devices?

The answer is that sending data to the cloud and getting back the results is often sub-optimal for the following reasons [23]:

- **Reason 1: A lot of pressure on the network**
  Imagine we want to process 224x224 videos from IoT cameras in real time at 30fps (1 frame every 30ms). We need to transmit 224x224x3 bytes to the cloud in 30ms (plus 1 byte for the class index). As a consequence the upload bandwidth required for **a single camera** is 40 Mbps. It is inefficient from the communication network's point of view, taking into account that the resolution considered is very low for today's standards and that usually there are 100s of cameras in the same network.

- **Reason 2: High and unpredictable latency**
  It is connected to the previous one, but introduces different critical aspects. The first aspect is the *low responsiveness*. Wireless WAN links have significant round-trip times, so even with high bandwidth, it can take 10-100 ms to receive the first packet. This delay is too long for real-time applications. The second issue is the *unpredictability*. Wireless connections can be unstable or unavailable in certain locations, making it impossible to guarantee receiving a response from the model.

- **Reason 3: High energy consumption**
  The energy required to perform *computing* tasks is constantly decreasing, but the energy required for *transmitting* data is not decreasing at the same pace. For this reason, it is much more efficient to do at least part of the computation locally, so that the amount of data transmitted to the cloud is reduced.

- **Reason 4: Privacy and Confidentiality**
  Transmitting raw data to the cloud raises privacy concerns due to the possibility of unauthorized access and data breaches, which could allow malicious actors to intercept and exploit sensitive information. It is true not only for personal devices, but also companies are often reluctant to let important data leave their premises.

To summarize, deploying ML/DL models directly on edge devices can offer significant benefits, including reduced latency, improved scalability, lower energy consumption, and enhanced privacy. However, this approach is challenging due to the limited hardware capabilities of edge devices, which are constrained by cost and energy efficiency considerations.

2.2.3 Main areas benefiting from Efficient ML technology

Efficient machine learning and deep learning are crucial across a variety of applications where resource constraints, performance demands, and real-time processing requirements are significant. As AI becomes increasingly integrated into everyday life, the need for efficient ML/DL will only intensify, driving innovation and enhancing the functionality of applications across diverse domains.

1. **Consumer Tech**
   One of the most prominent areas where efficient ML is applied is in consumer electronics, such as smartphones and smart home devices. These devices are increasingly equipped with AI capabilities to enhance user experiences. Efficient ML models are crucial here due to the limited computational resources and battery constraints of these devices. For example, real-time image and speech processing applications on smartphones, such as facial recognition and voice assistants, require ML models that are both lightweight and fast. Efficient ML techniques enable these models to run smoothly on consumer hardware without draining battery life or causing significant delays [24].
   Similarly, smart home devices, which perform tasks like voice control and environmental monitoring, benefit from efficient ML. These devices must process data quickly and accurately to provide real-time responses, which is achieved through efficient ML algorithms.

2. **Healthcare Devices**
   In healthcare, efficient ML is increasingly important for real-time diagnostics and personalized medicine. Medical devices, such as wearable health monitors and portable diagnostic tools, require ML models that can operate efficiently to provide timely and accurate health insights. For instance, wearable devices that monitor vital signs must analyze data continuously and make real-time predictions about potential health issues. Efficient ML algorithms allow these devices to process data quickly while conserving power, which is critical for wearable technology.
   Furthermore, in remote or underdeveloped regions where access to advanced medical infrastructure may be limited, efficient ML can facilitate the deployment of diagnostic tools on low-cost hardware. This enables healthcare providers to use AI-driven tools for tasks such as disease detection and monitoring without relying on expensive, high-performance computing systems [25].

3. **Autonomous Vehicles**
   Autonomous vehicles represent a complex application where efficient ML is vital. These vehicles rely on numerous sensors and cameras to gather data about their environment and make real-time driving decisions. The ML models used for object detection, lane tracking, and decision-making must be highly efficient to ensure that the vehicle can operate safely and swiftly.
   Efficient ML contributes to reducing the power consumption of onboard computing systems, which is crucial for the overall energy efficiency of the vehicle [26].

4. **Agriculture**
   Efficient ML is also transforming agriculture by enabling precision farming techniques. Farmers use ML models to analyze data from various sources, such as satellite images, soil sensors, and weather stations, to make informed decisions about crop management. Efficient ML models are crucial for processing this data in real-time to provide actionable insights, such as optimizing irrigation schedules, predicting crop yields, and detecting plant diseases [27].

5. **Transportation Infrastructure**: The integration of machine learning (ML) and deep learning (DL) models into transportation infrastructure is essential for creating smart, responsive urban mobility systems that can adapt to real-time conditions. Efficient ML/DL models are crucial for processing vast amounts of data generated by sensors embedded in roads, bridges, and tunnels, as they can identify patterns and anomalies that traditional systems might overlook. For example, sophisticated algorithms can analyze traffic flow, detect congestion in real time, and dynamically adjust traffic signals to optimize

vehicle movement and reduce delays. Additionally, these models can monitor the structural integrity of infrastructure by analyzing vibrations and stress patterns, predicting maintenance needs before catastrophic failures occur. The ability to deploy efficient ML/DL models at the edge minimizes latency and bandwidth usage, ensuring rapid decision-making and enhancing public safety.

6. **Smart Cities and Buildings**: The integration of efficient machine learning technology into smart cities and buildings is revolutionizing urban living and safety management. Infrared cameras play a crucial role in monitoring environmental conditions, occupancy, and potential risks in real time. By utilizing efficient ML algorithms, these systems can analyze data to enhance public safety and operational efficiency. For instance, ML models can process infrared data to detect human presence and movement, enabling rapid response in emergency situations, such as accidents or medical emergencies. This capability allows for the immediate notification of emergency services or the activation of safety protocols, significantly improving response times. Furthermore, the deployment of efficient ML solutions in smart city infrastructure can facilitate traffic management, waste collection, and security, leading to more sustainable and livable urban environments [28].

7. **Industries**: The integration of efficient ML and DL techniques with infrared sensors plays a crucial role in enhancing safety and reducing costs in industrial environments. IR sensors continuously monitor machinery and surrounding workspaces, enabling the detection of human presence near operating equipment. Efficient ML/DL algorithms analyze this data in real time, identifying potential risks and facilitating immediate actions, such as stopping machinery if a worker is detected within a dangerous proximity. This proactive approach not only mitigates the risk of accidents but also minimizes equipment damage and repair costs, leading to a safer and more cost-effective industrial workplace [29].

2.2.4 Main techniques in efficient ML/DL

The challenge of creating efficient models from computational, memory, and energy perspectives can be approached from two different angles, strictly related.
One key approach is to focus on the *hardware*, developing specialized devices that are optimized for specific tasks. These tailored devices can significantly enhance efficiency by reducing power consumption, minimizing memory usage, and improving computational speed. The second approach instead, involves optimizing the algorithms and *software* that run on these specialized devices. By refining the underlying code and leveraging advanced techniques it is possible to further enhance the efficiency of models. This software-focused strategy complements

hardware advancements, ensuring that both aspects work in harmony to achieve maximum performance and energy efficiency. Together, these two approaches provide a comprehensive solution to the challenge of creating efficient models.

It is important to underline that developing techniques that require complete redesigns for each specific device and task is costly and time-consuming. Such an approach limits scalability and accessibility due to the high cost of customization. Instead, the objective should be to create general-purpose techniques that are versatile and broadly applicable. These techniques can then be refined or fine-tuned to account for the specific devices on which they will run and the tasks they need to perform. This approach balances efficiency and adaptability, making the techniques more cost-effective and accessible while still optimizing performance for different scenarios. This thesis exclusively focuses on the software optimization of models and algorithms, explicitly incorporating hardware performance metrics into the optimization process.
The main optimization techniques are presented in this section.

The starting point is represented by the fact that critical computations in most NN layers are Matrix-Vector (MxV) or Matrix-Matrix (MxM) multiplications, also known as **GEMM**. Before passing to the main optimization techniques, it is necessary to introduce some basic concepts.

Operational Intensity and the Roofline Model

Operational Intensity is a key metric in evaluating the performance of computational applications, defined as the ratio of computational operations to data movement. Specifically, it measures the number of floating-point operations (FLOPs) performed per byte of data transferred [30].

$$OI = \frac{N.ofOperations}{BytesTransferred} \ [\frac{FLOP}{Byte}] \tag{2.10}$$

The OI does not depend just on the type of layer, but rather on its implementation. Therefore, working on low-level algorithms allows to increase the OI and make models more efficient.

The *Roofline Model* (2.3) is a performance visualization tool that helps understand the limits of an application as function of its Operational Intensity and takes as input [30]

- **Peak Performance (π)**: The upper limit of performance, defined by the hardware's computational capabilities. This is the maximum FLOPs the system can achieve under ideal conditions.

- **Memory Bandwidth** (β): The maximum rate at which data can be transferred between memory and the processor. This sets a constraint on performance for tasks with low operational intensity.

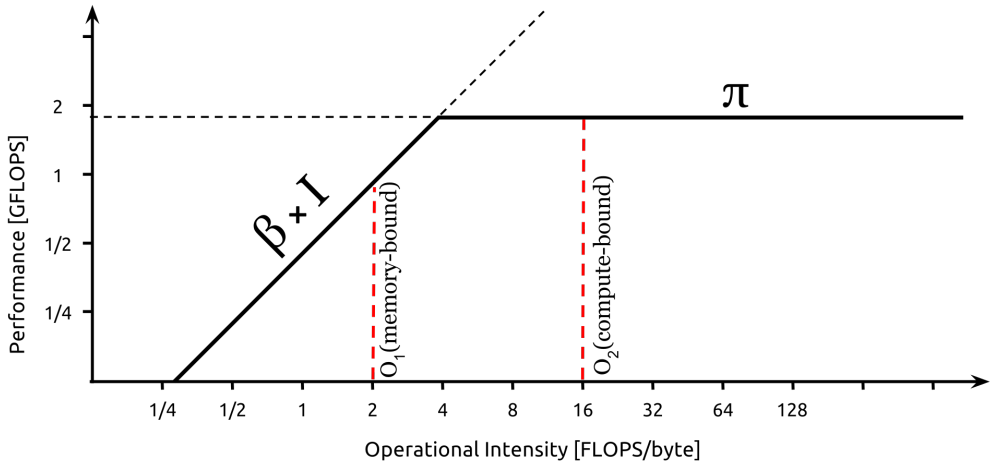


Figure 2.3: Roofline Model. [31]

It illustrates the maximum achievable performance for a given operational intensity. When operational intensity is low, performance is limited by memory bandwidth, meaning that the CPU could work more but it receives too less data per second. As operational intensity increases, performance can approach the system's peak performance limit, meaning that I could load data faster but the CPU is already working at 100%.
By using the Roofline Model, developers can identify performance bottlenecks, optimize their algorithms to improve operational intensity, and better align software with hardware capabilities.

Operational intensity, as mentioned earlier, depends on the specific implementation of a layer rather than its typology. Nevertheless, fully-connected layers tend to be memory-bound because they involve significant data movement relative to computation. In contrast, convolutional layers often fall into the compute-bound category. This is partly because convolutional layers benefit from weight sharing, which reduces the amount of data that needs to be moved.

The basic idea behind each optimization technique is to leverage the fact that deep neural networks are highly tolerant to approximations.

Why are DL models tolerant to approximations?

- **Reason 1**: Stochastic gradient-based training algorithms converge even in presence of small deviations from the correct gradient direction of a batch. In addition, it can be useful for escaping local minima.
- **Reason 2**: Often, it is not relevant the *exact* output value. For instance, the output of NN for multi-class classification are probability scores for each class. However, in most applications, the primary concern is identifying the class with the highest probability, rather than the specific value of that probability.
- **Reason 3**: DL models are highly *over-parametrized*. It means that the target function is approximated using many more parameters that those actually needed.

The subsequent discussion will highlight the major optimization techniques.

Efficient NN Layers

One effective way to optimize a NN is *replacing standard layers with more efficient alternatives* that could be exact or approximate. The focus, for this approach, will be exclusively on Convolutional layers and three main techniques will be presented. It is crucial to underline that in case of a standard Convolutional layer we have:

- **Number of weights**: it is equal to $K^2 \cdot C_{in} \cdot C_{out}$ where C_{in} and C_{out} represent respectively the input and output channels, while K the kernel size.
- **Number of MAC Operations**: it is given by $K^2 \cdot C_{in} \cdot C_{out} \cdot H_{out} \cdot W_{out}$ where H_{out} and W_{out} represent respectively the height and the width of the output image, in pixels.

The first technique proposed is replacing standard convolutions with **Spatially Separable** convolutions. The idea is to reduce a KxK 2D Convolution to the combination of Kx1 and 1xK 1D Convolutions. It means passing from $K^2 \cdot C_{in} \cdot C_{out}$ to $2 \cdot K \cdot C_{in} \cdot C_{out}$ number of weights. One negative aspect of this approach is that very few KxK kernels are spatially separable onto a combination of Kx1 and 1xK.

The second approach is represented by the **Depthwise Separable** convolutions. In this case, the concept is to separate not the spatial dimension but the channel one, therefore training each channel independently and then combining the different outcomes through a **Pointwise** (1x1) convolution [24]. In this case, the number of weights reduces to $(K^2 \cdot C_{in}) + (C_{in} \cdot C_{out})$.

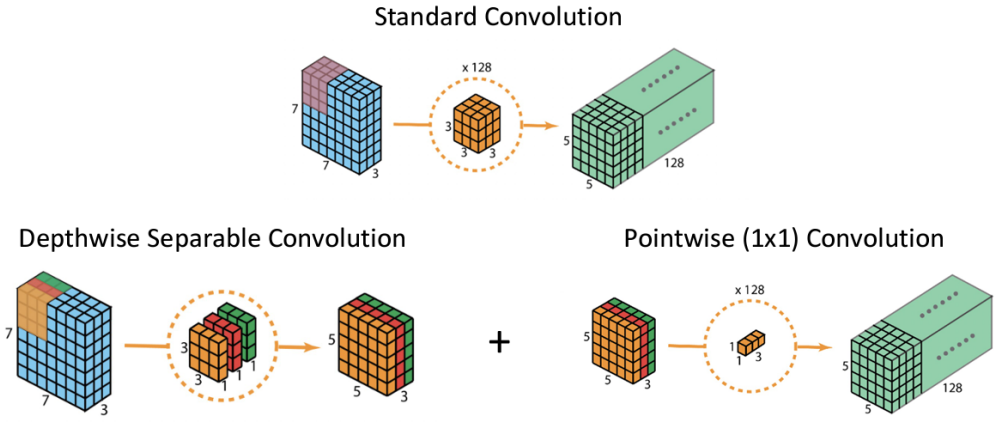


Figure 2.4: Comparison between standard convolution and DW + PW convolution. [32]

The last approach presented in the field of efficient layers, is the **Grouped Convolutions** that represents a generalization of DW convolutions in which each filter refers to a subset of input channels. As in the previous technique, after the first step it is necessary combining the results of different channels' subsets.

Neural Architecture Search (NAS)

Transitioning from classical Machine Learning to Deep Learning shifts the attention from *hand-crafted features* to *hand-crafted feature extractors*, and the complexity of modern neural networks makes manual architecture design increasingly inefficient. One of the most important techniques that has emerged to address it, is **Neural Architecture Search (NAS)**, which automates the design process of neural network architectures, optimizing them for performance and resource efficiency.

The two primary families of approaches for NAS are:

- **Classic NAS**: it is typically based on iterative processes like Reinforcement Learning (RL) or Evolutionary Algorithms (EA) [33]. It requires the definition of the search space and the search engine, in addition to the development of a performance estimator. The evaluating step is the crucial point that makes the process long, since it implies training the model and deploying it on the hardware in order to obtain accurate results. It is possible to use some approximations to speed-up the process. One option could be reducing the search space by considering fewer parameters and fewer possible values. Another option instead, could be acting on the training phase, by considering smaller dataset or lower number of epochs. About the evaluation step, it is possible to reduce the time spent, by replacing the deployment on the HW with simulation tools.
  Despite these approximations, Classic NAS remains computationally expensive and may require high-performance computing resources.

- **Differentiable NAS (DNAS)**: the idea is relaxing the search space to make it *continuous* and *differentiable* in order to use *gradient-based* algorithms not only for training the model, but also for optimizing the architecture.
  One of the most well-known examples of DNAS is DARTS [34]. It refers to the definition of a *SuperNet* (2.5): a NN with multiple alternatives for each layer, in which each of them is associated to a trainable weight that indicates how well the alternative works. During the training, the output is computed as the weighted sum of the results of the different alternatives.
  In addition, DNAS enables multi-objective optimization by adding a regularization term to the loss function that accounts for factors such as memory usage, latency, or energy consumption [35], as illustrated by the following equation:

$$FinalLoss = L(w; \alpha) + \lambda R(\alpha) \tag{2.11}$$

where the first term represents the task-specific loss, which depends on both the model weights and the selected alternatives, while the second term represents the cost, which depends exclusively on the chosen alternatives. The regularization parameter λ adjusts the trade-off between accuracy and cost. To summarize, the advantage of using DNAS is that it allows for optimizing both the architecture and training of the model within a single training loop. However, a drawback is that the initial neural network is often very large due to the presence of various alternatives.

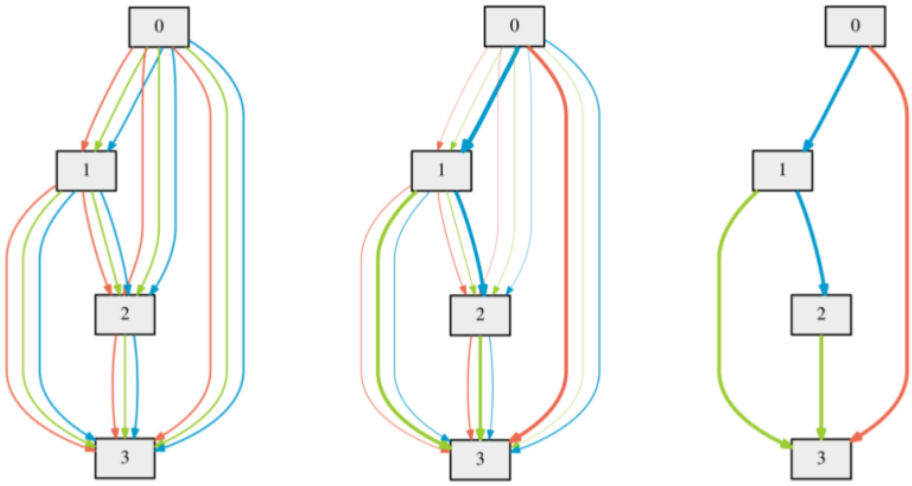


Figure 2.5: Supernet. [34]

Pruning

The **Pruning** technique involves removing redundant connections in a neural network and skipping the associated computations. It can be applied to weights, typically during training, and to activations, which represents a runtime optimization since activation values depend on the input, making the process more complex. The primary consequence of applying pruning is the creation of sparse models, which allows for the use of a *compressed format* to represent the data [36].
As a result, the main advantage is the reduction in storage size, while reductions in memory usage, latency, and energy consumption occur only if computations can be directly performed on the compressed format. Additionally, skipping computations

involving zero values can improve both energy efficiency and latency, but this benefit is only realized if the underlying hardware supports such optimizations. However, general-purpose hardware typically does not support these types of computations, which limits the effectiveness of pruning techniques.

About the weights pruning, there are two main strategies:

- **Magnitude-based Pruning**: the modern approach to weights pruning is based on iteratively eliminating the weights whose **absolute value** (magnitude) is smaller than a threshold. In some cases, more than 80% of the weights can be pruned without affecting accuracy. However, the caveat is that the model must be retrained to compensate for the pruned weights. In the magnitude-based approach, it is also possible to specify the desired percentage of weights to prune and then eliminate all weights with the lowest magnitude until that percentage is achieved [37].
- **Weights Saliency Pruning**: one of the earliest pruning approaches, which eliminates weights based on their impact on the loss function. The goal is to prune the weights that have the least impact on the value of the loss function [38].

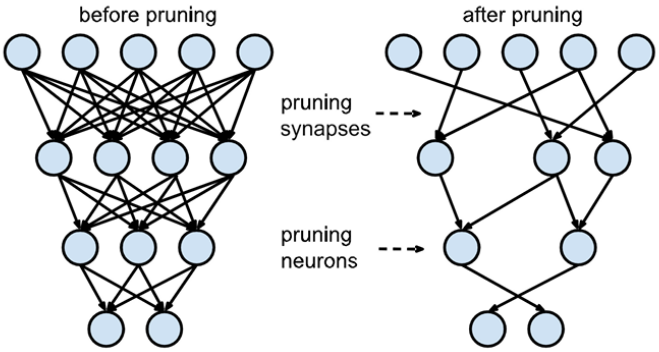


Figure 2.6: Weights and Activations pruning. [39]

In addition it is possible to classify the different weights pruning techniques based on the **granularity level**. There are two main approaches:

- **Unstructured Pruning**: it involves selectively removing individual weights from a neural network without following any specific pattern or structure. This approach offers greater flexibility, allowing the network to reduce its size while preserving performance, but it can be more challenging to optimize for hardware efficiency. In fact, the most important problems of this approach are that the number of non-zero elements in each row is *uneven* and their corresponding values are accessed with a random pattern, not allowing HW parallelism and caches. Therefore, even if it is theoretically possible skip some operations, in practice pruning may even make the model slower and less efficient.

- **Structured Pruning**: it focuses on removing entire groups of weights, such as neurons, channels, or layers, rather than individual ones. By maintaining a regular structure, this method is often more efficient for hardware optimization and easier to implement, though it may result in a larger performance drop compared to unstructured pruning. It can be applied at different granularities, targeting entire nodes, channels, or groups of weights, depending on the specific architecture and desired level of compression [40].

The last fundamental concept in exploiting pruning approaches is the **pruning schedule**. They are designed to manage the rate and extent of weight removal over time, optimizing the balance between efficiency and model performance. At the beginning, pruning is often easier due to the presence of numerous redundant weights, making it an ideal time to increase sparsity rapidly.
One commonly used schedule, polynomial decay, leverages this by accelerating the pruning rate initially and then gradually decreasing it as the pruning progresses.

Quantization

Quantization in neural networks refers to the process of reducing the precision of the weights and activations from floating-point representation to lower-bit formats, such as 8-bit integers [36]. One of the main benefits of using quantization is the reduction in model size, which has a significant impact from a storage perspective. While the advantages of quantization in terms of memory usage and computational efficiency are realized just when operations are performed directly in the reduced precision format, not always possible.

One of the most used quantization method is the *Integer Quantization*, which involves converting weights and activations to 8-bit integers, allowing for efficient computations on resource-constrained devices while maintaining an acceptable

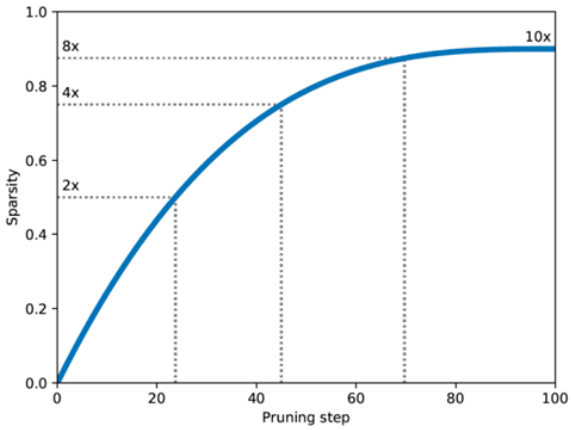


Figure 2.7: Polynomial Decay Pruning Schedule.

level of accuracy. This technique is particularly important for deploying models on embedded devices, especially those powered by microcontroller units (MCUs) that often lack a floating-point unit (FPU). In such cases, computations must be performed using the arithmetic logic unit (ALU), which accepts only integer numbers. Furthermore, operations executed by ALU are generally more efficient than those performed by FPU.

In this context, the most common approach for representing real number using an integer is the **Dynamic Fixed Point** representation. In this approach the term "fixed" refers to the fact that all elements of a tensor share the same *scaling factor* Δ and the same *zero offset* z. The term "dynamics" refers instead to the fact that different tensors could have different quantization parameters.

The integer value is obtained as follows:

$$integer\_value = round(\frac{real\_value}{\Delta} + z) \tag{2.12}$$

Note that the scaling factor and the zero offset have an impact on the the range of values that can be represented.

Understanding this representation is crucial for implementing quantization effectively, as there are two main strategies that differ in their application and timing:

- **Post-Training Quantization**: the model is trained using 32-bit precision and then weights (and optionally activations) are converted to a lower precision format [41]. The advantage is that it can be applied also to pre-trained models

without re-training and, in case of weights-only quantization, even without input data available.

- **Quantization-Aware Training**: the concept underlying this method, is avoiding accuracy drops by simulating quantization during training [42]. In this way, the optimizer can take into account the limited set of values that weights and activations can assume. It is necessary to simulate the effect of quantization (applying fake quantization) because real quantization could create problems during training. Particularly, in the back-propagation steps small weights updates would be impossible (values as 17.65 and 17.7 are indistinguishable), having as consequence the creation of a sticking point. The fake quantization is obtained as follows:

$$value_{fq} = \Delta * round(\frac{real\_value}{\Delta} + z) - z \tag{2.13}$$

  Despite the usage of fake quantization, one problem is still present: the gradient of the round() operation is zero in all differentiable points. It is solved with *Straight-Through Estimators (STEs)* [43]. In other words, using the fake quantization only in the forward pass and replace it with a differentiable function (e.g., identity) in the backward pass.

Mixed-Precision Quantization

The *Mixed-Precision Quantization* approach allows, unlike fixed-precision quantization, to assign different bit-widths to different layers in the neural network. This flexibility enables optimization tailored to the specific needs of each layer, as different layers may have varying levels of sensitivity to the quantization operation. By adapting precision based on the importance and function of each layer, it can achieve a more efficient balance between computational resources and model performance.

2.3 Infrared Sensors

Infrared (IR) sensors are devices capable to detect infrared radiation, i.e., a form of electromagnetic radiation emitted by objects in their surroundings. This radiation falls just beyond the visible spectrum, within wavelengths ranging from approximately 700 nanometers (nm) to 1 millimeter (mm). Although invisible to the human eye, infrared radiation is typically experienced as heat. This characteristic allows IR sensors to detect thermal variations and the presence of objects without requiring direct visual contact. As a result, IR sensors are widely employed in numerous applications, including environmental monitoring, thermal imaging, and motion detection systems.

2.3.1 Types of Infrared Sensors

Infrared sensors can be broadly categorized into two types: *active* and *passive* sensors.

1. **Active Infrared Sensors:** These sensors emit their own infrared radiation and detect the reflection or interruption of this radiation by nearby objects. They typically consist of two components:

   - **Emitter:** A source of infrared light, such as an IR LED.
   - **Detector:** A photodiode or phototransistor that senses the reflected IR radiation.

   Active sensors are commonly employed in proximity detection and short-range motion tracking.

2. **Passive Infrared Sensors:** In contrast to active sensors, these sensors do not emit any radiation. Instead, they detect infrared radiation naturally emitted by objects, particularly living beings, due to their body heat. PIR sensors are widely used in motion detectors and security systems, designed to sense the movement of objects that emit IR radiation.

2.3.2 Key Characteristics of Infrared Sensors

Several key characteristics determine the performance and applications of infrared sensors:

1. **Wavelength Range:** Infrared radiation spans a broad spectrum, and sensors operate in specific wavelength ranges:

   - Near-Infrared (NIR): 700 nm to 1400 nm

- Mid-Infrared (MIR): 1400 nm to 3000 nm
- Far-Infrared (FIR): 3000 nm to 1 mm

2. **Resolution:** Resolution refers to the sensor's ability to detect fine details. Low-resolution infrared sensors, commonly used in resource-constrained environments, have fewer pixels, which can make it difficult to detect precise shapes or small movements. These sensors, however, are often more affordable, consume less power, and are easier to integrate.

3. **Response Time:** This refers to how quickly the sensor can detect a change in infrared radiation. A fast response time is crucial for applications involving motion tracking.

2.3.3 Advantages and Limitations of Low-Resolution IR Sensors

Low-resolution infrared sensors offer several advantages, particularly for specific applications where cost, power consumption, and size are key considerations. One of the main benefits of these sensors is their low cost. They are generally more affordable than their high-resolution counterparts, making them well-suited for budget-constrained projects. Additionally, they consume less power, which is crucial in battery-operated systems or mobile devices. Their compact size is another advantage, allowing them to be easily integrated into devices with space constraints, such as wearable technology or embedded systems. Furthermore, low-resolution infrared sensors perform robustly in various lighting conditions, including low-light or dark environments, where visible light sensors might struggle.

However, these sensors also come with limitations. One significant drawback is their limited ability to detect fine details, as their lower resolution makes it challenging to differentiate between complex shapes. This can be particularly problematic in applications that require detailed recognition. Another limitation is their sensitivity to environmental temperature variations. Changes in ambient temperature can affect the sensor's performance, especially when the temperature of the surrounding environment is close to that of the objects being detected. Finally, low-resolution infrared sensors are more prone to noise and artifacts, which can complicate signal processing and require additional filtering techniques to achieve reliable results.

Chapter 3

Related Works

This chapter illustrates the main works conducted in the field of computer vision, focusing on both infrared (IR) and RGB images, as well as the most significant contributions related to efficient ML and DL methods.

3.1 Hand Detection Applications

Hand detection is a fundamental task in the field of computer vision and serves as a cornerstone to more advanced tasks such as hand tracking, gesture recognition, and hand-object interaction. Its importance is closely related to the wide range of applications in which it can be applied, including:

- **Human-Computer Interaction (HCI)**: In the field of HCI, hand detection enables touchless interfaces where users can interact with systems through natural hand movements. This technology is especially useful in scenarios where physical contact is impractical or undesirable, when minimizing contact became a priority [44].
- **Augmented Reality (AR) and Virtual Reality (VR)**: In AR and VR environments, hand detection is critical for creating immersive experiences. By accurately detecting and tracking hand movements, these systems allow users to interact with virtual objects as if they were manipulating physical ones. This has applications in gaming, training simulations, and collaborative design, where the ability to interact naturally with the virtual environment enhances the overall user experience [45].
- **Robotics and Automation**: In robotics, hand detection is used to enable robots to recognize and respond to human gestures or instructions. For example, in collaborative robotics, hand detection systems allow robots to

understand and react to human movements, facilitating safer and more intuitive human-robot interaction. This technology is particularly valuable in industrial settings, where robots and humans often work side by side [46].

- **Sign Language Recognition**: Hand detection plays a key role in sign language recognition systems, which aim to translate the movements of the hands and fingers into corresponding words or phrases. Accurate detection and tracking of hand shapes and positions are essential for these systems to function effectively, enabling communication for individuals who use sign language as their primary mode of expression [47].
- **Industrial Safety**: Hand detection represents a fundamental aspect in industrial environments, particularly for enhancing workplace safety. By integrating hand detection systems with machinery, immediate actions such as stopping operations can be triggered upon detecting a human hand in proximity. This approach not only increases safety levels but also serves as a significant tool for reducing repair costs associated with equipment damage and potential liabilities from workplace injuries.

To address this critical task in computer vision, a variety of techniques have been developed, incorporating both machine learning (ML) and deep learning (DL) methods. In the next sections, the key contributions and advancements made in the field of hand detection and efficient ML and DL will be reviewed.

3.2 Traditional ML Techniques Using RGB and IR Images

Traditional ML techniques refer to methods that, differently from DL, rely on handcrafted features. These techniques focus on the manual extraction and manipulation of raw data to identify patterns, rather than automatically learning representations from large dataset.

In this context, *the Histogram of Oriented Gradients* (HOG) feature descriptor has proven to be effective in various object detection tasks, including hand detection. HOG was first introduced by Dalal and Triggs [48] in their work on pedestrian detection, which demonstrated its robustness in capturing local shape and edge orientations. Due to its ability to effectively capture contours and shapes, HOG has been adapted and explored for various tasks in several studies. It has been used in combination with Support Vector Machines for vehicle classification [49] and for hand detection in real-time applications [50]. Additionally, HOG features have been applied to both RGB and infrared (IR) images. For example, [51] utilized HOG features for pedestrian detection based on infrared images.

Another widely used technique is *Local Binary Patterns* (LBP), a texture descriptor introduced by Ojala et al. [52]. LBP analyzes each pixel in an image by comparing its intensity with that of its neighboring pixels, generating a binary pattern based on whether each neighbor's intensity is greater or less than that of the central one. Its simplicity and computational efficiency make it particularly well-suited for real-time applications, even with low-resolution inputs.

Due to their foundational roles in feature extraction and pattern recognition, both HOG and LBP represent essential techniques from which various advanced variants have evolved. For instance, a combination of HOG and LBP features has been employed in the development of a hand gesture recognition method for mobile devices [53].

In addition to HOG and LBP, the *Scale-Invariant Feature Transform* (SIFT), proposed by Lowe in 1999 [54], has also gained prominence in hand detection tasks. SIFT detects key points and generates descriptors that are invariant to changes in scale, rotation, and illumination, making it particularly effective for detecting hands in varying poses and lighting conditions. For instance, studies have successfully applied SIFT to hand posture recognition and hand tracking in human-robot interaction, leveraging its ability to manage different orientations of the hand [55]. Additionally, in a different domain, SIFT has been exploited for fingerprint verification [56].

These methods, rooted in traditional machine learning, have laid the foundation for hand detection in environments where deep learning approaches are either computationally prohibitive or limited by small datasets. Each technique offers distinct strengths, i.e., HOG for capturing shape and edge features, LBP for texture encoding, and SIFT for handling transformations, making them valuable tools in a broad range of applications.

3.3 DL Techniques Using RGB and IR Images

In the field of Deep Learning, *Convolutional Neural Networks* are widely recognized as the primary architecture for computer vision tasks. Since the beginning with the influential work of Krizhevsky et al. on AlexNet [57], CNNs have consistently dominated a wide range of visual recognition challenges, including tasks like object detection, image classification, and human pose estimation. The hierarchical structure of CNNs, which allows them to learn low-level features such as edges and textures in early layers and high-level abstract concepts in later layers, makes them highly effective for image processing.

In this context, several architectures have been proposed over the years, such as YOLO, Faster R-CNN, and Inception which have become reference points for various applications, particularly considering the widespread adoption of transfer

learning methods.

- **YOLO**: It is known for its speed and efficiency. The core concept is to treat object detection as a single regression problem, where the model simultaneously predicts bounding boxes and class probabilities for each object in the image using a single CNN. [58]. By dividing the image into a grid and associating each grid cell with potential objects, YOLO can efficiently identify and localize multiple objects in real time, making it highly effective for dynamic environments. An example of a YOLO-based application is [59], which adopted a modified YOLOv3 model, called YOLO-Tomato, to detect tomatoes in complex environmental conditions.

- **Faster R-CNN**: It enhances the object detection pipeline by introducing a *Region Proposal Network* (RPN), which proposes candidate object bounding boxes and subsequently refines these proposals for classification and localization [60]. It has been used in different tasks including face detection as in [61] or object detection in medical images as in [62].

- **Inception**: it is a convolutional neural network architecture that employs a novel approach of using multiple filter sizes in parallel at each layer, allowing the model to capture features at various scales [63]. This design significantly enhances the network's efficiency while maintaining a relatively low computational cost. The applications range from traffic sign recognition [64] to breast cancer detection [65].

The applications mentioned represent only part of the broader range of use cases for these models. In the domain of Hand Detection, [66] explores the implementation of YOLO-based algorithms to identify hands in kitchen environments, leveraging a dataset derived from egocentric videos. Similarly, [67] investigates an innovative method for accurately detecting hands from single-color images, focusing on the challenges posed by the diverse appearances of human hands in cluttered scenes. The authors propose a hybrid detection and reconstruction framework based on a faster R-CNN architecture. However, it is important to note that despite their effectiveness for hand detection tasks, these models are complex and may be challenging to implement in resource-limited environments.

3.4 Efficient ML/DL Techniques

The defining characteristic of deep learning models is their ability to perform automated feature extraction, reducing the need for manual feature engineering. However, achieving high performance still heavily depends on the network's architectural design. This requirement often leads to manually crafted, over-parameterized architectures that, while effective for accuracy, introduce considerable challenges for deployment in resource-constrained environments due to their high computational and memory demands. This ongoing challenge has prompted significant research into techniques for *automating* and *optimizing* model design.

One promising solution to this challenge is *Neural Architecture Search* (NAS), which automates the process of designing optimal model architectures. Early approaches to NAS used reinforcement learning to explore the vast space of possible architectures [33] or evolutionary algorithms [68]. However, these methods were often computationally expensive. To overcome this, recent advancements have introduced Differentiable NAS (DNAS), which significantly reduces the search cost by making the architecture search process continuous and differentiable. Among DNAS methods, DARTS (Differentiable Architecture Search) [34] is particularly notable, as it enables architectures to be optimized using gradient descent, greatly enhancing efficiency compared to earlier NAS approaches.
Moreover, approaches like DMasking-NAS have refined DNAS by introducing masking mechanisms that further reduce computational overhead while maintaining search effectiveness [69]. In this case, differentiable trainable masks are used to tweak layers' parameters in order to explore different architecture during the search process. Several studies have shown the successful deployment of NAS-designed architectures in constrained environments, demonstrating that NAS not only automates architecture design but also produces models that are optimized for both accuracy and efficiency. For example, [70] explores the use of a lightweight DNAS method that learns independent precision assignments for each weight tensor channel in convolutional or fully connected layers, with the goal of developing deep neural networks capable of inference on constrained edge nodes. Additionally, [71] employed DNAS techniques to address the challenge of PPG-based blood pressure estimation in low-power wearable devices.

Naturally, NAS and its various implementations form the foundation for effective model design and optimization. They are often integrated with other optimization techniques, such as pruning and quantization, to further enhance model performance.

To the best of our knowledge, no previous work has considered the use case of hand detection on low-resolution infrared sensors.

Chapter 4

Methods

In this chapter, the key stages of the study will be outlined and discussed in detail, highlighting the approach taken, the methodologies applied, and the overall progression towards achieving the thesis objectives. Each step will provide insights into the decision-making process and the practical implementation of the work's core components.

The primary goal of this study is to develop efficient Deep Learning models capable of delivering strong performance in the task of hand detection with low-resolution infrared sensors.

4.1 Dataset Collection

The first step, before proceeding with the development of both ML and DL models, is the *Dataset Collection*, which forms the foundation for the entire modeling process. Without a well-constructed dataset, achieving accurate and effective model performance would not be feasible.

It is important to emphasize that in this work, the dataset was created entirely from scratch, without exploiting any pre-existing dataset.

4.1.1 IR Sensor, Raspberry Pi, Data Acquisition Script

The dataset consists of IR frames acquired using the Panasonic Grid-EYE (AMG8833) sensor. This sensor outputs an 8x8 array of IR data and has a 60° field of view [72]. To gather data, a system was implemented using a Raspberry Pi 3 connected to the Grid-EYE sensor. Data acquisition was automated with a Python script, enabling IR data collection at 10 frames per second in tabular format as CSV file. Subsequently, the tabular data was converted into images to serve as the final format for machine learning and deep learning model input.

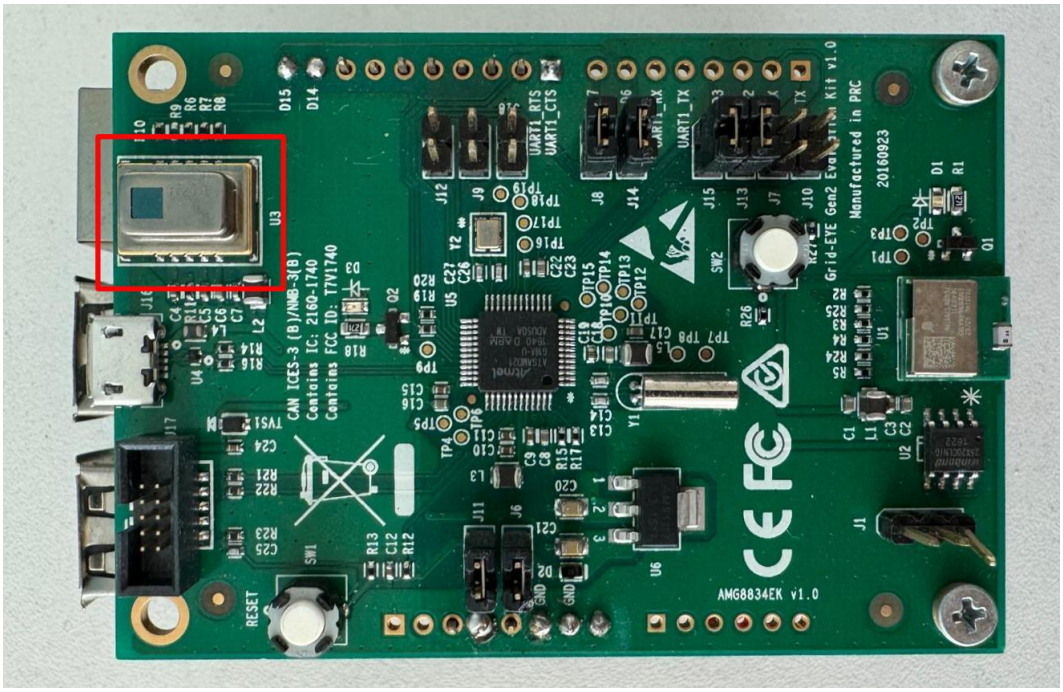


Figure 4.1: Panasonic Grid-EYE sensor (highlighted in red) integrated into Raspberry Pi 3.

4.1.2 Classes Identification

One of the most critical steps in the data collection process is identifying the appropriate classes, while keeping the final objective and primary use case in mind. In this work, four main classes have been identified:

1. **Empty Class**: This class represents images that contain no relevant content, just background.
2. **Hand Class**: This class includes all images that feature a hand. The hand may appear in various positions and can be located anywhere within the image.
3. **Object Class**: This class refers to images that contain warm objects. Similar to the Hand Class, these objects may be placed anywhere in the image. The specific objects used will be detailed in the next subsection.
4. **Hand with Object Class**: This class captures images that contain both a hand and a warm object interacting with each other.

4.1.3 Object Selection for Dataset

Once the classes were defined, the next step involved selecting the warm objects to be used for both the Object and Hand with Object classes. The goal was to choose objects that varied in terms of temperature, size, and shape, ensuring a more diverse dataset and contributing to the development of a robust model. Four specific objects were chosen for this purpose: a ***Coffee Cup***, a ***Glass***, a ***Small Pot***, and a ***Hair Diffuser***

The selection of these objects provided a range of temperature levels and different rates of heat diffusion, offering greater variability in the dataset. This variability was essential for improving the model's ability to generalize across different scenarios and object types, contributing to its overall effectiveness in detecting hands and warm objects in infrared images.

4.1.4 Image Collection and Dataset Structure

The image collection process was meticulously organized across six distinct sessions, spread over three days, with two sessions per day. During each session, images for all four identified classes were captured.

To maintain consistency and create a controlled environment, special attention was given to keeping the background temperature constant throughout the collection process. This decision was made to simulate a scenario in which the hand and objects could be clearly distinguished from the background, optimizing the conditions for hand detection in low-resolution infrared images. Furthermore, the infrared camera was positioned at a fixed distance from the objects in all sessions, ensuring uniformity in image composition across the dataset.

While maintaining a constant background temperature and a fixed camera distance may reduce variability, potentially making the task easier, these decisions were deliberate. The priority in this project was to establish a standardized baseline for hand detection using infrared images. By reducing unnecessary variability, this baseline provides a solid foundation for future works. Once this standard is well-defined and tested, the next step could involve gradually introducing more complexity, such as varying background temperatures or camera distances, to further challenge the model and enhance its robustness.

Moreover, it is important to note that if the background temperature were similar to that of the hand or object, it would be impossible to distinguish between them in the infrared images. For this reason, maintaining a distinct temperature difference was crucial to ensure clear detection. This controlled setup thus strikes a balance between making the task achievable while providing a realistic and practical scenario for the model to learn from.

To effectively structure the dataset, the six sessions were distributed as follows:

- **Training Set**: Images from four sessions were allocated to the training set. This provided a substantial and diverse dataset for the model to learn from, helping it to capture a wide range of variations in the different classes.
- **Validation Set**: One session was reserved exclusively for the validation set. This allowed for the fine-tuning of the model's parameters during the development phase, without exposing it to any images from the training set, ensuring unbiased feedback on model performance.
- **Test Set**: The images from the final session were used exclusively for the test set. This ensured that the test data remained completely independent, enabling a rigorous evaluation of the model's ability to generalize to new, unseen data.

The decision to allocate entire sessions to each dataset partition, rather than randomly splitting images from a single session into training, validation, and test sets, was made to avoid the risk of using consecutive frames in different phases of the model's development. Images captured within the same session tend to be more similar, as they show high temporal correlation. This similarity between frames could lead to a scenario where nearly identical images appear in both the training and test sets. Such overlap would compromise the integrity of the testing process, giving the model an artificial advantage by allowing it to generalize across very similar images.

By separating the sessions in this way, the training, validation, and test sets each contain unique data, ensuring that the test set provides a more accurate assessment of the model's true performance in real-world applications, where it must handle entirely new inputs without relying on previously encountered frames.

The tables below show the detailed composition of the Training, Validation, and Test sets, including the number of images for each class.

Class	**Number of Samples**
Empty	1734
Hand	5000
Object	4960
Hand With Object	5001

Table 4.1: Training Set

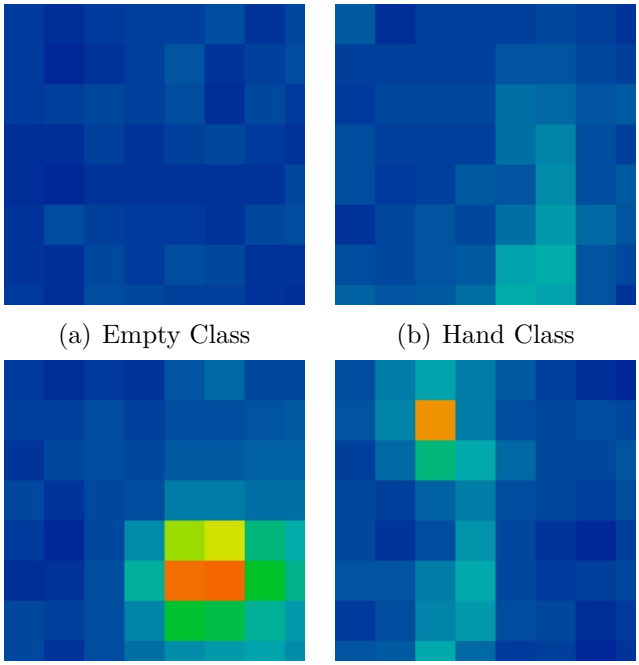
Class	Number of Samples
Empty	1500
Hand	1500
Object	3173
Hand With Object	3000

Table 4.2: Validation Set

Class	Number of Samples
Empty	1400
Hand	1500
Object	2217
Hand With Object	2884

Table 4.3: Test Set

The figures below present one representative 8x8 pixel image from each class. These images highlight the variation in data collected and the distinguishing characteristics of each class.



(a) Empty Class (b) Hand Class

(c) Object Class (d) Hand with Object Class

Figure 4.2: Images representing the four different classes.

4.2 OpTuna Framework

As mentioned in the section 2.1.6, hyperparameter tuning represents one the most crucial aspects in ML and DL since it is essential for optimizing model performance. Among the several alternatives, the *OpTuna* framework [73] has been chosen for this study. This section outlines the key features of OpTuna and provides a detailed examination of its default sampler, the *Tree-structured Parzen Estimator (TPE)*.

4.2.1 Main characteristics

Optuna is a sophisticated hyperparameter optimization framework designed to streamline the process of tuning machine learning and deep learning models.
One of its primary characteristics is its *flexible and intuitive API*, which allows users to define optimization problems as Python functions. This flexibility enables seamless integration with various machine learning libraries, making it accessible for both novel and experienced practitioners.

Optuna organizes the optimization process into *Studies* and *Trials*. A study serves as a container for a collection of trials that aim to optimize a specific objective function. Each trial represents a single execution of this objective function with a distinct set of hyperparameters, providing valuable performance feedback that informs subsequent trials. This structured approach facilitates efficient management of hyperparameter configurations and their associated performance metrics.

A standout feature of Optuna is its *pruning mechanism*, which enhances the efficiency of the optimization process. By monitoring trial performance in real-time, Optuna can identify underperforming trials and terminate them early, conserving computational resources and accelerating the overall search for optimal hyperparameters. This is particularly beneficial when working with complex models that require significant training time.

In addition, Optuna supports *multi-objective optimization*, allowing users to optimize multiple performance metrics simultaneously. This capability is crucial in scenarios where trade-offs between different objectives must be considered, enabling a more comprehensive evaluation of model performance. Users can define complex search spaces that accommodate various hyperparameter types, including integers, floats, and categorical variables, providing the flexibility needed for diverse modeling scenarios.

Optuna also offers built-in *visualization tools* that facilitate the analysis of optimization results. Users can generate visualizations such as optimization history, parameter importance, and parallel coordinate plots, which aid in understanding the relationships between hyperparameters and their impact on model performance. These visualizations not only enhance the interpretability of the optimization process but also assist in identifying patterns that can inform future modeling.

4.2.2 The Tree-structured Parzen Estimator (TPE)

The *Tree-structured Parzen Estimator* (TPE) is the default sampling algorithm used by Optuna for hyperparameter optimization. As a Bayesian optimization method, TPE models the distribution of the objective function in a non-parametric manner, leveraging past evaluations to inform future hyperparameter selections effectively. TPE excels in optimizing complex, high-dimensional search spaces by focusing the search on promising regions, significantly improving efficiency over traditional optimization methods like grid or random search [74].

TPE operates on the principle of probabilistic modeling and consists of the following key steps:

1. **Modeling the Objective Function**: TPE constructs two probability density functions (PDFs): one for promising configurations and another for unpromising ones. This dual modeling approach allows TPE to effectively distinguish between configurations based on their observed performance, resulting in a more informed optimization process. Specifically, the algorithm aims to maximize the ratio of these PDFs, $\frac{l(x)}{g(x)}$, where $l(x)$ represents configurations that perform well and $g(x)$ represents those that perform poorly.

2. **Parameter Selection**: For each new trial, TPE computes the expected improvement by selecting hyperparameters that maximize the ratio of the PDF for promising configurations to the one for unpromising ones. This selection strategy enables TPE to focus the search on areas of the parameter space that have historically led to better results, which enhances the likelihood of discovering optimal configurations.

3. **Tree Structure**: The term "tree-structured" refers to TPE's ability to efficiently handle hierarchical relationships between hyperparameters. In optimization problems where certain hyperparameters depend on the values of others (i.e., conditional hyperparameters), TPE builds a tree-like structure to model these relationships. The tree structure enables TPE to guide the optimization process by adaptively selecting which hyperparameters to evaluate based on past trials, considering the interdependencies between them. The tree structure not only helps in reducing the search space but also improves efficiency by focusing the search on the most promising hyperparameters at each node, effectively pruning the less relevant regions of the parameter space.

TPE offers several advantages over traditional optimization methods, including:

- **Sample Efficiency**: TPE is designed to be more efficient than methods such as grid search or random search. It achieves better results with fewer evaluations, making it particularly suitable for high-dimensional search spaces where each evaluation can be computationally expensive.

- **Adaptive Learning**: TPE learns from the outcomes of previous trials, continuously refining its sampling strategy to improve performance over time. This adaptability allows TPE to respond dynamically to the information gathered during the optimization process, focusing on the most promising regions of the search space.
- **Exploration and Exploitation Balance**: TPE effectively balances the exploration of the search space with exploitation of areas that have shown high performance. By focusing on promising regions and adjusting based on past evaluations, TPE accelerates convergence toward optimal solutions while minimizing computational cost.

4.3 Traditional ML Models

The initial phase of the experiments focused on traditional ML techniques, specifically SVM and RF. This section outlines the key aspects involved in the development of these models, taking into account that they serve as baseline for the next steps.

4.3.1 SVM - RF

For this work, SVM and RF have been identified as the traditional machine learning models used to define the baseline metrics. As with every traditional ML model, a key phase is the feature extraction process, which is crucial for performance. In this study, the Histogram of Oriented Gradients features described in Section 2.3.3 have been employed. Additionally, given the limited resolution of the images (8x8 pixels), raw pixel values were also tested as an alternative input for the classifiers.

For both the HOG and raw pixel representations, hyperparameter tuning was conducted for each classifier using the Optuna framework. This approach allowed for the systematic exploration of a broad hyperparameter space, ensuring that each classifier was optimized for its respective feature set.

The following lists present the fine-tuned hyperparameters and their respective search spaces for SVM and RF:

SVM Hyperparameters and Search Space:

- Regularization Parameter (C): 0.00001 - 100
- Kernel: linear, rbf, poly, sigmoid
- Gamma: scale, auto

RF Hyperparameters and Search Space:

- Number of Estimators: 10 - 100
- Maximum Depth: 2 - 32
- Minimum Samples Split: 0.1 - 1

It is important to underline that the Optuna optimization process was driven by validation accuracy, highlighting our focus on achieving robust model performance.

4.4 Deep Learning Models and their Optimization

After establishing a benchmark using SVM and RF, the next step involved developing a Convolutional Neural Network to assess whether transitioning from traditional machine learning models to more complex deep learning ones is justified in terms of accuracy improvements. In the initial phase, the CNN was constructed without prioritizing model efficiency, focusing solely on accuracy. The Optuna framework was employed also in this phase to conduct hyperparameter optimization and determine optimal parameter values. Two main approaches were explored. In the first approach, the number of layers and their respective channel counts were fixed, and the following hyperparameters were fine-tuned:

- Learning rate: 0.00001 - 0.1
- Dropout rate: 0.1 - 0.5

In the second approach, in addition to the hyperparameters specified above, the Optuna framework also optimized the number of channels for each layer. Specifically, for each of the three convolutional layers and the fully-connected one, the range varied between 32 and 128.
Due to the intrinsic capacity of deep learning models to automate feature extraction, raw images were directly input into the CNN without the need for extensive preprocessing.

This phase demonstrated that CNNs significantly enhance detection accuracy, suggesting they may be a superior choice for achieving the objectives of this study. These findings, which will be further detailed in 5.3.1, allowed us to proceed to the next stage: addressing additional challenges related to model complexity. The focus then shifts to optimizing the models by analyzing performance from the perspectives of energy consumption and memory usage, incorporating two main optimization techniques described in the following sections.

4.4.1 Regularizer

Before discussing the methods employed to design both effective and efficient models, it is important to address how model complexity was incorporated into the CNN training process. To achieve this, a model cost function that explicitly considers model complexity was defined. This complexity metric was then integrated as an optimizable parameter within the training loop, as outlined in Section 2.2.4.

In this context, two different alternatives exploit the loss computation during training by adding an additional term to the task-specific loss function:

- **Base Regularizer**: This approach incorporates model complexity directly into the loss function using the following term:

$$\lambda \times \text{cost} \tag{4.1}$$

  The parameter λ (known as strength) controls the weight given to model complexity in the optimization process. This regularizer enables the CNN to be optimized with model complexity taken into account. A limitation of this regularizer is its inability to set a minimum threshold for model complexity; in other words, it cannot ignore complexity considerations below a certain level. This can be problematic when strict model complexity requirements must be met.

- **DUCCIO Regularizer** [75]: This regularizer acts as a soft constraint on model complexity, making it suitable for cases where the Base Regularizer is insufficient. It is defined as:

$$\max(0, \text{cost} - \text{target}) \tag{4.2}$$

  Unlike the Base Regularizer, the DUCCIO Regularizer only factors model complexity into the loss function when the model does not meet the defined target complexity constraint.

In this study, the DUCCIO Regularizer was preferred because setting the target value makes it possible to focus on specific constraint requirements.
Regarding model cost metrics, among the various available options, two were employed in this work: the *number of model parameters* (params) and the *number of bits required to store the model's weights* (params_bit) computed as follows:

Standard 2D Convolutional Layer

$$params = C_{in} * C_{out} * K_x * K_y \tag{4.3}$$

$$params\_bit = C_{in} * C_{out} * K_x * K_y * w_{precision} \tag{4.4}$$

Fully-Connected Layer

$$params = C_{in} * C_{out} \tag{4.5}$$

$$params\_bit = C_{in} * C_{out} * w_{precision} \tag{4.6}$$

where, C_{in} and C_{out} represent the number of input and output channels of the considered layer, respectively; K_x and K_y denote the width and height of the kernel, while $W_{precision}$ indicates the number of bits used to represent each weight.

4.4.2 PIT

The first approach employed is *Pruning In Time* (PIT) [76]. It is a mask-based DNAS method focused on identifying the optimal number of channels for convolutional and fully-connected layers in a NN. Starting from a seed network, it prunes through a structured approach the weights with the lowest contribution to the model task performance, i.e., detection performance in the scope of our study. Additionally, PIT allows the definition of a model cost, which, when introduced as an additional term in the loss function, acts as a regularization factor, as those described in the previous section. This enables the creation of models that balance detection accuracy with model complexity. In this case, the model cost is represented by the total number of parameters and it has been combined with the DUCCIO regularizer (Eq. 4.2). Similar to traditional deep learning training loops, PIT has various hyperparameters that require tuning. In this study, this step is managed using the Optuna framework, in order to improve resource efficiency by analyzing only the most promising configurations. This approach helps to optimize both time and computational resources during training. In contrast to the initial phase, where model complexity was not a focus, the Optuna optimization process in this phase is driven by both validation accuracy and adherence to a model cost constraint. Specifically, at the end of each Optuna trial, the validation accuracy is adjusted by a penalty factor that considers whether the model complexity constraint has been met. The greater the gap between the obtained and target values, the larger the penalty applied. Similar to the concept of the DUCCIO regularizer, if the constraint is satisfied, the penalty term is set to zero.

The figure 4.3 illustrates the complete optimization process, which integrates PIT with the Optuna framework.

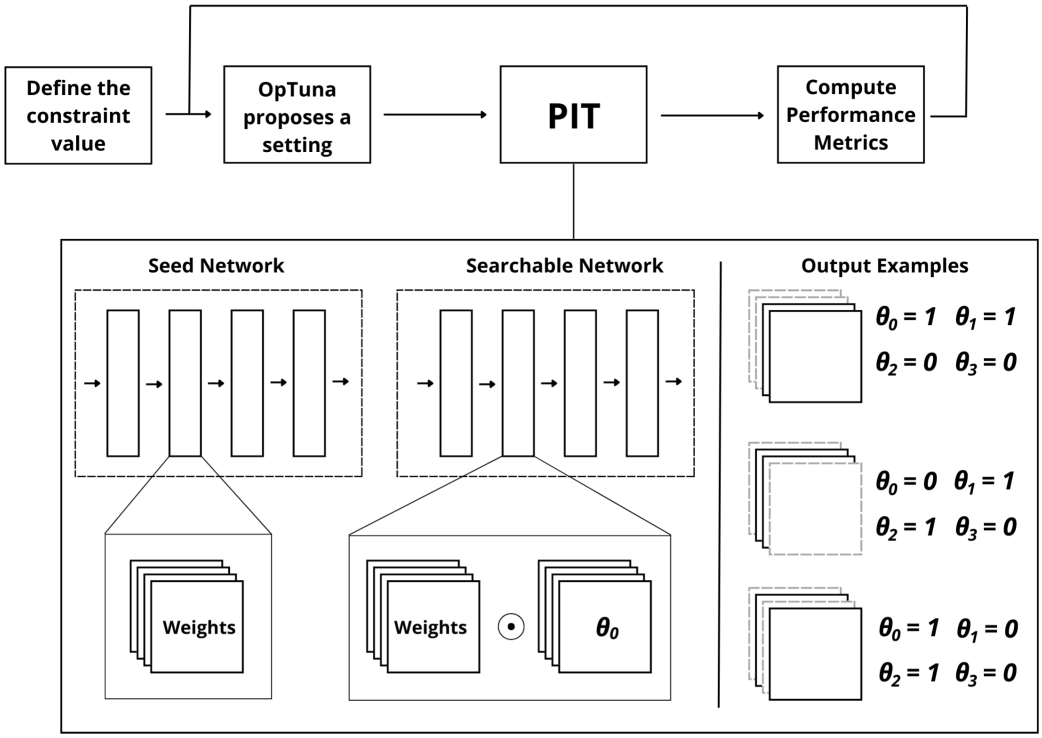


Figure 4.3: PIT in combination with Optuna.

4.4.3 MPS POST PIT

The second method employed is *Mixed-Precision Search* (MPS). It is a gradient-based tool that automates the exploration and assignment of integer quantization precision across various components of a deep neural network. Specifically, MPS independently determines and assigns precision levels for the weights and activations of convolutional and linear layers. An important feature of MPS is its ability to perform standard Quantization-Aware Training (QAT) [42] in cases where a single precision is specified, as illustrated by the use case in this study.

Starting from the optimal solutions (one for each constraint value) obtained via the PIT method, 32 distinct quantization configurations were applied to each solution. Specifically, this study investigates the 4-bit and 8-bit quantization formats. This method enabled consideration of model complexity in different terms,

transitioning from the number of parameters to the number of bits used to store them.

The Optuna framework was used to optimize the hyperparameters for MPS, focusing on a single architecture and a limited set of quantization configurations. The hyperparameter values identified were then consistently applied across various initial architectures and quantization configurations. Unlike the PIT method, the Optuna optimization process is guided solely by validation accuracy, as both the architecture and quantization settings are pre-defined.

This method facilitated an evaluation of whether it is more advantageous to start with a larger model and apply more aggressive quantization, or to begin with a smaller model and use a less restrictive quantization, as will be described in detail in the next chapter.

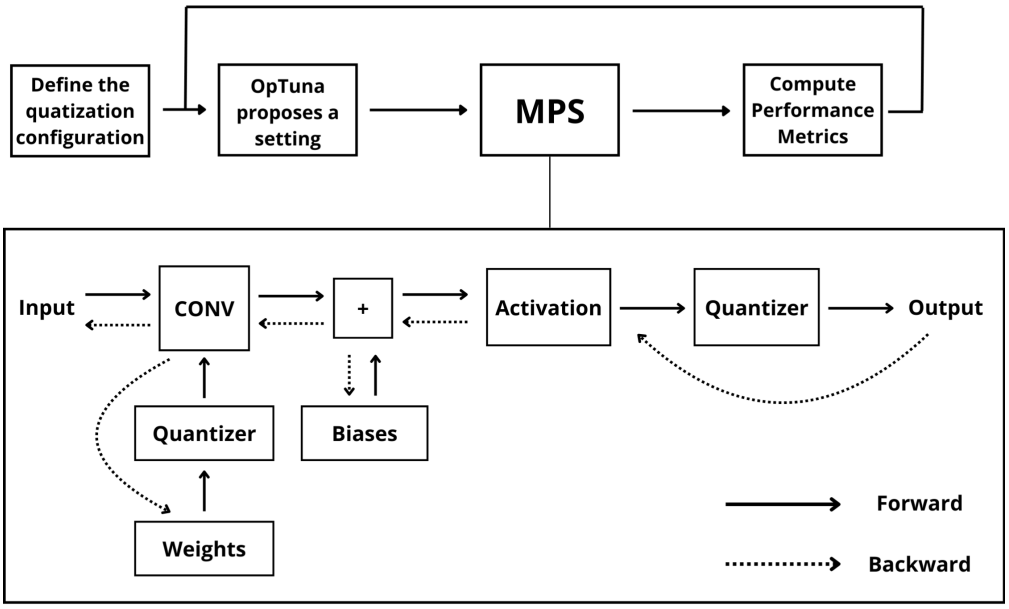


Figure 4.4: MPS as QAT tool in combination with Optuna.

4.4.4 PIT + MPS

The *PIT + MPS* method combines the preceding approaches in sequence, in an unique optimization process. This method employs Optuna not only to identify optimal hyperparameter values but also to strategically guide the optimization process toward an effective balance between pruning intensity and quantization precision. Specifically, this approach explores whether it is more advantageous to apply aggressive pruning early on in the PIT phase, followed by smoother

quantization in the MPS stage, or vice versa - a process in which a lighter pruning step is followed by a more restrictive quantization. By combining these strategies, PIT + MPS can effectively tailor the model to meet specific model size constraints without compromising accuracy.

In more detail, given a fixed model size constraint expressed in terms of number of bits required to store the model's weight, Optuna's role is to determine whether a larger model with more parameters from the PIT step, paired with a more restrictive quantization in the MPS phase, yields better results than a smaller model produced by more intensive pruning, which would allow for a less restrictive quantization setting.

By integrating Optuna directly into the optimization process, PIT + MPS avoids unpromising configurations early in the search and improves the efficiency of the resource allocation throughout the optimization. Similar to the MPS POST PIT method described in the previous section, this method enables the evaluation of model complexity in terms of the number of bits required to store the model's weights, facilitating a comparison between the two methods.

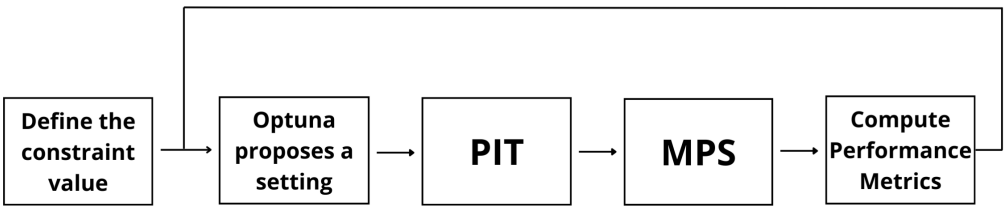


Figure 4.5: PIT + MPS method.

The figure 4.5 illustrates that, in this case, the PIT and MPS methods are employed within a single optimization process.

Chapter 5

Experimental Results

This chapter presents an analysis of the performance of both traditional machine learning and deep learning methods, highlighting the challenges encountered. Additionally, the effectiveness of the three methods introduced in Chapter 4 is evaluated in terms of reducing model complexity while maintaining high detection accuracy, with comparisons made across these approaches.

5.1 Experimental Setup

- **Programming environment:**
  - Python version 3.10.12
  - Traditional ML models: implemented using `scikit-learn` version 1.5.2 [77]
  - DL models: implemented using `PyTorch` version 2.5.1 [78]
- **Fine-tuning:**
  - Hyperparameter optimization performed with `Optuna` version 4.1.0 [73]
- **Optimization methods:**
  - `PIT`, `MPS`, and the `DUCCIO` regularizer were imported from the `PLiNIO` library [76]
- **Convolutional Neural Networks (CNNs):**
  - Trained with the `Adam` optimizer [13]
  - Training details:
    - **Number of epochs:** 10

    - **Batch size:** 32
    - **Learning rate scheduler:** `ReduceLROnPlateau` with factor equal to 0.3 and patience equal to 2 [79]

- **Task Performance metric:**
  - Accuracy
- **Execution environment:**
  - Experiments were conducted on `Google Colab` using a CPU

5.2 Traditional ML Techniques

As mentioned in the previous chapter, the first phase has been dedicated to assess the performance of traditional ML methods in order to define baseline metrics, in particular SVM and RF have been tested. The tables below present the results obtained by using both the HOG features and the raw input values as input. In particular, the reported Average Test Accuracy is obtained by first determining the optimal hyperparameter values through the Optuna framework and then using these values to train and test the classifier across 30 different random seeds to ensure a robust assessment of performance and stability.

RF Hyperparameters and Search Space:

- Number of Estimators: 10 - 100
- Maximum Depth: 2 - 32
- Minimum Samples Split: 0.1 - 1

SVM Hyperparameters and Search Space:

- Regularization Parameter (C): 0.00001 - 100
- Kernel: linear, rbf, poly, sigmoid
- Gamma: scale, auto

Num of Optuna Trials	Accuracy HOG	Accuracy Raw Pixels
50	46.28 +/- 0.55	79.23 +/- 0.43
100	46.00 +/- 0.40	78.95 +/- 0.86
150	46.44 +/- 0.17	79.63 +/- 0.24
200	46.56 +/- 0.11	79.22 +/- 0.63

Table 5.1: Results for RF

Num of Optuna Trials	Accuracy HOG	Accuracy Raw Pixels
50	69.08 +/- 0.05	85.39 +/- 0.38
100	69.09 +/- 0.06	85.31 +/- 0.24
150	69.10 +/- 0.05	85.31 +/- 0.24
200	69.10 +/- 0.05	85.18 +/- 0.02

Table 5.2: Results for SVM

Three main observations can be made:

- The performance achieved using SVM significantly outperforms that of RF classifier, regardless of whether HOG features or raw pixel values are used as input.
- For both classifiers, detection accuracy is higher when raw pixel values are used as input. This could be attributed to the small size of the images, which may limit the effectiveness of HOG descriptors in capturing the necessary shapes for accurate classification.
- The number of Optuna trials does not appear to have a significant impact on performance. The test accuracy values exhibit only minor variations, and no clear pattern can be identified.

5.2.1 Optimal Configurations

Varying the number of Optuna trials did not yield a single optimal configuration for the RF and SVM models. However, valuable insights were extracted from the results.
For RF, the optimal configurations spanned the entire search space, with the number of estimators ranging from 14 to 99 and the maximum tree depth between 9 and 32. In contrast, the minimum samples split consistently centered around 0.1, suggesting that this parameter had a greater impact on performance compared to the other two hyperparameters. These findings were consistent regardless of whether HOG features or raw pixel values were used as input.
For SVM, the situation differed. When using HOG features, two main configurations emerged: the first with a regularization term around 0.8 and a polynomial kernel, and the second with a regularization term around 5 and an RBF kernel. These findings changed when raw pixel values were used as input, where the RBF kernel was predominantly associated with a regularization term around 0.7. The polynomial kernel also appeared, though less frequently, with a regularization term around 3.

5.3 DL Techniques

After defining the benchmark using SVM and RF, the next step was the development of DL models, specifically CNNs. The deep learning phase is structured into two stages. The first stage aims to evaluate whether CNNs can achieve better detection accuracy compared to traditional machine learning models. The second stage focuses on improving the efficiency of the deep learning models in terms of resource utilization, while assessing the impact of this optimization on their predictive capabilities.

5.3.1 CNN

The following table presents the results obtained using CNNs. The investigated CNN architecture consisted of three convolutional layers, with the second and third layers followed by max pooling, and two fully connected layers. In addition to test accuracy, the table reports the number of model parameters and the hyperparameters explored by Optuna. For each experiment, the number of Optuna trials was set to 30, which was sufficient for convergence.

Test Accuracy	**N. of Parameters**	**Hyper-parameters Explored**
91.17	149504	Learning Rate (0.00001 - 0.01) Dropout Rate (0.1 - 0.5)
89.40	65748	Learning Rate (0.00001 - 0.01) Dropout Rate (0.1 - 0.5) N. of layers' channels (32 - 128)

It is possible to observe substantial improvements in detection accuracy compared to traditional ML techniques. Specifically, accuracy increases of approximately **5%** and **10%** have been noted when compared to SVM and RF, respectively.

5.3.2 PIT

The PIT method represents the first approach employed in the model resource optimization phase. As detailed in Section 4.4.2, the PIT method applies structured channel-wise pruning, starting from a seed network. In the experiments conducted, the seed network consists of three convolutional layers followed by two fully connected layers. The initial number of channels for each convolutional layer is set to 128, with the same number of units in the first fully connected layer.

The experiments were conducted as follows:

1. Twelve distinct values for the number of model parameters were identified to be passed as soft-constraints to the PIT method, which will exploit the DUCCIO regularizer features to meet the specified requirements.
2. For each constraint value, Optuna was run for a total of 100 trials to determine the optimal hyperparameter values and architecture topology for the specific target.
3. Once the optimal network topology was identified, it was fine-tuned for a total of 8 epochs across 10 different random seeds. This was done to obtain more robust results, including accuracy and standard deviation computations.

Before discussing the performance in terms of accuracy achieved, the table below presents the optimal network topology found for each of the different constraint values. The column related to the Architecture information reports the number of channels for each of the five layers.

Params Target	**Params Value**	**Architecture**
250	230	2 - 1 - 4 - 19 - 4
500	372	4 - 1 - 4 - 31 - 4
750	549	7 - 1 - 4 - 45 - 4
1000	453	3 - 2 - 6 - 30 - 4
1500	1418	38 - 1 - 39 - 13 - 4
2000	1083	9 - 1 - 14 - 45 - 4
2500	2098	24 - 1 - 33 - 40 - 4
3000	2772	34 - 1 - 36 - 49 - 4
4000	3552	6 - 2 - 40 - 68 - 4
5000	3513	32 - 1 - 44 - 56 - 4
10000	8295	11 - 2 - 68 - 102 - 4
15000	4890	29 - 2 - 43 - 81 - 4

Table 5.3: PIT

As shown in Table 5.3, it can be observed that the second convolutional layer consistently has the fewest number of channels, while the first fully connected layer has the highest number of channels for each of the parameter target values. Importantly, the last layer maintains a fixed number of channels equal to 4 for all constraint values, as it represents the number of output classes and is therefore immutable.

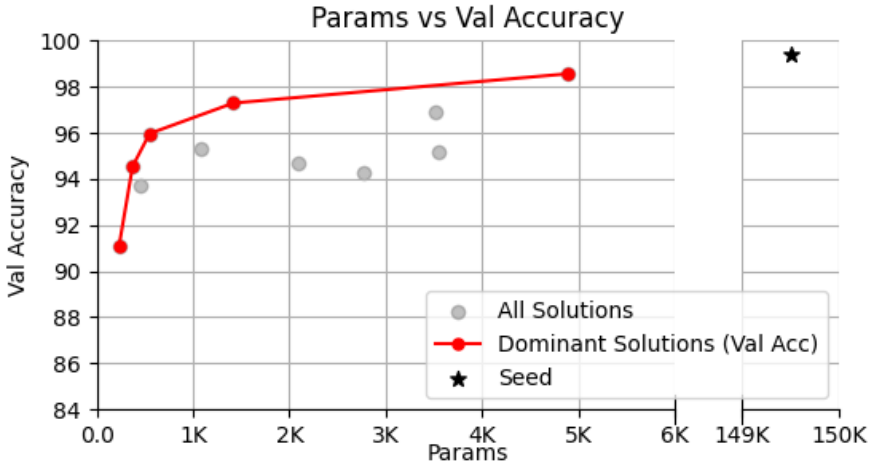
From the obtained optimal models, the fine-tuning has led to the results showed in the table 5.4. It can be observed that, although the trend is not particularly

pronounced, an increased number of parameters is associated with a greater discrepancy between validation and test accuracy. This outcome was anticipated, as the Optuna optimization process is driven by validation accuracy, and larger models tend to be more prone to overfitting.

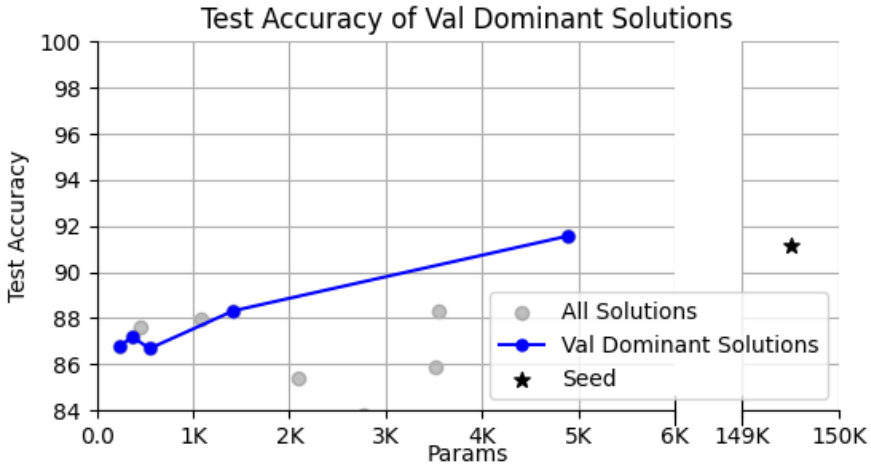
Params Value	**AVG Val Accuracy**	**AVG Test Accuracy**	**Δ(Val,Test)**
230	91.04	86.78	4.26
372	94.54	87.17	7.37
453	93.73	87.60	6.13
549	95.96	86.68	9.28
1083	95.33	87.93	7.40
1418	97.28	88.32	8.96
2098	94.68	85.37	9.31
2772	94.25	83.77	10.48
3552	95.17	88.31	6.86
3513	96.88	85.85	11.03
4890	98.55	91.56	6.99
8295	98.01	86.21	11.80

Table 5.4: PIT Results

In order to better assess the effectiveness of the PIT method, Figure 5.1 illustrates the impact of reducing the number of parameters on detection accuracy. In particular, the first plot displays the Pareto frontier in the Validation Accuracy vs. Number of Parameters space, while the second one illustrates the test accuracy values of validation-dominant solutions.
From the graphs, it is evident that reducing the number of model parameters impacts detection accuracy. Specifically, for validation accuracy, a transition from the largest validation-dominant configuration to the smallest one yields a compression ratio of **21x**, resulting in a **7.5%** drop in accuracy.
Interestingly, a more favorable outcome is showed in the second plot; the same compression ratio leads to only a **4.8%** decrease in test detection accuracy. These findings suggest that while parameter reduction significantly affects validation performance, it has a more limited impact on generalization, as observed in test accuracy.
The results show a greater degree of improvement when comparing the PIT-optimized models to the seed network. In this case, a similar drop in accuracy corresponds to a compression ratio of **500x**.



(a) Number of Parameters vs Validation Accuracy



(b) Test Accuracy values of Validation-dominant Solutions

Figure 5.1: PIT Pareto charts in the Params vs. Accuracy space.

5.3.3 MPS POST PIT

The following part of the experiments focused on evaluating the effectiveness of quantization in reducing model size while maintaining high accuracy, using the MPS method. As introduced in Section 4.4.3, MPS has been used exclusively as a QAT tool for this study. Additionally, it is important to note that the hardware target considered in this work only supports operations between tensors of the same precision.

For the MPS POST-PIT phase, the experiments were structured as follows:

1. Starting from a single optimal architecture obtained through the PIT method, 25 trials of the Optuna optimization process were conducted to identify optimal hyperparameter values for MPS. Eight different quantization settings were tested to gather robust data.
2. The optimal hyperparameters identified in the previous step were then used to apply MPS to the 12 architectures derived from the PIT method, testing 32 different quantization combinations for each of them. This process was repeated twice with two different random seeds to validate consistency in the results.

The following plots illustrate the test accuracy values for the validation-dominant solution across each starting architecture. During this initial phase, Pareto fronts based on validation accuracy values were calculated individually for each architecture to enable a detailed assessment of their potential.
The X-axis represents the average number of bits used to encode the model's weights, with the quantization strictly limited to 4-bit and 8-bit formats.

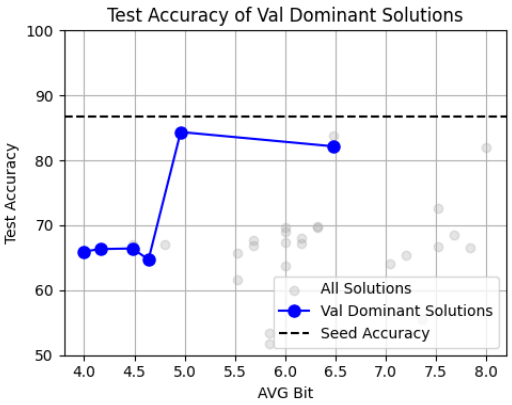


Figure 5.2: 230-params Seed

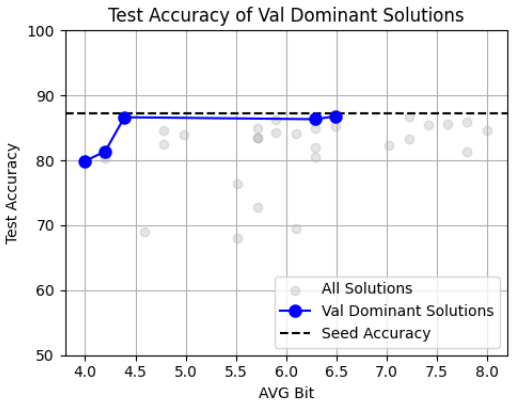


Figure 5.3: 372-params Seed

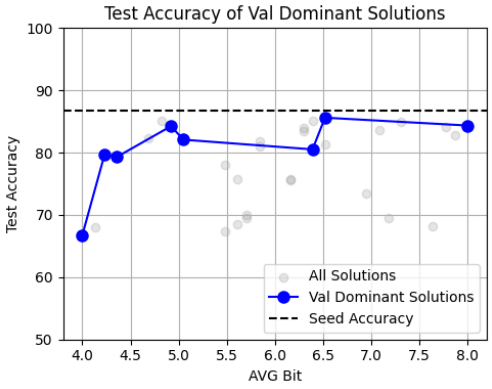


Figure 5.4: 549-params Seed

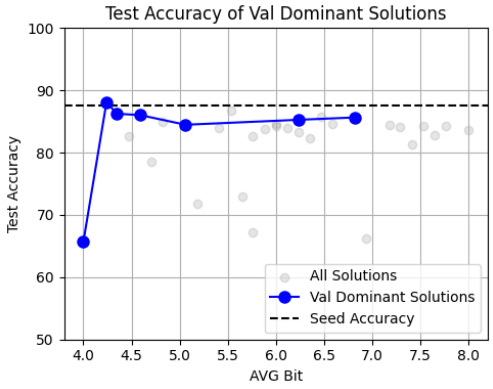


Figure 5.5: 453-params Seed

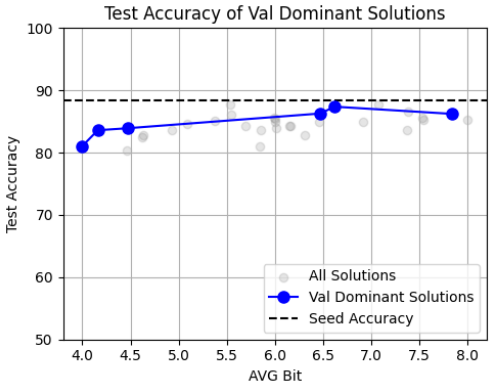


Figure 5.6: 1418-params Seed

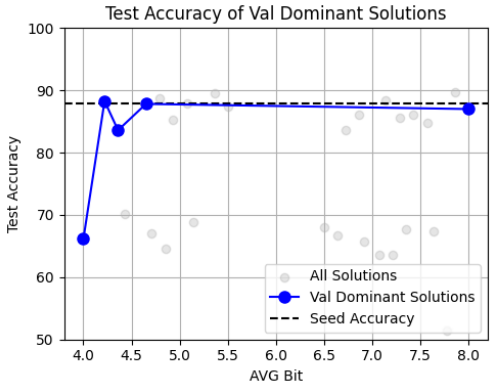


Figure 5.7: 1083-params Seed

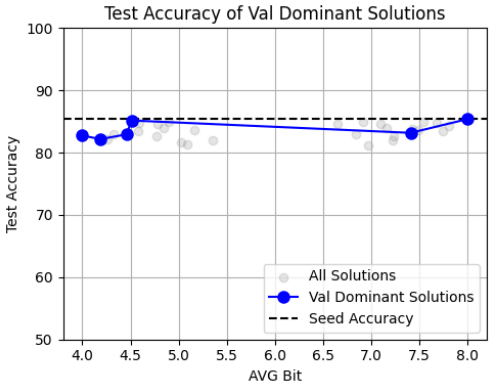


Figure 5.8: 2098-params Seed

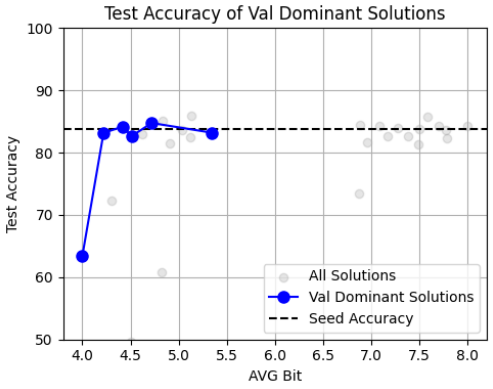


Figure 5.9: 2772-params Seed

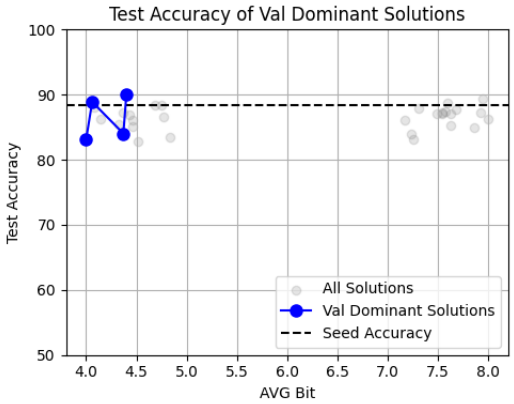


Figure 5.10: 3552-params Seed

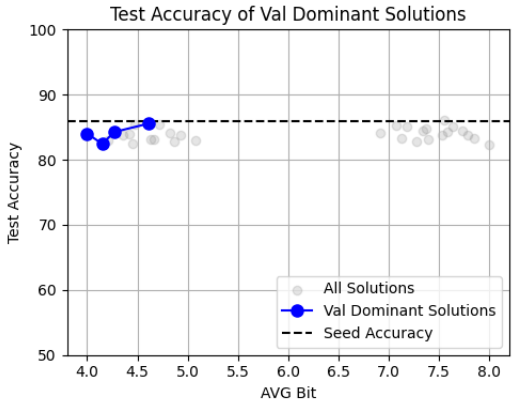


Figure 5.11: 3513-params Seed

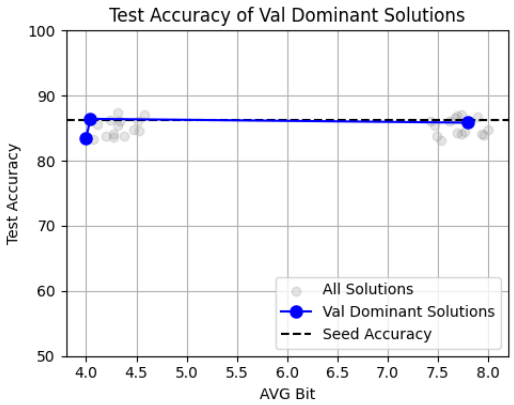


Figure 5.12: 8295-params Seed

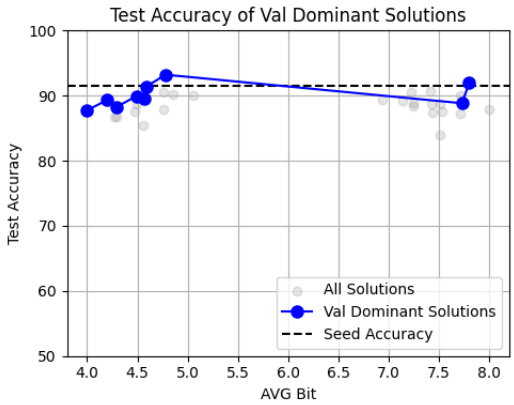
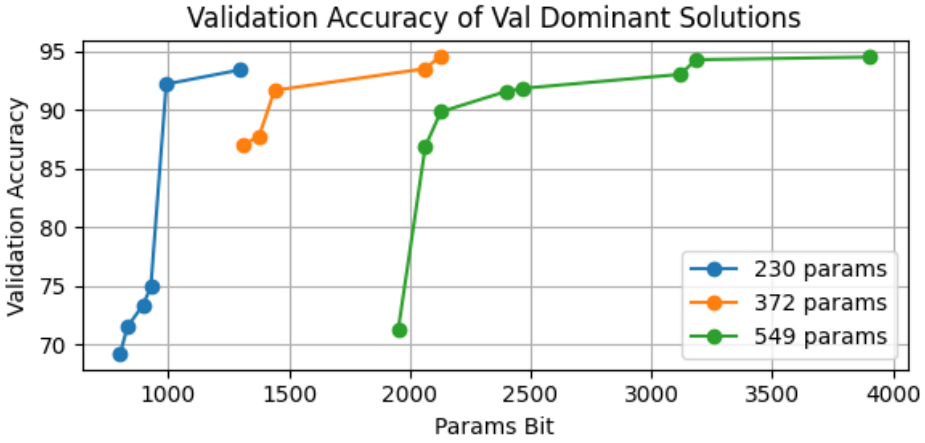


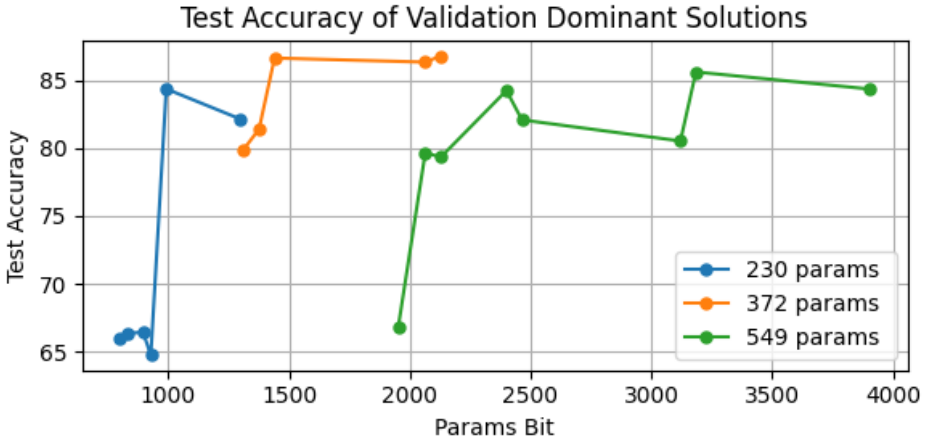
Figure 5.13: 4890-params Seed

The graphs above (Figures 5.2, 5.3, 5.4, 5.5, 5.6, 5.7, 5.8 5.9 5.10, 5.11, 5.12, 5.13) clearly show that, for certain quantization settings, accuracy remains comparable to that of the non-quantized (seed) network. This implies only a slight decrease in accuracy while achieving a compression ratio between **4x** and **8x** in model size.

In addition to evaluating the impact of quantization, the MPS POST PIT method was used to assess whether a model with a larger number of parameters, subjected to a more restrictive quantization scheme, performs worse than a model with fewer parameters but with a less stringent quantization approach, given a fixed model size defined by the number of bits required to store the model weights. The outcomes are showed in the following graphs (Figure 5.14).



(a) Validation-dominant Solutions

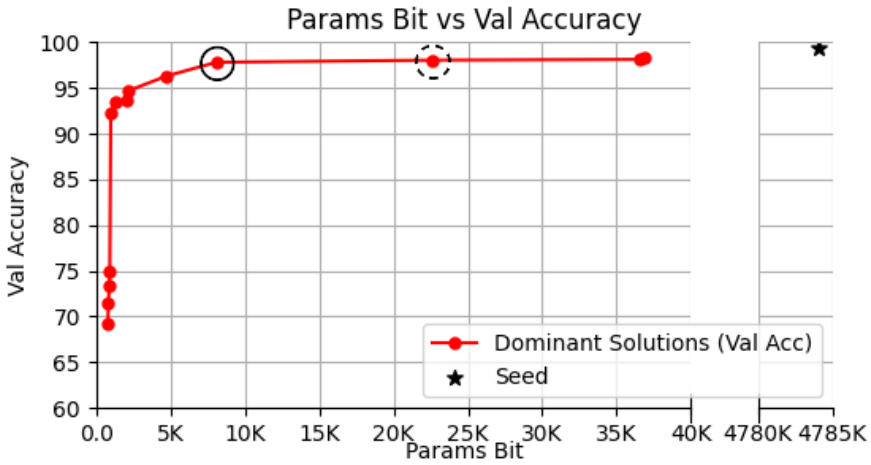


(b) Test Accuracy values of Validation-dominant Solutions

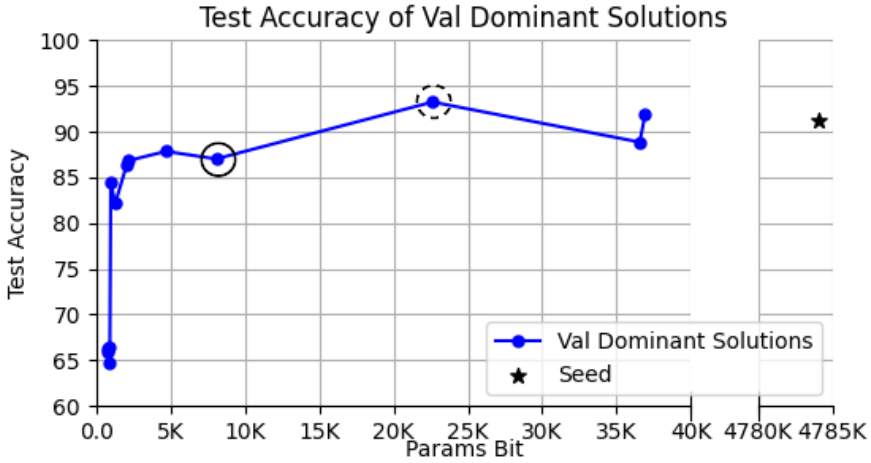
Figure 5.14: Comparison of performance achieved by applying MPS to different seed architectures.

As illustrated in the graphs (Figure 5.14), for a fixed model size, larger models subjected to more restrictive quantization schemes tend to perform worse compared to smaller models with less stringent quantization configurations. This trend is evident in both validation and test accuracy and persists even when considering larger seed networks.

To conclude the analysis, the following graphs (Figure 5.15) present the Pareto frontier in the Validation Accuracy vs. Model Size space, along with the Test Accuracy values for the Validation-dominant solutions. In this case, the Pareto frontier has been constructed considering all the seed architectures.



(a) Validation-dominant Solutions



(b) Test Accuracy values of Validation-dominant Solutions

Figure 5.15: MPS POST PIT Pareto charts, considering the different seed architectures.

A comparison between the seed network and the dominant solution circled in black demonstrates a compression ratio of **592x**, with a marginal drop of **1.6%** in validation accuracy and **4.1%** in test accuracy. An additional particularly noteworthy result is represented by the solution circled with a black dashed line, which exhibits an improvement in test accuracy of **2.1%** despite achieving a compression of up to **99.53%**. Furthermore, the plots in Figure 5.15 highlight the threshold (in terms of model size) below which performance critically declines, approximately 900 bits.

5.3.4 PIT + MPS

The last part of the experiments was dedicated to the PIT + MPS method described into Section 4.4.4. Similar to the PIT method, the experiments were conducted as follows:

1. Ten distinct model size constraint values were identified, with model size expressed in terms of the number of bits required to store the model's weights.
2. For each constraint value, Optuna was run for 50 trials to determine the optimal hyperparameters, topology, and the appropriate balance between the pruning and quantization phases.
3. The obtained optimal models were fine-tuned for a total of 8 epochs across 10 different random seeds to obtain more robust results, including accuracy and standard deviation computations.

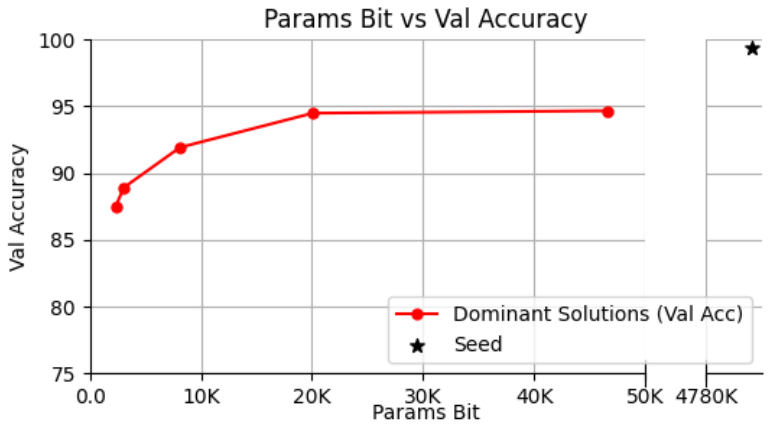
The table below presents the key results regarding the architecture topology identified during the PIT phase and the quantization configuration applied in the MPS phase.

Params Bit Target	Params	Params Bit	Architecture	Quantization
2500	482	2336	2 - 1 - 8 - 31 - 4	8 - 4 - 4 - 4 - 8
5000	565	3056	4 - 1 - 6 - 42 - 4	8 - 8 - 8 - 4 - 8
7500	493	3392	3 - 1 - 6 - 37 - 4	8 - 4 - 4 - 8 - 8
10000	1795	8120	3 - 1 - 23 - 58 - 4	4 - 4 - 8 - 4 - 8
15000	2867	12496	71 - 1 - 11 - 100 - 4	4 - 4 - 8 - 4 - 8
20000	4063	15940	3 - 1 - 51 - 67 - 4	8 - 8 - 4 - 4 - 4
25000	3279	16256	18 - 2 - 23 - 96 - 4	8 - 8 - 8 - 4 - 8
30000	4663	20084	6 - 2 - 47 - 79 - 4	8 - 8 - 8 - 4 - 4
50000	7107	29936	6 - 2 - 55 - 108 - 4	8 - 8 - 4 - 4 - 8
75000	6216	46592	37 - 4 - 55 - 70 - 4	4 - 8 - 8 - 8 - 8

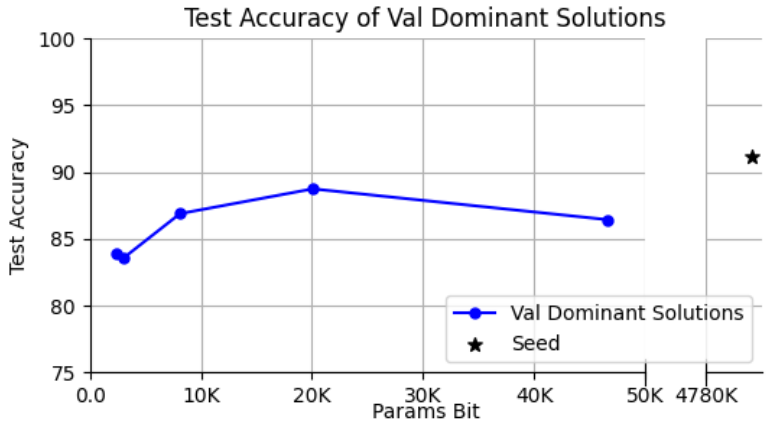
Table 5.5: PIT + MPS Architectures and Quantization Settings

Similar to the results obtained through the PIT method, the identified architecture topologies share similarities, particularly in the second convolutional layers, which have the lowest number of channels, and the first fully connected layer, which exhibits the greatest number of channels.
In addition, most of the quantization settings identified as optimal by Optuna tend to favor 8-bit configurations in each layer, reinforcing the notion that a more aggressive pruning phase followed by a less stringent quantization phase yields better results compared to the inverse approach.

Figure 5.16 illustrates the effectiveness of the PIT + MPS method in maintaining good performance while reducing model size. The graphs show the Pareto front in the Validation Accuracy vs. Model Size space and the Test Accuracy values of the validation-dominant solutions.



(a) Validation-dominant Solutions



(b) Test Accuracy values of Validation-dominant

Figure 5.16: PIT + MPS Results

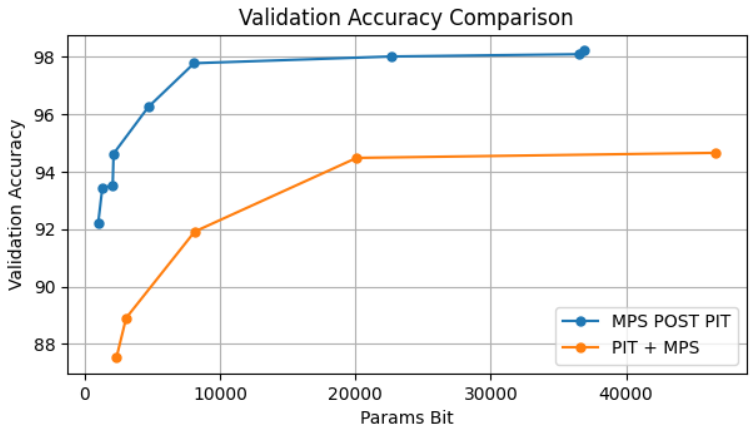
Three main observations can be made:

- The largest validation-dominant solution demonstrates significantly worse performance on the test set compared to smaller models. This may not pose an issue, as it could be excluded from the analysis due to the small improvement in validation accuracy corresponding to a substantial increase in model size relative to the smaller models.
- The drop in accuracy between the most and least performing models is larger

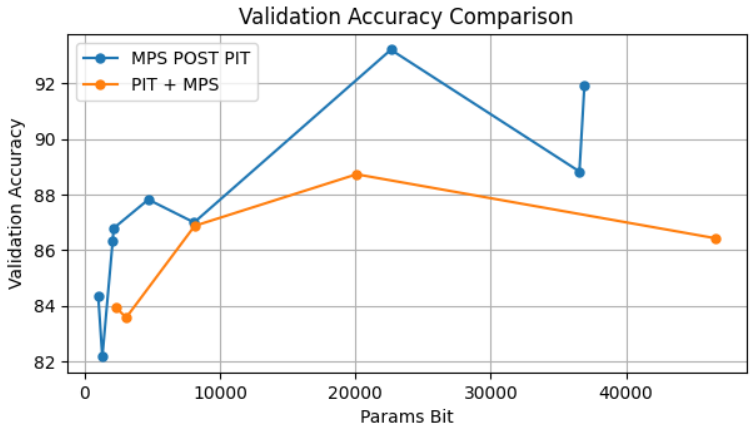
for validation accuracy compared to test accuracy. This could be attributed to the tendency of larger models to overfit, resulting in a reduced performance gap on the test set.

- Taking the Seed Network as a reference, a compression ratio of **239x** was achieved, resulting in a **4.9%** and **2.4%** decrease in validation and test accuracy, respectively, compared to the second-largest validation-dominant solution.

In conclusion, Figure 5.17 presents a comparison between the PIT + MPS method and the combination of PIT and MPS POST PIT. Specifically, the respective Pareto fronts, based on validation accuracy and the test accuracy values of their validation-dominant solutions, are shown.



(a) Comparison Validation-dominant Solutions



(b) Comparison Test Accuracy values of Val-dominant Solutions

Figure 5.17: Comparison between PIT + MPS and the combination of PIT and MPS POST PIT

The graphs highlight that the PIT + MPS method produces inferior results, as its entire Pareto front is dominated by that obtained through the combination of PIT and MPS POST PIT. This is particularly evident in validation performance. It is also important to note that the number of Optuna optimization trials conducted for the PIT + MPS method is significantly smaller, which partially accounts for its poorer performance.

5.3.5 Best Model Solutions

This section summarizes the results, focusing on the optimal solutions and their key characteristics. Table 5.6 presents the smallest models identified for each test accuracy drop level with respect to the seed network. It details their architecture topology, quantization schemes, the number of bits required to store the model's weights, the percentage reduction in model size, and the methods employed to achieve these outcomes.
The most noteworthy result is the solution that requires 22620 bits to store the model's weights. Despite a **99.53%** reduction in model size compared to the seed network, this solution achieves a **2.1%** improvement in test accuracy. Furthermore, the table highlights that all solutions, except one, are derived from the combination of PIT and MPS POST PIT, and that most quantization schemes use an 8-bit format for the convolutional layers.

These results underline, first of all, the effectiveness of the methods employed and, secondly, the potential drawbacks of developing models without considering model complexity, from different perspectives.

Test Accuracy Drop	Architecture	Quantization	Params Bit	Model Size Reduction	Method
No Drop	29 - 2 - 43 - 81 - 4	8 - 8 - 8 - 4 - 4	22620	99.53 %	MPS POST PIT
<2.5	6 - 2 - 47 - 79 - 4	8 - 8 - 8 - 4 - 4	20084	99.57 %	PIT + MPS
<3.5	9 - 1- 14 - 45 - 4	8 - 4 - 8 - 4 - 4	4696	99.90 %	MPS POST PIT
<4.5	4 - 1 - 4 - 31 - 4	8 - 8 - 8 - 4 - 8	2128	99.95 %	MPS POST PIT
<7	2 - 1 - 4 - 19 - 4	8 - 8 - 8 - 4 - 4	992	99.98 %	MPS POST PIT

Table 5.6: Summary of the smallest models for each test accuracy drop level, including architecture topology, quantization schemes, model size reduction, and methods.

Chapter 6

Conclusions and Future Works

Machine Learning and Deep Learning techniques have achieved remarkable maturity, enabling a wide range of complex applications. However, the focus is increasingly shifting from solely maximizing model performance to designing models that can operate efficiently within constrained computational environments. This shift is driven by the growing integration of Internet of Things devices into everyday life, where resource efficiency is paramount, and the reality that many organizations face significant limitations in computational resources. Enhancing the efficiency of ML and DL models is essential to making these technologies more accessible and scalable across diverse real-world scenarios.

This study specifically addresses the challenge of improving model efficiency in the context of hand detection, a fundamental computer vision task with applications in areas such as human-computer interaction and assistive technologies. A key distinguishing feature of this work is the use of IR sensors instead of high-resolution cameras. IR sensors were chosen for their low power consumption, cost-effectiveness, and privacy-preserving properties, aligning well with the study's emphasis on resource efficiency.

The research was structured into several stages. It initiated with data collection using an 8x8 IR sensor, followed by defining baseline performance metrics with SVM and RF. These benchmarks provided a solid basis for further analysis and exploration. Subsequently, the focus shifted to CNNs, first prioritizing predictive accuracy before addressing model complexity. The study explored two NAS techniques, PIT and MPS, leveraging the Optuna framework for hyperparameter optimization.

The results underscored the effectiveness of the proposed methods, achieving a compression ratio of up to 543× compared to the seed network, with only a minimal

reduction in accuracy. These findings highlight the potential for optimizing ML models to achieve a balance between performance and efficiency.

Future research could expand upon this work by tackling more complex tasks, such as hand gesture recognition. Additionally, investigating the impact of higher-resolution infrared sensors on both predictive performance and resource demands would provide valuable insights. Further exploration of MPS with additional 0-bit precision (i.e., channel pruning) which could offer an alternative to the methods explored, potentially reducing computational time and enabling faster deployment in real-time applications.

Bibliography

[1] Michael I. Jordan and Tom M. Mitchell. «Machine learning: Trends, perspectives, and prospects». In: *Science* 349.6245 (2015), pp. 255–260 (cit. on pp. 1, 4).

[2] Dana Harry Ballard and Christopher M. Brown. *Computer Vision*. 1st. Prentice Hall Professional Technical Reference, 1982. ISBN: 0131653164 (cit. on p. 1).

[3] Ethem Alpaydin. *Introduction to Machine Learning*. 3rd. Cambridge, MA: MIT Press, 2014 (cit. on p. 4).

[4] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. 1st. Cambridge, MA: MIT Press, 1998 (cit. on p. 5).

[5] Corinna Cortes and Vladimir Vapnik. «Support-vector networks». In: *Machine learning* 20.3 (1995), pp. 273–297. DOI: 10.1007/BF00994018 (cit. on pp. 5, 6).

[6] Christopher J. C. Burges. «A Tutorial on Support Vector Machines for Pattern Recognition». In: *Data Mining and Knowledge Discovery* 2.2 (1998), pp. 121–167. DOI: 10.1023/A:1009715923555. URL: https://doi.org/10.1023/A:1009715923555 (cit. on p. 7).

[7] Leo Breiman. «Random Forests». In: *Machine Learning* 45.1 (2001), pp. 5–32 (cit. on p. 8).

[8] Leo Breiman. «Bagging Predictors». In: *Machine Learning* 24.2 (1996), pp. 123–140 (cit. on p. 9).

[9] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016 (cit. on pp. 12, 14).

[10] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. «Deep Sparse Rectifier Neural Networks». In: *International Conference on Artificial Intelligence and Statistics*. 2011. URL: https://api.semanticscholar.org/CorpusID:2239473 (cit. on p. 13).

[11] Yinglong Guo, Shaohan Li, and Gilad Lerman. *The effect of Leaky ReLUs on the training and generalization of overparameterized networks*. 2024. arXiv: 2402.11942 [cs.LG]. URL: https://arxiv.org/abs/2402.11942 (cit. on p. 13).

[12] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. «Learning representations by back-propagating errors». In: *Nature* 323.6088 (1986), pp. 533–536. ISSN: 1476-4687. DOI: 10.1038/323533a0. URL: https://doi.org/10.1038/323533a0 (cit. on p. 15).

[13] D. P. Kingma and J. Ba. «Adam: A Method for Stochastic Optimization». In: *International Conference on Learning Representations (ICLR)*. 2015. URL: https://arxiv.org/abs/1412.6980 (cit. on pp. 15, 55).

[14] Yann LeCun, Léon Bottou, Yoshua Bengio, and Pierre Haffner. «Gradient-Based Learning Applied to Document Recognition». In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324 (cit. on p. 15).

[15] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri. «Learning Spatiotemporal Features with 3D CNNs». In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. 2015, pp. 4489–4497. URL: https://openaccess.thecvf.com/content_iccv_2015/html/Tran_Learning_Spatiotemporal_Features_ICCV_2015_paper.html (cit. on p. 15).

[16] https://www.rncm.ac.uk/research/research-activity/research-centres-rncm/prism/prism-blog/a-short-history-of-neural-synthesis/. November 4th, 2024. (cit. on p. 16).

[17] L. Alzubaidi, J. Zhang, A. J. Humaidi, et al. «Review of deep learning: concepts, CNN architectures, challenges, applications, future directions». In: *Journal of Big Data* 8 (2021), p. 53. DOI: 10.1186/s40537-021-00444-8. URL: https://doi.org/10.1186/s40537-021-00444-8 (cit. on p. 17).

[18] C. Tan, F. Sun, T. Kong, W. Zhang, C. Yang, and C. Liu. «A Survey on Deep Transfer Learning». In: *Artificial Neural Networks and Machine Learning – ICANN 2018*. Ed. by V. Kůrková, Y. Manolopoulos, B. Hammer, L. Iliadis, and I. Maglogiannis. Vol. 11141. Lecture Notes in Computer Science. Springer, Cham, 2018, pp. 27–39. DOI: 10.1007/978-3-030-01424-7_27. URL: https://doi.org/10.1007/978-3-030-01424-7_27 (cit. on p. 17).

[19] Haibo He and Edwardo A. Garcia. «Learning from Imbalanced Data». In: *IEEE Transactions on Knowledge and Data Engineering* 21.9 (2009), pp. 1263–1284. DOI: 10.1109/TKDE.2008.239 (cit. on p. 18).

[20] C. Shorten and T. M. Khoshgoftaar. «A survey on Image Data Augmentation for Deep Learning». In: *Journal of Big Data* 6.1 (2019), p. 60. DOI: 10.1186/s40537-019-0197-0. URL: https://doi.org/10.1186/s40537-019-0197-0 (cit. on p. 18).

[21] Scott M. Lundberg and Su-In Lee. «A Unified Approach to Interpreting Model Predictions». In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2017, pp. 4765–4774. URL: https://arxiv.org/abs/1705.07874 (cit. on p. 18).

[22] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. «Why Should I Trust You? Explaining the Predictions of Any Classifier». In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2016, pp. 1135–1144. URL: https://arxiv.org/abs/1602.04938 (cit. on p. 18).

[23] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. «Edge Computing: Vision and Challenges». In: *IEEE Internet of Things Journal* 3.5 (2016), pp. 637–646. DOI: 10.1109/JIOT.2016.2579198 (cit. on p. 19).

[24] Andrew G. Howard et al. «Mobilenets: Efficient convolutional neural networks for mobile vision applications». In: *arXiv preprint arXiv:1704.04861* (2017). URL: https://arxiv.org/abs/1704.04861 (cit. on pp. 20, 26).

[25] Sohail Saif, Mayurakshi Jana, and Suparna Biswas. «Recent Trends in IoT–Based Smart Healthcare Applying ML and DL». In: *Emerging Technologies in Data Mining and Information Security*. Ed. by João Manuel R. S. Tavares, Satyajit Chakrabarti, Abhishek Bhattacharya, and Sujata Ghatak. Singapore: Springer Singapore, 2021, pp. 785–797. ISBN: 978-981-15-9774-9. DOI: 10.1007/978-981-15-9774-9_72 (cit. on p. 21).

[26] M. Bojarski, D. Del Testa, D. Dworakowski, A. Falco, M. Finzi, E. Gallo, and et al. «End to End Learning for Self-Driving Cars». In: *arXiv preprint arXiv:1604.07316* (2016). URL: https://arxiv.org/abs/1604.07316 (cit. on p. 21).

[27] Andreas Kamilaris and Francesc X. Prenafeta-Boldú. «Deep learning in agriculture: A survey». In: *Computers and Electronics in Agriculture* 147 (2018), pp. 70–90. ISSN: 0168-1699. DOI: https://doi.org/10.1016/j.compag.2018.02.016. URL: https://www.sciencedirect.com/science/article/pii/S0168169917308803 (cit. on p. 21).

[28] Ya-Li Hou and Grantham K. H. Pang. «People Counting and Human Detection in a Challenging Situation». In: *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans* 41.1 (2011), pp. 24–33. DOI: 10.1109/TSMCA.2010.2064299 (cit. on p. 22).

[29] Shahid Latif, Maha Driss, Wadii Boulila, Zil e Huma, Sajjad Shaukat Jamal, Zeba Idrees, and Jawad Ahmad. «Deep Learning for the Industrial Internet of Things (IIoT): A Comprehensive Survey of Techniques, Implementation Frameworks, Potential Applications, and Future Directions». In: *Sensors* 21.22 (2021). ISSN: 1424-8220. DOI: 10.3390/s21227518. URL: https://www.mdpi.com/1424-8220/21/22/7518 (cit. on p. 22).

[30] Samuel Williams, Andrew Waterman, and David A. Patterson. «Roofline: An insightful visual performance model for multicore architectures». In: *Communications of the ACM* 52.4 (2009), pp. 65–76. DOI: 10.1145/1498765.1498785 (cit. on p. 23).

[31] https://en.wikipedia.org/wiki/Roofline_model. November 4th, 2024. (cit. on p. 24).

[32] https://iq.opengenus.org/separable-convolution/. November 4th, 2024. (cit. on p. 26).

[33] Barret Zoph and Quoc V. Le. *Neural Architecture Search with Reinforcement Learning*. 2017. arXiv: 1611.01578 [cs.LG]. URL: https://arxiv.org/abs/1611.01578 (cit. on pp. 27, 39).

[34] Hanxiao Liu, Karen Simonyan, and Yiming Yang. *DARTS: Differentiable Architecture Search*. 2019. arXiv: 1806.09055 [cs.LG]. URL: https://arxiv.org/abs/1806.09055 (cit. on pp. 27, 28, 39).

[35] Matteo Risso, Alessio Burrello, Luca Benini, Enrico Macii, Massimo Poncino, and Daniele Jahier Pagliari. «Multi-Complexity-Loss DNAS for Energy-Efficient and Memory-Constrained Deep Neural Networks». In: *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design*. ISLPED '22. ACM, Aug. 2022, pp. 1–6. DOI: 10.1145/3531437.3539720. URL: http://dx.doi.org/10.1145/3531437.3539720 (cit. on p. 27).

[36] Tailin Liang, John Glossner, Lei Wang, Shaobo Shi, and Xiaotong Zhang. *Pruning and Quantization for Deep Neural Network Acceleration: A Survey*. 2021. arXiv: 2101.09671 [cs.CV]. URL: https://arxiv.org/abs/2101.09671 (cit. on pp. 28, 30).

[37] Jaeho Lee, Sejun Park, Sangwoo Mo, Sungsoo Ahn, and Jinwoo Shin. *Layer-adaptive sparsity for the Magnitude-based Pruning*. 2021. arXiv: 2010.07611 [cs.LG]. URL: https://arxiv.org/abs/2010.07611 (cit. on p. 29).

[38] Kaveena Persand, Andrew Anderson, and David Gregg. «Taxonomy of Saliency Metrics for Channel Pruning». In: *IEEE Access* 9 (2021), pp. 120110–120126. DOI: 10.1109/ACCESS.2021.3108545 (cit. on p. 29).

[39] https://blog.paperspace.com/neural-network-pruning-explained/. November 4th, 2024. (cit. on p. 29).

[40] Sajid Anwar, Kyuyeon Hwang, and Wonyong Sung. *Structured Pruning of Deep Convolutional Neural Networks*. 2015. arXiv: 1512.08571 [cs.NE]. URL: https://arxiv.org/abs/1512.08571 (cit. on p. 30).

[41] Ron Banner, Yury Nahshan, and Daniel Soudry. «Post training 4-bit quantization of convolutional networks for rapid-deployment». In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett. Vol. 32. Curran Associates, Inc., 2019. URL: https://proceedings.neurips.cc/paper_files/paper/2019/file/c0a62e133894cdce435bcb4a5df1db2d-Paper.pdf (cit. on p. 31).

[42] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. *Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference*. 2017. arXiv: 1712.05877 [cs.LG]. URL: https://arxiv.org/abs/1712.05877 (cit. on pp. 32, 52).

[43] Penghang Yin, Jiancheng Lyu, Shuai Zhang, Stanley Osher, Yingyong Qi, and Jack Xin. *Understanding Straight-Through Estimator in Training Activation Quantized Neural Nets*. 2019. arXiv: 1903.05662 [cs.LG]. URL: https://arxiv.org/abs/1903.05662 (cit. on p. 32).

[44] Gaurav Sinha, Rahul Shahi, and Mani Shankar. «Human Computer Interaction». In: *2010 3rd International Conference on Emerging Trends in Engineering and Technology*. 2010, pp. 1–4. DOI: 10.1109/ICETET.2010.85 (cit. on p. 35).

[45] Fan Zhang, Valentin Bazarevsky, Andrey Vakunov, Andrei Tkachenka, George Sung, Chuo-Ling Chang, and Matthias Grundmann. *MediaPipe Hands: On-device Real-time Hand Tracking*. 2020. arXiv: 2006.10214 [cs.CV]. URL: https://arxiv.org/abs/2006.10214 (cit. on p. 35).

[46] Hongyi Liu and Lihui Wang. «Gesture recognition for human-robot collaboration: A review». In: *International Journal of Industrial Ergonomics* 68 (2018), pp. 355–367. ISSN: 0169-8141. DOI: https://doi.org/10.1016/j.ergon.2017.02.004. URL: https://www.sciencedirect.com/science/article/pii/S0169814117300690 (cit. on p. 36).

[47] Razieh Rastgoo, Kourosh Kiani, and Sergio Escalera. «Sign Language Recognition: A Deep Survey». In: *Expert Systems with Applications* 164 (2021), p. 113794. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2020.113794. URL: https://www.sciencedirect.com/science/article/pii/S095741742030614X (cit. on p. 36).

[48] N. Dalal and B. Triggs. «Histograms of oriented gradients for human detection». In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*. Vol. 1. 2005, 886–893 vol. 1. DOI: 10.1109/CVPR.2005.177 (cit. on p. 36).

[49] Seung-Hyun Lee, MinSuk Bang, Kyeong-Hoon Jung, and Kang Yi. «An efficient selection of HOG feature for SVM classification of vehicle». In: *2015 International Symposium on Consumer Electronics (ISCE)*. 2015, pp. 1–2. DOI: 10.1109/ISCE.2015.7177766 (cit. on p. 36).

[50] Jiang Guo, Jun Cheng, Jianxin Pang, and Yu Guo. «Real-time hand detection based on multi-stage HOG-SVM classifier». In: *2013 IEEE International Conference on Image Processing*. 2013, pp. 4108–4111. DOI: 10.1109/ICIP.2013.6738846 (cit. on p. 36).

[51] Li Zhang, Bo Wu, and Ram Nevatia. «Pedestrian Detection in Infrared Images based on Local Shape Features». In: *2007 IEEE Conference on Computer Vision and Pattern Recognition*. 2007, pp. 1–8. DOI: 10.1109/CVPR.2007.383452 (cit. on p. 36).

[52] T. Ojala, M. Pietikäinen, and T. Maenpää. «Multiresolution gray-scale and rotation invariant texture classification». In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 24.7 (2002), pp. 971–987. DOI: 10.1109/TPAMI.2002.1017623 (cit. on p. 37).

[53] Houssem Lahiani and Mahmoud Neji. «Hand gesture recognition method based on HOG-LBP features for mobile devices». In: *Procedia Computer Science* 126 (2018). Knowledge-Based and Intelligent Information Engineering Systems: Proceedings of the 22nd International Conference, KES-2018, Belgrade, Serbia, pp. 254–263. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2018.07.259. URL: https://www.sciencedirect.com/science/article/pii/S1877050918312353 (cit. on p. 37).

[54] D.G. Lowe. «Object recognition from local scale-invariant features». In: *Proceedings of the Seventh IEEE International Conference on Computer Vision*. Vol. 2. 1999, 1150–1157 vol.2. DOI: 10.1109/ICCV.1999.790410 (cit. on p. 37).

[55] Chieh-Chih Wang and Ko-Chih Wang. «Hand Posture Recognition Using Adaboost with SIFT for Human Robot Interaction». In: *Recent Progress in Robotics: Viable Robotic Service to Human: An Edition of the Selected Papers from the 13th International Conference on Advanced Robotics*. Ed. by Sukhan Lee, Il Hong Suh, and Mun Sang Kim. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 317–329. ISBN: 978-3-540-76729-9. DOI: 10.1007/978-3-540-76729-9_25. URL: https://doi.org/10.1007/978-3-540-76729-9_25 (cit. on p. 37).

[56] Unsang Park, Sharath Pankanti, and A. K. Jain. «Fingerprint verification using SIFT features». In: *Biometric Technology for Human Identification V*. Ed. by B.V.K. Vijaya Kumar, Salil Prabhakar, and Arun A. Ross. Vol. 6944. International Society for Optics and Photonics. SPIE, 2008, 69440K. DOI: 10.1117/12.778804. URL: https://doi.org/10.1117/12.778804 (cit. on p. 37).

[57] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. «ImageNet classification with deep convolutional neural networks». In: *Communications of the ACM* 60 (2012), pp. 84–90. URL: https://api.semanticscholar.org/CorpusID:195908774 (cit. on p. 37).

[58] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. *You Only Look Once: Unified, Real-Time Object Detection*. 2016. arXiv: 1506.02640 [cs.CV]. URL: https://arxiv.org/abs/1506.02640 (cit. on p. 38).

[59] Mubashiru Olarewaju Lawal. «Tomato detection based on modified YOLOv3 framework». In: *Scientific Reports* 11.1 (Jan. 14, 2021), p. 1447. ISSN: 2045-2322. DOI: 10.1038/s41598-021-81216-5. URL: https://doi.org/10.1038/s41598-021-81216-5 (cit. on p. 38).

[60] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. 2016. arXiv: 1506.01497 [cs.CV]. URL: https://arxiv.org/abs/1506.01497 (cit. on p. 38).

[61] Huaizu Jiang and Erik Learned-Miller. «Face Detection with the Faster R-CNN». In: *2017 12th IEEE International Conference on Automatic Face Gesture Recognition (FG 2017)*. 2017, pp. 650–657. DOI: 10.1109/FG.2017.82 (cit. on p. 38).

[62] Yang Liu, Zhuo Ma, Ximeng Liu, Siqi Ma, and Kui Ren. «Privacy-Preserving Object Detection for Medical Images With Faster R-CNN». In: *IEEE Transactions on Information Forensics and Security* 17 (2022), pp. 69–84. DOI: 10.1109/TIFS.2019.2946476 (cit. on p. 38).

[63] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. «Going deeper with convolutions». In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, pp. 1–9. DOI: 10.1109/CVPR.2015.7298594 (cit. on p. 38).

[64] Chunmian Lin, Lin Li, Wenting Luo, Kelvin C. P. Wang, and Jiangang Guo. «Transfer Learning Based Traffic Sign Recognition Using Inception-v3 Model». In: *Periodica Polytechnica Transportation Engineering* 47.3 (2019), pp. 242–250. DOI: 10.3311/PPtr.11480. URL: https://www.pp.bme.hu/tr/article/view/11480 (cit. on p. 38).

[65] Mohammed Abdulla Salim Al Husaini, Mohamed Hadi Habaebi, Teddy Surya Gunawan, Md Rafiqul Islam, Elfatih A. A. Elsheikh, and F. M. Suliman. «Thermal-based early breast cancer detection using inception V3, inception V4 and modified inception MV4». In: *Neural Computing and Applications* 34.1 (2022), pp. 333–348. ISSN: 1433-3058. DOI: 10.1007/s00521-021-06372-1. URL: https://doi.org/10.1007/s00521-021-06372-1 (cit. on p. 38).

[66] Joshua van Staden and Dane Brown. «An Evaluation of YOLO-Based Algorithms for Hand Detection in the Kitchen». In: *2021 International Conference on Artificial Intelligence, Big Data, Computing and Data Communication Systems (icABCD)*. 2021, pp. 1–7. DOI: 10.1109/icABCD51485.2021.9519307 (cit. on p. 38).

[67] Chi Xu, Wendi Cai, Yongbo Li, Jun Zhou, and Longsheng Wei. «Accurate Hand Detection from Single-Color Images by Reconstructing Hand Appearances». English. In: *Sensors (Basel)* 20.1 (Dec. 2019). The authors declare no conflict of interest., p. 192. DOI: 10.3390/s20010192 (cit. on p. 38).

[68] Noor Awad, Neeratyoy Mallik, and Frank Hutter. *Differential Evolution for Neural Architecture Search*. 2021. arXiv: 2012.06400 [cs.NE]. URL: https://arxiv.org/abs/2012.06400 (cit. on p. 39).

[69] Ariel Gordon, Elad Eban, Ofir Nachum, Bo Chen, Hao Wu, Tien-Ju Yang, and Edward Choi. *MorphNet: Fast Simple Resource-Constrained Structure Learning of Deep Networks*. 2018. arXiv: 1711.06798 [cs.LG]. URL: https://arxiv.org/abs/1711.06798 (cit. on p. 39).

[70] Matteo Risso, Alessio Burrello, Luca Benini, Enrico Macii, Massimo Poncino, and Daniele Jahier Pagliari. «Channel-wise Mixed-precision Assignment for DNN Inference on Constrained Edge Nodes». In: *2022 IEEE 13th International Green and Sustainable Computing Conference (IGSC)*. 2022, pp. 1–6. DOI: 10.1109/IGSC55832.2022.9969373 (cit. on p. 39).

[71] Alessio Burrello, Francesco Carlucci, Giovanni Pollo, Xiaying Wang, Massimo Poncino, Enrico Macii, Luca Benini, and Daniele Jahier Pagliari. *Optimization and Deployment of Deep Neural Networks for PPG-based Blood Pressure Estimation Targeting Low-power Wearables*. 2024. arXiv: 2409.07485 [eess.SP]. URL: https://arxiv.org/abs/2409.07485 (cit. on p. 39).

[72] *Panasonic High Performance Grid-EYE Sensors Reference Specification*. 2024 (cit. on p. 40).

[73] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. *Optuna: A Next-generation Hyperparameter Optimization Framework*. 2019. arXiv: 1907.10902 [cs.LG]. URL: https://arxiv.org/abs/1907.10902 (cit. on pp. 45, 55).

[74] Shuhei Watanabe. *Tree-Structured Parzen Estimator: Understanding Its Algorithm Components and Their Roles for Better Empirical Performance*. 2023. arXiv: 2304.11127 [cs.LG]. URL: https://arxiv.org/abs/2304.11127 (cit. on p. 46).

[75] Alessio Burrello, Matteo Risso, Beatrice Alessandra Motetti, Enrico Macii, Luca Benini, and Daniele Jahier Pagliari. «Enhancing Neural Architecture Search With Multiple Hardware Constraints for Deep Learning Model Deployment on Tiny IoT Devices». In: *IEEE Transactions on Emerging Topics in Computing* 12.3 (2024), pp. 780–794. DOI: 10.1109/TETC.2023.3322033 (cit. on p. 50).

[76] Daniele Jahier Pagliari, Matteo Risso, Beatrice Alessandra Motetti, and Alessio Burrello. «PLiNIO: A User-Friendly Library of Gradient-Based Methods for Complexity-Aware DNN Optimization». In: *2023 Forum on Specification & Design Languages (FDL)* (2023), pp. 1–8. URL: https://api.semanticscholar.org/CorpusID:259982600 (cit. on pp. 51, 55).

[77] Fabian Pedregosa et al. *Scikit-learn: Machine Learning in Python*. 2018. arXiv: 1201.0490 [cs.LG]. URL: https://arxiv.org/abs/1201.0490 (cit. on p. 55).

[78] Adam Paszke et al. *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. 2019. arXiv: 1912.01703 [cs.LG]. URL: https://arxiv.org/abs/1912.01703 (cit. on p. 55).

[79] PyTorch Documentation. *ReduceLROnPlateau Learning Rate Scheduler*. https://pytorch.org/docs/stable/generated/torch.optim.lr_scheduler.ReduceLROnPlateau.html. Accessed on November 22nd, 2024 (cit. on p. 56).