POLITECNICO DI TORINO

Master's Degree in Aerospace Engineering

Master's Degree Thesis

Advanced Optical Navigation Strategies Based on AI Algorithms

Supervisors

Prof. Fabrizio STESINA

Ing. Antonio D'ORTONA

Co-Supervisors

Ing. Lucrezia LOVAGLIO

Candidate

Daniele BOCCACCIARI

Academic Year 2023/2024

Abstract

The deployment of neural networks in space has opened new possibilities for achieving high-precision proximity operations, which are crucial for inspection, maintenance, and debris removal in the context of on-orbit servicing (OOS). A core requirement for these missions is accurate pose estimation, defined as the capability of an active spacecraft to estimate the relative position and orientation of a non-cooperative one. It involves substantial technological challenges in sensor architecture selection and algorithm development. For this purpose, visual navigation employs monocular cameras, favoured for their compact form and low resource demands, which serve as ideal sensors for capturing visual data in space environments. By leveraging Convolutional Neural Networks (CNNs), this research aims to enhance pose estimation capabilities, addressing the unique challenges posed by non-cooperative targets under varying lighting conditions and complex orbital backgrounds.

This work aims to provide an optimization strategy for neural networks to increase their accuracy, performance, and efficiency for deployment on hardware with limited capacity. Particular emphasis is placed on advanced hyperparameter optimization and compression techniques, such as pruning, to streamline the network while preserving high levels of accuracy. The network model was trained and validated using two synthetic image datasets, each representing distinct, mission-critical phases of proximity navigations.

The results demonstrate the effectiveness of hyperparameter optimization (HPO) and pruning techniques in enhancing the performance and efficiency of neural networks for space-based pose estimation tasks. The application of HPO led to a marked improvement in pose accuracy, with optimized networks achieving higher performance while downgrading image resolution, thereby minimizing computational cost and accelerating convergence during solution search and network training. Furthermore, structured pruning techniques were successfully applied, reducing model size without compromising accuracy. These optimizations validate theoretical expectations and establish practical benefits for on-orbit neural network deployment, showing that highly accurate and resource-efficient networks can be realized by reducing resolution and leveraging structured pruning. This work provides a foundation for further exploration into optimization techniques that maintain high performance even on constrained hardware, highlighting efficient strategies for real-time space applications.

Table of Contents

List of Tables

List of Figures

Acronym

OOS On-Orbit Servicing

ADR Active Debris Removal

DL Deep Learning

ML Machine Learning

SLAB Stanford's Space Rendezvous Laboratory

CNN Convolutional Neural Network

ANN Artificial Neural Network

FC Fully Connected

SROC Space Rider Observer Cubesat

SR Space Rider

BBO Black-box optimization

HPO Hyperparameter optimization

TPE Tree-Structured Parzen Estimator

CMA-ES Covariance Matrix Adaptation-Evolution Strategy

INGO Independent Natural Gradient Optimization

BO Bayesian Optimization

ASHA Asynchronous Successive Halving Algorithm

SHA Successive Halving Algorithm

PTQ Post-Training Quantization

QAT Quantization Aware Training

KD Knowledge Distillation

DPF Dynamic Pruning with Feedback

RNP Rank-Normalized Pruning

IoU Intersection over Union

FA Final Approach

WSE Walking Safety Ellipse

NNCF Neural Network Compression Framework

Chapter 1

Introduction

1.1 Visual Navigation for proximity operation

In recent years, the number of satellites launched into orbit has increased significantly, driven by lower launch costs and easier access to space. Each mission has distinct goals that define the satellite's size, functionality, and lifespan. Most satellites complete their missions as planned, after which they are either relocated to a graveyard orbit or left to re-enter Earth's atmosphere. Non-cooperative satellites represent a category of satellites that do not actively participate in mission operations, either due to their design or because of unforeseen malfunctions. These satellites can pose significant risks to other space infrastructure, especially when malfunctions hinder the success of the mission. To address these challenges, there has been growing interest in On-Orbit Servicing (OOS) missions, which focus on the inspection, maintenance, and repair of in-orbit spacecraft. OOS helps extend the operational life of spacecraft and ensures their continued functionality. In parallel, Active Debris Removal (ADR) missions are also critical for mitigating the risks posed by space debris, including defunct satellites. ADR involves removing non-functional satellites and other debris from Earth's orbit, which is essential to maintaining the safety and sustainability of space operations. Both OOS and ADR missions require spacecraft to perform complex rendezvous and proximity manoeuvres around the target before carrying out mission-specific tasks.[1]

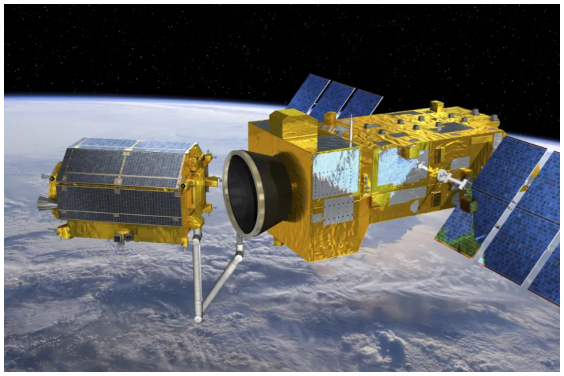


Figure 1.1: Conceptual image of OOS

A key requirement for these autonomous operations is highly precise pose estimation, which involves determining the relative position and orientation of the target spacecraft. In many cases, the target is non-cooperative, meaning it does not actively communicate or provide signals to assist with navigation. As these missions often involve small spacecraft, visual navigation systems based on monocular cameras are commonly used due to their compact size and low power consumption. Visual navigation enables spacecraft to estimate their position and orientation by analyzing visual cues, such as features on the target object. Traditional methods for pose estimation in space have relied on manually designed feature-matching techniques. These methods utilize feature descriptors and detectors such as SIFT, SURF, KAZE, and AKAZE [2] to identify key features like corners and edges, subsequently leveraging their 3D correspondences to determine the relative pose. However, these techniques face significant limitations in the space environment due to challenges like low signal-to-noise ratios and high contrast lighting conditions. These limitations reduce the effectiveness of conventional approaches, making them less reliable for autonomous operations. To address these issues, more advanced techniques, such as neural networks, are found to improve robustness and generalization in space-based visual navigation, providing more accurate pose estimation even in the harsh conditions encountered in space operations.[3]

Deep learning (DL)-based approaches have shown significant potential in various applications, but they largely depend on labelled data, which is challenging to acquire in certain domains. To address this, synthetic data generation and laboratory-based data collection have emerged as feasible methods for training and testing DL algorithms, as demonstrated by Park et al.

[4]. However, these techniques often face performance issues when applied to real-world images, a problem commonly referred to as the domain gap. This discrepancy between training and real-world performance highlights the need for strategies to bridge the gap. Although some methods have been proposed, such as those by Park et al. [4] and Lovaglio [3], they are often either too computationally demanding for onboard use or fail to achieve the necessary accuracy for practical implementation.

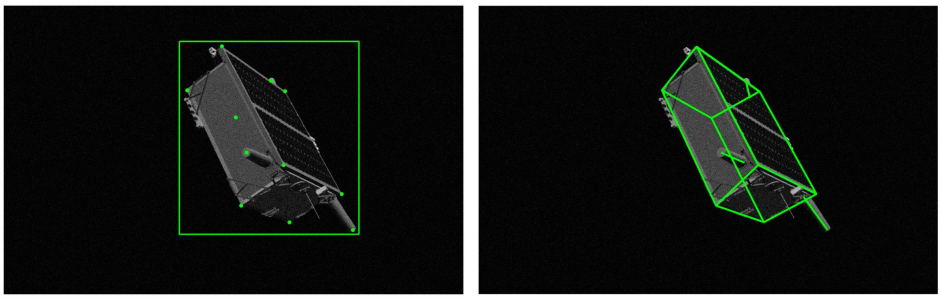


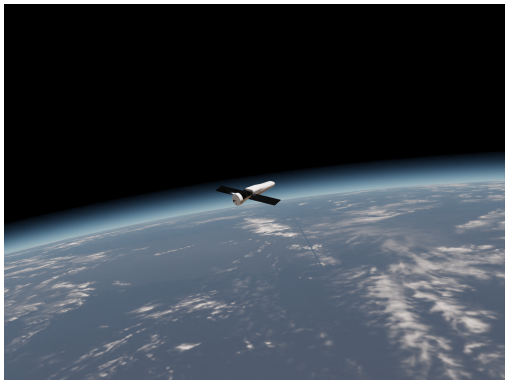
Figure 1.2: Pose estimation [5]

This thesis aims to optimize neural networks for pose estimation by focusing on hyperparameter tuning to develop a solution that is both computationally efficient and lightweight while meeting the accuracy requirements for pose determination. The optimization process extends beyond selecting appropriate training parameters, addressing the need to adapt the neural network for compatibility with limited processing power onboard processors equipped on satellites, ensuring both timeliness and precision. A methodological approach is proposed to optimize such neural networks to establish a standard for optical navigation in future space missions.

1.1.1 Domain Gap

In space exploration, the Domain Gap presents a critical challenge for applying machine learning (ML) models in space-borne vision applications. This gap arises from the significant difference between the environments in which ML models are trained, typically simulated, and the actual conditions encountered in space. Acquiring a good dataset of space-borne images for model training is challenging because, even when images are collected from space, they will always differ from those relevant to the specific mission being studied. This variability arises from differences in spacecraft and environmental conditions, making it difficult to obtain a consistent and representative set of data for

training purposes. This limitation hinders the ability to thoroughly evaluate the performance of ML models on real space images before deployment. Consequently, most of the existing methods rely on training models with synthetic images and extensive data augmentation or domain randomization to simulate space conditions. However, the evaluation of these models on actual space-borne images remains limited due to the scarcity of such data. Furthermore, the computational and memory constraints of satellite avionics make it impractical to perform domain adaptation during the mission.



(a) Space Rider - synthetic image

(b) Cygnus - space-born image

Closing the domain gap is crucial to guarantee that models developed and tested in Earth-like conditions can operate reliably in space. Although advancements have been made, a complete solution to this issue is still lacking, highlighting the need for more sophisticated domain adaptation methods designed to address the specific challenges of space exploration.
It's valuable to mention the work of the SPEED/SPEED+ benchmark, developed at Stanford's Space Rendezvous Laboratory (SLAB) [6]. The aim is to create a large dataset that includes 59960 synthetic photos: 6740 for emulating diffuse light in Earth's orbits and 2791 images for simulating the sun's influence on spacecraft. This groundbreaking work has become a benchmark for the training and testing of neural networks for space, as well as a standard procedure for future missions. However, even though these images so other techniques need to be implemented in order to bridge the gap.[4]

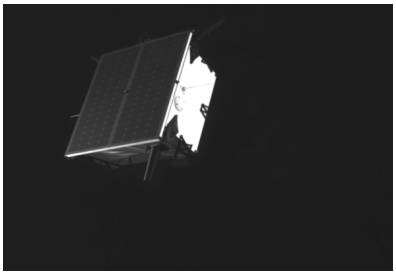


Figure 1.4: SPEED+ mock-up [7]

1.2 CNN for pose estimation

The present thesis will concentrate on enhancing the performance of the following convolutional neural network architecture [3] tailored for the ESA's mission Space Rider Observer Cubesat (SROC).
SROC mission is designed to showcase essential technologies and capabilities required for conducting rendezvous and docking missions within a safety-critical environment. The system is tasked with executing proximity operations near the Space Rider (SR) vehicle, ultimately docking with the mothership before re-entering Earth's atmosphere. The mission statement is outlined as follows: "To operate a CubeSat in LEO to demonstrate capabilities in the close-proximity operations domain in a safety-critical context, including rendezvous and docking with another operational spacecraft."[8].

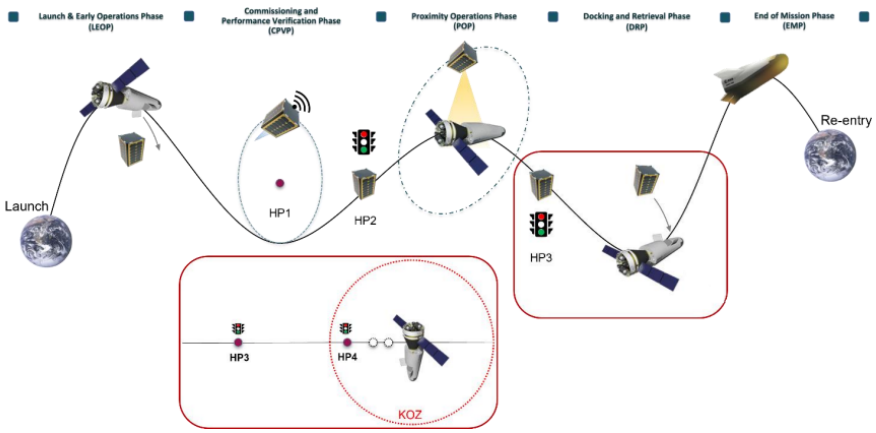


Figure 1.5: SROC mission [8]

As an in-orbit demonstrator, the SROC project aims to develop and validate novel technologies specific to proximity operations in space. Additionally, the advancements achieved through the SROC mission will contribute to other space operations, such as in-orbit servicing and debris mitigation.[8]
The following sections will introduce the main architecture of this CNN, outlining its key components

- Preliminary Classification CNN: It's a basic binary classifier, trained to recognize the presence of the target labelling the images as 1 defines if the target is present, while 0 if not.
- Pose Estimation CNN: It estimates the pose of an uncooperative target by computing the rotation and translation through regression over two heads, EfficientPose and Heatmap.

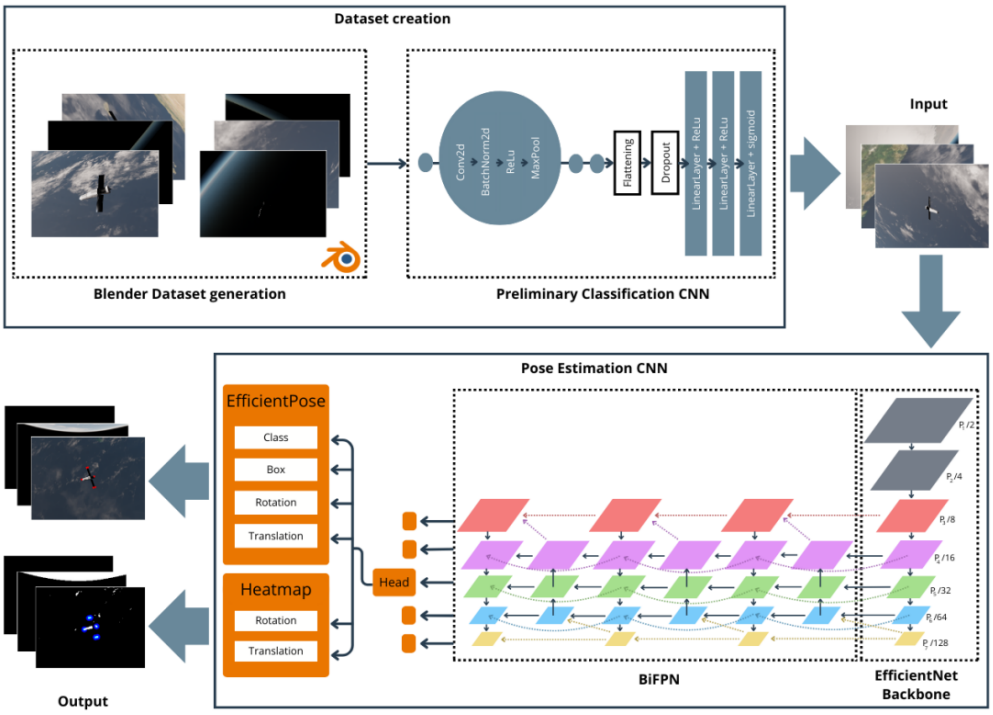


Figure 1.6: EfficientPose Estimation Network [3]

1.2.1 Binary Classification

The CNN is entirely custom-designed and classifies augmented, pre-processed images with a binary output of 1 or 0, indicating the presence or absence of the target. The overall architecture consists of four convolutional layers, each utilizing a 3x3 kernel and a stride of 1. The first layer accepts 3 input channels (corresponding to RGB) and produces 16 output channels, each representing a unique filter that detects different patterns. In subsequent hidden layers, the number of output channels is doubled, progressing from 16 to 32, 64, and 128 filters. The figure below provides a detailed schematic of the architecture, illustrating the various operations applied to the layers, such as MaxPooling for output size reduction, activation functions, and batch normalization.

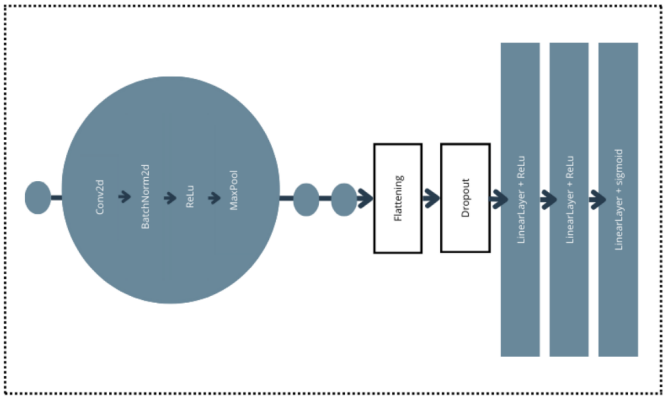


Figure 1.7: Binary Classification Network [3]

1.2.2 EfficientPose

The CNN follows the approach designed by Park et al [4], based on the EfficientDet feature encoder, comprising EfficientNet as a backbone and the Bidirectional Pyramid Network (BiFPN), which output is given to the multiple prediction heads. These heads have different works as bounding box detection, keypoints prediction, target rotation and translation regression. There's also the Heatmap head, designed to process images maintain high resolution and produce heatmaps as outputs.

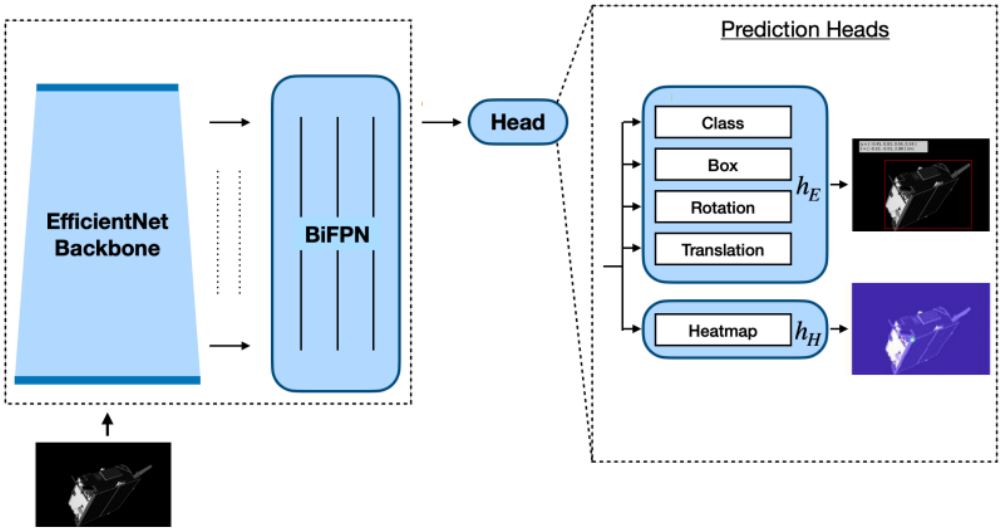


Figure 1.8: Park's proposed architecture [4]

Starting from EfficientNet, it was selected as the backbone for the model due to its efficiency and high performance in image detection. The backbone features extracted by EfficientNet give detailed information about the object shapes, textures, and contexts being effective in capturing meaningful representations of input images.
The BiFPN is a lightweight and efficient module consisting of a top-down pathway that aggregates features from higher levels of the network and a bottom-up pathway that aggregates features from lower levels. These pathways are strictly connected by fusion nodes that combine features from any level using a weighted sum. During the training phase, the weights of these nodes are learned to optimize the feature fusion process.

Chapter 2

Mathematical Background

2.1 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) represent a specialized class of deep neural networks designed to leverage the spatial structure present in grid-like data, making them particularly well-suited for tasks involving image-driven pattern recognition tasks, such as classification, detection and segmentation. CNNs stand out as one of the most powerful types of Artificial Neural Networks (ANNs), deriving their name from convolution, a linear mathematical operation they employ to extract features from input data.[9]
The architecture of CNNs is inspired by the visual perception system of living beings and is composed of one or more groups of convolution and pooling layers, followed by fully connected (FC) layers and an output layer. In the subsequent sections, the characteristics of these three key building blocks will be explored in detail.

2.1.1 Convolutional Layer

A convolutional layer can be seen as the "eyes" of a CNN, as each neuron in this layer is responsible for detecting specific features within the input. Typically, the input to a convolutional layer is either the original image, in the case of the first layer, or the output from a previous layer in the network. Convolutional layers perform convolution operations on the input data using learnable filters, also known as kernels. These filters are small matrices of weights that act as feature detectors, designed to identify specific patterns or characteristics in the input data. During the convolution process, the filter is systematically applied to the input data, and at each step, the dot product

between the filter and the corresponding region of the input is computed. Each filter is trained to focus on localized areas of the input to extract particular features. The result of this process is a set of feature maps that highlight the most relevant patterns and features present in the input.

2.1.2 Pooling layer

Pooling is a mathematical operation required in CNNs. A pooling operation replaces the output of the convolution operation at a certain location down-sampling the spatial dimensions of the input feature maps. The result is the reduction of the number of parameters and computational complexity of the network. The pooling layer, usually inserted between convolutional layers, makes the representations more computationally efficient and reduces overfitting, in particular when it's dealing with large-scale datasets.[9]
The most popular pooling operation is max pooling, it summarizes the input as the maximum within a rectangular neighbourhood but does not introduce any new parameter. So, the advantage is the aggressive reduction of the parameters. The figure below illustrates a simple example of how max pooling functions. The input consists of a 4x4 slice, with a 2x2 filter applied and a stride of two. In the first 2x2 region, the maximum value is 7, which is stored in the output channel. As the filter slides over by 2 pixels, the maximum value in the next region (highlighted in green) is again 7. Once the edge is reached, the filter resets to the left and moves down by 2 pixels. In the yellow region, the maximum value is 6, which is stored in the output channel, and the same process is applied to the neighbouring region. After completing the operation, the result is a 2x2 down-sampled representation of the input slice, while maintaining the depth of the volume at its original size.

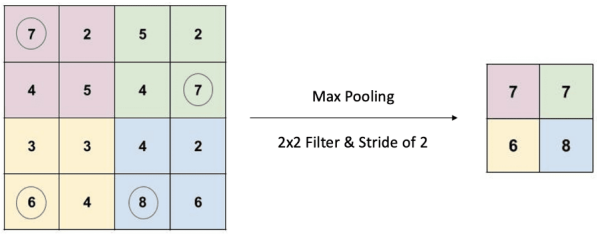


Figure 2.1: MaxPooling example

2.1.3 Fully connected Layer

The final component of every CNN architecture consists of fully connected layers, commonly referred to as dense layers. These fully connected layers receive input from the last convolutional or pooling layer, known as the feature map. This feature map is flattened into a vector, which is subsequently fed into the fully connected layer to produce the final output of the CNN for classification or regression tasks. In fully connected layers, each neuron receives input from all neurons in the previous layer and generates an output that is transmitted to all neurons in the subsequent layer.

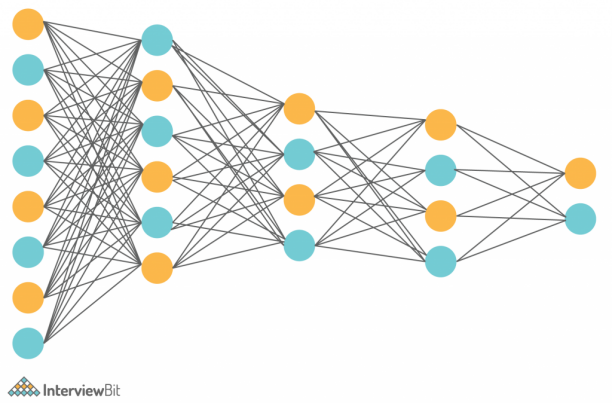


Figure 2.2: Fully connected layer network

The number of neurons in this layer determines the dimensionality of the learned feature space, while the weights associated with each connection are adjusted during the training process. This type of connectivity enables the network to learn complex patterns and relationships within the input data effectively.

2.1.4 Back-propagation

Once outlined the general architecture of a neural network, it is essential to understand how it operates, particularly the training process through which the network learns to perform the required task. It's possible to describe it in two distinct operations: the forward pass and the back-propagation. The forward pass begins by propagating the input data through the network, starting from the input layer, passing through the hidden layers, and finally reaching the output layer where the network's predictions are made. The

error is then calculated based on the difference between the predicted output and the actual target values.[10] Next, the backward pass is initiated. During this phase, the error is propagated backwards through the network, from the output layer to the input layer, updating the weights along the way to improve the network's performance.

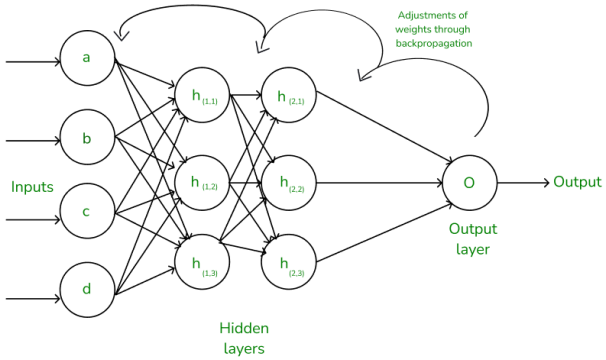


Figure 2.3: Back-propagation pass [10]

This process, known as back-propagation, is an iterative algorithm designed to minimize the cost function by determining which weights and biases need adjustment. In each iteration or epoch, the model learns by updating these weights and biases, using the gradient to guide how much each parameter should be modified to reduce the error in the next forward pass.

2.1.5 Loss Functions

This section examines the primary loss functions utilized in neural networks for object detection and pose estimation, which are crucial for evaluating network performance. Loss functions act as essential tools to guide the learning process, measuring how accurately network predictions align with actual data. Here, particular focus is placed on the Intersection over Union (IoU) and SPEED loss, each tailored to capture different performance metrics [3].

The IoU quantifies the overlap between predicted and actual regions, providing a ratio that reflects prediction accuracy relative to the ground truth. This metric is especially effective because it penalizes both false positives and false negatives, fostering balanced prediction outcomes.

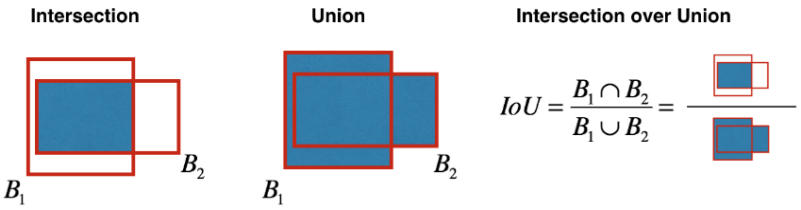


Figure 2.4: IoU [11]

The SPEED loss serves as the official performance metric for the Satellite Pose Estimation Challenge [12], specifically designed to evaluate predicted satellite pose in terms of both positional accuracy and orientation.

For position, the error e_t is calculated as the 2-norm difference between the ground truth position t_{gt} and the predicted position t_{pr}. To account for variations in target distance, the normalized error is computed, which heavily penalizes positional errors when the target satellite is closer. [12]

$$\bar{e_t} = \frac{||t_{gt} - t_{pr}||_2}{||t_{gt}||_2} \tag{2.1}$$

Orientation error e_q, in contrast, is derived as the angular distance between the true and predicted quaternions q_{gt} and q_{pr}, respectively.

$$e_q = 2arccos|q_{gt} \cdot q_{pr}| \tag{2.2}$$

The overall pose error is then determined by combining the normalized position error with the orientation error, providing a comprehensive metric for evaluating satellite pose accuracy.

$$e_{pose} = \frac{||t_{gt} - t_{pr}||_2}{||t_{gt}||_2} + 2arccos|q_{gt} \cdot q_{pr}| \tag{2.3}$$

2.2 Hyperparameters

Hyperparameters play a crucial role in determining how well a model generalizes from training data to unseen data.
A Machine Learning model is a mathematical model with several parameters that need to be learned from the data. These are referred to as model parameters, which are internal to the model and are automatically adjusted by the algorithm during the training phase. These include weights and biases, and they cannot be manually set; they are learned through the training process and later used for making predictions.
The training phase focuses on selecting the most suitable hyperparameters, which guide the learning algorithm to achieve optimal results. Unlike model parameters, the user explicitly defines hyperparameters to manage the learning process [13], and they must be set before training begins. These top-level parameters influence crucial aspects of the model, such as its complexity and the speed at which it learns. It is essential to note that hyperparameters are external to the model and cannot be altered once training starts.
Tuning hyperparameters significantly impacts the model's performance, as determining the ideal values for a specific problem is often challenging. Poor hyperparameter selection has been recognized as a major obstacle to progress in machine learning research. To minimize errors in hyperparameter selection, it is important to understand the most commonly used hyperparameters and their influence on model performance. [13]

The following section will explore eight key hyperparameters commonly employed in tuning machine learning models.

Learning rate
The learning rate controls the size of the steps a model takes during gradient descent, the method used to minimize the loss function. It also determines the frequency of cross-checking with model parameters. Choosing an optimal learning rate is challenging because a high learning rate allows the model to learn quickly by taking larger steps, but it risks overshooting the optimal solution. Conversely, a low learning rate ensures slower, more precise progress but can cause the model to get stuck in local minima.

Learning rate factor
The learning rate factor is a multiplier that adjusts the learning rate dynamically throughout training. It helps maintain stability and improve convergence

by gradually reducing the learning rate as training progresses.

Scheduler
Schedulers are tools that adjust the learning rate over time to enhance training efficiency and model performance. They address issues like slow convergence or overshooting by adapting the learning rate based on the training's progress. The choice of scheduler depends on the model's architecture and specific goals. [3]

- Step: The step scheduler lowers the learning rate by a specific factor at predetermined epochs. The learning rate stays constant until the scheduled epoch is reached, after which it drops by the set factor. This method is useful for gradually decreasing the learning rate to aid in convergence during training.

- Cosine: The cosine scheduler adjusts the learning rate based on a cosine function throughout the training process. It begins with the learning rate at its highest value and then gradually reduces it in a smooth, cosine-shaped curve until it approaches near zero by the end of training.

- Exponential: The exponential scheduler reduces the learning rate exponentially throughout training. This is done by applying a fixed decay rate after each epoch, resulting in a steady exponential decrease in the learning rate as training progresses.

Batch size
The batch size refers to the number of samples the model processes before updating its gradients. Batch sizes can range from a single sample to the entire dataset. Typically, a range of 1 to a few hundred samples offers the fastest training, but this depends on hardware like the GPU.

Number of epochs
An epoch can be defined as the complete cycle for training where the model uses the entire dataset once, performing both forward and backward passes. The number of epochs needed varies by model and is determined by monitoring validation errors. A single epoch in training is not enough. It leads to underfitting so the number of epochs is increased until there is a reduction in a validation error until there is no improvement for the consecutive epochs.

Epochs for learning rate decay
This hyperparameter specifies the number of epochs after which the learning rate is reduced. It can be set to any value between 0 and the final epoch of training.

Weight decay interval amplitude
The hyperparameter defines the amplitude of the weight decay interval in case of the step scheduler is chosen.

Optimizer
Optimizers play a crucial role in minimizing the loss function by iteratively updating a model's weights and biases. They adjust the parameters to guide the model toward convergence. The most relevant optimizer algorithms are:

- The Adam optimizer is adaptive, calculating individual learning rates for each parameter by adjusting the step size based on the square root of the accumulated second moments of the gradients.
- AdamW is a variation of Adam that separates weight decay regularization, which helps prevent overfitting, from the optimization step. This decoupling allows the model to better focus on minimizing the loss function.

Chapter 3

Methodology and tools: Optimization of Convolutional Neural Networks

Optimizing neural networks is essential to enhance performance and efficiency. By fine-tuning hyperparameters and reducing model complexity, optimization helps ensure that networks generalize well to unseen data while minimizing computational resources. This is particularly important in real-world applications, where high accuracy and reduced processing time are critical for tasks like image recognition and natural language processing.

3.1 Hyperparameters tuning: Black-box optimization

Hyperparameters are important in determining the performance of CNN-based algorithms. However, finding the optimal set of hyperparameters is often a complex and inefficient process and traditionally relies on subjective trial-and-error methods. In this context, black-box optimization (BBO) has emerged as a powerful approach to tackle optimization problems where the objective function is unknown, complex, or does not have a clear mathematical representation.

BBO is therefore particularly suitable for hyper-parameter optimization, an important challenge aimed at improving model performance. The objective function in black-box optimization $f : X \to R$ must be optimized within a limited evaluation budget, where the term 'black-box' means that there is no additional information about the function beyond its evaluations at specific points. In scenarios where each evaluation is resource-intensive, it is essential to carefully select input values to efficiently generate a set of inputs that converges to a global optimum.
The BBO algorithm is particularly effective in determining optimal operating parameters in systems where performance depends on tunable parameters. The application of BBO to hyperparameter optimization (HPO) provides an automated, systematic approach that overcomes the shortcomings of heuristic methods and significantly improves the performance of CNN-based algorithms. By systematically exploring the hyperparameter space, HPO methods search for configurations that maximize performance metrics. Furthermore, these methods allow the integration of domain knowledge and prior knowledge, enabling more informed and efficient optimization decisions.

3.1.1 Optuna: Define by-run optimization network

Identifying optimal hyperparameters is often a complex task that may require inefficient trial-and-error approaches. HPO offers a more structured approach to enhance the performance of CNNs. In this study, the Optuna library is employed as the HPO framework. Optuna is an advanced, open-source optimization tool developed under the define-by-run paradigm. It utilizes sophisticated algorithms for hyperparameter sampling and effectively prunes unpromising trials, enhancing exploration efficiency.
The define-by-run approach, a contemporary concept in deep learning, allows users to build and modify deep neural networks during execution dynamically. In the context of optimization frameworks, this term refers to a design that permits users to construct the search space dynamically, eliminating the need to predetermine all aspects of the optimization process. In Optuna, hyperparameter optimization is formalized as the minimization or maximization of an objective function, which takes a set of hyperparameters as input and returns a corresponding score.
Optuna organizes this process into *studies*, where each objective function evaluation is referred to as a *trial*. During each trial, the objective function interacts with a "trial object," progressively constructing the function as the

trial unfolds. The search spaces for hyperparameters are thus defined dynamically through interactions with the trial object during the runtime of the objective function.
Attention is focused on the sampling and pruning algorithms employed in constructing the experimental setup with the Optuna framework. These algorithms are essential for optimizing the hyperparameter search process, enabling efficient exploration of the parameter space and dynamic elimination of suboptimal trials, thereby enhancing computational efficiency and overall performance.

Sampler algorithms

The efficient parameter-sampling algorithms allow the first step for the user-customization framework. Optuna offers a variety of samplers that it's possible to divide into two types of sampling methods: independent sampling which samples each parameter independently and relational sampling which exploits the correlations among the parameters.
Independent sampling methods such as the Tree-Structured Parzen Estimator (TPE) are recognized for their strong performance even without utilizing parameter correlations. The Covariance Matrix Adaptation Evolution Strategy (CMA-ES) represents an example of a sampling algorithm that leverages relationships within the parameter space. In this approach, candidate solutions, corresponding to hyperparameters, are drawn from a multivariate Gaussian distribution. Following the evaluation of all sampled solutions, they are ranked based on their performance metrics. The parameters of the Gaussian distribution are subsequently updated under this ranking to refine the sampling process [14]. The cost-effectiveness of both relational and independent sampling is contingent on the environment and the task at hand. Optuna can handle various independent sampling methods like TPE as well as relational sampling methods such as CMA-ES or Independent Natural Gradient Optimization (INGO).
However, Optuna offers a naive option for the TPE algorithm, the reasons why it became so popular is for its great flexibility and outstanding performance demonstrated for HPO in DL models. In this paragraph, it will be discussed the algorithm's intuition and the role of its control parameters. TPE is a variant of Bayesian optimization (BO), where in general the goal of BO is to minimize the objective function, such as finding the optimal configuration for the hyperparameter. It iteratively searches for optimum using the acquisition function to trade off the degree of exploration (research of unseen regions) and exploitation (research region near promising observation). [15]

In the TPE algorithm, the search space is divided into regions following a tree-like structure, where each node represents a region. At each iteration, it explores different regions of the search space to discover promising areas that may contain the optimal solution. This exploration is helped by prioritizing regions that have not been explored yet. Then, a probabilistic model is built to estimate the performance of the configurations of every node (region).
Going deeper into the functionality and the role of its parameters, the TPE can be divided into the splitting algorithm, the weighting algorithm, and the kernel functions for building the surrogate model.

1. Splitting algorithm: utilizes the gamma splitting function to divide the search space into two regions as the better and worse group, which are then explored separately to identify the optimal hyperparameters. The quantile is used to determine the proportion of the search space allocated to each child node. [15]

2. Weighting algorithm: evaluates the objective function at each node in the tree and calculates the weight based on its evaluation score and the scores of its parent and child nodes. Then, it assigns the calculated weights to each node. The functions model the relationship between hyperparameters and the evaluation function, enabling the algorithm to search for optimal hyperparameters efficiently. [15]

3. Kernel functions: transform hyperparameters into a higher-dimensional feature space, where the relationship between hyperparameters and the evaluation function is more linear. Several kernel functions can be used and it depends on the type of hyperparameter, for example, categorical or numerical. But there's also another aspect that is important to mention and it's the difference between univariate and multivariate kernels. The first one can handle conditional parameters because of the independence of each dimension instead the multivariate cannot but the implementation of the latter enhanced the general performance at the expense of conditional parameter handling.[15]

Pruner Algorithms

The pruning algorithm is crucial for maintaining the *cost* aspect of cost-effectiveness. Generally, the pruning mechanism operates in two stages: it regularly checks the intermediate objective value and stops any trials that do not satisfy a set condition. Optuna, like the samplers, provides various pruning

algorithms such as MedianPruner, HyperOpt, and RandomPruner, but the Asynchronous Successive Halving (ASHA) algorithm has shown exceptional performance compared to the others.
ASHA is a robust enhancement of the Successive Halving (SHA) method. It allows each worker to perform aggressive early stopping asynchronously based on the provisional ranking of trials. This makes it well-suited for parallel computation, enabling the simultaneous processing of multiple trials without delays. The SHA is based on distributing a limited budget to each hyperparameter configuration, assessing all configurations, and keeping the best ones. The budget for each configuration is then increased by a factor of η, and this is repeated until the maximum budget R is achieved.[16]
This optimization process is influenced by several parameters, including the number of hyperparameter configurations, the minimum and maximum budget for each configuration, the reduction factor η, and the early-stopping rate s. The early-stopping rate is particularly significant, as lower values lead to more aggressive early stopping, impacting the resources allocated to each configuration. This optimization can be visualized as a series of "rungs" or levels, where each rung signifies an increase in the budget for each configuration and a decrease in the number of configurations retained. [16]
Parallelization of SHA is essential for enhancing its efficiency in large-scale scenarios, but strategies like running multiple instances of SHA on each worker are ineffective in such contexts. Therefore, the ASHA algorithm was developed, utilizing asynchronous to address stragglers and maximize parallelism. The key distinction is that ASHA advances configurations to the next rung whenever feasible, rather than waiting for their completion before moving on to the next stage.

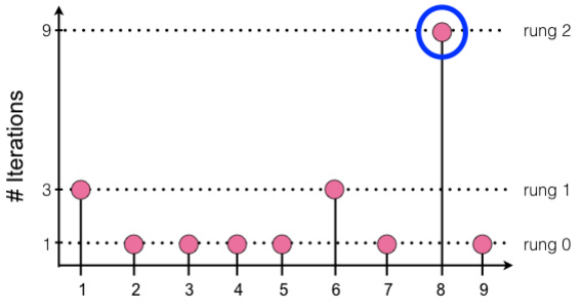


Figure 3.1: ASHA promoting rungs [16]

3.2 Model reduction

CNNs have demonstrated outstanding performance across a wide range of applications. This success is often attributed to their increasingly deep architectures, which come at a high computational cost. These architectures can contain millions of parameters, demanding substantial computing power and making them difficult to deploy on hardware with limited resources. Model compression offers a potential solution to this issue by reducing the number of parameters, computational demands, and memory usage. This work explores various strategies to implement the most effective method for compressing a large, custom CNN. The techniques examined include pruning, quantization, and knowledge distillation.

Quantization

One of the most effective techniques to achieve this is quantization, which involves reducing the precision of weights and activations. By lowering precision, quantization helps to decrease both the computational burden and memory requirements of a model, making it a popular approach in various machine-learning applications. Quantization offers several advantages that make it a versatile optimization technique. Notably, it can be applied across a wide range of models and use cases without necessitating changes to the underlying architecture. This flexibility allows developers to take a pre-trained floating-point model and convert it into a fixed-point quantized model with minimal accuracy loss. For example, 8-bit quantization is a widely used approach that can reduce the model size by a factor of four while maintaining accuracy levels close to those of the original floating-point model. Also, moving 8-bit data is four times more efficient than 32-bit floating-point data, which is particularly beneficial where memory access is a significant source of power consumption.[17]

There are two primary approaches to quantization: Post-Training Quantization (PTQ) and Quantization Aware Training (QAT). PTQ is a simple and effective method that quantises models after training, with limited data required for the process. Since it does not involve modifying the training process, PTQ is an attractive option when time or computational resources are limited. However, the simplicity of PTQ can sometimes come at the cost of accuracy. In certain cases, the loss in precision introduced by quantization can lead to unacceptable performance degradation, especially in more complex

models.[17]
In scenarios where accuracy loss from PTQ is intolerable, QAT offers a more robust solution. It integrates quantization into the training process itself, allowing the model to adapt to the lower precision environment during its learning phase. This is achieved by inserting fake quantization modules into the model during training, which simulate the effects of quantization—such as clamping the values to the reduced precision range. So, the model learns to accommodate the limitations of quantized representations, leading to a final model that better retains accuracy when converted into a fully quantized version. Once the training is complete, the model is converted into a quantized integer model using the information captured by the fake quantization modules. [18]

Knowledge Distillation

Knowledge Distillation (KD) is a model compression technique employed in CNNs that facilitates the transfer of knowledge from a larger, more complex model known as the teacher to a smaller, more efficient model known as the student. [19] The process involves training the student model not only on the original training dataset but also utilizing the labels generated by the teacher. This richer information helps the student model learn the patterns and relationships in the data more effectively.
By mimicking the teacher's behaviour, the student can achieve comparable performance while being significantly smaller in size, leading to faster inference times and reduced computational resource requirements. This technique is particularly advantageous for deploying CNNs on resource-constrained devices, such as embedded systems, where computational efficiency and speed are relevant.

Pruning

The objective of pruning is to minimize the number of parameters without significantly affecting the performance of the models. Most research on pruning has been done on CNNs for the image classification task, which is the foundation for other computer vision tasks.

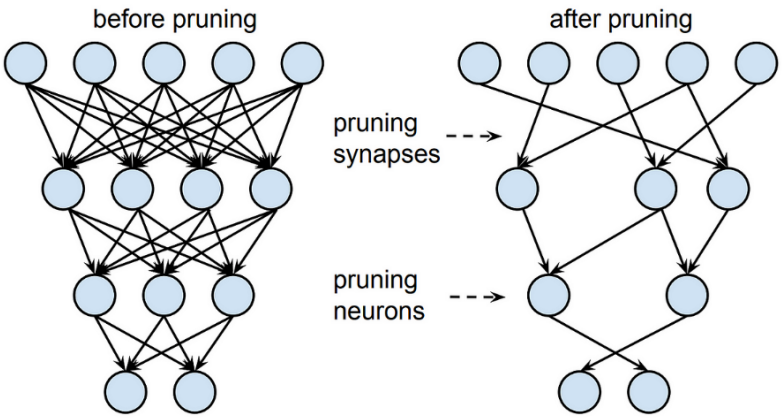


Figure 3.2: Visual pruning scheme [20]

Pruning can be categorized into unstructured and structured pruning.
The unstructured pruning removes individual weights or connections within the network based on predefined criteria, resulting in a sparse model representation. This irregular sparsity can be more difficult to optimize on certain hardware, as the remaining parameters are not organized in a predictable pattern. [21] Additionally, this method often necessitates fine-tuning and retraining to preserve the performances. On the other hand, in structured pruning, entire components such as channels, layers, or filters are removed. Because large structural elements are pruned, the resulting model is often smaller and maintains a regular architecture, which is beneficial for optimizing hardware acceleration. [21] Such pruning often leads to significant speedups during inference, especially on GPU.

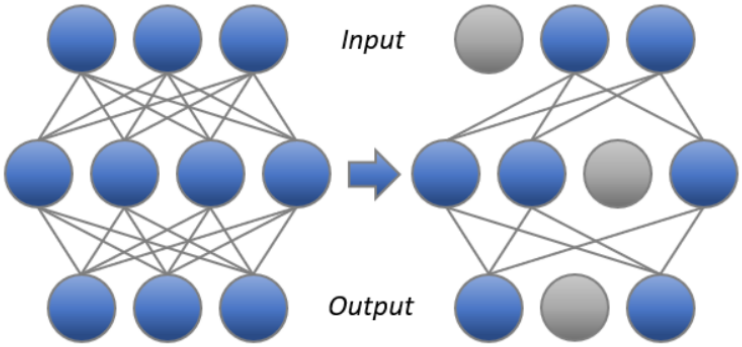


Figure 3.3: Visual structuring pruning scheme [20]

Since structured pruning tends to be more effective on larger networks, and this work aims to reduce the number of parameters while maintaining efficiency, we will explore its methods and applications in detail. Structured pruning is a technique designed to reduce the complexity of deep convolutional neural networks by systematically removing structural elements such as filters, channels, or entire layers. This method improves computational efficiency and reduces memory usage, which is critical for deploying deep learning models in environments with limited resources. Filter pruning removes entire convolutional filters based on criteria like weight magnitude or filter relevance. This approach is effective because filters that contribute less to the final prediction can be eliminated without a significant loss in accuracy. This strategy simplifies the network, minimizing the resources required for both training and inference.
Methods such as channel pruning take a slightly different approach by focusing on the removal of channels, or feature maps, that represent intermediate representations of the input. Channels that carry redundant or non-informative features can be pruned, leading to a streamlined network architecture. This reduces the size of intermediate data passed between layers, saving both memory and computational power.
It's possible to implement a more aggressive approach, such as layer pruning, that involves eliminating entire layers from the CNN, which is feasible in cases where some layers contribute little to model performance. This method is often guided by architectural search strategies that balance the trade-off between pruning too many layers and maintaining network efficacy.[22]
Pruning can be integrated during different stages of the network life cycle.

Dynamic pruning during training is one approach, where the network is iteratively pruned while it is still learning. This dynamic adaptation allows the model to retain high performance by gradually learning to optimize itself with fewer parameters. Additionally, techniques like Dynamic Pruning with Feedback (DPF) utilize feedback loops to further refine which components of the network can be pruned during training.
Alternatively, pruning can be applied after the network has been fully trained, often known as pruning during inference. In this case, techniques like Rank-Normalized Pruning (RNP) are used to remove filters or channels that are less important during the inference phase. This method is advantageous for reducing the computational load during real-time deployment, where minimizing latency is a priority. Here, the focus is not on adjusting the network during training but on trimming unnecessary components after the model has already learned.[22]

Chapter 4

Test and Results

In this section, the tests and results of the optimization experiments are presented. Each subsection details the experimental setup, the outcomes of the conducted experiments, and a consequent discussion of the obtained results. This discussion highlights the strengths of the implemented optimization while addressing the challenges encountered, serving as a basis for future improvements.
The mean performances, i.e., the average errors evaluated on the inference dataset, are presented in tables. The reported errors are Intersection over Union (IoU) and the SPEED loss, which includes position error, orientation error, and pose error designed to evaluate the predicted satellite pose in terms of both position and orientation (refer to 2.1.5 for more details).

4.1 HPO

This section explores the approach used to optimize hyperparameters, leveraging the Optuna tool as the primary method for conducting this task. The results obtained by Lovaglio [3] serve as the baseline for this optimization process, providing a reference point for evaluating improvements.

4.1.1 Setup and Test

The initial step in the optimization strategy involves identifying the specific hyperparameters that will be targeted for tuning. These hyperparameters

influence the model's performance, and optimizing them effectively can significantly enhance accuracy and efficiency. Once the relevant hyperparameters have been identified, the next task is to establish a suitable range of values that each hyperparameter can assume. It is essential to strike a balance between exploring a broad spectrum of configurations and maintaining focus on ranges known to produce desirable results. The table below provides a detailed overview of the hyperparameters chosen for optimization, along with the specific ranges of values considered during the search process.

HPO	**Values**
Scheduler	Step, Exponential, Cosine Decay
Learning rate	$[10^{-5}; 10^{-3}]$
Learning rate factor	$[0.1; 0.9]$
Batch size	$[3; 8]$
Number of epochs	$[100; 150]$
Optimizer	Adam, AdamW
Weight decay interval amplitude	$[1; 2]$
Epochs for learning rate decay	[0; End epoch]

Table 4.1: Range of hyperparameters for HPO

For instance, the learning rate must remain relatively small, as increasing it beyond 10^{-3} would not be practical or effective. Similarly, choosing an appropriate learning rate scheduler and determining a suitable range for the batch size is equally important. For the batch size, it is well understood that increasing it to a certain threshold, typically greater than 1, can lead to improved performance due to more stable gradient estimates and better utilization of hardware resources.
This careful balance between range exploration and constraint is essential to ensure that the optimization process is both thorough and efficient, leading to improved model performance without overstepping practical bounds.

4.1.1.1 Final Approach Dataset

The dataset consists of around 3000 RGB images, split into training, validation, and testing sets in a 70-15-15 ratio. The images represent a mission scenario where the chaser follows a straight-line trajectory toward the target along the InTrack axis to achieve mating conditions. This scenario, known as the Final

Approach (FA), covers distances ranging from 8 to 80 meters from the target.

To speed up the training and inference processes without sacrificing quality, the original image resolution of 2048x1536 has been reduced to 512x384. This reduction is supported by Lovaglio's work [3], which demonstrated that a graphical downgrade does not negatively impact performance. The findings in table 4.8 confirm this theory, showing that the reduced resolution maintains the model's performance while improving processing efficiency.

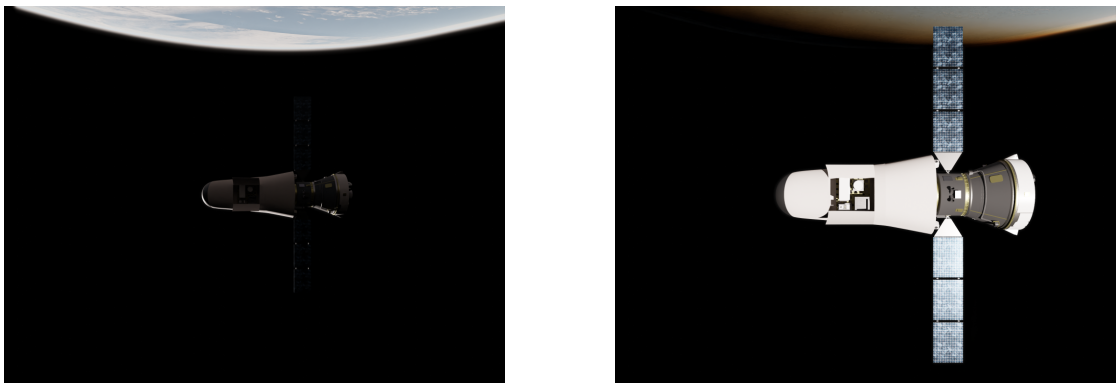


Figure 4.1: Images from FA Dataset

As discussed in 1.1.1, training with synthetic images can be particularly challenging. Data augmentation techniques are essential to improve training performance and minimize overfitting. To this end, using the available data, the Albumentations library has been applied to enhance the diversity of training samples.

Setup

The following section outlines the methodology and options selected for the optimizer, which are tailored to the specific characteristics of the network, such as its size, the objective function to minimize, and the nature of the hyperparameters involved (integer, categorical, or float). For more in-depth information, refer to the theory discussed in 2.2 and the Optuna documentation [23]. For this thesis, the multivariate TPE sampler was chosen due to its robustness and versatility, making it well-suited for integration with pruners, an essential requirement to accelerate the optimization process. ASHA was selected as the pruner to discard unpromising trials due to its ability to evaluate trials asynchronously and promote promising configurations without waiting for all rounds to finish. The table below outlines the strategy and options

employed.

TPE Sampler	Multivariate: true	consider magic clip: true
ASHA Pruner	min resource: auto	reduction factor: 4

Table 4.2: Optuna Sampler and Pruner configuration

Two optimization simulations are conducted with differing numbers of trials, specifically 10 and 20. The reason behind this is to assess whether the TPE algorithm requires more than 10 trials, the minimum needed for the algorithm to explore the hyperparameter space effectively.

Test case with 10 trial

In this case, the following optimal hyperparameters were identified.

HPO	**Values**
Scheduler	Exponential
Learning rate	0.000757
Learning rate factor	0.783
Batch size	4
Number of epochs	115
Optimizer	AdamW
Weight decay interval amplitude	1
Epochs for learning rate decay	27

Table 4.3: Set of optimized hyperparameters - FA 10 trials

It is important to note that the objective function is relatively easy to minimize when using the FA dataset. Consequently, the optimization process is efficient, requiring only the time needed for a single evaluation round. This is facilitated by the ASHA pruner, which promotes promising configurations asynchronously, thus conserving resources by avoiding less promising alternatives. The following Optuna plots further illustrate the relationships between the hyperparameters, their significance, and the optimization history.

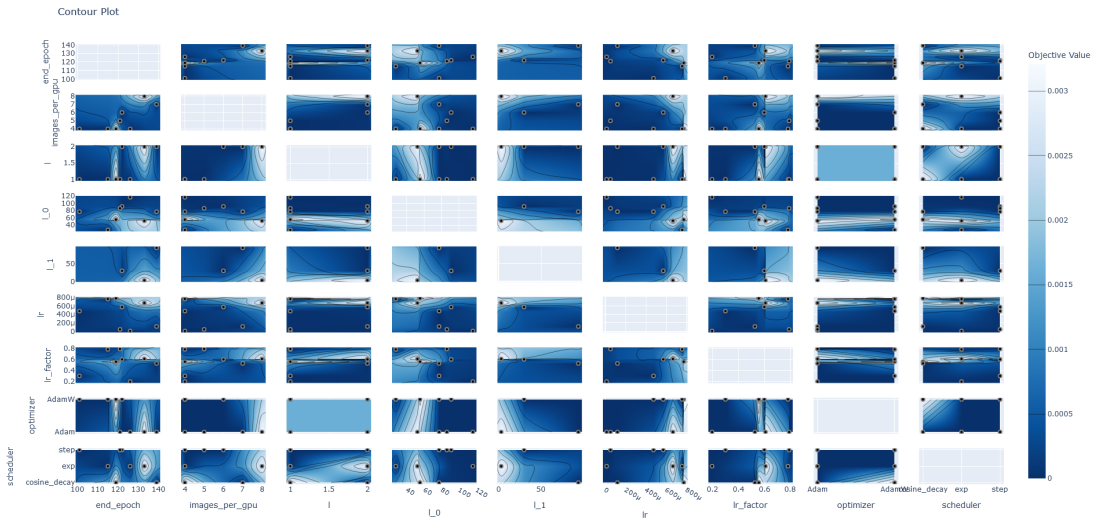


Figure 4.2: Parameter relationship - FA Dataset trial 10

The contour plot in the upper figure illustrates the relationships between various hyperparameters and the combinations of values that can minimize the objective function. In the figure below, the importance of each hyperparameter is displayed, ranked by their relative contributions to error reduction.

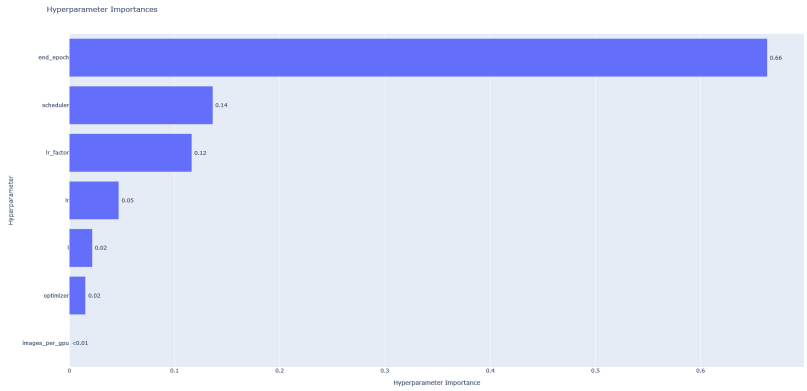


Figure 4.3: Hyperparameters importances - FA Dataset trial 10

As shown in the lower figure, the timeline demonstrates that the objective function was minimized as early as the first trial, while the second and ninth trials were disregarded and pruned. As the process progressed, the optimization consistently favoured the configuration from trial 1, which led to the minimization of pose error.

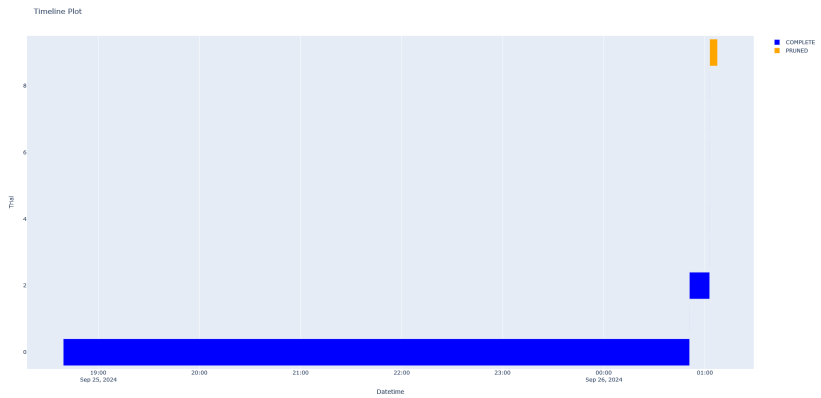


Figure 4.4: Timeline of the study - FA Dataset trial 10

This figure further confirms the previous observations regarding the timeline, showing that the objective function was minimized by evaluating only a limited number of trials.

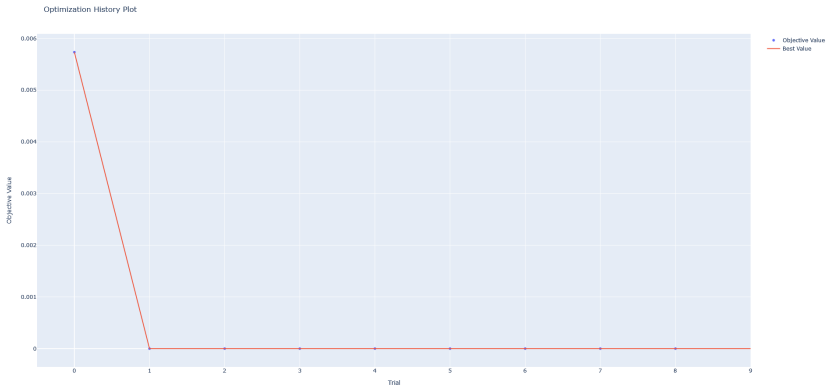


Figure 4.5: Optimization history - FA Dataset trial 10

Test case with 20 trials

This test confirmed the results of the first test, though it yielded a different set of parameters. As a result, extending the optimization process is unnecessary, as the objective function is effectively minimized within the early configurations of the first round.

HPO	Values
Scheduler	Step
Learning rate	0.000324
Learning rate factor	0.176
Batch size	5
Number of epochs	104
Optimizer	Adam
Weight decay interval amplitude	2
Epochs for learning rate decay	[11;104]

Table 4.4: Set of optimized hyperparameters - FA 20 trials

The contour plot in this case shows different relationships between various hyperparameters and combinations of values as expected.

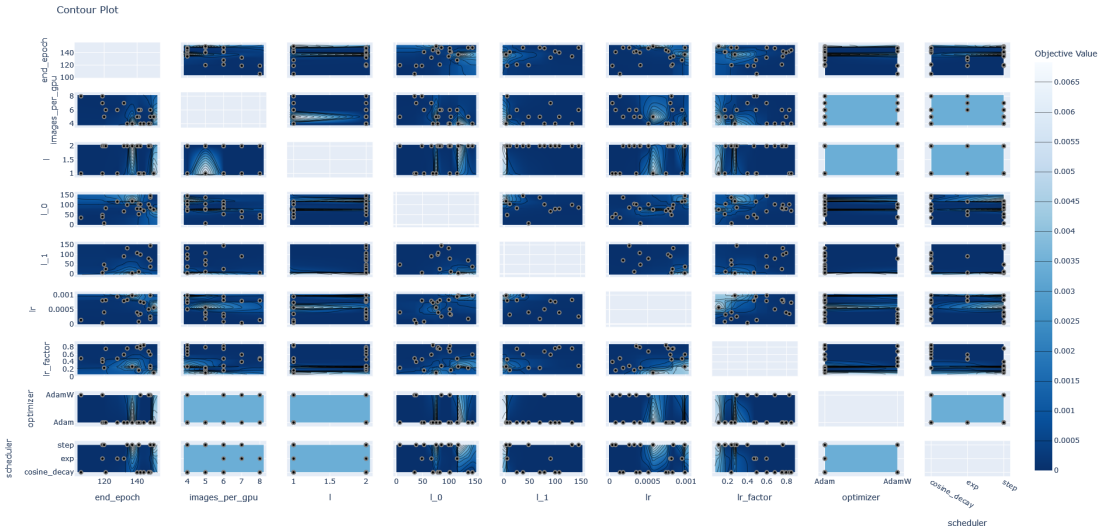


Figure 4.6: Parameter relationship - FA Dataset trial 20

In the figure below, the importance of each hyperparameter is displayed and it's also changed

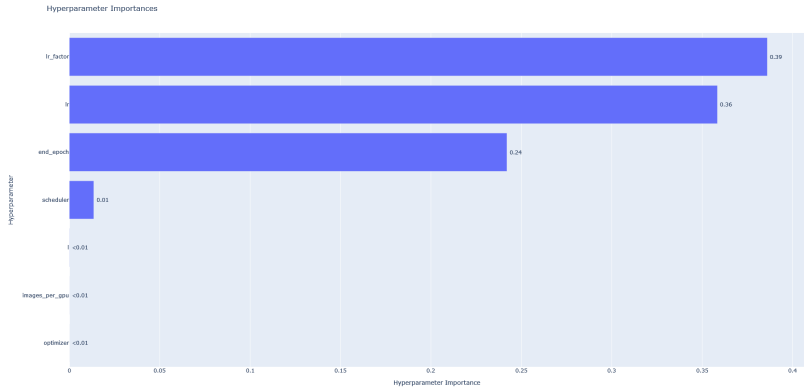


Figure 4.7: Hyperparameters importances - FA Dataset trial 20

As shown in the lower figure, the timeline demonstrates that the objective function minimization was achieved by exploring the first trial and evaluating others trials but the pruning was not needed.

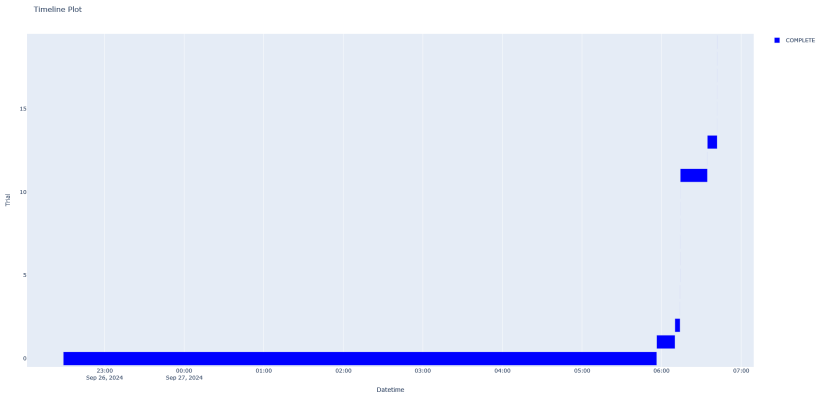


Figure 4.8: Timeline of the study - FA Dataset trial 20

As in the previous trial, the objective function was minimized by evaluating only a limited number of trials.

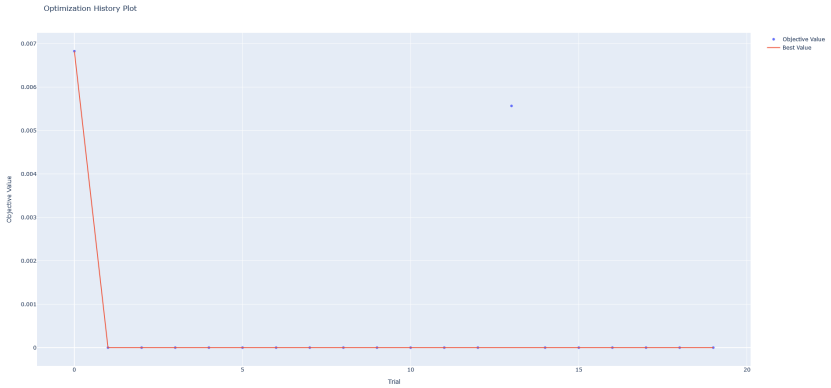


Figure 4.9: Optimization history - FA Dataset trial 20

4.1.1.2 Walking Safe Eclipse Dataset

The dataset comprises approximately 9500 RGB images related to the Walking Safety Ellipses (WSE) for Space Rider observation. In this dataset, the target spacecraft is placed in random positions and orientations relative to the chaser, with distances ranging from 600 to 100 meters. Like the previous dataset, the original resolution of 2048x1536 pixels is reduced to 512x384 for efficiency purposes. The dataset is split with an 80-20 ratio for training and validation/testing, and data augmentation using the Albumentations library is applied.

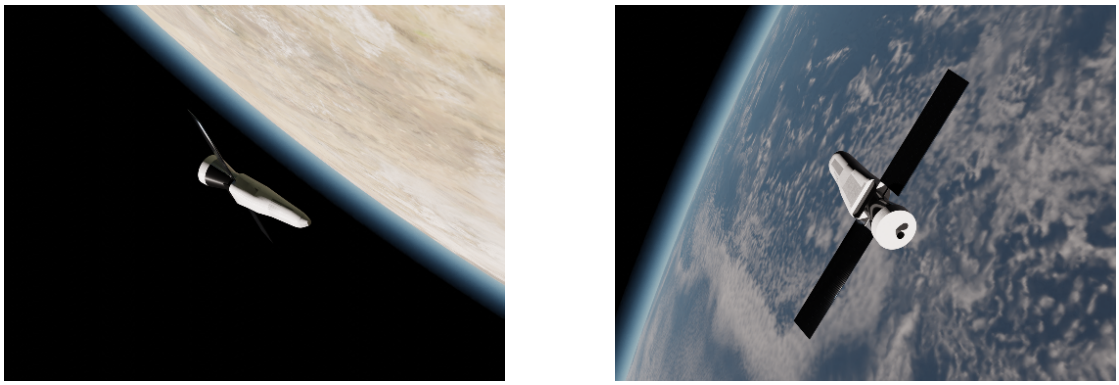


Figure 4.10: Images from WSE Dataset

Setup

The method applied to this dataset remains unchanged, utilizing the synergy between the TPE sampler and the ASHA pruner, with the number of trials set to 10, as it was confirmed unnecessary to increase this number in the previous dataset.
However, significant differences exist compared to the previous dataset, and it is expected that the objective function will be more challenging to minimize due to the increased distance from the target, and the fact that the target's orientation is not fixed but varies as the chaser follows an elliptic orbit around SR. As a result, the optimization process will probably take longer and the sampler will need to evaluate all trials, pruning only those deemed unpromising to minimize the pose error as much as possible.

Test case with 10 trial

In this case, the following optimal hyperparameters were identified.

HPO	**Values**
Scheduler	Cosine Decay
Learning rate	0.000151
Learning rate factor	0.418
Batch size	3
Number of epochs	116
Optimizer	AdamW
Weight decay interval amplitude	2
Epochs for learning rate decay	[3;110]

Table 4.5: Set of optimized hyperparameters - WSE

The contour plot in this test case shows an unexpected relationship between various hyperparameters and combinations of values, in particular, it seems that batch size and the optimizer do not correlate with other hyperparameters

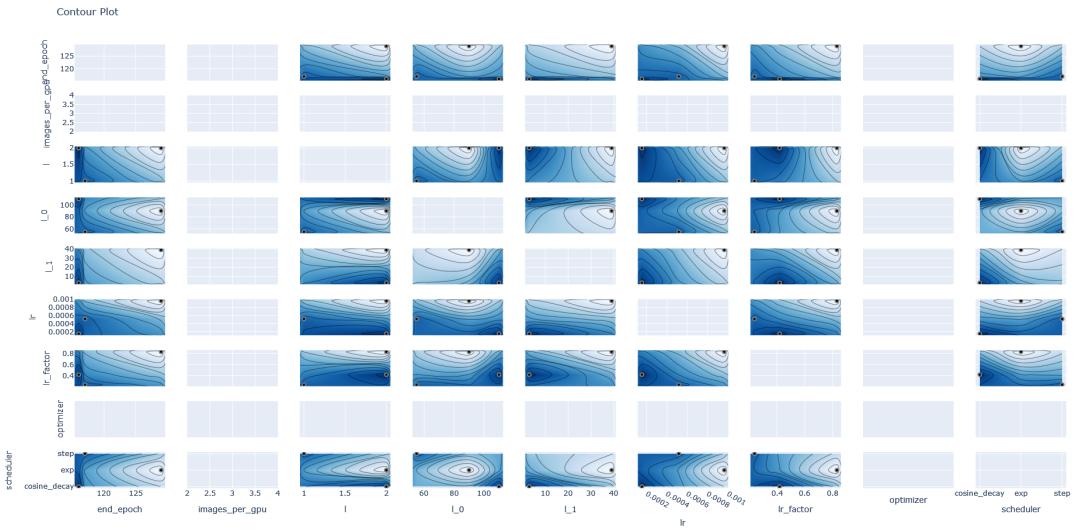


Figure 4.11: Parameter relationship - WSE Dataset trial 10

In the figure below, the significance of each hyperparameter is illustrated, explaining the lack of correlations with batch size and optimizer. This is due to their importance being ranked very low. This remains true when considering a ranking of hyperparameters that most significantly influence the minimization of the objective function. However, batch size plays a crucial role in calculating the error, which this ranking does not account for. This will be elaborated on later, but it is essential to highlight that the optimal batch size is heavily dependent on the structure of the images in the dataset. For example, the FA dataset's images are quite similar, as the model follows a straight path toward the target. This is not the case with the WSE dataset, where greater variability exists. As a result, the network likely requires a larger sample of images per iteration to compute back-propagation accurately.

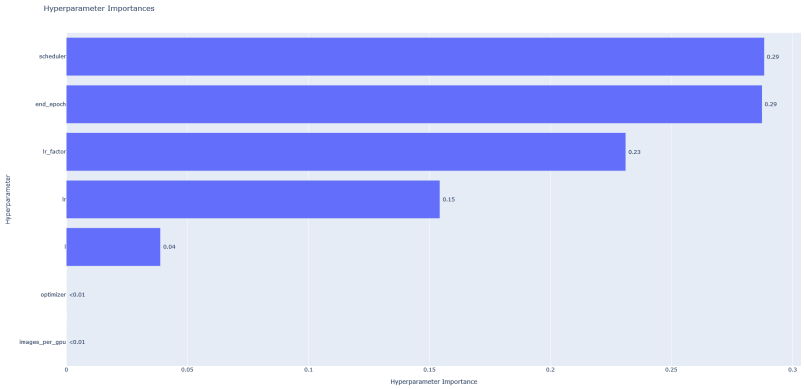


Figure 4.12: Hyperparameters importances - WSE Dataset trial 10

As shown in the lower figure, the timeline demonstrates that the objective function minimization was achieved by exploring all ten trials and pruning the most trials.

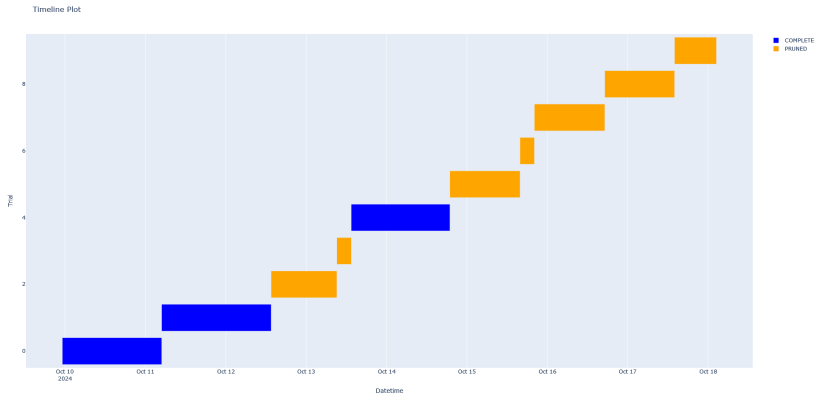


Figure 4.13: Timeline of the study - WSE Dataset trial 10

As in the previous trial, the objective function was minimized by evaluating the fourth trial.

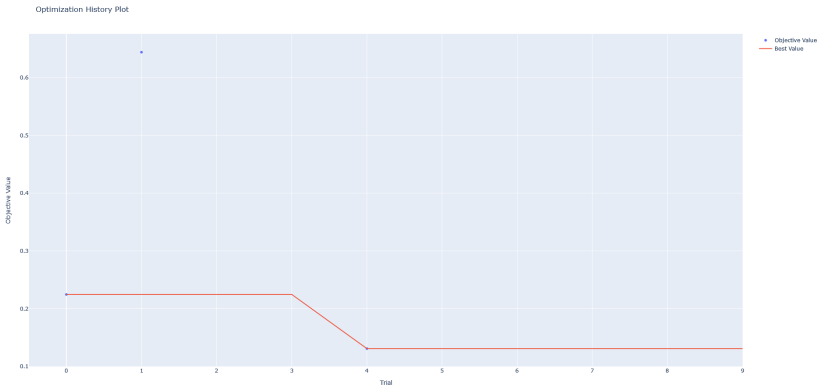


Figure 4.14: Optimization history - WSE Dataset trial 10

4.1.2 Results

FA Dataset

Regarding the optimization performed on the FA dataset, the following two tables present the mean performances, evaluated on the inference data, of the neural network optimized via HPO with 10 and 20 trials, respectively.

Method	**Dimension (pxl)**	**IoU (%)**	E_{TN} **[m]**	E_q **[deg]**	E_{pose}
Optimized - 10 trials	512x384	99.5	0.003	0.029	0.003

Table 4.6: Mean performances of FA optimization - 10 trials

Method	**Dimension (pxl)**	**IoU (%)**	E_{TN} **[m]**	E_q **[deg]**	E_{pose}
Optimized - 20 trials	512x384	99.1	0.005	0.04	0.005

Table 4.7: Mean performances of FA optimization - 20 trials

It is noteworthy that, in addition to the performances falling within an acceptable range, 10 trials are sufficient for the TPE sampler to effectively

and efficiently explore the hyperparameter space. Therefore, for subsequent work, we decided to proceed with 10 trials. Nonetheless, the most promising result was obtained with the 10-trial optimization, and this will be used for comparison with the original network. It is important to highlight, however, that the number of trials does not affect the success of finding the optimal solution, but rather the depth of the search. The choice of 10 trials represents a trade-off between computational time and the quality of results obtained.

The table below demonstrates that hyperparameter optimization is particularly effective in minimizing pose error, yielding better results than the baseline. It is of particular interest that the optimized results were achieved while reducing the image resolution by 75 %. Thus, through HPO, it is possible to achieve smaller losses by significantly decreasing the image resolution, which translates into lower computational cost and faster convergence during both the search for optimal solutions and network training.

Network	**Dimension (pxl)**	**IoU (%)**	E_{TN} **[m]**	E_q **[deg]**	E_{pose}
Not Optmized	1024x768	99.2	0.004	0.035	0.005
Optimized	512x384	99.5	0.003	0.029	0.003

Table 4.8: Comparison of FA mean performances

To sum up the improvements obtained through the optimization:

- The IoU loss experiences no improvement or decay
- The $\bar{e_t}$ error see an improvement of 25%
- The e_q error has an improvement of 17%
- The e_{pose} error experience improvement of 40%

It is therefore essential to investigate whether increasing the resolution back to the original, while using the optimized hyperparameters, will yield better results or if we have reached a limit beyond which further improvements are not feasible. The table compares the performance of the optimized network trained with low-resolution images (512x384) and high-resolution images (1024x798) to explore potential improvements.

Network	Dimension (pxl)	IoU (%)	E_{TN} [m]	E_q [deg]	E_{pose}
Optimized	512x384	99.5	0.003	0.029	0.003
Optmized	1024x768	99.2	0.004	0.025	0.004

Table 4.9: Comparison of FA optimized mean performances between different resolutions

The results indicate that no significant improvement was observed, and the mean performances remain within an acceptable range. Therefore, working with low-resolution images is a strategy that allows significant computational savings and reduced training time without degrading performance.

WSE Dataset

Based on the results obtained from the previous scenario and the applied methodology (refer to 4.1.1.2), the optimization was conducted using 10 trials and then, a new test case to solve some issues.

Test case - 10 trials

The following tables present the mean performances of the optimized network, as well as a comparison with the results of the original network trained on the current dataset.

Method	Dimension (pxl)	IoU (%)	E_{TN} [m]	E_q [deg]	E_{pose}
Optimized - 10 trials	512x384	93.5	0.020	5.65	0.11

Table 4.10: Mean performances of WSE optimization - 10 trials

Network	Dimension (pxl)	IoU (%)	E_{TN} [m]	E_q [deg]	E_{pose}
Not Optmized	1024x768	93.5	0.043	2.386	0.085
Optimized	512x384	93.5	0.020	5.65	0.11

Table 4.11: Comparison of WSE mean performances

The table above shows that the optimization process yields results that deviate from expected outcomes. Specifically, while the normalized error significantly decreases, a contrasting increase in orientation error occurs, which subsequently impacts the pose error as well. This unexpected outcome is likely attributable to an improper combination of hyperparameters. As mentioned in 4.1.1.2, the WSE dataset poses particular challenges for target orientation estimation, as the EfficientPose prediction head struggles to achieve high precision in attitude estimation at extended distances, often prioritizing the minimization of one error over the other, in this case, focusing on reducing distance estimation error. This results in variability in error calculation that is highly sensitive to both training and the hyperparameter set. Consequently, the baseline presented by Lovaglio [3] demonstrates a lower orientation error; however, as extensively shown, the distance error decreases with hyperparameter optimization. This issue remains an open avenue for future research, as it will be essential to identify an alternative strategy for selecting a hyperparameter set that can stabilize orientation estimation accuracy.

To sum up, the improvements obtained through the new optimization:

- The IoU loss experiences no improvement or decay
- The $\bar{e_t}$ error see an improvement of 53.4%
- The e_q error has a decay of 57%
- The e_{pose} error experience an overall decay of 22%

4.2 Model size reduction

As anticipated, the present work focuses on dynamic structured pruning to achieve a more compact and efficient neural network while enhancing the accuracy previously attained through hyperparameter optimization.

4.2.1 Setup and Test

Specifically, the pruning methods employed in the network leverage weight-dependent criteria to assess the significance of individual filters within the architecture. These techniques are relatively simple to implement and come with lower computational costs. Weight-dependent criteria can be divided into two main categories: filter norm and filter correlation. This research focuses on the first category, where filter norm values are used as the key metric for pruning. In general terms, the l_p-norm of a filter can be expressed as:

$$\left\|\mathbf{F}_i^l\right\|_p = \sqrt[p]{\sum_{n=1}^{N_l}\sum_{k_1=1}^{K_l}\sum_{k_2=1}^{K_l}|\mathbf{F}_i^l(n,k_1,k_2)|^p} \tag{4.1}$$

where i represents the i-th filter in l-th layer, N_l is the input channel size and K_l the kernel size. The choice of p is arbitrary and represents the order of the norm, in general, the common norms are l_1-norm and l_2-norm. [22] In this work, the methodology involves the l_1-norm which calculates the sum of the absolute values of the filter's weights. It has been chosen because it promotes sparsity by selecting filters with fewer non-zero weights, making it a good option for identifying the less important ones.
Below are two pruning methods applied to the previously optimized neural network, confirming the theoretical results.

L1-norm method

In this case, a function was implemented using PyTorch's native module, enabling structured pruning based on the p-norm. The objective is to prune filters in the convolutional layers of the pre-trained network with a pruning rate of 40%. This approach significantly reduces the number of parameters, from 12 million to approximately 7.5 million, while improving the accuracy of error pose.

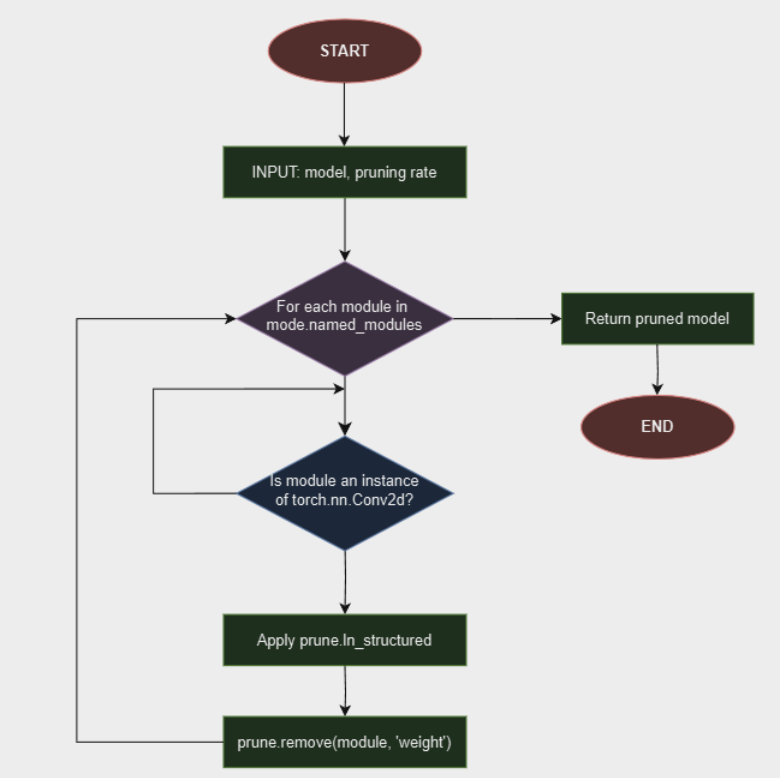


Figure 4.15: Flow-chart of the algorithm

The algorithm is implemented before the training phase, and the pruned model is subsequently retrained for 50 epochs, which is sufficient considering that the neural network was pre-trained for 115 epochs.

Filter Pruning method

In this test case, the Neural Network Compression Framework (NNCF) was assessed. NNCF is designed to offer both post-training and training-time algorithms for neural network optimization, focusing on model compression while minimizing accuracy loss. The framework functions through a set of modules that implement various algorithms, with these modules drawing configuration details from a configuration file that outlines the optimization strategy. In this instance, a weight-dependent filter pruning algorithm was chosen, based on the work "Pruning Filters for Efficient ConvNets (PFEC)" [24]. The configuration applies pruning across the entire network, beginning with a pruning rate of 10%, which escalates to 40% after 10 epochs. The reason for this progressive

pruning strategy is its ability to reduce the model's complexity while maintaining performance. Initially, the network was heavily dependent on the weights and filters it had learned. Starting with an excessively aggressive pruning rate could risk removing too many crucial connections, leading to a significant decline in performance. The final setting to configure is the filter importance criterion. Pruning can be applied either globally or layer-wise. The distinction lies in whether to prune layers independently, selecting filters with the least importance in each layer separately, or to conduct global pruning, choosing the least important filters across the entire network, potentially removing more filters from one layer than another, depending on their relative importance.

4.2.2 Results of model reduction

In this section, the results obtained by applying the pruning methods described in section 4.2 to the neural network optimized via HPO for both datasets will be discussed. The objective is to demonstrate that current structured pruning methods can be an effective tool for reducing the size of large neural networks without compromising their accuracy. The outcome of this process will result in a lighter, faster, and more efficient neural network.
Since three pruning strategies were implemented, the following table presents the mean performances of the pruned network, providing an overview of the effectiveness of this technique.

FA Dataset

The table presents the results obtained by applying the three pruning strategies, noting that the network with optimized hyperparameters was pretrained, subsequently pruned, and then trained for half the epochs used in the initial training. The results confirm theoretical expectations: there is no performance drop despite a 40% reduction in model size.

Pruning method	**Dimension (pxl)**	**IoU (%)**	E_{TN} **[m]**	E_q **[deg]**	E_{pose}
L1-norm	512x384	99.3	0.003	0.027	0.004
Global Filter	512x384	99.5	0.003	0.023	0.003
Layerwise Filter	512x384	99.4	0.003	0.023	0.003

Table 4.12: Comparison of FA mean performances between different pruning methods

The table demonstrates that all three pruning strategies yield promising results, even with the reduction in architectural parameters. Each approach maintains robust performance, highlighting the effectiveness of pruning in preserving model accuracy while streamlining the network structure. This underscores the potential of pruning as a technique for optimizing neural networks without compromising their core capabilities

Method	**Dimension (pxl)**	**IoU (%)**	E_{TN} **[m]**	E_q **[deg]**	E_{pose}
Original	512x384	99.5	0.003	0.029	0.003
Pruned	512x384	99.5	0.003	0.023	0.003

Table 4.13: Comparison of FA mean performances between original and pruned network

To further clarify the improvement level, a comparison between the original network and the globally pruned network with filter pruning is presented. This comparison can help establish a standard procedure for optimizing such neural networks. Specifically, after determining the optimal hyperparameter configuration, structured pruning is applied to reduce model size. It is essential to note that the pruning rate should be carefully selected based on the specific neural network, as smaller or task-specific networks may not benefit from an aggressive pruning approach.

WSE Dataset

The table presents results obtained using the same training strategy as applied in the previous dataset, in combination with one of the three pruning algorithms, specifically, the L1-norm approach, though this choice is arbitrary. The results confirm theoretical expectations: there is no performance drop despite a 40% reduction in model size.

Method	**Dimension (pxl)**	**IoU (%)**	E_{TN} **[m]**	E_q **[deg]**	E_{pose}
Original	512x384	93.5	0.020	5.65	0.11
Pruned	512x384	92.4	0.020	5.93	0.12

Table 4.14: Comparison of WSE mean performances between original and pruned network

4.2.2.1 Comparison with smaller neural network

The objective of this thesis is to demonstrate the substantial benefits of optimizing neural networks to enhance their efficiency while simultaneously reducing their size without compromising performance. This section presents a comparative study of two neural networks: the first is a larger network that undergoes optimization and pruning, while the second is a smaller network trained with the same set of hyperparameters. Architecturally, the networks are identical, differing only in the backbone used. The first network, employed throughout this study, utilizes the EfficientDet D3 model with 12 million parameters, whereas the second network uses the D2 model with 8 million parameters. The hypothesis is that the performance of the D3 network, optimized with hyperparameter tuning and pruned by 40%, resulting in approximately 8 million parameters, will outperform the smaller network with fewer connections. The study is based on the premise that a properly optimized neural network, effectively pruned of non-contributive connections, will retain the performance advantages of a larger network.

The initial test involved training both networks with the same hyperparameter set derived from the D3 network's optimization. Results align with expectations, demonstrating that the optimized and pruned network outperforms the network with fewer parameters on the FA dataset.

Method	**Dimension (pxl)**	**IoU (%)**	E_{TN} **[m]**	E_q **[deg]**	E_{pose}
D3	512x384	99.5	0.003	0.023	0.003
D2	512x384	98.8	0.006	0.029	0.007

Table 4.15: Comparison of FA mean performances between D3 optimized and pruned and smaller D2 network

This raises the question of whether different hyperparameter sets, each obtained through HPO specific to the original network, could influence results. The expectation is to confirm prior outcomes: an optimized and subsequently pruned network should consistently deliver high performance, preserving its ability to generalize and identify patterns. This approach supports adopting large neural networks, maintaining their core characteristics while significantly reducing their size.

So, the first step is to conduct HPO on the D2 network, utilizing the same range of hyperparameters outlined in Table 4.1. The network is then trained with the optimized set, obtaining the following set of optimized parameters

HPO	**Values**
Scheduler	Cosine Decay
Learning rate	0.000705
Learning rate factor	0.482
Batch size	5
Number of epochs	137
Optimizer	AdamW
Weight decay interval amplitude	1
Epochs for learning rate decay	42

Table 4.16: Set of optimized hyperparameters on D2 network - FA Dataset

Subsequently, the mean performances between the two networks, both now configured to have the same number of parameters, are compared in the following table.

Method	**Dimension (pxl)**	**IoU (%)**	E_{TN} **[m]**	E_q **[deg]**	E_{pose}
D3	512x384	99.5	0.003	0.023	0.003
D2 opt	512x384	99.4	0.003	0.017	0.004

Table 4.17: Comparison of FA mean performances between D3 optimized and pruned and smaller D2 optimized network

The results presented in the table indicate that the two networks exhibit nearly identical performance levels, underscoring the critical importance of hyperparameter optimization as a foundational step for their effective deployment. A separate discussion is warranted regarding pruning: while it is advantageous in large neural networks, where it is generally advisable to prune connections that contribute minimally to predictions, no significant performance difference is observed when compared to a smaller, optimized version of the same network. For this thesis, the choice of network architecture is arbitrary and highly contingent on hardware constraints.

Chapter 5

Simulation of mission's inference

This chapter will discuss the results of a simulation designed to approximate real-world conditions by testing a progressive, sequential image dataset. This dataset simulates the images that the camera would capture during the SROC mission. The aim is to assess whether the neural network's performance meets the mission's relative position estimation error requirements. Specifically, the requirement stipulates that, for each range of relative distance from the target, an error margin below 2% of the smallest approach distance in the manoeuvre conducted shall be achieved. For the FA manoeuvre, the table below shows the requirements for each range of distance.

Relative distance range	**Performance**
$[60; 40]$ m	0.8 m
$[40; 20]$ m	0.4 m
$[20; 10]$ m	0.2 m
$[10; 5]$ m	0.1 m

Table 5.1: AOCS requirement for relative position

5.1 Setup of the neural network

Since this point, the focus has been on optimizing the neural network by training with two separate datasets to demonstrate the replicability and effectiveness of the method across various scenarios.
However, the ultimate objective is to deploy this neural network for the SROC mission, where it must accurately recognize and estimate the target's relative position in any orientation during manoeuvres. Consequently, priority is given to achieving a network with a robust generalization capability, that is its ability to make accurate predictions on data it has not encountered during training.

5.1.1 Overfitting and troubleshooting

The model's primary task is to generalize to unseen data; therefore, a neural network with strong generalization capabilities will be less sensitive to noise or irrelevant details in the training dataset, making it better suited to real-world applications where data rarely matches training conditions precisely.

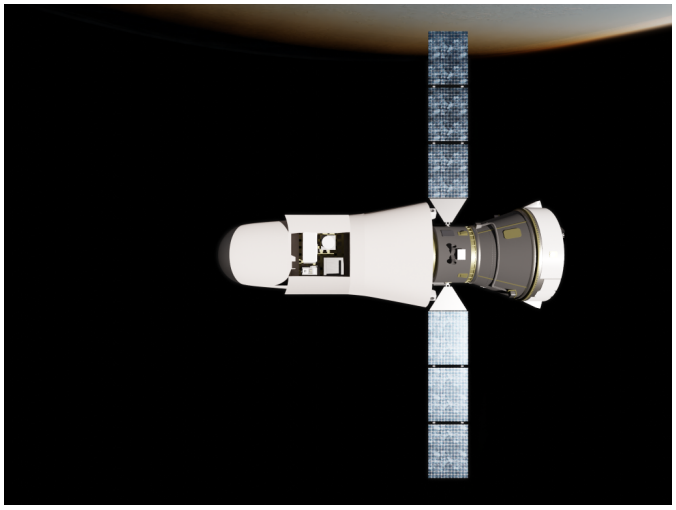
The primary issue to mitigate is overfitting, a phenomenon where the network over-specializes in training data, memorizing details rather than learning generalizable patterns. A well-generalizing network can capture the data's underlying structures rather than each sample's specific features. To prevent and reduce overfitting, the following strategies were implemented:

1. Dataset Division: The data was split into training, validation, and test sets to assess model performance and prevent overfitting objectively.

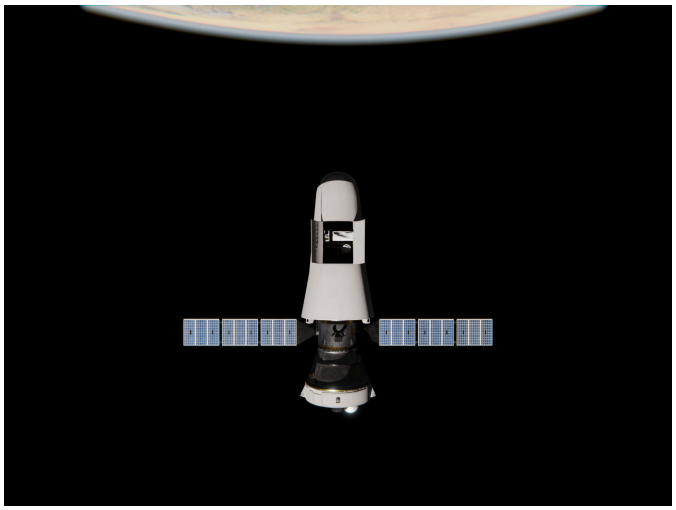
2. Data Augmentation: To increase data diversity and quantity for training, we applied data augmentation techniques, using the Albumentations library based on Lovaglio's approach [3]. The main limitation here is the lack of rotation and flipping of the images, which would significantly enhance the network's generalization capacity by exposing it to varying target orientations during training.

3. Epoch Regulation: The number of training epochs can also negatively affect generalization if set too high. Overextending training epochs can lead the network to memorize specific training images rather than identifying broader patterns.

Testing the optimized neural network trained on only a single dataset with a set of unseen images, where the target position deviates from the training dataset yields unsatisfactory results with low performance, as shown in the images.
Despite hyperparameter optimization, the network fails to generalize effectively to a completely different dataset due to its inability to capture the underlying patterns in the training data. This issue is largely attributed to target orientation; as shown in the images, the target is rotated by 90 degrees compared to the training dataset.



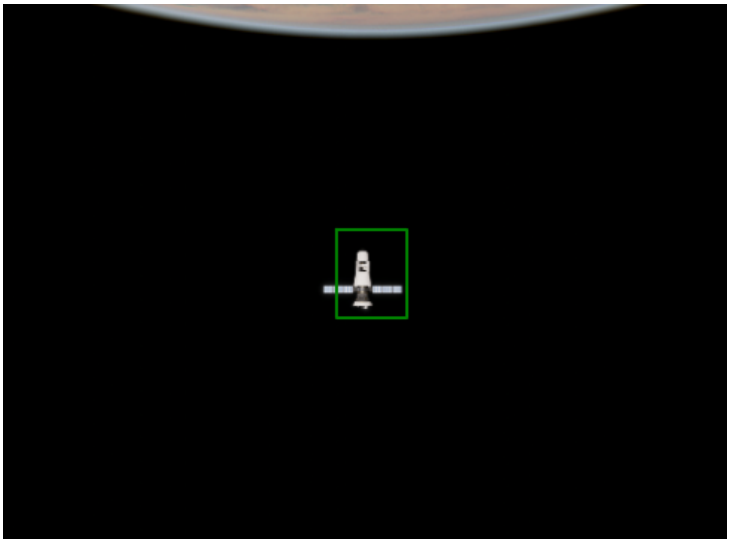
(a) Training Dataset



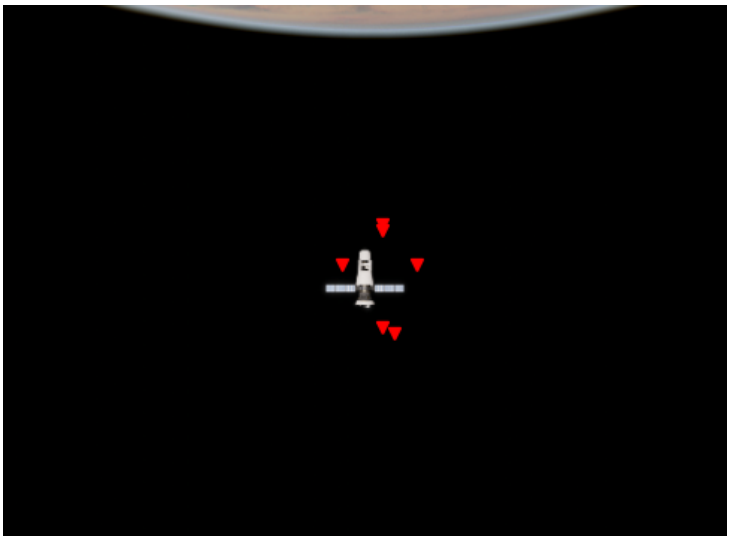
(b) Inference (unseen) Dataset

Figure 5.1: Comparison between Training and Inference datasets

Although seemingly trivial, the lack of angular rotation for data augmentation results in poor pattern recognition and spatial estimation ability when the target's orientation differs.



(a) Bounding box error



(b) Keypoints error

Figure 5.2: Committed prediction error by the neural network

The second test relies on the following strategy. The neural network is trained, with the same hyperparameters of the table 4.5, on an expanded WSE dataset, which includes over 40000 additional SR images captured during the WSE manoeuvre. These images were acquired using identical camera configurations as those utilized for the FA dataset, ensuring consistency in data characteristics. Horizontal and vertical flips were applied to the images as data augmentation techniques to increase variability and improve the model's generalization capabilities.

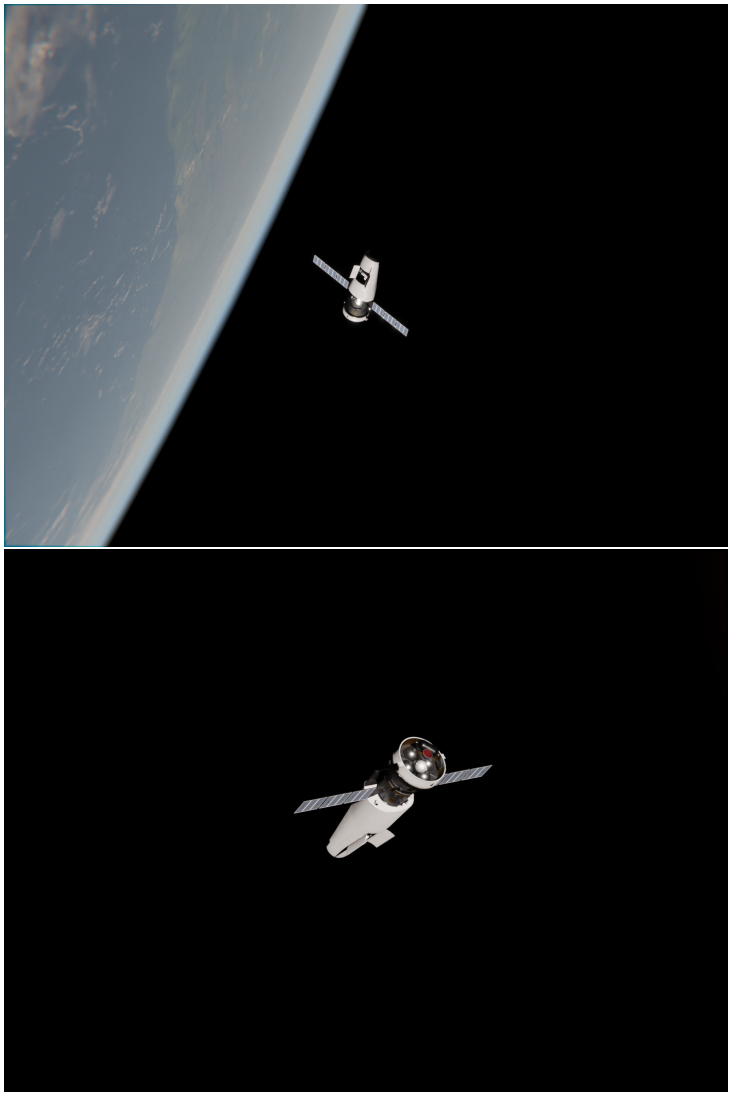


Figure 5.3: New WSE Dataset

The following table presents the neural network's performance evaluated on its inference dataset, highlighting significant improvements in positional and orientation error as well as in the IoU percentage.

Method	**Dimension (pxl)**	**IoU (%)**	E_{TN} **[m]**	E_q **[deg]**	E_{pose}
D3	512x384	95.0	0.014	4.27	0.08

Table 5.2: Performance of the Neural Network on WSE test dataset

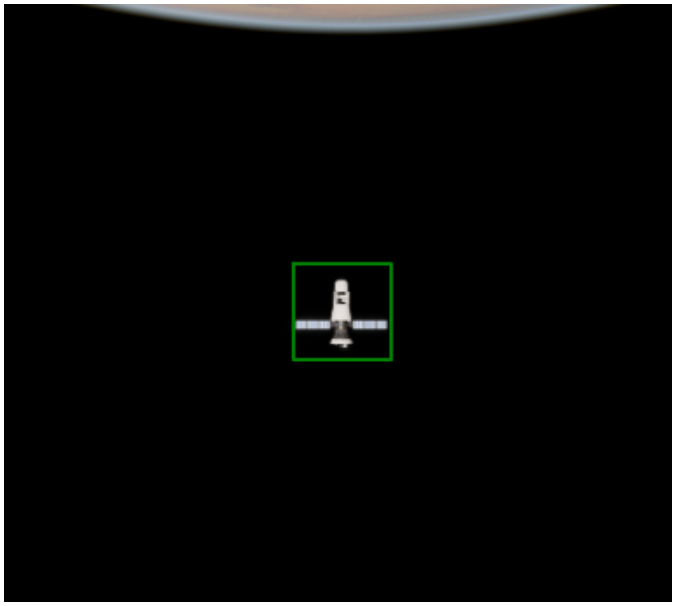
The true benchmark for the network, however, is the progressive dataset generated earlier, which consists of images the neural network has never encountered. The expectation is that the network has enhanced its ability to generalize.

Method	**Dimension (pxl)**	**IoU (%)**	E_{TN} **[m]**	E_q **[deg]**	E_{pose}
D3	512x384	76.0	0.043	15.6	0.30

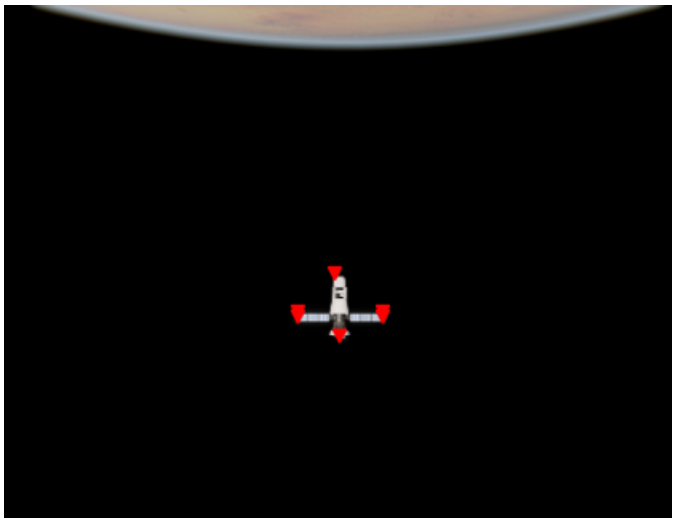
Table 5.3: Performance of the Neural Network on FA progressive dataset

The table indicates improved results in terms of translational and orientation errors, as the training dataset was designed to be more generalized, encompassing all possible configurations.
Nevertheless, in terms of error, these performance levels remain insufficient to satisfy the requirements. Further training on a specialized dataset with diverse angular orientations is necessary, as previously addressed in earlier chapters of this thesis. Training the network with a fixed setup fails to provide adequate generalization, leading the model to memorize a single configuration. This limitation is compounded by the inability to apply data augmentation techniques, such as rotation, to rectangular images.

Addressing this issue is crucial for future developments. This work serves as a starting point for tackling the challenges of overfitting and improving the network's generalization capabilities, with particular attention to the target's orientation.



(a) Bounding box error



(b) Keypoints error

Figure 5.4: Committed prediction error by the new WSE-trained neural network

5.2 Test and Results

Regardless of the previously obtained results, this section demonstrates that a neural network with a high degree of generalization, such as the one trained on the FA dataset in Section 4.1.1, can meet the requirements outlined in Table 5.1.
The experimental setup involves testing 450 images simulating the satellite's approach toward the target. For each evaluated image, the network outputs the same image annotated with the predicted bounding box and assesses whether the distance prediction satisfies the specified requirement. The evaluation script first calculates the prediction error across the three axes compared to the ground truth, applying a tolerance of 0.1 %. Next, it assesses the satellite's position, applies the corresponding requirement, and verifies whether the network's prediction falls below the threshold. If the requirement is not met, the image is annotated with "requirement not satisfied" in red; otherwise, "requirement satisfied" appears in green.

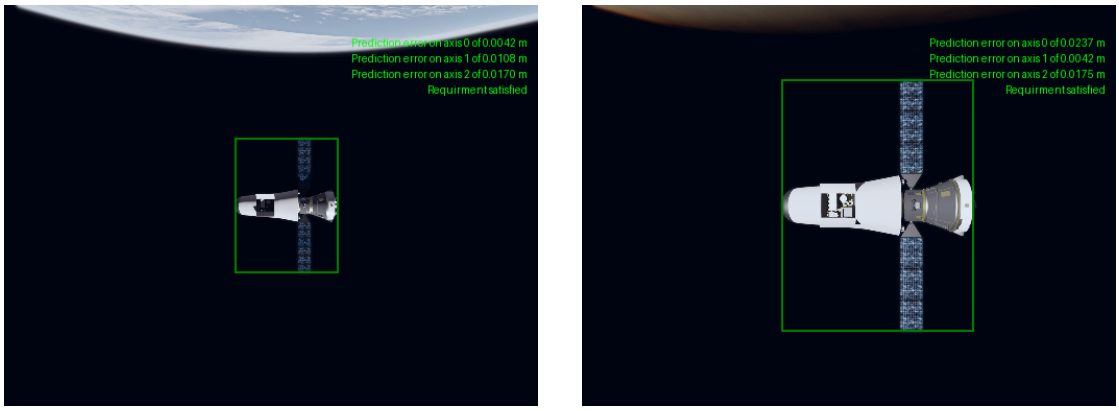


Figure 5.5: Output of the NN based on requirements

Upon testing, it was observed that 449 out of 450 images met the requirement, meaning 99.78% of the network's predictions were within the acceptable threshold. This result supports the assertion that a well-generalized network capable of recognizing target patterns in space can fulfill mission requirements with a confidence level exceeding 90%.

Chapter 6

Conclusions and future work

This study has conducted a comprehensive investigation into optimizing CNNs for optical navigation in space, with a particular emphasis on pose estimation for non-cooperative spacecraft. By leveraging advanced optimization techniques, including hyperparameter tuning and structured pruning, the research has achieved notable advancements in both accuracy and computational efficiency, paving the way for more effective and practical applications in space exploration. A key strength of this work lies in the systematic methodology adopted for CNN optimization. The use of Optuna as a framework for hyperparameter tuning enabled an exhaustive exploration of the hyperparameter space, leading to optimized model performance without relying on heuristic or manual methods. This rigorous approach resulted in a substantial reduction in pose estimation errors, demonstrating the critical role of precise hyperparameter selection in enhancing model generalization to unseen data.
In addition to hyperparameter optimization, structured pruning played a pivotal role in reducing model complexity while preserving high levels of performance. Pruning has produced lighter neural networks by significantly decreasing the number of parameters, making them well-suited for deployment on resource-constrained hardware, such as spacecraft systems.

The testing outcomes have confirmed that these optimization techniques not only preserved the models' accuracy but, in some cases, enhanced it. The optimized networks were capable of delivering robust performance even at lower resolutions, without compromising output quality. This is a critical accomplishment, as it demonstrates the feasibility of maintaining high performance in resource-limited settings.
This project has highlighted the importance of an integrated approach to optimizing neural networks for space applications. The techniques developed and tested here offer substantial potential for enhancing the efficiency and reliability of optical navigation operations in complex environments.

Building upon these achievements, several open works for future research have been identified. A deeper exploration of model reduction techniques is warranted, with a focus on incorporating quantization alongside advanced pruning methods to minimize model size and weight further. This could enhance the deployability of neural networks on even more constrained hardware platforms.
Moreover, accelerating the hyperparameter optimization process by developing faster and more efficient tuning algorithms could significantly reduce computational overhead and time requirements. This would be particularly valuable for large-scale applications.
Another critical aspect of improvement involves strategies to mitigate overfitting. A fundamental approach to preventing the network from focusing on a single pattern is the generation of a robust image dataset characterized by diverse scenarios that deviate from nominal conditions. Such a dataset must include multiple target configurations as well as images where the target is not centered within the camera's frame. This strategy is essential to ensure that the network does not rely excessively on a single pattern but, through appropriate training, develops the ability to generalize across a wide range of scenarios and conditions. This includes suboptimal target positions and orientations, all while maintaining accuracy. In addition, testing these optimized designs on actual spacecraft hardware is an essential step to validate their performance under real-world operational constraints.
In conclusion, this thesis has laid a solid foundation for the optimization of CNNs in space navigation, highlighting the potential of advanced neural network methodologies. The identified future directions hope to refine and extend these contributions, advancing the field of space-based AI systems.

Bibliography

[1] Leo Pauly, Wassim Rharbaoui, Carl Shneider, Arunkumar Rathinam, Vincent Gaudillière, and Djamila Aouada. «A survey on deep learning-based monocular spacecraft pose estimation: Current state, limitations and prospects». In: *Acta Astronautica* (2023) (cit. on p. 1).

[2] Shaharyar Ahmed Khan Tareen and Zahra Saleem. «A comparative analysis of sift, surf, kaze, akaze, orb, and brisk». In: *2018 International conference on computing, mathematics and engineering technologies (iCoMET)*. IEEE. 2018, pp. 1–10 (cit. on p. 2).

[3] Lucrezia Lovaglio. «AI-Based Optical Navigation for Rendezvous and Proximity Operations (RPOs) Missions of Small Satellites». MA thesis. Politecnico di Torino, 2024 (cit. on pp. 2, 3, 5–7, 12, 15, 27, 29, 42, 50).

[4] Tae Ha Park and Simone D'Amico. «Robust multi-task learning and online refinement for spacecraft pose estimation across domain gap». In: *Advances in Space Research* 73.11 (2024), pp. 5726–5740 (cit. on pp. 3, 4, 7, 8).

[5] Bo Chen, Jiewei Cao, Alvaro Parra, and Tat-Jun Chin. «Satellite pose estimation with deep landmark regression and nonlinear pose refinement». In: *Proceedings of the IEEE/CVF international conference on computer vision workshops*. 2019, pp. 0–0 (cit. on p. 3).

[6] Tae Ha Park, Marcus Märtens, Gurvan Lecuyer, Dario Izzo, and Simone D'Amico. «SPEED+: Next-generation dataset for spacecraft pose estimation across domain gap». In: *2022 IEEE Aerospace Conference (AERO)*. IEEE. 2022, pp. 1–15 (cit. on p. 4).

[7] Tae Ha Park, Juergen Bosse, and Simone D'Amico. «Robotic testbed for rendezvous and optical navigation: Multi-source calibration and machine learning use cases». In: *arXiv preprint arXiv:2108.05529* (2021) (cit. on p. 5).

[8] S Corpino, G Ammirante, G Daddi, F Stesina, F Corradino, A Basler, A Francesconi, F Branz, J Van den Eynde, et al. «Space Rider Observer Cube–SROC: a CubeSat mission for proximity operations demonstration». In: *Proc. 73rd International Astronautical Congress (IAC)*. 2022 (cit. on pp. 5, 6).

[9] *Convolutional Neural Network (CNN): A Complete Guide* (cit. on pp. 9, 10).

[10] *GeeksforGeeks Website* (cit. on p. 12).

[11] *Evaluating detection (Intersection Over Union)* (cit. on p. 13).

[12] Mate Kisantal, Sumant Sharma, Tae Ha Park, Dario Izzo, Marcus Märtens, and Simone D'Amico. «Satellite pose estimation challenge: Dataset, competition design, and results». In: *IEEE Transactions on Aerospace and Electronic Systems* 56.5 (2020), pp. 4083–4098 (cit. on p. 13).

[13] *Hyperparameters in Machine Learning* (cit. on p. 14).

[14] Nikolaus Hansen. «The CMA evolution strategy: A tutorial». In: *arXiv preprint arXiv:1604.00772* (2016) (cit. on p. 19).

[15] Shuhei Watanabe. «Tree-structured parzen estimator: Understanding its algorithm components and their roles for better empirical performance». In: *arXiv preprint arXiv:2304.11127* (2023) (cit. on pp. 19, 20).

[16] Liam Li, Kevin Jamieson, Afshin Rostamizadeh, Ekaterina Gonina, Jonathan Ben-Tzur, Moritz Hardt, Benjamin Recht, and Ameet Talwalkar. «A system for massively parallel hyperparameter tuning». In: *Proceedings of Machine Learning and Systems* 2 (2020), pp. 230–246 (cit. on p. 21).

[17] Raghuraman Krishnamoorthi. «Quantizing deep convolutional networks for efficient inference: A whitepaper». In: *arXiv preprint arXiv:1806.08342* (2018) (cit. on pp. 22, 23).

[18] Jonghoon Kwak, Kyungho Kim, Sang-Seol Lee, Sung-Joon Jang, and Jonghee Park. «Quantization aware training with order strategy for CNN». In: *2022 IEEE International Conference on Consumer Electronics-Asia (ICCE-Asia)*. IEEE. 2022, pp. 1–3 (cit. on p. 23).

[19] Yufan Liu, Jiajiong Cao, Bing Li, Weiming Hu, Jingting Ding, and Liang Li. «Cross-architecture knowledge distillation». In: *Proceedings of the Asian conference on computer vision*. 2022, pp. 3396–3411 (cit. on p. 23).

[20] *Understanding the Difference: Structured vs Unstructured Neural Pruning* (cit. on pp. 24, 25).

[21] Hugo Tessier. *Neural Network Pruning 101* (cit. on p. 24).

[22] Yang He and Lingao Xiao. «Structured pruning for deep convolutional neural networks: A survey». In: *IEEE transactions on pattern analysis and machine intelligence* (2023) (cit. on pp. 25, 26, 43).

[23] *Optuna* (cit. on p. 29).

[24] Determine Filters'Importance. «Pruning Filters for Efficient ConvNets». In: (2016) (cit. on p. 44).