POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Multi-class Dataset Generation and Spacecraft Detection with Deep Learning to aid Pose Estimation

Tesi di Laurea Magistrale in
Space Engineering - Ingegneria Aerospaziale

Author: **Eshwar Bhargav Bhupanam**

Student ID: 940164
Advisor: Prof. Mauro Massari
Academic Year: 2022-23

Abstract

The research on vision-based pose estimation algorithms using monocular images for Rendezvous and Proximity Operations has been widely discussed in recent years. Deep Learning (DL) techniques perform well in such research, but they require labelled spacecraft datasets. The existing datasets contain a single spacecraft, and the algorithms are inaccessible to reproduce the dataset with any desired spacecraft. This research is proposed to overcome these shortcomings by introducing an algorithm to generate a multi-spacecraft dataset named POSSE Dataset. Since the state-of-the-art (SOTA) models are trained to detect the very same or similar featured spacecraft, this research also aims to substantiate these DL models' feasibility using a convolution neural network (CNN) in detecting the spacecraft with disparate features eventually assisting in pose estimation. The dataset generation algorithm written in Python makes use of Relative Orbital Elements for the camera path planning and is developed to integrate with open-source software using physical-based rendering techniques. Moreover, the proposed CNN for detection termed as Spacecraft Detection Network (SDN) employs modified SOTA object detection architectures to predict multiple spacecraft. As a result, the POSSE Dataset includes 8237 novel synthetic images of Aqua and Cluster that consist of sharp and smooth features along with their annotations for a bounding box, keypoints and pose data. In addition, the dataset includes challenging conditions such as long-range, low visibility, strong shadows, and partial views of the spacecraft with stars and Earth in the background. Furthermore, the POSSE Dataset is partitioned into another four datasets by removing the stars in the background to validate its robustness in comparison, with the proposed SDN in different scenarios. Overall, the SDN is proven to be efficient in detecting a single spacecraft with a dark background at a mean Average Precision (mAP) of 78.03% and 5 Frames Per Second (FPS). However, this performance is reduced when tested on the POSSE Dataset to 36.57% of mAP and 4 FPS.

Keywords: synthetic image generation; relative orbital elements; deep learning; spacecraft detection; relative pose estimation

Abstract in lingua italiana

La ricerca sugli algoritmi di stima della posa basati sulla visione utilizzando immagini monoculari per le Operazioni di Rendezvous e Prossimità è stata ampiamente discussa negli ultimi anni. Le tecniche di Deep Learning (DL) danno buoni risultati, ma richiedono dataset etichettati dei satelliti. I dataset esistenti contengono un solo satellite, e gli algoritmi sono inaccessibili per riprodurre i dataset con qualsiasi satellite desiderato. Lo scopo di questa ricerca è superare questi svantaggi introducendo un algoritmo per generare un dataset multi-satellite chiamato POSSE Dataset. Poiché i modelli allo stato dell'arte (SOTA) sono addestrati per rilevare lo stesso satellite o simili, questa ricerca mira anche a corroborare la loro fattibilità di DL utilizzando una rete neurale convoluzionale (CNN) nella rilevazione dei satelliti con caratteristiche disparate, aiutando infine nella stima della posa. L'algoritmo di generazione del dataset scritto in Python fa uso degli Elementi Orbitali Relativi per la pianificazione del percorso della telecamera ed è sviluppato per integrarsi con software open-source utilizzando tecniche di rendering basate sulla fisica. Inoltre, la CNN proposta per la rilevazione, denominata Spacecraft Detection Network (SDN), impiega architetture di rilevamento oggetti modificate dallo SOTA per prevedere più satelliti. Di conseguenza, il POSSE Dataset include 8237 nuove immagini sintetiche di Aqua e Cluster che consistono in caratteristiche nitide e lisce insieme alle loro annotazioni per bounding box, keypoints e dati di posa. Inoltre, il dataset include condizioni difficili come lunghe distanze, bassa visibilità, forti ombre e viste parziali dei satelliti, con stelle e la Terra sullo sfondo. Inoltre, il POSSE Dataset è suddiviso in altri quattro dataset rimuovendo le stelle sullo sfondo per convalidare la sua robustezza in confronto con la proposta SDN in scenari diversi. Complessivamente, la SDN è dimostrata efficiente nel rilevare un singolo satellite su sfondo scuro con una mean Average Precision (mAP) del $78,03\%$ e 5 Frame Per Second (FPS). Tuttavia, questa prestazione è ridotta quando testata sul POSSE Dataset al $36,57\%$ di mAP e 4 FPS.

Parole chiave: generazione di immagini sintetiche; elementi orbitali relativi; deep learning; rilevamento di satelliti; stima della posa relativa

Contents

List of Figures

List of Tables

Nomenclature

Non-dimensional Quantities

$\boldsymbol{A}$	State matrix	$[-]$
$\boldsymbol{B}$	Control input matrix	$[-]$
$\boldsymbol{I}$	Identity matrix	$[-]$
$\boldsymbol{T}$	Transformation matrix	$[-]$
$\boldsymbol{b}$	Bias term	$[-]$
$\boldsymbol{\delta\alpha}$	Relative Orbital Elements	$[-]$
δ	Four quadrant inverse tangent	$[-]$
$\delta\vec{\boldsymbol{e}}$	Relative eccentricity vector	$[-]$
γ	Vernal equinox	$[-]$
$\mathbf{R}$	Direction Cosine Matrix	$[-]$
e	Eccentricity	$[-]$

Physical Quantities

$\boldsymbol{I}_{h\times w\times c}$	Image array	[px]
$\boldsymbol{K}_{h\times w\times c}$	Kernel matrix	[px]
$\vec{\boldsymbol{X}}$	Relative state vector	$[-]$
$\delta\vec{\boldsymbol{i}}$	Relative inclination vector	[rad]
$\delta\lambda$	Relative mean argument of longitude	[rad]
Ω	Right Ascension of the Ascending Node (RAAN)	[rad]

ω	Argument of perigee	[rad]
θ	True argument of latitude of plane-change maneuvering point	[rad]
φ	Relative pericenter	[rad]
$\vec{r}$	Spacecraft's distance vector	[m]
a	Semimajor axis	[m]
i	Inclination	[rad]
M	Mean anomaly of the orbit	[rad]
n	Mean Motion	[rad/s]
S	Stride length	[px]
$u = \omega + M$	Mean argument of latitude	[rad]

Acronyms

ADRIOS	Active Debris Removal/ In-Orbit Servicing
AI	Artificial Intelligence
ANN	Artificial Neural Network
BGD	Batch Gradient Descent
CNN	Convolution Neural Network
CoM	Center of Mass
DCM	Directional Cosine Matrix
DL	Deep Learning
DNN	Deep Neural Networks
ECI	Earth Centered Inertial
GEO	Geostationary Equatorial Orbit
IoU	Intersection over Union
KPEC	Kelvins - Pose Estimation Challenge
KRN	Keypoint Regression Network
LEO	Low Earth Orbit

LiDAR	Light Detection and Ranging
LVLH	Local Vertical and Local Horizontal
MEV	Mission Extension Vehicle
MLP	Multi-Layer Perceptron
ML	Machine Learning
NED	North East Down
NMS	Non-Maximum Suppression
ODN	Object Detection Network
POSSE	un**P**erturbed **O**rbital **S**imulator for **S**pac**E**craft
ReLU	Rectified Linear Unit
ROE	Relative Orbital Elements
RoI	Region of Interest
RPN	Region Proposal Network
RPO	Rendezvous and Proximity Operations
RTN	Radial Tangential Normal
SBRF	Spacecraft Body Reference Frame
SDN	Spacecraft Detection Network
SGD	Stochastic Gradient Descent
SLAM	Simultaneous Localization and Mapping
SOTA	State of the Art
SSN	Space Surveillance Network
VBN	Vision Based Navigation

1 | Introduction

The space bodies such as asteroids, space debris and satellites are classified as "Cooperative" or "Uncooperative" based on whether they have a priori information or not (markers/communication). This research focuses on the "Uncooperative" artificial satellites in particular. The presence of 3000 out of 4500 inactive satellites [9] and more than 27000 pieces of space junk pose a grave threat if fragmented, as estimated by the Department of Defense's global Space Surveillance Network (SSN). The Kessler syndrome is most prevalent in Low Earth Orbit (LEO) and Geostationary Equatorial Orbit (GEO).

The most effective way to combat this chain reaction and stabilise the debris population is to remove large debris from Space. The EU-funded `REMOVEDEBRIS` project [10] deployed in June 2018 successfully demonstrated the in-orbit technologies for active debris removal. In 2019, ESA's Ministerial Council named Space19+ initiated the Active Debris Removal/ In-Orbit Servicing (ADRIOS) project [9] with two main goals: the removal of human-made objects in Space as well as developing the competence necessary for in-orbit servicing, which can extend the lifetime of infrastructure in Space.

One of the most critical aspects of such a mission is the ability to estimate the relative pose, which refers to the position and orientation of the targets in six-dimensional space. Considering a real-time Rendezvous and Proximity Operations (RPO) scenario, ground support in navigation is expensive and compromises operational safety. On-board Vision-Based Navigation (VBN) could address these issues whilst introducing the possibility for accurate target classification, detection, and inspection.

To mention a few missions related to the topic - The *VBN experiment* [11], regarding the `REMOVEDEBRIS` project, built a leap step to undertake relative pose measurements with the use of active and passive imaging systems, such as Light Detection and Ranging (LiDAR) and passive cameras. The *Mission Extension Vehicle* (MEV) series, launched successively in 2020 and 2021 by Northrop Grumman Space Systems and Space Logistics LLC, successfully demonstrated the RPO and Docking capabilities using a combination of the visible, long-wave infrared spectrum, and active LiDAR [12].

Following these footsteps, future missions such as NASA's Restore-L, alias OSAM 1 [13], and ESA's ClearSpace-1 [14] aim to fulfil the goals of the ADRIOS project. Furthermore, companies such as *Astroscale*, *ClearSpace SA* and *Infinite Orbits* have also emerged to co-operate in ESA's Clean Space initiative. The missions' requirements also sparked interest in seeding companies that create modular satellites and on-orbit assemblable spacecraft components with cutting-edge robotics.

1.1. Problem statement

As previously stated, this study focuses on uncooperative targets, with known features and an insufficient docking system: it would be significant in pose estimation to use the passive cameras with visual feedback. Few traditional choices employ both multiple imagers in the electromagnetic spectrum and LiDAR for depth mapping delivers precise measurements, at high mass and energy costs. In the Section 3.3, the detailed discussion on the choice of a passive monocular camera is presented.

Grayscale space imagery for pose estimation has gained popularity over recent years due to its ease of use and lower computational cost as if the images were instantaneous compared to world mapping techniques such as Simultaneous Localization and Mapping (SLAM). According to [12], the target is pixelated at relative ranges exceeding 30km and is sensitive to eclipses for target identification. Concerning the reflective materials used in aerospace, there are illumination challenges under strong/weak intensities and shadow overlays, for less than 5km ranges. The disadvantages include low signal-to-noise ratio, background noise, and zero visibility in eclipse. These factors induce complexity in image processing and feature extraction, which are detailed in Section 1.2.

From the survey [1], most of the complexities in image processing mentioned above get resolved with the help of Deep Learning (DL) techniques, specifically Convolution Neural Networks (CNN). The CNN training requires adequate labelled images, known as datasets. These trained models are believed to be robust in spacecraft detection. The lack of publicly available tools for dataset generation and labelling of desired spacecraft limits this approach. Resolving this issue is also an implicit goal of the research. As a result, an un**P**erturbed **O**rbital **S**imulator for **S**pac**E**craft (POSSE) is developed for rendering images.

In summary, the research problem statement is to develop a **dataset generation algorithm** for generating a **multi-spacecraft dataset** that consists of sharp and smooth features. In addition, these synthetic images are labelled with bounding boxes, keypoints and pose data such that the CNN can perform **spacecraft detection**. By partitioning the main dataset the proposed network is set to validate the detection network's ability and robustness of the CNN in different scenarios, and this network eventually assists further in the development of a multi-spacecraft pose estimation algorithm.

Motivation

An interesting opportunity in this research is to solve the problem statement with the support of State-of-the-Art approaches using minimal and simplistic hardware resources, that will result in lower mission cost and complexity. In optical navigation, working with images alone can make it very difficult to develop the RPO mission-oriented strategy in estimating the pose, given the unfavourable conditions and utmost limitations in Space. The research is motivated by advancements in artificial intelligence (defined in Section 2.3) over computer vision[1] in developing adequate autonomous navigation competence in terrestrial applications. With advanced computational techniques, this research hopes to deliver a robust and accurate algorithm for detecting spacecraft with different geometrical shapes, such that the pose estimation can be performed seamlessly on multiple spacecraft, eventually contributing to the advancement of spacecraft's autonomous capability in the optical navigation domain.

1.2. State of the art

The idea is to present the State of the Art (SOTA) specific to pose estimation in aerospace with monocular imaging using deep-learning techniques alone in this section, but it is important to discuss in brief the pose estimation through computer vision techniques for better understanding. The typical estimation procedure involves identifying, locating, and tracking a number of keypoints on an object. For objects, the keypoints could be corners or other significant features. The process of detecting, extracting and evaluating the correspondence score of these features is termed "image processing", and transforming the potential keypoints from the 2D image plane to the object's 3D space results in "pose estimation". It is to be noted that the image processing algorithms are a subset of computer vision.

The progression of research in "image processing" started in the past five decades from the development of detectors for simple features such as lines and curves by Duda and Hart [15]. Another paper presented the use of Hough transformation[2] in detecting the lines

[1]The field of high-level understanding of computers from digital images or videos.

[2]The purpose of the Hough transform is to perform groupings of edge points into object candidates by performing an explicit voting procedure over a set of parameterized image objects.

and curves from the digitized image, and the very same authors have mentioned in their journal about the Sobel-Feldman Operator [16] which aid in edge detection by estimating the gradient intensity function of the image. Canny [17] had also made a significant contribution with his computational approach on edge detection. Upon this success in feature detection and feature extraction: The concept of perceptual organisation[3], introduced by Lowe [18] to perform 3D-object recognition, is further extended with the novel voting-based methods for simple feature matching implemented by Horaud [19].

In the late 1980s, Dhome et al. [20] introduced a seminal approach to three-dimensional attitude determination from images. They proposed an eighth-degree equation for correspondence analysis, wherein features are extracted based on the triplet interpretation principle. This principle involves considering any image lines as the perspective projection of a triplet of linear ridges. Subsequently, the method was evaluated to determine the corresponding attitude based on these projections. A similar approach with a feature-based technique followed by a model-based verification is published [21] in which, the feature-based process relies on the extraction of an attitude-dependent signature, and then this signature is matched by means of cross-correlation against a stored library of signatures with known attitude. To date, the feature-based techniques on satellite pose estimation are well-advanced with the above-mentioned works [15–21] as baselines; and much more reliable detectors, extraction and matching methods were developed, but a focused discussion on this topic is beyond the scope of this research.

Deep-learning techniques fulfil the image processing tasks and aid in satellites' pose estimation. The techniques are categorised into two: direct and indirect deep-learning

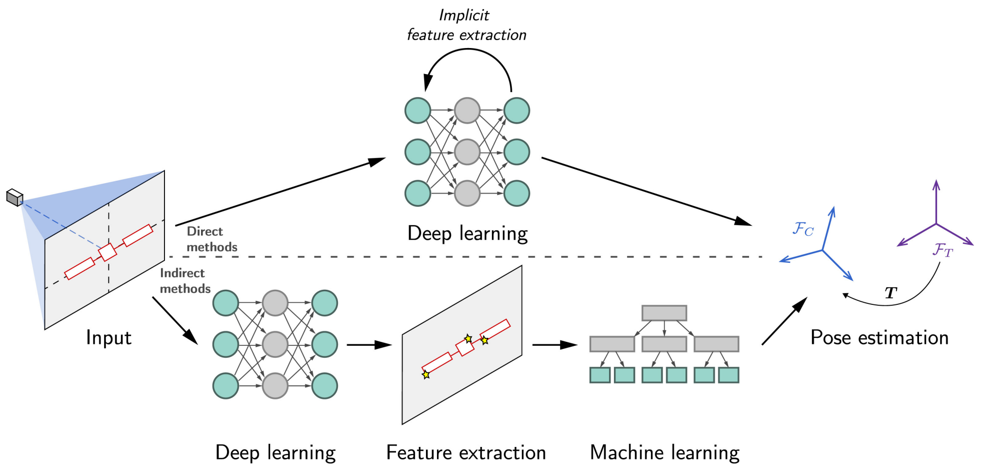


Figure 1.1: Direct and Indirect deep learning Song et al. [1]

[3]Detecting groupings and structures in the image that are likely to be invariant over wide ranges of viewpoints

methods (see Figure 1.1) by Song et al. [1], in their survey, where the former refers to the direct end-to-end Deep Neural Networks (DNN) pipeline (i.e., from the image input to the pose estimate); and the latter uses DNN for image processing functions alone. Further, the pose estimation is achieved by combining this output with other methods, such as classical Machine Learning (ML), geometry-based optimisation, and Kalman filtering.

From here on, the SOTA is detailed on DNN alone for better clarity. It is better to begin with the ESA's Kelvins - Pose Estimation Challenge (KPEC)[4], that shares an exact problem statement of the study; but with Stanford's dataset[5] provided. The challenge had made good progress on the complexities mentioned in Section 1.1. The outcomes from this challenge had a significant impact on this research, thus mindful attention is given to the few studies that are reported below.

Direct deep learning methods

In 2018, Sharma, Beierle, and D'Amico [24] first published the CNN-based approach with an assumption of a known 3D model of the satellite to provide an initial guess of the pose in real-time. Secondly, it contributes an introduction to the image synthesis pipeline that tailors to generate high-fidelity images of any spacecraft 3D model. The proposed end-end CNN adopts AlexNet [6] network architecture consisting of five convolution layers and three fully connected layers (detailed in Section 2.4.4), of which the convolution layers are trained with the ImageNet dataset [25] by adopting transfer learning approach. The last three fully connected layers are trained with synthetically generated images of the Tango satellite of the PRISMA mission such that, the minimal neurons in the last few layers adapt the network for space imagery, and there is no need to train a network completely from scratch for spaceborne applications. The resulting dataset from the image synthesis pipeline is augmented to ten different datasets and tested with the proposed CNN. This had better performance than traditional techniques, but applicability to coarse initialisation alone is the only limitation.

As an outcome of KPEC, one-year later, Sharma and D'Amico introduced Spacecraft Pose Network (SPN) [2] along with the SPEED Dataset [22, 23], and validated their network on SPEED. A consecutive publication [26] in the following year extended the SPN training to Apogee Motor, Imitation-25, and PRISMA-25 datasets. To discuss the SPN network briefly, the relative position and relative attitude are computed in two stages

1. The first stage is set to classify and regress the 2D bounding box and relative attitude by employing three separate branches that are preceded by 5 convolution

[4]KPEC Official website: `https://kelvins.esa.int/satellite-pose-estimation-challenge/`
[5]SPEED Dataset: Kisantal et al. [22], Park et al. [23]

layers as shown in Figure 1.2. The feature maps generated by the convolution layers are processed in Branch 1 using Region Proposal Network (i.e., an object detection algorithm, see Section 2.5.1) for probability-based classification and regression of the 2D bounding box. Whereas, the other two branches are fully connected layers, which use the feature maps along with the bounding box selected from the *ROIPooling* operation as the inputs. Branch 2 identifies the closest probability of the predefined attitude classes for the image input. The corresponding attitude classes are further evaluated in Branch 3 with the popular machine learning loss functions (i.e., softmax and L2 regularization) based on the angular differences from the true class, to find the relative weights that result in estimating the relative attitude.

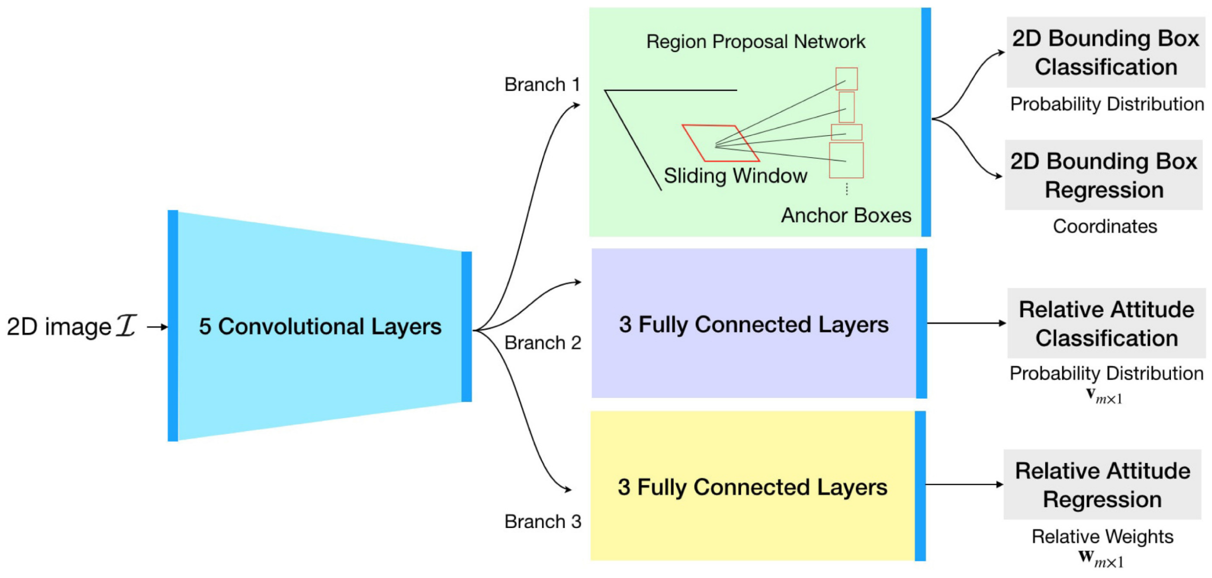
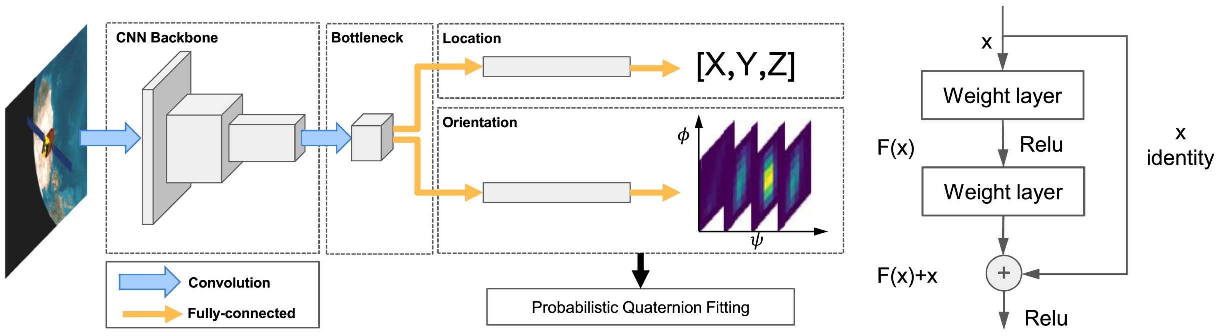


Figure 1.2: Spacecraft Pose Network (SPN) [2]

2. and the second stage obtains the first stage outputs as its inputs, with added geometrical constraints posed from the Perspective-n-Point problem (see Section 2.6). Upon solving this minimization problem, the relative position is achieved. For this, the diagonal characteristic length of the 3D wireframe model and the 2D bounding box are used to coarse estimate the relative position. Later on, this estimate is used as an initial guess along with the azimuth and elevation angles, that are derived from the bounding box coordinates and sensor properties. The minimization problem to estimate the relative position is solved using the Gauss-Newton algorithm.

It is noted that decoupling the relative attitude and position estimation along with the transfer learning will infer the target's pose even with smaller datasets. When tested with the SPEED's test dataset, which contains real images of Tango, the SPN network resulted in degree-level relative attitude and cm-level relative position errors.

Proença and Gao [3] also contested the challenge with the aim of proposing a simplistic approach. Their framework consists of a modified ResNet architecture (detailed in Section 2.4.4) with pre-trained COCO weights. In order to keep spatial feature resolution, the fully connected and the global average pooling layers at the network's end are omitted; and replaced with a bottleneck layer of one extra double-strided 3×3 convolution.



(a) Simplified architecture model (b) ResNet block

Figure 1.3: CNN pipeline of Proença and Gao [3]

Figure 1.3 depicts the framework. The key contribution also includes a simulator built with Unreal Engine 4, which generates its own synthetic image dataset of Soyuz. The idea of soft classification for probabilistic orientation regression, and a simple regression branch with two fully connected layers for position estimation are presented. where, the latter minimizes the relative translation error $(t_{est} - t_{gt})$, corresponding to the total loss function (L_{total}).

$$L_{total} = \beta_1 \sum_{i}^{m} \frac{||t_{est}^{(i)} - t_{gt}^{(i)}||_2}{||t_{gt}^{(i)}||_2} + \beta_2 L_{ori} \quad \text{where: } (\beta_1, \beta_2) \text{ are fine-tuned loss weights.}$$

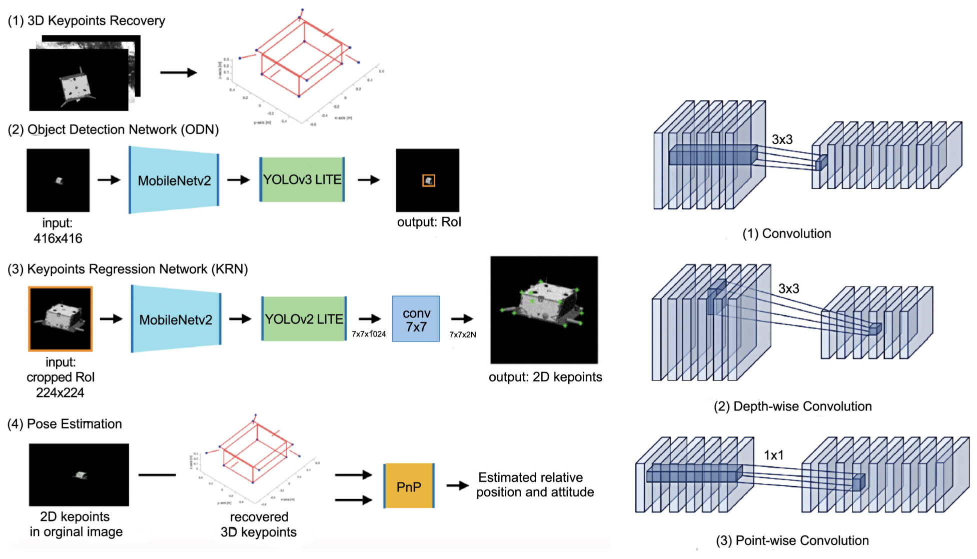
As for the orientation, three different methods were experimented on: direct regression, probabilistic soft classification and multimodal estimation. However, the interest of the study is limited to the problem addressed, rather than the resolution. In short, at first, the paper states the quaternions are non-injective, and direct regression with L_2, L_1 loss results in inaccurate angular distances for orientation representation. Secondly, A continuous estimation approach is experimented, with the idea of encoding each label (q_{gt}) as a Gaussian random variable in (ψ, ϕ) space, so that the network trains to output probability mass functions for Probabilistic Quaternion Fitting (PQF). However, due to the ambiguous views (if present) in the training set, there exists a multimodal distribution (i.e., many solutions for one problem). For this, the Expectation-Maximization (EM) method is proposed.

The CNN pipeline is trained and tested on Soyuz, Dragon and Tango spacecraft, with

4 different architectures, varying image resolution and dataset image augmentation technique; resulting in a decimeter and less than 10° mean errors for location and orientation. The drawback of this network is that there are no RoI checks (i.e., no bounding box). Therefore, multiple objects in one image result in unreliability. Although, the framework should be credited for its simplicity and compatibility with multiple network architectures. As a first ablation study, this has potential baseline capabilities for our work.

Indirect deep learning methods

At the end of 2019, after introducing the SPN network [2], Park, Sharma, and D'Amico [4] refined their pose estimation process with the indirect methods, which utilizes deep-learning for feature extraction alone (i.e., object detection and keypoint regression). Subsequently, the number of trainable parameters is drastically reduced in comparison with the end-to-end DL approach. In addition, a Neural Style Transfer technique for texture randomization is implemented to insist CNN on learning the global shape of the object resulting in robustness to varying illumination and inter-spacecraft distance ($3 - 30m$). An overview of their proposed CNN pipeline in four key steps is shown in Figure 1.4.


(1) 3D Keypoints Recovery
(2) Object Detection Network (ODN)
MobileNetv2
YOLOv3 LITE
input: 416x416
output: RoI
(3) Keypoints Regression Network (KRN)
MobileNetv2
YOLOv2 LITE
7x7x1024
conv 7x7
7x7x2N
input: cropped RoI 224x224
output: 2D kepoints
(4) Pose Estimation
2D kepoints in orginal image
recovered 3D keypoints
PnP
Estimated relative position and attitude
3x3
(1) Convolution
3x3
(2) Depth-wise Convolution
1x1
(3) Point-wise Convolution

(a) Pose estimation network (b) Different convolution operations

Figure 1.4: CNN pipeline of Park et al. [4]

The pipeline begins with the manual assignment of 11 keypoints as in Figure 1.4 (subjected to the 3D wireframe model availability). The Object Detection Network (ODN) is tasked to detect the 2D bounding box. For this, the input image is resized to 416×416, which is processed through the YOLOv3 detection network, with MobileNetv2 as the network's backbone. The conventional convolutions are replaced with depth-wise separable convolutions (i.e., depth-wise, point-wise convolutions) as shown above, that are followed by batch normalization and ReLU activation resulting in reduced computation by a factor of 8 or 9. The corresponding architecture uses 9 pre-defined anchor boxes for training and 3 for inference without non-max suppression, to predict the bounding box alias the RoI.

Furthermore, the cropped image is resized to a typical architecture demand of 224×224, and fed into Keypoint Regression Network (KRN) built on a combination of MobileNet and YOLOv2 architectures similar to ODN. The output of the 7×7×1024 tensor is further convoluted with the 7×7 kernel to predict the visible keypoints in the image and their corresponding coordinates. The Mean Squared Error (MSE) loss function is employed for the KRN. Based on the body frame referenced coordinates on the wireframe model and the regressed keypoints by using the PnP algorithm, the 2D coordinates are re-projected to estimate the 6D pose.

From the observation, it is rational that the accurately cropped RoI for KRN preserves the features of interest and moreover allows for spacecraft classification before the pose estimation process, if required. On the other hand, the decoupled estimation process will provide an opportunity to detect outliers and reduce the computational load in general, at the expense of increased risk due to inaccurate estimates of keypoints.

To outline the SOTA, development has primarily focused on estimating the single spacecraft at extreme possible conditions. In view of a refuelling or servicing mission to restore multiple spacecraft by the chaser, it is certain that these missions necessitate instantaneous **multi-spacecraft** estimation capabilities to precisely classify the target in the image frame rather than the single-spacecraft-specific algorithms tested on various spacecraft with similar features. Given the challenges to be addressed, this topic has intrigued the author's interest in developing an algorithm by training a model with datasets consisting of at least two satellites **with different geometrical features**, such as sharp and smooth features.

1.3. Thesis outline

The document is divided into six chapters. So far, Chapter 1 has already been discussed, which refers to the introduction and the state-of-the-art of the research topic.

- The supportive coordinate systems in developing POSSE for novel synthetic dataset generation, and an in-depth description of the DL operations and its general architecture are discussed in Chapter 2. Furthermore, it includes a discussion on SOTA networks used for localization and object detection tasks with their performance indicators. Mathematical background, along with a brief discussion on the Perspective transformations completes this chapter.

- Chapter 3 discusses mainly the POSSE simulator and sensor models implemented. The idea behind the target spacecraft selection, and the propagator model assumptions for the spacecraft, camera and the sun are also discussed here. The outlook over the generated dataset along with its filtering, annotations and re-partition decision making completes the discussion.
- The proposed CNN architecture for Spacecraft Detection is comprehensively discussed in Chapter 4. Since the network is a modified SOTA architecture, these modifications are clearly explained.
- Chapter 5 present the rendered images from the POSSE simulator and the use of these images for the experiments. In addition, the overall Spacecraft detection performance and robustness with the proposed CNN in Chapter 4 is analysed in this chapter.
- In total Chapter 6 concludes the research outcomes with final comments and discusses the potential ideas to overcome the drawbacks for improving the accuracy and efficiency of the proposed network.

2 | Mathematical background

2.1. Coordinate Systems

The unperturbed propagation of the spacecraft orbit and attitudes require the following reference frames.

2.1.1. Geocentric-Equatorial Coordinate System

This frame is also known as the Earth-Centered Inertial frame (ECI). It is defined in three dimensions using the Cartesian coordinate system, with Earth at the centre of an imaginary celestial sphere, as shown in Figure 2.1.

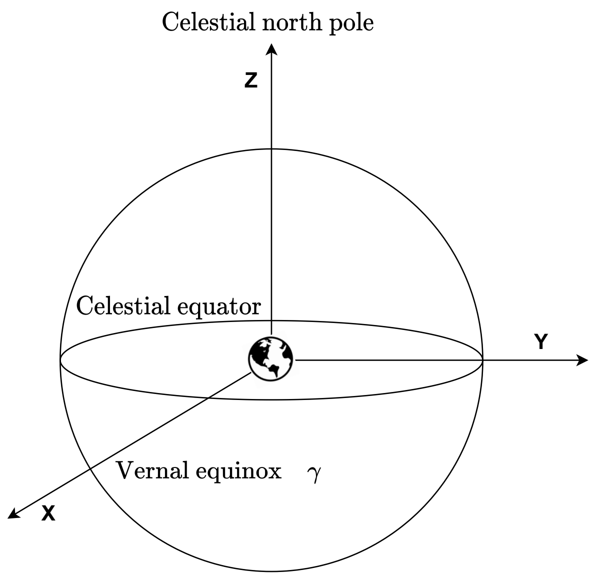


Figure 2.1: ECI Coordinate System

The celestial equator is the greater circle on the xy plane that is aligned with the Earth's equator. The vernal equinox is defined as the position of the rising sun in the sky on the first day of spring. The x-axis points to the vernal equinox (γ point), the z-axis is perpendicular to the xy-plane pointing to the Celestial North Pole (CNP), and the y-axis is orthogonal to the xz-plane following the right-hand thumb rule.

2.1.2. North East Down Coordinate System (NED)

The NED coordinate system is a popular navigational coordinate system. It is in the non-inertial frame, fixed to the spacecraft's centre. The NED axes can be compared to the XYZ axes respectively as in the Figure 2.2.

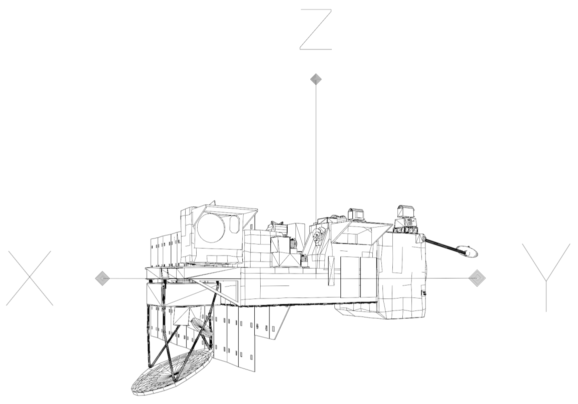


Figure 2.2: Spacecraft Body Reference Frame

These are defined in reference to the earth's surface, as shown below.

- $\hat{\mathbf{N}}$ points north, parallel to the earth's surface in the polar direction.
- Following the right-hand rule, $\hat{\mathbf{E}}$ points east, parallel to the earth's surface, and orthogonal to $\hat{\mathbf{N}}$.
- $\hat{\mathbf{D}}$ points downward, in the opposite direction to the normal of the earth's surface.

The NED coordinate system is very easy to define with the knowledge of the distance vector $\vec{r}_{ECI}$ of the spacecraft in the ECI frame as described in Jonsson [27]. The equations for defining the unit vectors $\hat{\mathbf{N}}$, $\hat{\mathbf{E}}$ and $\hat{\mathbf{D}}$ are rewritten below,

$$\hat{\mathbf{D}} = -\frac{\vec{r}_{ECI}}{|\vec{r}_{ECI}|} \tag{2.1}$$

$$\hat{\mathbf{E}} = \begin{bmatrix} -sin(\delta) & cos(\delta) & 0 \end{bmatrix} \tag{2.2}$$

where: $\delta = atan2(\vec{r_y}, \vec{r_x})_{ECI}$ from the four-quadrant-inverse-tangent of first two components in $\vec{r}_{ECI}$.

$$\hat{\mathbf{N}} = \hat{\mathbf{E}} \times \hat{\mathbf{D}} \tag{2.3}$$

The Directional Cosine Matrix (DCM) rotating the vector from ECI to the NED coordinate system would be,

$$\mathbf{R}_{NED}^{ECI} = \left[\hat{\mathbf{N}} \mid \hat{\mathbf{E}} \mid \hat{\mathbf{D}}\right]^T \tag{2.4}$$

The spacecraft's orientation is computed by transforming the vectors with DCM from NED to body axes.

Spacecraft Body Reference Frame (SBRF)

The right-hand coordinate system is used to refer to the Spacecraft Body Reference Frame. The centre of the frame is on the spacecraft's Center of Mass (CoM). The XYZ-axis is orthogonal, and each axis passes through the different faces of the spacecraft as shown in Figure 2.2. The spacecraft orientation is computed relative to the fixed body axes in the reference frame of interest.

2.1.3. Local Vertical and Local Horizontal (LVLH) Coordinate System

In the scenario of Rendezvous and Proximity Operations (RPO), the use of an LVLH frame is critical. To specifically discuss the inter-satellite distances and relative states of the target and chaser spacecraft, the frame uses a target-centric right-handed coordinate system as in the Figure 2.3.

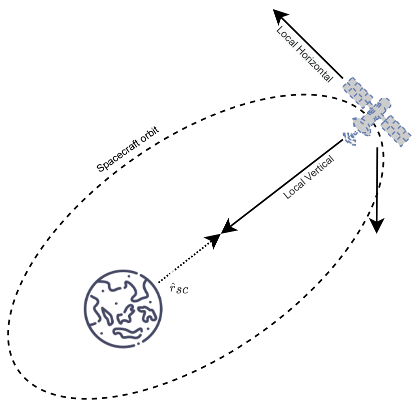


Figure 2.3: Local-Vertical-Local-Horizontal Frame

The Local-Vertical vector, also known as the R-bar, points towards the Earth (opposite to the spacecraft's radial direction). The Local-Horizontal points, in the spacecraft velocity vector direction in the orbital plane, are referred to as the V-bar. The other vector is perpendicular to the spacecraft's orbital plane and orthogonal to both the V-bar and R-bar directions which is referred to as H-bar.

2.2. Relative Orbital Elements

The Orbital Elements or Keplerian elements define the spacecraft's orbit in the inertial frame. Likewise, the relativity between the two spacecraft orbits, which require a set of non-singular Keplerian elements, are defined in D'Amico [5] and are reported below in Equation (2.5).

$$\boldsymbol{\alpha} = \begin{bmatrix} a & u & e_x & e_y & i & \Omega \end{bmatrix}^T \tag{2.5}$$

where a is the semimajor axis, and $u = \omega + M$ denotes the mean argument of latitude, of which ω is the argument of perigee, and M defines the mean anomaly of the orbit. The e_x and e_y are the x and y components of the eccentricity vector which implies: $e_x = e\ cos\ \omega$ and $e_y = e\ sin\ \omega$. Finally, the last two terms i and Ω represent the inclination and right ascension or longitude of the ascending node (RAAN).

The relative states between the target-chaser spacecraft are defined with the quantities of Equation (2.5); and are known as Relative Orbital Elements (ROE) denoted with symbol $\boldsymbol{\delta\alpha}$. The ROEs are dimensionless and are formulated [5] in Equation (2.6).

$$\boldsymbol{\delta\alpha} = \begin{bmatrix} \delta a \\ \delta\lambda \\ \delta e_x \\ \delta e_y \\ \delta i_x \\ \delta i_y \end{bmatrix} = \begin{bmatrix} (a_c - a_t)/a_t \\ (\omega_c - \omega_t) + (M_c - M_t) + (\Omega_c - \Omega_t)\ cos\ i_t \\ e_c\ cos\ \omega_c - e_t\ cos\ \omega_t \\ e_c\ sin\ \omega_c - e_t\ sin\ \omega_t \\ i_c - i_t \\ (\Omega_c - \Omega_t)\ sin\ i_t \end{bmatrix} \tag{2.6}$$

where δa represents the relative semimajor axis in which the difference is normalized with the target quantity (a_t) to be dimensionless. The relative mean argument of longitude is termed as $\delta\lambda$. Later on, two-phase angles $(\varphi,\ \theta)$ are defined as

1. the angle φ denotes the relative pericenter (i.e., $\angle(\overrightarrow{\boldsymbol{\Omega_{rel}}}, \overrightarrow{\boldsymbol{\delta e}_{rel}})$).
2. the true argument of latitude of the target-chaser plane-change manoeuvring point $(\boldsymbol{\Omega_{rel}})$ from the target's ascending node (see Figure 2.4), is denoted as θ.

such that the relative eccentricity $(\delta\vec{\boldsymbol{e}})$ and relative inclination $(\delta\vec{\boldsymbol{i}})$ vectors are expressed in polar coordinates as in Equation (2.7). It is important to note that the formulation for $\delta\vec{\boldsymbol{e}}$, $\delta\vec{\boldsymbol{i}}$ vector in Equation (2.6) are valid under the assumption of relative difference in inclination and RAAN quantities are small (i.e., $\|i_c - i_t\|$ and $\|\Omega_c - \Omega_t\| \ll 1$).

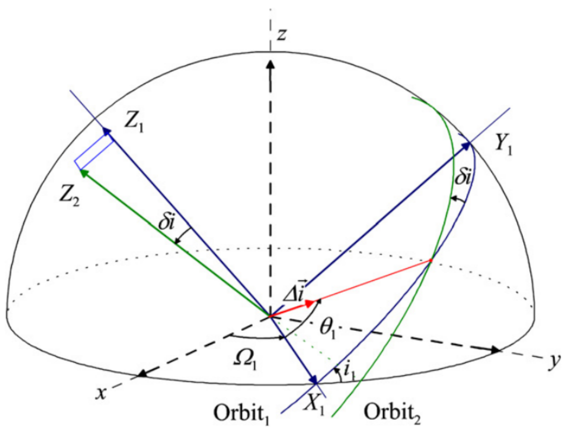


Figure 2.4: Relative E/I vectors [5].

$$\delta\vec{\boldsymbol{e}} = \begin{pmatrix}\delta e_x \\ \delta e_y\end{pmatrix} = \delta e \cdot \begin{pmatrix} cos\ \varphi \\ sin\ \varphi\end{pmatrix}$$
$$\delta\vec{\boldsymbol{i}} = \begin{pmatrix}\delta i_x \\ \delta i_y\end{pmatrix} = \delta i \cdot \begin{pmatrix} cos\ \theta \\ sin\ \theta\end{pmatrix} \tag{2.7}$$

2.2.1. Keplerian motion propagation

The linear-system dynamics for the evolution of Relative Orbital Elements is written in Equation (2.8), in which the control acceleration term $\boldsymbol{B\tilde{u}}(t)$ is also included for the sake of completeness. In reality, the study implements unperturbed propagation of Keplerian motion i.e., the control accelerations are zero in the system. The trajectory design of the camera follows such a system that will be discussed in Section 3.2.4.

$$\begin{gathered}\boldsymbol{\delta\dot{\alpha}}(t) = \boldsymbol{A\delta\alpha}(t) + \cancelto{0}{\boldsymbol{B\tilde{u}}(t)} \\ \text{Initial condition: } \boldsymbol{\delta\alpha}(t_0) = \boldsymbol{\delta\alpha}_0\end{gathered} \tag{2.8}$$

where $\boldsymbol{A}$ represents the state matrix that determines the change of ROEs with respect to Keplerian motion and Orbital perturbations; and $\boldsymbol{B}$ is the control input matrix referring to Radial-Tangential-Normal (RTN) frame as defined in Equation (2.9) [28].

$$\boldsymbol{A} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ -\frac{3}{2}n & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} + \cancelto{0}{\boldsymbol{A_{perturb}}} \qquad \boldsymbol{B} = \frac{1}{na}\begin{bmatrix} 0 & 2 & 0 \\ -2 & 0 & 0 \\ sin\ u & 2cos\ u & 0 \\ -cos\ u & 2sin\ u & 0 \\ 0 & 0 & cos\ u \\ 0 & 0 & sin\ u \end{bmatrix} \tag{2.9}$$

RTN frame is in close reference to the LVLH frame (see Section 2.1.3), with the negative direction of the R-bar and H-bar representing Radial and Normal components, and the V-bar is parallel to the Tangential component in RTN frame.

The unified ROE-HCW Framework for the Keplerian motion in the close-range domain is detailed in Gaias and Lovera [28], which explains the time-variant Lyapunov transformation for the change of variables by introducing the transformation matrix $\boldsymbol{T}(t)$ as in Equation (2.10).

$$\boldsymbol{T}(t) = \begin{bmatrix} 1 & 0 & -cos(nt) & -sin(nt) & 0 & 0 \\ 0 & 1 & 2sin(nt) & -2cos(nt) & 0 & 0 \\ 0 & 0 & 0 & 0 & sin(nt) & -cos(nt) \\ 0 & 0 & nsin(nt) & -ncos(nt) & 0 & 0 \\ -\frac{3}{2}n & 0 & 2ncos(nt) & 2nsin(nt) & 0 & 0 \\ 0 & 0 & 0 & 0 & ncos(nt) & nsin(nt) \end{bmatrix} \tag{2.10}$$

where n represents the target's orbit mean motion, and time travelled by the spacecraft from the initial point of the integration is denoted by t. The transformation matrix is non-singular and continuously differentiable; the rows of T(t) transform the ROEs in the RTN components for determining the relative position ($\vec{\boldsymbol{r}}_{\boldsymbol{rel}}$) and relative velocity ($\vec{\boldsymbol{v}}_{\boldsymbol{rel}}$) of the chaser. It is denoted in vector form as $\vec{\boldsymbol{X}}(t)$, such that

$$\vec{\boldsymbol{X}}(t) = \boldsymbol{T}(t) \cdot a_t \cdot \boldsymbol{\delta\alpha}(t) \quad \text{where: } \vec{\boldsymbol{X}} = \begin{bmatrix} \vec{\boldsymbol{r}}_{\boldsymbol{rel}} \\ \vec{\boldsymbol{v}}_{\boldsymbol{rel}} \end{bmatrix}_{6\times1}. \tag{2.11}$$

Finally, the term $\boldsymbol{\delta\alpha}(t)$ can be defined from the definition of differential calculus as:

$$\boldsymbol{\delta\alpha}_{t+1} = \boldsymbol{\delta\dot{\alpha}}_{t+1} \cdot \Delta t + \boldsymbol{\delta\alpha}_{t-1}$$

substituting with $\boldsymbol{\delta\dot{\alpha}}(t)$ from Equation (2.8) in the above, results in (2.12)

$$\boldsymbol{\delta\alpha}(t) = \boldsymbol{A\delta\alpha}(t) \cdot \Delta t + \boldsymbol{\delta\alpha}(t_0) \tag{2.12}$$

where Δt is the integration time step, such that $t = t_0 + \Delta t$. Integration for the initial condition at $t = t_0$ results as

$$\begin{aligned} \boldsymbol{\delta\alpha}(t) = \boldsymbol{\delta\alpha}(t_0) = \boldsymbol{\delta\alpha}_0 : &\quad \boldsymbol{\delta\alpha}(t_0) = (\boldsymbol{A}\Delta t + \boldsymbol{I}) \cdot \boldsymbol{\delta\alpha}_0 \\ \text{At } t = t_0 + \Delta t : &\quad \boldsymbol{\delta\alpha}(t) = (\boldsymbol{A}\Delta t + \boldsymbol{I}) \cdot \boldsymbol{\delta\alpha}(t_0) \end{aligned}$$

such that, the Equation (2.11) can be linearized and rewritten as in Equation (2.13)

$$\vec{\boldsymbol{X}}(t) = \boldsymbol{T}(t) \cdot a_t \cdot \underbrace{(\boldsymbol{A}\Delta t + \boldsymbol{I}) \cdot \boldsymbol{\delta\alpha}(t_0)}_{\boldsymbol{\delta\alpha}(t)} \tag{2.13}$$

2.3. Artificial Intelligence

Artificial Intelligence (AI) is defined as the ability of machines and computing systems to demonstrate intelligence, thinking and decision-making like humans by recursive training of the models. Machine Learning is the technique of training machines in a programmatic approach through algorithms. For this, the resources are either the existing available data stored in the dataset or from the newly fed data over time in a closed loop. Later, the model is set to understand the evolution of the trends during the learning process, called the ‘training’ phase. The trained model with the intelligence developed is set to predict the output for any model’s unknown input; this process is known as ‘inference’.

The supervised, semi-supervised, unsupervised and reinforcement learning techniques are the various machine learning techniques that are classified based on the input data availability and the algorithm’s functioning. One such simple network well-known as Multi-Layer Perceptron (MLP), consisting of input/output layers and nodes shown as in Figure 2.5, that is artificially built-on to replicate the neural functioning of the human brain is known as Artificial Neural Networks (ANNs).

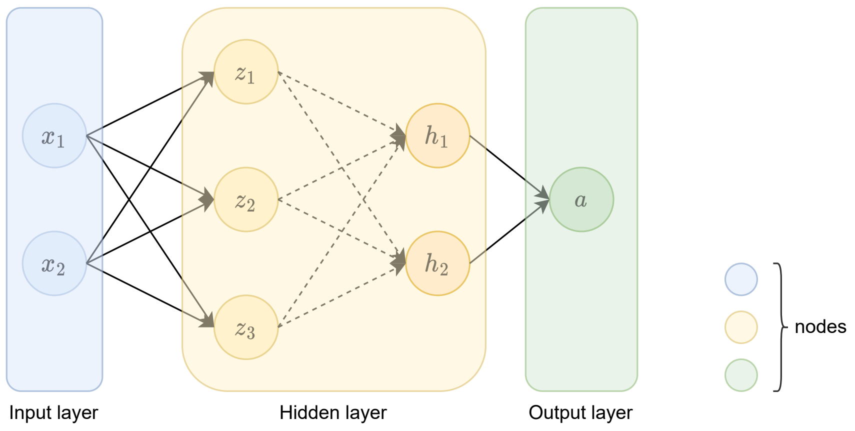


Figure 2.5: A Computational graph representing ANNs

ANNs introduce the conceptual development for Deep Learning, in which the complex neural networks are packed into multiple hidden layers consisting of nodes resembling the neurons (i.e., $z_{1-3}, h_{1,2}$ in Figure 2.5) of the human brain. The more the number of hidden layers or neural networks, the greater the depth. In simple words, Deep Learning is a super-set of neural networks and machine learning algorithms, being structured in layers. The following Sections 2.4 to 2.6 will discuss a particular DL technique and a few significant network architectures related to the study. Furthermore, the networks and algorithms used for object detection, and perspective transformations are followed.

2.4. Deep learning: Convolution neural networks

The CNNs are one among the various DL techniques and share a similar information flow process like Feed Forward Neural Networks, for example, MLP is a feed-forward network. In the later part each neuron is connected to all the neurons in the next layers only in forward direction, whereas in the former, the connections are filtered out by the convolution and pooling operations. This set of filtering operations benefits the CNN model to avoid an over-fitting problem and increase the robustness to identify the spatial relationship of the data, which is especially suited for image processing tasks in computer vision applications.

2.4.1. Convolution

A convolution is a mathematical operation that combines two functions to produce a third function that describes how the shape of one is modified by the other. In the context of image processing and computer vision for a 2D grayscale image of $\boldsymbol{I}_{h\times w}$, convolution is a process of applying a small matrix $\boldsymbol{K}_{h\times w}$ called a kernel (or filter) to an image, by sliding it over the entire image from left to right and top to bottom, and meanwhile computing the dot product between the image's pixel values and the kernel's values at each position. These results are summed along both dimensions (i.e., height and width), to build the pixel values for the output map $\mathbf{O}_{h\times w}$, such that the output pixel values ($\boldsymbol{O}_{ij}$) are,

$$\boldsymbol{O}_{ij} = \sum_{i=0}^{I_h-K_h} \sum_{j=0}^{I_w-K_w} \boldsymbol{I}_{(i:i+K_h)\times(j:j+K_w)} \cdot \boldsymbol{K}_{h\times w} \tag{2.14}$$

It is called convolution because it can be thought of as 'sliding' the kernel over the image, and the output can be thought of as the 'overlap' of the kernel and the image. In general, the above mathematical operation in Equation (2.14) is termed as cross-correlation in pattern recognition and machine learning. Whilst convolution and cross-correlation share a common operational idea, the subtle difference between the two is the way the kernel is used. In convolution, the kernel is flipped vertically and horizontally. The reason for flipping the kernel is to ensure that the convolution operation is commutative and the sliding direction of the kernel is from left to right over the image.

The kernel depth dimension 'c_k' would be the same as the depth of the convolution output 'c_0' for an n-dimensional image with a 'c_i' number of channels. The output is calculated on a kernel-by-kernel basis, where each kernel creates a 'c_i' number of shallow copies and performs convolution on the input. This yields a 3D output with 'c_i' channels, which is then summed over the last dimension resulting in a 2D output of that particular kernel.

In deep learning with CNNs, the convolution layers undertake the convolution operation with an added scalar bias term $\boldsymbol{b}$ to each element of the output map, such that $\boldsymbol{O}_{ij}$ is:

$$\boldsymbol{O}_{ij} = \boldsymbol{b} + \sum_{i=0}^{I_h-K_h} \sum_{j=0}^{I_w-K_w} \boldsymbol{I}_{(i:i+K_h)\times(j:j+K_w)} \cdot \boldsymbol{K}_{h\times w}$$

The bias term allows the model to shift the output of each filter and supports building a better model with complex relationships between the input and output. In practice, the bias term is learned along with the weights (a.k.a. kernel matrix) using a back-propagation algorithm[1] during model training.

Feature map and receptive field

In CNNs, a feature map is a set of two-dimensional output maps $\boldsymbol{O}_{h\times w\times c}$ that represent the activation of a particular set of filters $\boldsymbol{K}_{h\times w\times c}$ applied to an input image or feature map itself. Each element $\boldsymbol{O}_{i\times j\times c}$ in the feature map represents the response of a particular filter $\boldsymbol{K}_{h\times w\times c}$ at a particular location in the input. The number of feature maps are equal to the number of filters applied to the convolution layer's input. In theory, the output map of any 2D convolution layer can be called a 'feature map'.

Whereas the receptive field is discussed at the neuron/nodal level and it can be thought of as the portion of the input that is 'seen' by the neuron. For example, the receptive field of an element $\boldsymbol{O}_{i\times j\times c}$ in the feature map is the portion of the input (i.e., $\boldsymbol{I}_{(i:i+K_h)\times(j:j+K_w)}$) from which the neuron originated from. If there are any intermediary inputs between the neuron and the input, those elements are also accounted for in the neurons' receptive field. A larger receptive field can be useful for capturing global information and patterns in the input, while a smaller receptive field can be useful for capturing local features.

Padding and stride

Padding and striding are key concepts in convolution layers because they affect the output size and the receptive field of the layer, which can have a significant impact on the performance of a neural network. The padding mode, stride length, size of the input image and the kernel impact the output dimension of the feature map $\mathbf{O}_{h\times w\times c}$, computed as:

$$\boldsymbol{O}_{h,c} = \frac{\boldsymbol{I}_h + 2P_h - \boldsymbol{K}_h}{S_h} + 1; \quad \boldsymbol{O}_{w,c} = \frac{\boldsymbol{I}_w + 2P_w - \boldsymbol{K}_w}{S_w} + 1 \tag{2.15}$$

where: P - padded pixels and, S - stride length.

[1]An optimization technique that uses gradient descent to minimize error in the predictions.

If padding is applied, the input image is padded with zeros around its edges to preserve the spatial dimensions of the output. The amount of padding is controlled by the padding mode, such as ‘valid’, ‘full’, and ‘same’. In ‘valid’ mode padding is not applied to the input, eventually, the kernel will not be applied to the input edges. In contrast, the ‘full’ and ‘same’ modes apply padding either to increase or maintain the same size of the output respectively. The amount of padding applied to the input in their respective modes is:

$$\begin{aligned}
\text{valid: } (P_h, P_w) &= (0, 0) \\
\text{same: } (P_h, P_w) &= (\frac{\boldsymbol{K}_h - 1}{2}, \frac{\boldsymbol{K}_w - 1}{2}) \\
\text{full: } (P_h, P_w) &= (\boldsymbol{K}_h - 1, \boldsymbol{K}_w - 1)
\end{aligned}$$

Stride is defined as a pixel measure of the kernel movement along each dimension, over the input. For instance, the default value of $S = 1$ implies that the kernel moves one pixel at a time. The parameter S is called stride length.

2.4.2. Pooling

Pooling is another simple and significant operation in CNNs. In this operation, when the sliding window of a fixed shape called the ‘pool size’ slides over the input with a certain stride length, the result from each stride is aggregated. The process of aggregating information yields coarser maps and allows the neural model to access information on a global scale. It mainly addresses common problems in image processing such as invariance to scale and translation.

Unlike the use of kernels in convolution operations, here there is no such need. Instead, the computations occur by channel-wise pooling either the maximum or the average value among the elements in the sliding window $\boldsymbol{I}_{(i:i+PO_h)\times(j:j+PO_w)}$ and are termed Maximum pooling and Average pooling respectively. By definition, it is clear that the latter confers some degree of invariance to the output. However, the other parameters such as padding and striding impact the pooling layers’ output dimension similar to the convolution layer as in Equation (2.15), where the kernel parameter ($\boldsymbol{K}_h$, $\boldsymbol{K}_w$) alone is replaced by the height and width of the pool size i.e., (PO_h, PO_w).

The difference between the above two layers (i.e., Convolution and Pooling) is the channel dimension, which remains unchanged in the pooling. Secondly, the sole purpose of the pooling layer is to reduce the input’s spatial resolution (downscaling), which implicitly reduces the computational load and contributes to building deeper networks. Whereas the convolution layer only adjusts the number of filters for better feature recognition.

Fully connected

The term 'fully connected' indicates that every neuron in a layer is connected to every neuron in the preceding layer, forming a dense network of connections. For this reason, they are also known as 'dense layers'. This connectivity allows information to flow freely between neurons, enabling the network to learn complex patterns and make high-level representations of the input data. Hence, the output of each neuron in this layer is determined by a weighted sum of the inputs from the previous layers.

Fully connected layers are commonly used in CNNs, to perform tasks such as classification, regression, or feature extraction. For instance, if it is a case to classify 1000 classes in the dataset, there exist 1000 neurons in the dense layer, in which each neuron represents a class score of a particular class in the dataset.

Activation

Activation is a mathematical function applied to the output map $\mathbf{O}_{h\times w}$, to introduce non-linearity to the network. This is a non-trainable layer and only transforms the numerical value of the output map into a meaningful representation or value. The output map could be the result of any of the operations mentioned in the Sections 2.4.1 to 2.4.2. Some commonly used basic non-linear activation functions are:

1. Sigmoid Function (Logistic Function): $f(\mathbf{O}_{i\times j}) = \frac{1}{1+e^{-\mathbf{O}_{i\times j}}}$, it is mainly used for predicting the probability of an object belonging to a specific class with a range to be between 0 and 1 (i.e., single-class classification).

2. Softmax Function: $f(\mathbf{O}_{i\times j}) = \frac{e^{\mathbf{O}_i}}{\sum_{j=0}^{w} e^{\mathbf{O}_j}}$, this performs multi-class classification by assigning probabilistic value in the range of $[0, 1]$ for each class in the dataset, such that the sum of all values equal to 1.

3. Hyperbolic Tangent: $f(\mathbf{O}_{i\times j}) = \tanh(\mathbf{O}_{i\times j})$, the result is zero bounded and ranges in between [-1, 1]. For very large or small input values, the gradients become close to zero and raise problems related to the vanishing gradients similar to the sigmoid.

4. Rectified Linear Unit (ReLU): $f(\mathbf{O}_{i\times j}) = \max(0, \mathbf{O}_{i\times j})$, the function simply transforms the negative values from the output to zeros and maintains the rest. Unlike other activation functions, this does not have any upper bound. It is typically implemented after the convolutional layer to avoid the vanishing gradient problem (explained in Section 2.4.4) when the CNN depth is increased. In this way, the weights between the layers are adjusted according to their respective errors and do not propagate any further.

There are also other activation functions derived from the basic functions listed above. In general, the combination of fully connected and activation layers behave as decision-making layers to produce final predictions.

2.4.3. Transfer Learning

Transfer learning is a model initialization technique in ML and DL, where the data (i.e., kernel weights) obtained from training a similar but different task is used as a priori. It leverages the pre-trained models that are trained on large-scale datasets, which contain millions of labelled images from various categories. Although these categories do not include spacecraft directly, they do share common low-level features to recognize objects, shapes, and textures effectively for spacecraft localization tasks.

2.4.4. Architecture

The first ever CNN architecture named LeNET5 is introduced in the late 20^{th} century for handwritten digit recognition, in which the architecture consists of several convolutional layers, max-pooling layers, and fully connected layers. Literally, this simple conceptual idea laid the foundation for several advanced architectures for image classification such as AlexNet [6], VGG (Visual Geometry Group) [29] and ResNet (Residual Networks) [7]. Due to the importance of these architectures as the backbone of object detection and localization networks, a brief discussion is intended. VGG and ResNet are highly deeper networks capable to learn more complex features, on the contrary, the numerous trainable parameters result in expensive computation. Considering the similarities between the two, only the top-error performance architecture (i.e., ResNet) is discussed in this section.

As mentioned before in Section 1.2, the first published CNN-approach in the spacecraft domain by Sharma et al. [24] included Alexnet [6] in their network as a backbone. A snapshot of its architecture consisting of five convolutional layers, followed by three fully connected layers is shown in the Figure 2.6

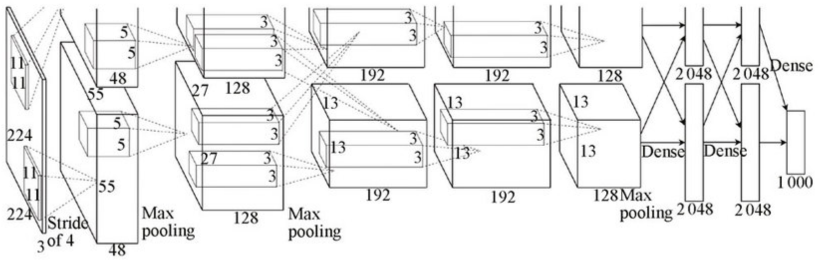
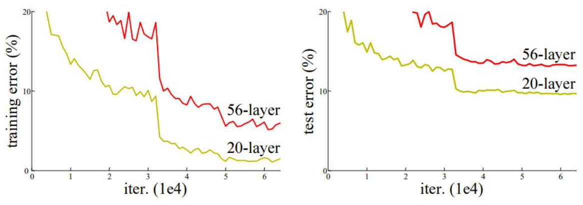


Figure 2.6: AlexNet architecture [6] illustrating delineated operations on two GPUs

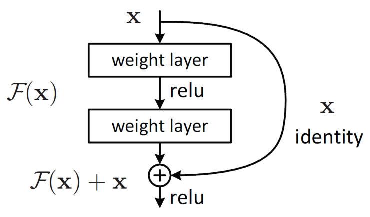
A three-dimensional image of input dimension 224×224×3 fed into the network. The first two convolutional layers perform spatial reduction with 96 and 256 kernel depth, and sliding window sizes of 11 and 5 respectively. Effective padding and a stride length of 4 are applied to the input alone. Later, each convolutional layer is followed by the max-pooling of size 3 and stride length 2. Further, the last three convolution layers are connected only to the previous layer, which resides on the same GPU except for the third layer, which is interconnected to the second. The convolutional output is fed into two dense layers of 4096 neurons each, and finally, a dense layer of 1000 neurons along with the softmax activation is applied for the classification of 1000 classes.

In addition, the introduction of novel techniques such as the use of non-linear Rectified Linear Units (ReLU) activation function, dropout regularization, and data augmentation for reducing overfitting problems are unique in the AlexNet architecture. It incorporates normalization, and pooling layers in the network, and performs multi-GPU training to delineate the responsibilities between the GPUs (see Figure 2.6).

Later in 2015, He, Zhang, Ren, and Sun [7] introduced the ResNet architecture to discuss the issues in the plain deeper networks (i.e., Alexnet, VGG). Primarily, the problem is regarding the vanishing and exploding gradients. While model training, after each iteration in the epoch, the 'n' trainable layers' weights need to be updated by multiplication followed by gradient computation. Depending on the multiplying factor, whether the factor is small or large will result in the very small (vanish) or very large (explode) gradient problem. In Figure 2.7a, the significant difference between the 20-layer and 56-layer 'plain' networks is shown, in which the deeper network has higher training and test error in comparison.



(a) Performance comparison in 'plain' networks.


x
weight layer
relu
weight layer
$\mathcal{F}(\mathbf{x})$
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

(b) ResNet building block.

Figure 2.7: ResNet Architecture [7]

The problem is addressed by the deep residual learning framework, on the hypothesis that it is easier for the solvers to optimize the residual mapping $F(x) := H(x) - x$ instead of the original mapping $F(x) + x$. The key difference between the VGG and ResNet architecture is the layer arrangement, in which the latter contains the building block with a 'shortcut' connection $x_{identity}$ (see Figure 2.7b) in series and the former extends the depth by stacking-up the convolutional layers. To conclude the best among the plain and residual networks, it is proven that the model architecture formed by the deep residual nets performs the same operation as VGG at a very low FLOP rate and better accuracy, without any performance degradation.

In summary, the ResNet architecture, in particular, is clearly the best solution for improved accuracy and reduced train/test error. Furthermore, Proença and Gao [3] also included a modified version of ResNet in their framework and successfully performed feature recognition. Moreover, it is also implemented in the backbone architecture of this work. Therefore, it is certain to discuss ResNet's working architecture in this section.

First, the ResNet architecture is defined with two key building blocks called residual and identity, such that the residual modifies the tensor shape, and the identity preserves the spatial dimensions of the input feature maps without any transformations. These two blocks commonly contain either two or three convolution layers in series depending on the architecture variant (shallower/deeper), followed by batch normalization[2]. The layers in both blocks, other than the first layer, have similar convolution layer properties. Whereas, the kernel parameter 'K' of the first convolution layer is 3 for shallow and 1 for deeper variants. Along with this, the residual block's first convolution layer reduces the tensor shape to half by striding with $S = 2$. The block architecture of the deeper variants along with their default properties are shown in Figure 2.8.

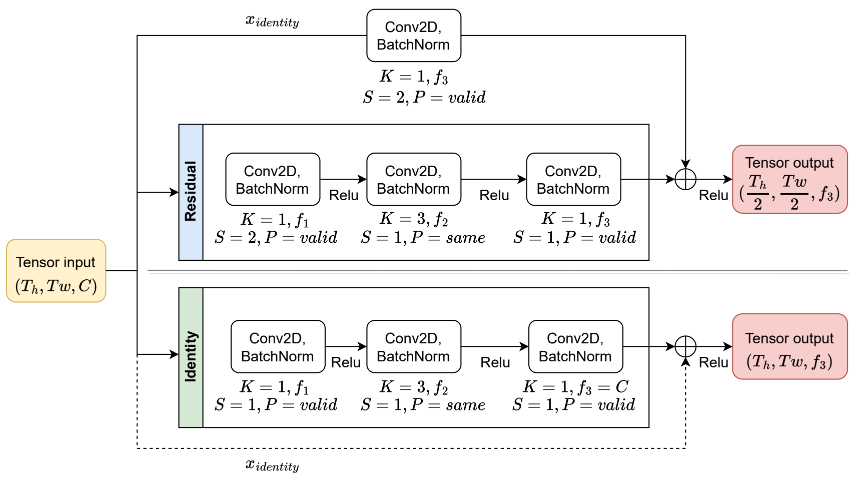


Figure 2.8: Residual and identity block architecture in deeper variants of ResNet

The kernel depth of the `Conv2D` layers in the building blocks for shallow architectures is constant. Whereas, the restoration of the required channel depth is only necessary for the deeper variants (i.e., ResNet-50/101/152) and the final layer in both the building blocks, which fulfils the requirement. The $x_{identity}$ branch in building blocks represents the

[2]Normalization of outputs by subtracting the mean and dividing by the standard deviation.

'shortcut' connection, similar to the one in Figure 2.7b. Since the residual block results in a reduced tensor shape, there is a need for another convolution layer with $S = 2$ and a filter depth similar to the last layer of the building block.

Once the building blocks are defined, it is now possible to introduce the ResNet-xx. The 'xx' refers to the depth of the ResNet architecture, which is determined by the total number of convolutional layers used in the network. In total, five variations in ResNet with a depth of 18, 34, 50, 101 and 152 are proposed by He et al. [7]. In this section, we discuss one such variant called ResNet-18. The stage arrangement of this architecture is shown in Figure 2.9. An input image of 224×224×3 is fed into a five-stage ResNet-18 network. After that, a zero-padding of $P = 3$ is applied to the input, in order to avoid border information loss. Further, the spatial reduction is followed by the convolution and the max-pooling layers with $S = 2$. The layer properties are: $K = 7 \times 7$, $f = 64$, $P = valid$ for convolution, and $PO = 3{\times}3$, $P = same$ for max-pool. All the convolutional layers in this architecture are followed by batch normalization and ReLU activation.

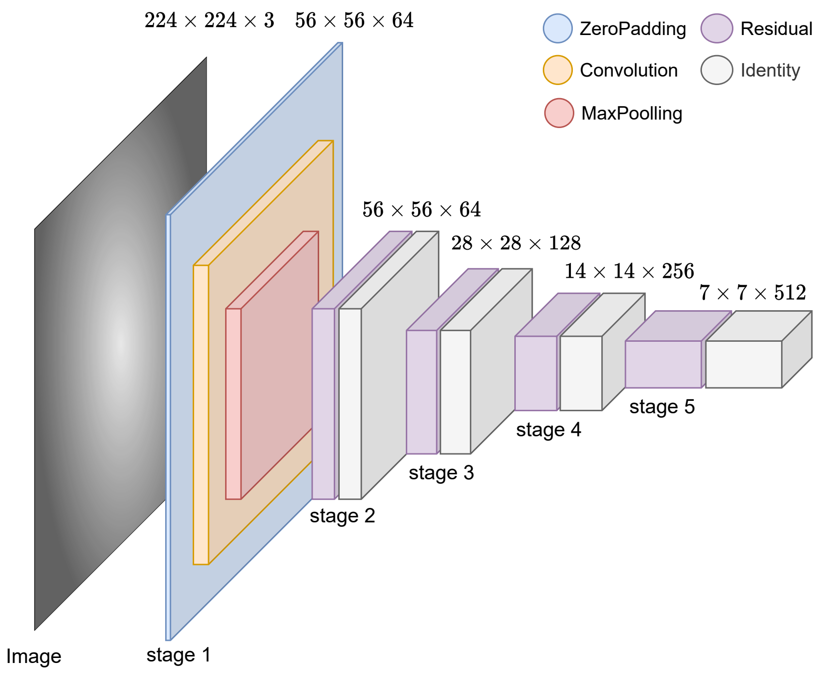


Figure 2.9: Architecture of ResNet-18

The key function of other stages (i.e., stages 2-5) is to extract a hierarchical representation. Each subsequent stage captures higher-level and more abstract features. At every stage, the kernel depths of the building blocks are equal. Whilst, the kernel depths of the `Conv2D` operation in the building block per stage are $f_{stage2} = \{64, 64\}$, $f_{stage3} = \{128, 128\}$, $f_{stage4} = \{256, 256\}$ and $f_{stage5} = \{512, 512\}$. In addition, at stage 2, there is an exception for the residual block's default sliding window value from 2 to 1.

After Stage 5, the typical ResNet architecture pools the resulting high-level features into a 1×1 feature map using a global-average pooling layer of $PO = 7 \times 7$. Later, the feature map of $1 \times 1 \times 2048$ is fed into a dense layer of 'n' neurons, with softmax activation for multi-class classification of 'n' classes in the dataset.

Popular Datasets

Dependency between the supervised learning models and the annotated datasets is vital to produce good DL models. The performance metrics of the architectures discussed so far are evaluated on the publicly available popular datasets. Projects such as ImageNet [30], COCO [31] and Open Images [32] etc., gather images along with the annotations of various classes for image classification, localization, segmentation and keypoint detection tasks to support the model training. Each project is focused on a certain task and has developed the datasets in particular categories with their own format for encoding the labelled annotations. The knowledge of the format and nomenclature of the annotations is very important while working with any SOTA architectures through transfer learning.

2.4.5. Model Training

In supervised learning such as CNN, the model is trained to reduce the losses between the ground truth (a.k.a targets) and the estimated values (i.e., predictions); this is achieved by optimizing the kernel weights (i.e., elements of kernel matrix $\boldsymbol{K}_{h \times w \times c}$) of the convolution layers. The former loss computation is carried through loss functions and the latter optimization task is performed by the optimizers. We discuss some most commonly used loss functions and optimizers in this section. The loss functions and optimizers in ML/DL can be metaphorically compared to an observer and controller in control system dynamics.

Loss functions

At the end of the CNN architecture, the dense layers result in model predictions, which could be an integer or float value depending on the applied activation function. Loss functions measure the model's prediction accuracy based on the desired prediction tasks. Cross-entropy loss, or log loss is mainly used for classification tasks, in which the dissimilarity between the predicted and the target probability distribution is logarithmically measured. A batch-level cross-entropy loss for M mini-batches with P predictions each is computed as in Equation (2.16)

$$CE_{loss}(y_p, y_t) = \frac{1}{M} \sum_{k=1}^{M} \frac{1}{P} \sum_{j=1}^{P} \left[- \sum_{i=1}^{N} y_{ijk,t} \log(y_{ijk,p}) \right] \tag{2.16}$$

where, y_p and y_t are the predicted and targeted probability for N classes. For $N = 2$, it is called binary cross-entropy loss and is used for binary classification. The categorical and sparse categorical cross-entropy are the variants of the CE_{loss} introduced for $N > 2$, that are classified based on target label interpretation (i.e., one-hot encoding or integer-based).

Similarly, the Huber loss, Mean Squares Error (MSE) loss and Mean Absolute Error (MAE) are widely used for regression tasks. Huber loss is less sensitive to outliers than others and prevents exploding gradients, therefore mainly implemented in robust regression. This loss function combines the best properties of MAE (L_1) and MSE (L_2) losses, such that it is quadratic for small prediction errors and linear for larger ones. The Huber batch loss is computed for the regression of four bounding box coordinates, as follows

$$Huber_{loss}(y_p, y_t) = \frac{1}{M}\sum_{k=1}^{M} \frac{1}{P}\sum_{j=1}^{P} \left[\sum_{i=1}^{4} \begin{cases} \frac{1}{2}(\Delta)^2 & \text{if } |\Delta| < \delta \\ \delta \cdot \left(\Delta - \frac{1}{2}\delta\right) & \text{otherwise} \end{cases} \right] \tag{2.17}$$

where, $\Delta = y_{ijk,t} - y_{ijk,p}$ is a residual or a prediction error, and δ is the quadratic to linear transition threshold. The Huber loss with $\delta = 1$ is called Smooth-L_1 loss, which converges to a constant zero loss. Most of the two-stage detectors implemented these two loss functions in their study. For an explanation of other loss functions, it is suggested to refer to the ML glossary [33].

Optimizers

For T trainable layers, the model's best-fit parameters (i.e., kernel weights W and bias b) are put in place by the optimizers, upon minimizing the loss function a.k.a cost function denoted by $L(y_p, y_t)$. A regularization cost is included in the cost function, in order to avoid the model overfitting to the learned features alone. This cost is computed for each trainable layer (i.e., $T(W, b)$) as the mean of the squared-Frobenius norm of T layers' weights and bias matrices, multiplied by the weight decay parameter λ, such that

$$R = \lambda \sum_{l=1}^{T} \left[\frac{\|T_l^W\|^2}{size(T_l^W)} + \frac{\|T_l^b\|^2}{size(T_l^b)} \right]$$

Some of the most common optimizers are Adagrad, Adam, RMSProp, Gradient Descent and Stochastic Gradient Descent. The Gradient Descent a.k.a Batch Gradient Descent (BGD) is a basic optimizer that adjusts the model parameters (i.e., $T_{new}(W, b)$) by taking

$$T_{new}(W, b) = T_{old}(W, b) - \alpha \frac{\partial}{\partial T} \left[L(y_p, y_t) + R \right]$$

the derivative of the loss function with respect to the trainable parameters, and later updates the weights in the direction of the negative gradient.

where α is the learning rate that controls the number of steps required for the optimizer to achieve the local minimum of the convex cost function. Different optimizers contain distinct control variables. In ML/DL they are termed as Hyperparameters (e.g., α, λ).

Stochastic Gradient Descent (SGD) is a variant of BGD, whereupon evaluating model performance, instead of updating weights after a single-batch iteration consisting of an entire dataset, the model parameters are updated at the end of each mini-batch M, based on a random sample chosen across M. SGD works efficiently with larger datasets and result in faster convergence, on the contrary, yield a noisy and unstable convergence.

2.5. Image classification and localization

The task of identifying the target object on the image plane is known as object localization. Region of Interest (RoI) is the area covered by the Bounding Box, a rectangular box with the target at its centre and is delimited by the top-left and bottom-right pixel coordinates. One or more objects that can be found in an image are known as instances. The study is performed on a multi-class labelled dataset with single-instance images. Likewise, object classification is a task to identify the object in the RoI.

Training the same architecture to detect similar objects involves repeating the massive computing task. Therefore, most object detection algorithms adopt the transfer learning technique, particularly in the backbone architecture. This work also implemented a similar procedure for training the CNN's backbone.

In this context, the outcome from some popular challenges such as ILSVRC (ImageNet Large Scale Visual Recognition Challenge), and COCO (Common Objects in COntext) are proven to be useful. While the research teams evaluated their algorithms in the competitions, AlexNet won the ILSVRC'12, the deeper ResNet-152 won the ILSVRC'15 and COCO'15 challenge in object detection, localization, and segmentation. Later, these architectures set out as a baseline for building several state-of-the-art classification and localization methods.

These state-of-the-art approaches are broadly categorised into Region-based (a.k.a. two-stage) and Single shot (a.k.a. one-stage) detectors. The categorization is based on the number of stages required to predict the object bounding boxes, class probabilities, and their respective strategy for localizing the object. The Figure 2.10 outlines the functional difference between them. At first, the two-stage detectors generate region proposals and then perform predictions with refinement over the proposals. Whereas, a one-stage detector processes both tasks in a single pass.

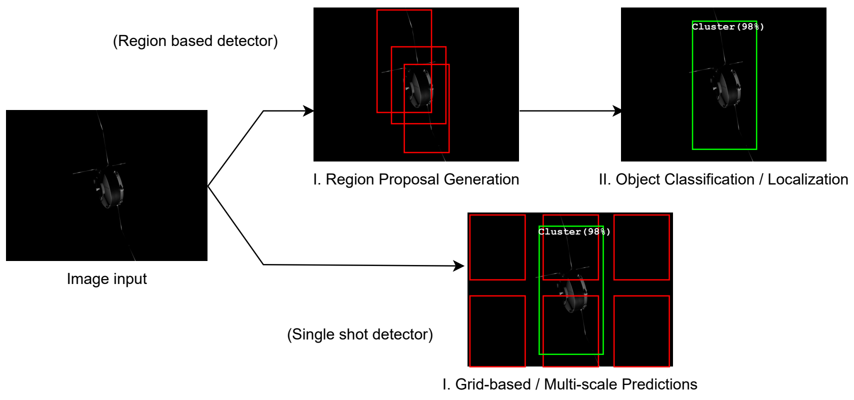

(Region based detector)
I. Region Proposal Generation
Cluster (98%)
II. Object Classification / Localization
Image input
Cluster (98%)
(Single shot detector)
I. Grid-based / Multi-scale Predictions

Figure 2.10: Image classification and localization categories

For example, R-CNN [34], Fast R-CNN [35], Faster R-CNN [36], and Mask R-CNN [37] methods belong to the two-stage detectors, in which, a region proposal technique is implemented in the first stage. Later in the second stage, the CNN-based classifier and regressor predict the object class and refine the bounding box coordinates. Each of the method's network architecture is widely published, and available on public domains; hence a detailed explanation is thought to be exhaustive and repetitive. Since all the methods have the same functional task to be fulfilled, of course by using various approaches, it is believed to be a good idea to focus solely on the unified framework and explain the different approaches in the key modules of two-stage detectors in this section.

The unified framework mainly consists of three modules: the CNN backbone for feature extraction; Secondly, the region proposal generator to locate RoIs; and lastly, the post-processing of the cropped feature map or an image, which depends on the method to implement. A snapshot depicting these key modules in a generic two-stage detectors pipeline is shown in Figure 2.11.

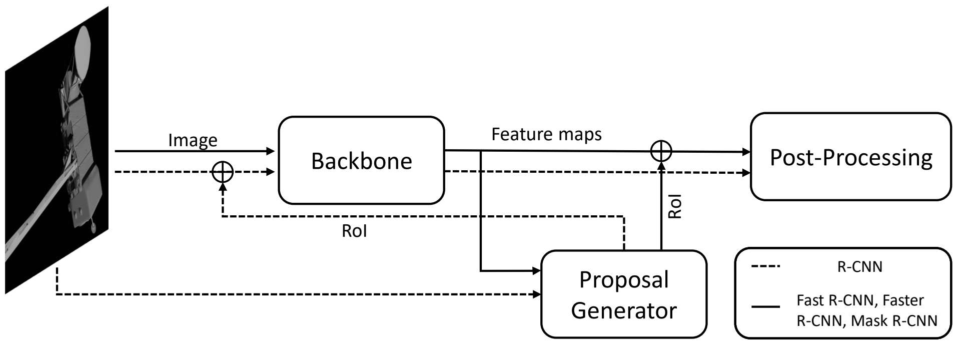


Figure 2.11: Two-stage detectors unified framework

The RoIs are the cropped regions proposed by the proposal generator. First, from the figure, the difference between the primarily introduced region-based Convolution Neural Network (i.e. R-CNN), and the others is easily noticeable. In R-CNN, it is observed that the RoIs crop pixels from the raw image. This process of feeding the cropped raw image as many times as the number of RoIs (i.e. typical value is $\sim$ 2000) through the backbone degrades the performance of R-CNN. Whereas, in the other methods, the pixels are cropped from a feature map, that is emerged at the end of different stages in the backbone in a single pass. This approach was computationally efficient and remained the same for all the successive methods.

Each method of this detector contains different architecture in the backbone module when published, such as AlexNet for R-CNN, VGG for Fast R-CNN and, ResNet-xx for both Faster R-CNN and Mask R-CNN. Similarly, the proposal generator module uses different techniques, such as the selective search technique and a simple convolution-based Region Proposal Network (RPN). Since one of them contributes to generating proposals in our object detection pipeline, the Section 2.5.1 will discuss these two techniques in detail. Finally, the last module (i.e., post-processing) receives cropped feature maps of either stacked or plain stages of multiple scales and shapes of different aspect ratios as input.

Therefore, after the RoI crop operation, these feature maps of different scales and sizes must be resized to ensure consistent input dimensions for subsequent layers in the network. Fixed-size RoIs simplify the processing and enable the use of fully connected layers or convolutional layers with fixed input sizes, for classification and regression tasks. For this, *Warping*, *ROIPool* and *ROIAlign* operations are used particularly in two-stage detectors. However, there are differences in how they handle the alignment and interpolation of features within the RoIs. In R-CNN, *warping* simply reshapes the region into 227×227×3 size images. Whereas, in Fast-RCNN, *ROIPool* divides the RoI into a fixed grid of sub-regions and performs max pooling within each sub-region to obtain a fixed-size output feature map. *ROIPool* does not consider any misalignment between the input feature map and the RoI, resulting in the misalignment of RoI boundaries with the grid. To resolve this issue, *ROIAlign* is introduced in Faster R-CNN that addresses the misalignment issue by using bilinear interpolation to extract features from the input feature map at sub-pixel locations. Instead of quantizing the RoI into a fixed grid, *ROIAlign* computes the exact sampling locations on the input feature map for each output feature map location. This allows for more accurate alignment and preserves spatial information better than *ROIPool*. The ROI of height h and width w extracts the feature map from the level k of the multi-level backbone pyramid, following the equation: $k = k_0 + log_2(\frac{\sqrt{h \times w}}{224})$; where k_0 is the initial level typically set to 4, and 224 is the canonical ImageNet fixed size.

The post-processing module primarily contains bottlenecks and network heads, that usually consist of fully connected layers followed by the respective classifier and regressor depending on user choice and application. It is also observed in Section 1.2, the authors modify especially the last few layers to adapt the network for both feature detection and pose estimation. Also in R-CNN, the classification of N classes in 2000 RoI proposals are performed with the Support Vector Machine (SVM) classifier with 4096×N weights. The 2000×4096 features are extracted from the last two dense layers of AlexNet. The SVM classifier tends to find an optimal hyperplane that separates different classes in a feature space. The goal is to maximize the margin between the classes, making the classifier more robust to unseen data. However, the Fast R-CNN and its successors contain two fully connected layers in common, resulting in a shared RoI feature vector. These features are further fed through the linear classifier[3] with softmax activation (i.e., a dense layer of N neurons) for classification and linear regressor[4] for the bounding box coordinates (i.e., a dense layer of $4\times N$ neurons).

Similarly, YOLO [38], SSD (Single Shot MultiBox Detector), and RetinaNet are other examples of one-stage detectors. These employ a predefined set of default bounding boxes (a.k.a anchor boxes) at different scales and aspect ratios, by using algorithms such as K-means clustering on the dataset. The anchor boxes are evenly distributed over the image or at each grid cell of a feature map as in Figure 2.10. For a grid cell of size S×S, the potential areas of the objects' existence are identified in the first iteration per grid, which generates objectness score and bounding box predictions along with their corresponding class probabilities in a single pass. Likewise the R-CNN family of networks, the YOLO method also has a history of eight official and unofficial versions to date. Instead of discussing the difference between the methods in one-stage detectors, we briefly discuss YOLO alone, since it is more efficient than the others in terms of speed and accuracy. Throughout the versions, YOLO significantly improved the object detection accuracy from 21.6% to 51.4% on the COCO dataset. One-stage detectors also consist of the same three modules in a series. The proposal filtering technique (i.e., Non-Maximum Suppression) is the same in both of the detectors. However, the backbone architecture generating feature map is different in YOLO, such as DarkNet-19/53 and CSPDarknet53 (Cross Stage Partial Network), [39–41] which explains the architecture in detail. In addition, the task performed in the post-processing module is included in the YOLO architecture as additional layers, called "detection heads" or "prediction heads".

[3]The linear classifiers also include logistic regression, which uses a logistic function to model the probability of an instance belonging to a particular class.

[4]Linear regressor contain a linear function that minimizes the sum of squared differences between the predicted values and the actual target values.

In summary, single-shot detectors are designed to be more efficient with faster inference times than region-based detectors, as they eliminate the need for a separate region proposal stage. However, they face challenges in accurately detecting small objects and objects in dense scenes. The region-based detectors, on the other hand, include the pre-defined boxes of very-low to high scales with a typical stride of 1 or 2 pixels, and perform detection on a regional basis, such that the possibility of missing an object in the image frame is minimal. Unlike YOLO, where each grid cell only predicts two boxes and can only have one class, the two-stage detectors do not impose strong spatial constraints on bounding box predictions. In this work, the best accuracy is of the highest priority over faster inference times. Losing an object in the image frame costs much higher than predicting more frames per second. Therefore, the two-stage detectors are preferred.

2.5.1. Proposal generator

Apart from a filtering technique to refine the top score boxes, both detectors do not share the proposal generator, as stated in the previous section. As a result, in this section, we will look at how the two-stage detectors generate proposals. The generation procedure consists mostly of three steps: the first is to determine the anchor boxes (i.e., pre-defined boxes), and the second is to predict the objectness scores and refine deltas for each of the anchor boxes. The following step is to use filtering algorithms to remove the redundant anchor boxes and generate RoIs of about 2000 for training and 1000 for testing.

The first step is introduced by Fast R-CNN to improve efficiency and speed up the process. Before that, the original R-CNN used selective search (see below) to generate region proposals. Anchor boxes are determined for an image of shape $H \times W$. The key parameters to define the boxes of a certain pixel area are their *aspect ratio* and *anchor stride* value, which determine the spatial arrangement of the anchor boxes. For example, if the anchor stride value is 2, the centre of the adjacent anchor boxes will be 2 pixels apart in both horizontal and vertical directions. The aspect ratio is defined as the ratio of the box width to the box height. Typical aspect ratio values utilized in and after Fast R-CNN are $[0.5, 1, 2]$ representing tall, square and wide boxes. The square root of the anchor box's pixel area defines the anchor box *scale*; the typical values are $[32, 64, 128, 256, 512]$ square pixels over the image. The anchor box dimensions are computed as

$$height = \frac{scale}{\sqrt{ratios}} \ , \qquad width = scale \times \sqrt{ratios}$$

that is, at each pixel position, for 'm' scales and 'n' ratios, there are '$m \cdot n$' anchor boxes. In addition, the same fashion of spatial arrangement is applicable to feature maps as well.

Selective search

Selective search is a bottom-up approach that aims to identify potential object regions in an image by grouping similar regions together, as an external method. It performs object localization by using an exhaustive search approach and also performs segmentation of colours in the given image. The exhaustive search uses sliding filters of different sizes to extract the objects. Here, the computation effort increases with an increase in filters. This algorithm uses the greedy algorithm to grow the region by locating similar colours in the regions and merging them together. The measure of similarity between pairs of neighbourhood regions is computed as

$$S(a,b) = \underbrace{S_{texture}(a,b)}_{\text{texture similarity}} + \underbrace{S_{size}(a,b)}_{\text{spatial proximity}}$$

The hierarchy of region clusters obtained from the final region grouping is used to generate a set of potential object region proposals. The underlying idea to introduce this in R-CNN and Fast R-CNN is that these candidate regions reduce the search space, and aid in improving efficiency and computational performance. The idea of candidate proposals served as a valuable method, however considered as a separate pre-processing step and does not fully integrate with the CNN object detection pipeline. In addition, it is an unsupervised algorithm that generates region proposals without considering the precise localization of objects. The Region Proposal Network (RPN) overcomes these limitations.

Region Proposal Network

Region Proposal Network (RPN) was introduced in the Faster R-CNN by Ren, He, Girshick, and Sun [36] in 2015, later, also employed in Mask-RCNN. The RPN generates proposals in an end-to-end manner eliminating selective search. The key distinction is that the selective search operates on the raw image, whereas, RPN operates on feature maps extracted from the backbone. For example, consider the backbone architecture as ResNet, which results in multi-scale feature maps at the end of each stage. Let's call them $FM = \{c_2, c_3, c_4, c_5\}$; the RPN processes these set of feature maps. It achieves a shared feature vector by sliding a kernel of shape 3×3, over the FM with $P = same$, $S = anchor\ stride$ and $f = 256$ or 512 followed by ReLU activation. The concept of creating shareable convolution is obtained from the Zeiler-Fergus (ZF) and VGG-16 models, which have 5 and 13 shared layers with a depth of 256 and 512 filters respectively. The shared feature vector is further connected to two sibling layers, one for objectness scores and the other to refine the bounding box coordinates. The term *objectness score* indicates the likelihood of an object being present within a kernel (implicitly indicates anchor box) and allows the network to discriminate between object and background (i.e., 2 classes). Simultaneously, bounding box regression refines the four coordinates of the pre-defined anchor boxes, improving localization accuracy. The assignment strategy and parameters of these pre-defined anchor boxes in RPN to process FMs are as follows:

- In the bottom-up pathway, the earlier stages consist of fine-grained information (i.e., low-level features), and more abstract representations (i.e., high-level features) appear at the final stages. Hence, the smaller scale boxes are mapped to the earlier stage FMs. For example, the anchor *scale* of 32 is mapped to c_2. Hence, typically the length of the *scale* vector equals the number of stages' outputs in the backbone (i.e., $len(scale) = len(FM)$).
- The *feature stride* or *backbone stride* is defined in Mask R-CNN as the ratio of CNN-fed image dimension to the feature map dimension. When the kernel slides 'S' pixels apart on the feature map c_2, it indicates that the kernel has moved *feature stride* pixels over the image (i.e., for an image of dimension 224×224 and c_2 map of 56×56, the c_2 map's convolution kernel implicitly overlooks the region of the image 224/56 pixels apart).
- In total, the number of anchor boxes employed by the RPN is:

$$\#anchor\ boxes = \sum_{i=0}^{len(scale)} \frac{FM[i]_{dim}[0] \times FM[i]_{dim}[1]}{S} \times len(ratios)$$

  and the modified height and width of the anchor box in the feature map would be

$$height_{FM} = \frac{scale}{backbone\ stride \times \sqrt{ratios}} \quad , \quad width_{FM} = \frac{scale}{backbone\ stride} \times \sqrt{ratios}$$

Therefore, the sibling layer properties that predict objectness scores for $len(ratios)$ anchors at each kernel location are $K = 1 \times 1$, $P = valid$ and $f = len(ratios) \times 2$ with softmax activation. Similarly, the layer properties for bounding box coordinates' linear regression are $K = 1 \times 1$, $P = valid$ and $f = len(ratios) \times 4$ with linear activation. Overall, selective search is a fixed algorithm that is not easily adaptable or optimized for specific datasets. On the other hand, the RPN is designed to be adaptable and trainable on different datasets. It offers better generalization and optimization by leveraging the training process and the shared convolutional backbone network. Whilst, the RPN assigns objectness scores for the $\#anchor\ boxes$. The final proposals are generated by filtering the redundant anchor boxes into two levels.

In the first level, simply, the top 'A' anchor boxes[5] with high-objectness scores are selected as the candidate regions or proposals. Later, at the second level of filtering, these $\#A$ proposals are further filtered with a technique called Non-Maximum Suppression (NMS), which is a common technique employed in both one-stage and two-stage detectors.

The NMS workflow is iterative, where, the $\#A$ proposals are iteratively filtered out, based on the proposal box's objectness scores and the amount of overlap threshold within its own proposals' set 'A'. The amount of overlap between any two boxes is computed by a popular metric known as Intersection over Union (IoU). In general, it is defined as the ratio of the intersection area to the union area between the two boxes. This IoU metric is mathematically represented as

$$IoU = \frac{a_{intersection}}{a + \tilde{a} - a_{intersection}} \tag{2.18}$$

where a and $\tilde{a}$ are the first and second box's area, computed by $[(y_2 - y_1) \times (x_2 - x_1)]$ and, $[(\tilde{y_2} - \tilde{y_1}) \times (\tilde{x_2} - \tilde{x_1})]$ respectively. Simultaneously, the intersection area (i.e., $a_{intersection}$) is calculated as $[\min(y_2, \tilde{y_2}) - \max(y_1, \tilde{y_1})] \times [\min(x_2, \tilde{x_2}) - \max(x_1, \tilde{x_1})]$.

Let's denote the scalar IoU threshold with 'T' and the final set of $\sim$ 2000 proposals as 'B'. The NMS would first consider a proposal box from 'A' with a high-objectness score, and eliminate it from 'A' whilst adding it to 'B'. Later, compute the IoU of this proposal box with all the other boxes in 'A' and eliminate the boxes in 'A' with $IoU > T$, considering them identical, and hence redundant. This iterative loop continues until the proposal set 'A' is empty. When the final set 'B' does not have enough proposals, zero padding is applied to maintain the tensor shape.

The NMS technique is sensitive to the 'T' value irrespective of the objectness score, like any threshold-based technique. For instance, if there are two objects side by side, one of them would be eliminated and the proposal with a lower threshold is still kept even though its score is very low. Another variant of NMS called *Soft-NMS* resolves this issue by reducing the proposal score proportional to the IoU value (i.e., *new score* = *old score* $\times$ *(1 - IoU)*), instead of complete removal.

To outline the filtering techniques, they are not fail-proof because their combination is highly sensitive to the RPN's objectness score prediction. However, considering our object classification and localization problem that focuses on multi-class labelled datasets with single-instance images, these techniques are adequate.

[5]Typical value of 'A' in Faster R-CNN and Mask R-CNN is $\sim$ 6000

2.5.2. Feature Pyramid Network

The Deep CNNs backbone such as ResNet-xx is effective at extracting high-level features. Along with this, the methods with a pre-defined anchor-based approach solve the translation-invariant problem. However, they do not explicitly address the issue of scale invariance. For this reason, Feature Pyramid Networks (FPN) is introduced by Lin, Dollár, Girshick, He, Hariharan, and Belongie [42], as a subsystem of the backbone module. The primary goal of FPN is to generate a feature pyramid that captures multi-scale information from the input image. It achieves this by combining features (i.e., FMs) of different stages from the Bottom-Up pathway (i.e., backbone CNN hierarchy).

FPN introduces a Top-Down pathway, which performs nearest neighbour upsampling of factor 2 (i.e., 2× of the FM, for instance, consider c_5) on the high-level semantics to maintain the same spatial size as the relatively lower-level semantic feature map (i.e., c_4), and merge them together through lateral connections. The lateral connection involves a convolution layer with $K = 1 \times 1$, $f = 256$ and $P = valid$, that adjusts the depth of both feature maps. Furthermore, to reduce the aliasing effect of upsampling a 3×3 convolution layer is attached, resulting in the final FPN feature maps (i.e., $\{p_2, p_3, p_4, p_5\}$). It is also possible to extend the pyramid, to an extra coarser level (p_6), with a maximum-pooling layer of size $PO = 2$ to capture information on larger objects. The building block of FPN, representing its core structure is shown in the Figure 2.12 below

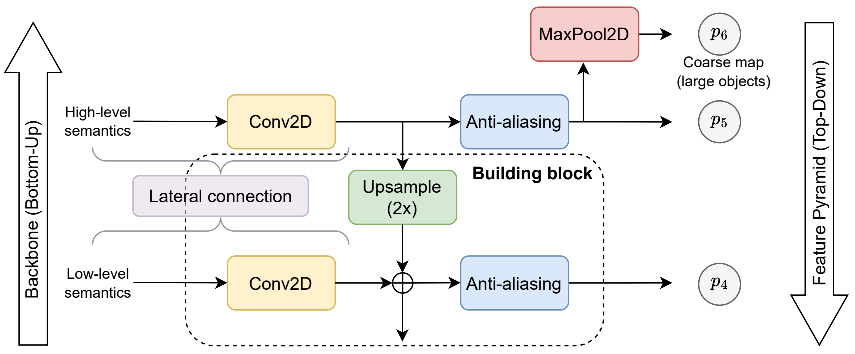


Figure 2.12: Building block of Feature Pyramid Networks (FPN)

FPN achieves a better balance between semantics and spatial precision and, enhances the network's ability to detect objects at various scales. The Fast and Faster R-CNN have an increment in their object detection performance of about 3.3% on the COCO dataset when implemented with FPN; the Mask R-CNN also implements FPN in their backbone module. FPN is a versatile and robust framework that can be integrated with various CNN architectures, it is widely adopted in the computer vision community.

2.5.3. Performance Indicators

The model predictions on an image are classified by use of a confusion matrix, composed of four elements, such as True Positive (TP), False Positive (FP), True Negative (TN) and False Negative (FN). Consider a multi-classification object detection, where the targeted class objects in the image are called the positive instances and the remaining class objects are negative instances. The predicted class and the IoU metric computed as in Equation (2.18) between the prediction and ground truth box are the conditional variables to define these elements as follows:

- True Positives are the correctly identified positive instances, that is, when IoU $\geq$ set threshold $\wedge$ $class_{predict} \equiv class_{target}$.
- False Positives are the incorrect detections, where the negative instances are wrongly identified as positive instances (i.e., IoU $<$ threshold $\vee$ $class_{predict} \neq class_{target}$)
- The correctly identified negative instances with IoU $\geq$ threshold $\wedge$ $class_{predict} \equiv class_{non-target}$ called True Negative.
- A prediction that wrongly identifies a positive instance as a negative instance and such missed detections (i.e., IoU $<$ threshold $\vee$ $class_{predict} \neq class_{non-target}$) are defined as False Negatives.

The image classification and localization model's prediction performance is indicated by two key indicators known as precision and recall. These indicators are usually evaluated on each image of the dataset. The indicators provide a confidence score between $0 - 1$. The measure of predicting a positive class correctly is called precision, and the model's ability to detect all available positive class instances is recall, these are denoted as

$$Precision : P = \frac{\#TP}{\#TP + \#FP} \qquad Recall : R = \frac{\#TP}{\#TP + \#FN} \tag{2.19}$$

Dataset-level precision is represented by the $P - R$ cure, for this, the precision set is rearranged in a monotonically decreasing manner and the area under the $P - R$ Curve is defined as Average Precision (AP). Average Recall (AR) at a certain threshold is computed as a mean value of the recall set obtained from evaluating R at each image.

As a common practice in object detection, all the above indicators (i.e., P, R, AP, AR) are computed separately either for each class or at varying thresholds (i.e., 0.5 - 0.95 with step 0.05), and even both. Whilst, the average recall of a dataset is computed as the maximum AR value obtained across different thresholds. The mean of the average precision across multiple classes or thresholds is called mean Average Precision (mAP). Note that for a single instance object detection problem, the mAP and AP are equal.

2.6. Perspective transformation

Perspective transformation refers to "*Perspective Projection*" and "*Perspective-n-Point*" (PnP). *Perspective Projection* is a process of mapping 3D points in the world coordinate system to their corresponding positions on the 2D image plane of a camera. In 1981, Fischler and Bolles [43] coined the specific problem of 2D to 3D transformation as "Perspective-n-Point" (PnP), where 'n' represents the keypoints in the image plane.

2.6.1. Perspective Projection

The projection of a 3D point on the 2D plane involves four coordinate systems, that are: World Coordinate System (WCS), Camera Coordinate System (CCS), Image Coordinate System (ICS), and Pixel Coordinates System (PCS). The WCS is a global space used to describe the positions, orientations, and dimensions of objects or points in cartesian coordinates (i.e., X, Y, Z).

The CCS is a reference frame for describing the viewpoint and perspective of the camera. The origin of this system is at the camera's position or the focal point. Typically in computer vision applications, its Z_c faces outward, in other words, inward to the camera lens (i.e., camera principal axis). Whilst, the Y_c lies in the opposite direction to the Y_w and, X_c is parallel to X_w. The 3D point $(X, Y, Z)^T_w$ in WCS is converted to CCS with the 4×4 transformation called *camera extrinsic matrix*, as shown in Equation (2.20); where R^c, t^c are the Euler rotation matrix and translation vector of the camera in WCS.

$$\begin{Bmatrix} X \\ Y \\ Z \\ 1 \end{Bmatrix}_c = \underbrace{\begin{bmatrix} R^c_{3\times3} & t^c_{3\times1} \\ 0_{1\times3} & 1_{1\times1} \end{bmatrix}}_{\text{Extrinsic matrix}} \cdot \begin{Bmatrix} X \\ Y \\ Z \\ 1 \end{Bmatrix}_w \tag{2.20}$$

Origin's location is a key difference between the ICS and PCS plane, in which, the plane's origin lies at the centre and the top-left respectively. The Z_c passes through the origin, and the 2D plane axes $[X_i, Y_i]$ for ICS and $[u, v]$ for PCS, which are parallel to X_c and Y_c. In addition, the ICS follows SI units and the PCS units are pixels. The distance from the focal point to the image plane along Z_c is called the *focal-length (f)*. Transforming a point in the perspective view (i.e., CCS), to either ICS or PCS involves another 3×4 matrix called *camera intrinsic matrix*. This 3D to 2D projection is a lossy transformation (i.e., loss of depth information), and is achieved via the Pinhole model, which assumes a simple no-lens camera but with a tiny aperture. The model projections are based on the law of similar triangles, over the rays between the point, camera and image. Therefore, results in $\frac{X_i}{f} = \frac{X_c}{Z_c}$ and $\frac{Y_i}{f} = \frac{Y_c}{Z_c}$; rewriting this relationship in matrix form derives the intrinsic matrix as in Equation (2.21)

$$\begin{Bmatrix} X_i \\ Y_i \\ Z_c \end{Bmatrix} = \underbrace{\begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}}_{\text{Intrinsic matrix (ICS)}} \cdot \begin{Bmatrix} X \\ Y \\ Z \\ 1 \end{Bmatrix}_c \tag{2.21}$$

Finally, the transformation between the ICS and PCS is simpler because the only difference is an origin shift. In terms of converting SI units to pixel representation, it is highly dependent on the sensor model and the *pixel-pitch* parameter (defined in Section 3.3.1). In short, *pixel-pitch* is the ratio of the sensor dimension to the number of pixels allocated in that dimension. Let's call them ρ_u, ρ_v (i.e., for a square-shaped sensor $\rho_u \equiv \rho_v$). Similarly, consider the shift along X_i and Y_i as u_0, v_0 in pixels, such that, $u = \frac{X_i}{\rho_u} + u_0 = \frac{1}{\rho_u} f \frac{X_c}{Z_c} + u_0$ and $v = \frac{Y_i}{\rho_v} + v_0 = \frac{1}{\rho_v} f \frac{Y_c}{Z_c} + v_0$. As a result, the improved intrinsic matrix for transforming the 2D point from ICS to PCS is shown in Equation (2.22). The projection matrix denoted by $\mathbf{P}_{3\times4}$ fuses the three transformations in Equations (2.20) to (2.22) as $T_{PCS} \cdot T_{ICS} \cdot T_{CCS}$.

$$\begin{Bmatrix} u \\ v \\ w \end{Bmatrix} = \underbrace{\begin{bmatrix} \frac{1}{\rho_u} & 0 & u_0 \\ 0 & \frac{1}{\rho_v} & v_0 \\ 0 & 0 & 1 \end{bmatrix}}_{\text{Intrinsic matrix (PCS)}} \cdot \begin{Bmatrix} X_i \\ Y_i \\ Z_c \end{Bmatrix} \tag{2.22}$$

3 | Dataset generation

3.1. Spacecraft selection

The key drivers in selecting the spacecraft include a minimum of two different shapes, stronger shadows, and varied illumination conditions. The idea is to register the images with these effects naturally by considering the real data of the spacecraft and the sun's orbital position. The options for shape categorization are in Table 3.1.

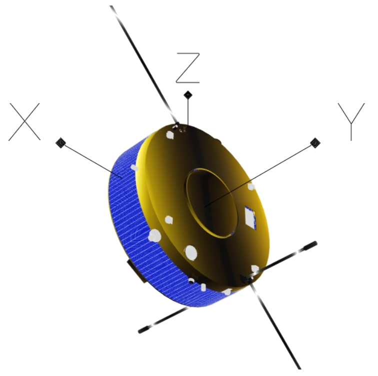
Category	Shape	Spacecraft	Orbit	Inc [°]	Alt [km]	Operational
Sharp	rectangle	Aqua	SSO	98.19	700	Y
	square	proba-2	SSO	98.2	720	Y
	polygon	acrimsat	SSO	98.3	720	N
	trapezoid	cygnss	LEO	35	520	Y
		sentinel-6	LEO	66	1336	Y
Smooth	circle	Cluster	HEO	135	16000*116000	Y
	cylinder	aquarius	SSO	98	657	N

Table 3.1: Shape categorization

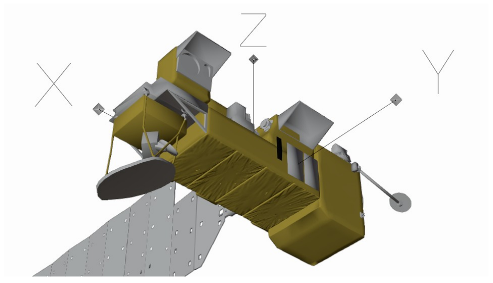
The aim is to have disparate features in the dataset, hence including one from sharp and smooth produces robustness to most generic shapes available, that may possibly extend to rocket boosters, cuboids and more. Keeping in mind the lighting conditions and the amount of background composition (Earth/Dark starry) to be included in the dataset, it is the best choice to select spacecraft from HEO and SSO. The respective inclination, altitude and operational status of the spacecraft are also reported in Table 3.1. Whereas, concerning the shape: perfect-cylindrical and rectangular shapes provide an opportunity to address the problem of symmetrical ambiguities in shape through deep-learning.

3.1.1. 3D models

The three-dimensional geometrical models of the Cluster and Aqua as shown in Figure 3.1, are acquired from the official ESA's Scifleet[1] and NASA[2] archives. The raw model of the Cluster includes a main body, four 50m wire booms, two 5m rigid booms in XZ-plane, and two antenna booms along Y-axis [44]. On the basis of considering the close proximity scenario, such a scale of geometry is definitely out of Field Of View (FOV). The Cluster is spin-stabilized with the large wire booms rotating at 15 rpm. This rotating nature can result in feeding ambiguities to the supervised and pre-trained CNN. Hence, 50m wire booms are neglected and the modified model is reported in Figure 3.1a. Whereas, the Aqua model shown in Figure 3.1b does not have any modifications.



(a) Cluster.



(b) Aqua.

Figure 3.1: Orthographic view of spacecraft 3D models.

The geometrical properties of the two spacecraft are tabulated in Table 3.2. The Centre of Mass (CoM) denotes the surface parameter, not the inertial mass parameter. For the 3D Bounding Box, the dimensions are scaled to one. Here, the tabulated values include the rigid booms and antenna parameters for the Cluster, and solar panels for the Aqua.

	Body axis	Cluster	Aqua
Dimensions [m]	X, Y, Z	[2.9, 1.3, 2.9]	[8, 16.7, 4.8]
Bounding Box [-]	X, Y, Z	[0.616, 0.378, 1.000]	[0.529, 1.000, 0.286]
Centre of Mass [m]	X, Y, Z	[-0.0335, 0.1078, -0.0474]	[1.5508, 0.2342, -0.1669]

Table 3.2: Dimensional data of 3D models

[1]`https://www.cosmos.esa.int/web/esac-cmso/scifleet`
[2]`https://nasa3d.arc.nasa.gov/models`

3.2. Propagator

The orbit propagator model as in Figure 3.2 follows the ECI frame and does not include perturbation dynamics. The model is developed in `MATLAB and Simulink` environment with filename `propagator_script.m`. The relative distance of the spacecraft and the sun from the earth, including the spacecraft orientation are detailed in Sections 3.2.1 and 3.2.2. Whereas, the camera path trajectory is derived from the relative dynamics models in a spacecraft-centre frame, as in Section 3.2.4, to extract the camera's position.

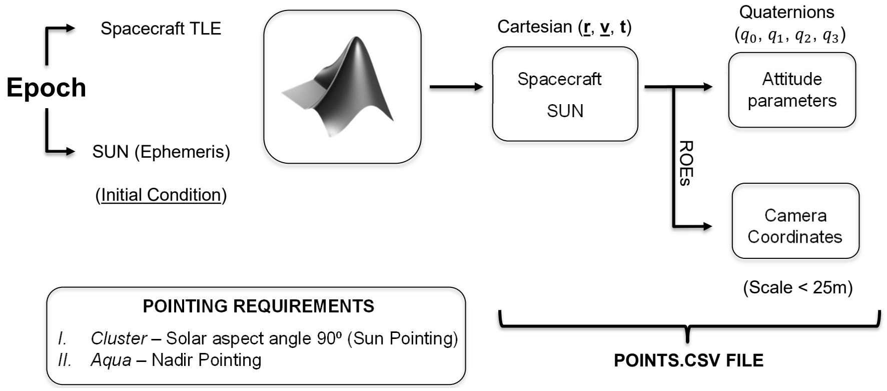


Figure 3.2: Propagator model

The model works with the Two-Line-Element (TLE) set of the desired spacecraft at certain epoch from CelesTrak, which are the only input parameters required for the target spacecraft and sun propagation. Whereas, the camera path is derived by the desired non-dimensional relative orbital elements (δROE). The output parameters of the model are stored in the comma-separated value format in `points.csv` and are reported below.

1. Normalised camera position relative to spacecraft in LVLH frame.
2. Distance vector from the spacecraft to the sun.
3. Spacecraft's relative orientation with Earth.
4. Sun visibility with respect to the sun.
5. Direct solar flux acting on the spacecraft.
6. Relative distance vector of the spacecraft with the earth.
7. Earth's reflected radiation.
8. Earth's rotation on its spin axis.

3.2.1. Spacecraft orbit

The spacecraft orbit is defined from the TLE set in ECI frame with the following parameters: inclination, longitude of ascending node, eccentricity, argument of perigee, mean anomaly at epoch and mean motion of the orbit. The epoch is considered to be the initial state for the propagation and denoted with subscript $(\square_0)$. The TLEs of the Cluster and Aqua are recorded at an epoch of 21334.84382759 and 22031.08504144 respectively, and are reported below in Tables 3.3 and 3.4.

i [°]	Ω [°]	e [−]	ω [°]	M_0 [°]	n [rad/s]
133.35	345.92	0.52	195.45	0.38	3.21e-05

Table 3.3: Cluster - Two Line Elements set

i [°]	Ω [°]	e [−]	ω [°]	M_0 [°]	n [rad/s]
98.23	334.40	8.98e-05	52.44	318.10	1.05e-03

Table 3.4: Aqua - Two Line Elements set

The orbit's semi-major axis and time period are computed as in Equation (3.1).

$$a = \sqrt[3]{\frac{\mu_E}{n^2}}, \qquad T = \frac{2\pi}{n} \tag{3.1}$$

Upon neglecting the orbital perturbations in the model, the orbit is simulated through the integration of true anomaly as in Equation (3.2).

$$\dot{\theta}(t) = \frac{n(1 + e \cdot \cos(\theta(t)))^2}{(1 - e^2)^{3/2}} \tag{3.2}$$

where, $\theta(t)$ is the true anomaly in time starting with initial condition θ_0 that is derived from the Equation (3.3). The integration time-step is fixed to a value of $\frac{t_{sim}}{N}$, where the number of instances recorded is denoted by N, and t_{sim} represents simulation time i.e., computed as a multiple of the orbital time period (T).

$$\theta_0 = M_0 + (2e - 0.25e^3) \cdot sin(M_0) + 1.25e^2 \cdot sin(2M_0) + \tfrac{13}{12}e^3 \cdot sin(3M_0) \tag{3.3}$$

The distance of the spacecraft from the Earth is computed from $r_{s/c} = \frac{a(1-e^2)}{1+e \cdot \cos(\theta(t))}$.

With the knowledge of keplarian elements and true anomaly, the position and velocity components in the Perifocal Frame (PF) are retrieved.

$$\vec{\boldsymbol{r}}_{\boldsymbol{PF}} = r_{s/c} \begin{bmatrix} \cos(\theta) \\ \sin(\theta) \\ 0 \end{bmatrix} \qquad \vec{\boldsymbol{v}}_{\boldsymbol{PF}} = \sqrt{\frac{\mu}{a(1-e^2)}} \begin{bmatrix} -\sin(\theta) \\ e+\cos(\theta) \\ 0 \end{bmatrix}$$

The rotation matrices of the ECI to PF are transposed and multiplied to the above, to compute the position and velocity components in the ECI Frame.

$$\begin{aligned} \vec{\boldsymbol{r}}_{\boldsymbol{s/c}} = \vec{\boldsymbol{r}}_{\boldsymbol{ECI}} &= \boldsymbol{R}_3^T[\Omega] \cdot \boldsymbol{R}_1^T[i] \cdot \boldsymbol{R}_3^T[\omega] \cdot \vec{\boldsymbol{r}}_{\boldsymbol{PF}} \\ \vec{\boldsymbol{v}}_{\boldsymbol{ECI}} &= \boldsymbol{R}_3^T[\Omega] \cdot \boldsymbol{R}_1^T[i] \cdot \boldsymbol{R}_3^T[\omega] \cdot \vec{\boldsymbol{v}}_{\boldsymbol{PF}} \end{aligned}$$

3.2.2. Spacecraft attitude

The attitude estimation model implemented in the propagator is free of control and disturbance torques. The underlying assumption in the model is that the spacecraft is perfectly oriented towards the pointing direction, in other words, the respective body axis is aligned with the pointing requirement for the spacecraft. A decision is made such that the propagator requirement is to orient the spacecraft towards the pointing direction in the instance of image rendering, and hence it is not necessary to implement the entire dynamic model. Instead, it is needed to compute the directional cosines of the spacecraft at the instance of rendering and align the body axis accordingly.

The pointing requirements of the two spacecraft: Cluster and Aqua, are referenced from the official documentation [44, 46], in which Cluster attitude is ensured at a solar aspect angle of about 90° (i.e., sun-pointing), and Aqua's positive z-axis in body frame is pointed towards the earth's nadir direction. Propagation of Cluster at 10 orbital positions pointing the Sun is shown in Figure 3.3. To implement these requirements NED coordinates frame (see Section 2.1.2) is utilized, where the unit column vectors $\hat{\mathbf{E}}$ and $\hat{\mathbf{N}}$ stands perpendicular to the pointing direction and is computed as in Equation (2.2) and Equation (2.3). The required pointing vectors for Cluster and Aqua are $\hat{\boldsymbol{r}}_{\boldsymbol{R_s}}$ and -$\hat{\boldsymbol{r}}_{\boldsymbol{s/c}}$, that are aligned with the unit vector direction $\hat{\mathbf{D}}$. Accordingly, the directional cosines matrix in Equation (2.4) is evaluated by transforming the reference frames from ECI to NED body frame.

Throughout the research, the 'quaternions' are the preferred attitude parameters which can avoid singularities in comparison to the DCMs. Hence, the evaluated DCMs are also parameterized into quaternions following the Reference [see 27, Section 2.6.3]. To verify the correctness, the unity norm of the quaternions at each instance is also evaluated.

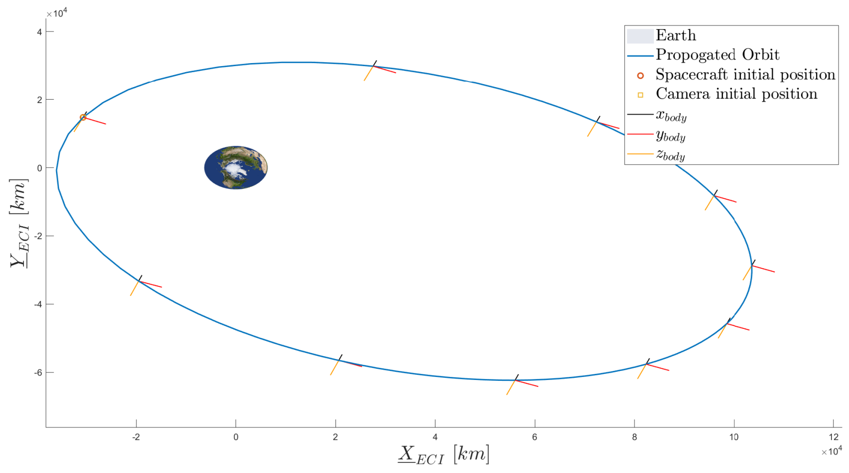


Figure 3.3: Orbital position and pointing direction of the spacecraft.

3.2.3. Sun orbit

The position of the sun's ephemeris in ECI is computed for the instance of the spacecraft simulation time and modelled as in Equation (3.4).

$$t_{sun} = t_{epoch} + t_{s/c} \qquad \text{where: } t_{s/c} \text{ is represented in days.} \tag{3.4}$$

Later, the Matlab function block `epoch2date` transforms the t_{sun} in TLE format to the Gregorian date, and finally to the Julian date. The Planetary Ephemeris block (see Figure 3.4) in Simulink implements NASA's JPL Chebyshev coefficients of the DE405 ephemeris model for the position and velocity estimates [8].

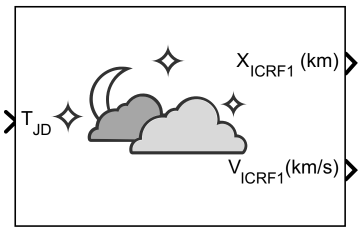


Figure 3.4: Planetary ephemeris [8]

The sun is the only source of illumination in our case, so it is important to compute the spacecraft eclipse due to the Earth in between. For this aspect, three different angles and a distance vector are evaluated in between the three bodies: sun, spacecraft and earth with the set of equations as in Equation (3.5).

$$\begin{cases} \alpha = \arccos \dfrac{\vec{\boldsymbol{r}}_{s/c} \cdot \vec{\boldsymbol{R}}_s}{\left\|\vec{\boldsymbol{r}}_{s/c}\right\| \cdot \left\|\vec{\boldsymbol{R}}_s\right\|}, \\ \beta = \arccos \dfrac{R_E}{\left\|\vec{\boldsymbol{r}}_{s/c}\right\|}, \\ \gamma = \arccos \dfrac{R_E}{\left\|\vec{\boldsymbol{R}}_s\right\|}, \\ \vec{\boldsymbol{r}}_{\boldsymbol{R}_s} = \vec{\boldsymbol{R}}_s - \vec{\boldsymbol{r}}_{s/c}. \end{cases} \tag{3.5}$$

where $\vec{\boldsymbol{r}}_{s/c}$ and $\vec{\boldsymbol{R}}_s$ denote the position vector of the spacecraft and the sun in ECI frame; the constant value of earth's radius is denoted by R_E. Among the three angles: α is the angle between the spacecraft and the sun, β and γ are the angles computed with the magnitude of earth's tangent to the spacecraft and the sun respectively. The logical value of the eclipse is calculated with an angles-based approach, such that $\alpha > (\beta + \gamma)$ denotes the spacecraft eclipse in which the light illuminated over the spacecraft is zero. The distance vector from the sun to the spacecraft is denoted by $\vec{\boldsymbol{r}}_{\boldsymbol{R}_s}$.

Radiation

The solar and orbit-specific radiation over the spacecraft surface is also considered to account for proper illumination. Solar flux acting on the surface is a parameter that cannot be accounted the same as of the earth, because of the widely varying semi-major axis in the two different orbits: HEO and LEO. Hence, this value is computed from the relative distance between the spacecraft and the sun ($\hat{r}_{R_s}$). The formulae are reported below in Equation (3.6).

$$Solar flux = \frac{1367}{\left\|\vec{\boldsymbol{r}}_{\boldsymbol{R}_s}\right\|^2} \; \frac{W}{m^2} \qquad \text{where: } \left\|\vec{\boldsymbol{r}}_{\boldsymbol{R}_s}\right\| \text{ is in astronomical units.} \tag{3.6}$$

Regarding the orbit-specific radiation: only the solar radiation reflected over the earth's surface (albedo) is parameterized. For this ESA's SPace ENVironment Information System SPENVIS model data is utilized. Through one-dimensional interpolation of the data, an instance-specific albedo value is also added to the total radiation and all the other contributions over the spacecraft are ignored concerning the monocular camera being operated in the visible spectrum alone.

$$\text{Total radiation} = \text{solar} + \text{albedo} + \xcancel{\text{IR}} + \xcancel{\text{internal}}$$

3.2.4. Camera trajectory

In order to replicate a close relation with the real-time rendezvous scenario, the design of the camera trajectory is derived based on the Relative Orbital Elements (ROEs) (see Section 2.2). The research is mainly focused on pose estimation such that, developing an accurate propagator is an implicit goal. It is important to understand that building an arbitrary trajectory does not compromise the aim, and is designed only to construct the geometrical path of the camera. The camera location planning followed uniformly distributed, space discretization techniques [24] to generate the SPEED dataset. Instead, the idea in this study is to make use of the concept of unperturbed relative dynamics, and possibly demonstrate the advantage of creating a mission-specific dataset with the known ROEs alone, that could support pre-mission simulations and also to train/evaluate the neural model.

In this study, the closed-loop relative dynamics for the target-chaser's relative motion propagation is implemented by introducing the parameter H, which represents the number of different perspectives from which the camera is registering an image at N^{th} spacecraft instance. In analogy with the parameter N for the spacecraft propagation (see Section 3.2.1), there exist H relative orbits around a spacecraft over the integration time span. The selection of initial conditions for these H relative orbits in terms of ROEs define a strategy, and a simple one that is applied in the tool is as follows

1. The relative orbits should be spacecraft centric (i.e., $\delta\lambda = 0$) and maintain the same shape (i.e., $\Delta a = 0$) and the same size (i.e., $\Delta e = \text{const}$), that refers to the design of bounded-centric orbits.
2. Considering the relative RAAN ($\Delta\Omega$) and argument of perigee ($\Delta\omega$) as the control parameters, it is observed that the perturbation applied on one can define another as if the relative mean argument of longitude is zero i.e., $\Delta\omega + \Delta M + \Delta\Omega\ cos\ i_t = 0$, where $\Delta M = \Delta\Omega$.
3. From the literature [5], an increase and decrease of the inclination vector parameters ($i.e., \delta i_x,\ \delta i_y$) resemble harmonic oscillations of relative orbit in RN and TN plane respectively.

Therefore, the algorithm is framed to contain two user inputs of which one is the difference in eccentricity Δe that controls the relative distance in magnitude, and the other is to define $\delta\vec{e}$ with either the $\Delta\Omega$ or $\Delta\omega$ input value. The relative inclination δi_x and δi_y are varied at a step size of $\frac{H}{4}$ relative orbits with a magnitude of $\pm\frac{2\delta i_{y0}}{H}$ and $\pm\frac{\delta i_{y0}}{H}$ respectively. The highlight of this approach is that there exist four camera locations for one instance on

$\pm R$, $\pm T$ axes that could support resolving symmetrical ambiguities in pose estimation.

Algorithm 3.1 Camera relative motion propagation

```
 1: Initial ROEs : $\boldsymbol{\delta\alpha_0} = [0, 0, \delta e_{x_0}, \delta e_{y_0}, \delta i_{x_0}, \delta i_{y_0}]^T$
 2: for $h = 1 : H$ do
 3:     refine the initial values for varied relative inclination.
 4:     if $h < 0.25H$ then
 5:         $\delta i_x = \delta i_{x_0} + \frac{2\delta i_{y_0}}{H} \times h$
 6:     else if $0.25H \leq h < 0.5H$ then
 7:         $\delta i_x = \delta i_{x_0} - \frac{2\delta i_{y_0}}{H} \times h$
 8:     else if $0.5H \leq h < 0.75H$ then
 9:         $\delta i_y = \delta i_{y_0} + \frac{\delta i_{y_0}}{H} \times h$
10:     else if $h \geq 0.75H$ then
11:         $\delta i_y = \delta i_{y_0} - \frac{\delta i_{y_0}}{H} \times h$
12:     end if
13:     Refined ROEs: $\boldsymbol{\delta\alpha}(t_0) = [0, 0, \delta e_x, \delta e_y, \delta i_x, \delta i_y]^T$
14:     for $k = 1 : N$ do
15:         get $\boldsymbol{T}(t)$ matrix as in Equation (2.10) for $k^{\text{th}}$ instance.
16:         evaluate $\vec{\boldsymbol{X}}(t)$ following the Equation (2.13) for $k^{\text{th}}$ instance.
17:         update $\boldsymbol{\delta\alpha}(t_0)$ for next $\vec{\boldsymbol{X}}(t)$ with $\boldsymbol{\delta\alpha}(t)$ of the previous step, to form closed-loop system.
18:     end for
19:     Therefore, the relative states $\vec{\boldsymbol{Y}}_{(N(h-1)+1:Nh,\ :)}$ are $\vec{\boldsymbol{X}}(t)$.
20: end for
```

With respect to the above discussion on a trial and error basis, Δe is set to a value of 0.005 with the requirement of relative distance in magnitude to be less than $25m$ such that, the resulting camera path for the Aqua is reported in the Figure 3.5. To summarize the working idea, imagine that the spacecraft rotating in time is mounted on a simulator bed and the camera is revolving relative to the spacecraft at each spacecraft rotation step,

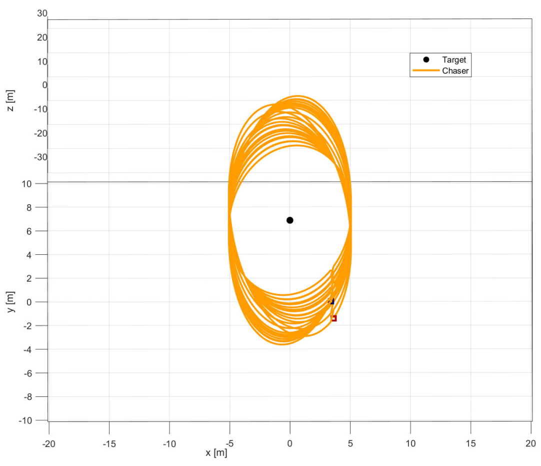


Figure 3.5: Camera trajectory

in which the position is derived from the Keplerian motion propagation in Cartesian form. Later, the pose data of the camera is registered in the RTN frame for generating the labelled dataset.

3.3. Monocular camera

The technological challenges in pose estimation are related to the selection of the best-suited sensor architecture for the given environments. As previously discussed in Chapter 1 the significant advantage of having a visual aid in close-proximity situations for full pose estimation, the Section 3.3 will detail the specific use of the monocular camera among the other Electro-Optical sensors (EO), and a brief discussion about the sensor model utilized for the POSSE simulator is reported in the Section 3.3.1.

The EO sensors collect reflected/emitted radiation in the electromagnetic spectrum, of which the range of frequencies from $0.37\mu m$ - $0.75\mu m$ and $0.75\mu m$ - $1000\mu m$ belong to the visible light and infrared radiation, respectively. In spacecraft relative navigation, the above two are widely applicable and classified into passive (Monocular/Stereo cameras) and active (LiDAR) systems.

Among the passive imagers, infrared imaging allows good Line-Of-Sight (LOS) measurements but it is poorly textured for pose estimation. On the other hand, the images in the visible band contain abundant data to process the full pose estimates [see 48, Section 2]. In analogy, the active systems that work with a back-scattering principle in the FOV are even better in accuracy and also have their pros and cons on this topic; upon respecting the mass and power factors as limitations in the study, the choice is discarded. Even though stereo cameras are good at acquiring 3D information, to simplify the process of dataset generation and the scalable approach of having RGB/Grayscale images as widely acceptable for pose estimation applications, a monocular camera is preferred.

In the past, such cameras were used in Hubble Space Telescope Servicing Mission 4 (SM4) for relative navigation towards an uncooperative target - telescope. Furthermore, many authors in [2, 3, 24, 26] presented their works on cooperative and uncooperative targets pose estimation from monocular imaging resulting in promising outcomes and further challenges with the single-image-based approach. To understand the state-of-the-art on the monocular model-based pose estimation better, the authors Pasqualetto Cassinis et al. [48] and Opromolla et al. [49] presented a detailed review for further reference. Moreover, monocular imagers are computationally less expensive in terms of processing algorithm onboard and hardware complexity. In addition, these are good at detecting and tracking objects up to a very close range, which is best suitable for this study.

3.3.1. Sensor model

The characteristics of the monocular vision are to be well understood to define the sensor model. It mainly consists of sensor properties such as focal length, field-of-view and sensor dimensions. In addition, there exist image properties such as resolution and pixel pitch that cross-correlate with the sensor. The definition of the parameters is as follows

- Resolution: The term 'pixel' in digital imaging refers to the smallest single component either in square or dot shape. Several pixel overlay over the image plane creates an image. The number of pixels that are aligned along the image plane axes defines the resolution and is often measured in pixels. So, a higher resolution means greater image clarity.
- Pixel pitch: It defines the density of pixels along the axes i.e., the distance in millimetres from the centre of a pixel to the centre of the adjacent pixel. A smaller pixel pitch means there is less empty space between pixels. Therefore, the higher the pixel density better the image resolution.
- Field of view: The observable area of the sensor measured angularly in the horizontal and vertical direction defines field-of-view in degrees.
- Sensor dimension: It is the physical measurement of the sensor lens in millimetres.
- Focal length: The distance between the sensor lens and the image plane/focal point in millimetres, along the principal axis i.e., the orthogonal axis of the image plane. The focal length (f) can be computed in relation to the sensor dimension and FOV such that, $f = \frac{\text{sensor width}}{2\arctan(\frac{FOV_h}{2})}$.

It is widely known that the creation of synthetic images employs a virtual camera for the simulation. The SPEED dataset [22] contains both the rendered and real images with a resolution of 1920×1200 pixels captured using the Point Grey Grasshopper 3 camera with a Xenoplan 1.9/17 mm lens that replicates the PRISMA onboard hardware. Similarly, the URSO dataset [3] utilises a virtual camera with the 90° horizontal FOV generating RGB images of 1080×960 in resolution. Likewise, the above two as baselines, the virtual camera model for the POSSE is desired to have a square-pixels of mid-range resolution and wide FOV as reported in Table 3.5.

Field of View [°]	Sensor width [mm]	Resolution [px]	Pixel Pitch [$\mu m/px$]
90	36	1280×960	28.12

Table 3.5: Sensor model

3.4. Simulator

As previously stated in Section 1.1 regarding the significance and lack of availability of the labelled spacecraft datasets for training the Neural Networks (i.e., supervised learning), the unPerturbed Orbital Simulator for SpacEcraft (POSSE) is developed as an implicit goal of the study. The POSSE is set up to render the images, synthetically using the free and open-source 3D computer graphics software toolset known as Blender 3.0[3] developed by Blender Online Community. The software is released under *GNU General Public Licence*[4] thus eliminating the user-access restrictions on using POSSE. The important tasks that the POSSE relied upon include 3D Modeling, UV Editing, Raster Graphics, Geometry Nodes, and Rendering, all of which Blender 3.0 is capable of performing like every other graphics software but with enhanced precision and control over the parameters. Another aspect that must be emphasized on the software is that it is coded in `C`, `C++` and `Python` languages, and make use of `Python 3` for scripting its API[5] (i.e., `bpy` and `mathutils` modules). Henceforth, it provides an edge to automate the dataset generation tasks through coding with `Python` programming language. Therefore, this results in the development of `simulator.py` file to perform the rendering operation, followed by the execution of the file on Blender 3.0 in the background via the Command-line interface (CLI) leveraging the Blender API.

In order to define the POSSE working environment, PyCharm© 2021 (Professional Edition) editor is used along with the interpreter that is identical to the Blender 3.0 i.e., `Python 3.9` for the tool development. It is preferred to detail the simulator framework by dividing it into three parts. The first part follows as in the Section 3.4.1, which focuses on building a scene with the necessary three-dimensional models, camera and illuminating objects. Secondly, the rendering set-up and the sequence of operations involved in the image rendering process are demonstrated in Section 3.4.2. Finally, the last part which is Section 3.4.3 will discuss the strategy for extracting the annotations (i.e., predefined keypoints and 2D bounding box) for the rendered image.

The file `simulator.py` is designed to request two key user inputs of the rendering operation to be performed, one is the name of the spacecraft and the other regards to the type of background needed. A simple schematic layout of the POSSE is presented in the Figure 3.6. The code flow in the file is organised in such a way that the input parameters whilst declaring the user-defined functions, are the sole source to control the operational set-up of the simulator.

[3]Downloaded from: `https://download.blender.org/release/Blender3.0/`
[4]The license terms can be referred at `https://www.gnu.org/licenses/gpl-3.0.html`
[5]Blender 3.0 Python API Documentation: `https://docs.blender.org/api/3.0/`

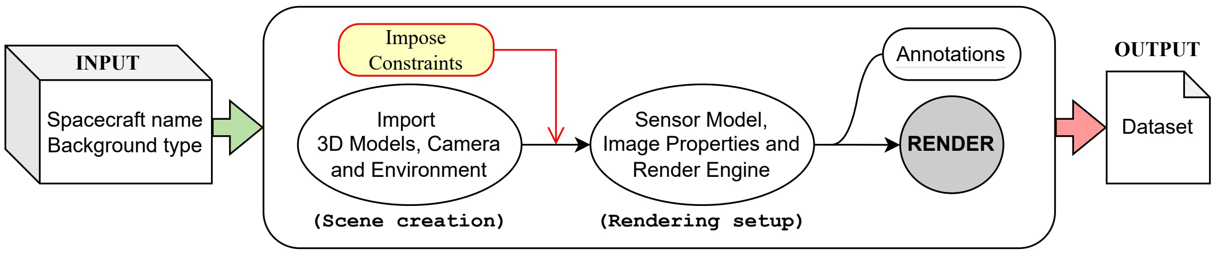


Figure 3.6: POSSE Framework

3.4.1. Scene

Generically speaking, a 'scene' is described as an anticipated circumstance in which the relevant elements are organized such that the event can take place. Whereas, in our scenario, the organisation of such elements in the event, typically address the following

1. The position and orientation of the traceable objects relative to the camera, and also related to the geometrical and materialistic properties of the scene objects themselves.
2. Lighting conditions are defined by the type of light sources in the scene and their characteristics such as colour, power and intensity of the light.
3. and the Environment in which the scene is defined, which usually indicates the objects involved in the foreground/background, which represents the Earth in our case.

Blender allows to arrange the main scene elements in the *.blend* file into multiple scenes *collections*, that can either store the entire set or a subset of the scene elements. As regards to such a feature: the meshes, cameras and light sources are organised with the name of *sc_ collection* as in Figure 3.7 in the main scene. All the elements in the scene *collection* are stored as libraries in the *.blend* file, which is similar to the folders on a PC.

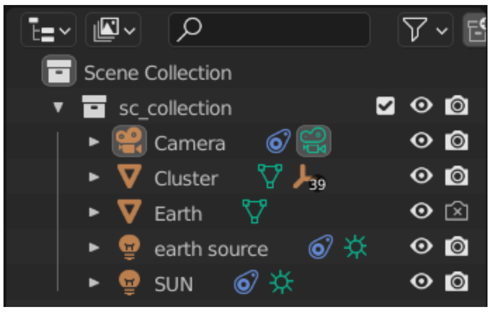

Scene Collection
sc_collection
Camera
Cluster
Earth
earth source
SUN

Figure 3.7: Scene collection

To understand better, it is preferred to break down the elements in the *sc_ collection* one after the other, along with their necessary characteristics and sub-elements as follows.

Spacecraft

As previously mentioned in Section 3.1.1, the downloaded spacecraft models are imported into the Blender graphical interface. The one-time edits are performed on the raw model; of-which includes 3D modelling, shading and texture editing for the materials. The idea is to create a *'spacecraft_name'.blend* file with single Blender *Object*[6] consisting of the spacecraft mesh geometry along with their respective materials. Concerning the geometrical editing, *Edit Mode* contain most of the required tools for 3D modelling; Aqua desires no modifications in the raw model resulting in Figure 3.1b, whereas for Cluster as discussed in the Section 3.1.1 it is modified by slicing the large wire-boom edges and carrying out the *join* operation to form a single mesh geometry. Later on, with regard to the materials, all are textured from the high resolution images with an exception to the Cluster's solar panels. For Cluster, solar panels are set out as a potential feature in the image processing step. The solar panels in the raw file are poorly textured as shown in Figure 3.8a. To overcome this, another feature of Blender is used that is *Shader Editor*[7]. In this way a node tree is built, by which the UV map of the panel geometry is utilized to generate the procedural pattern of Voronoi Texture over the surface such that the texture regions resemble the solar cells as in the Figure 3.8b. On the other hand, Aqua is textured ideally with an aluminium sheet, shielded entirely over the surface of the main body.

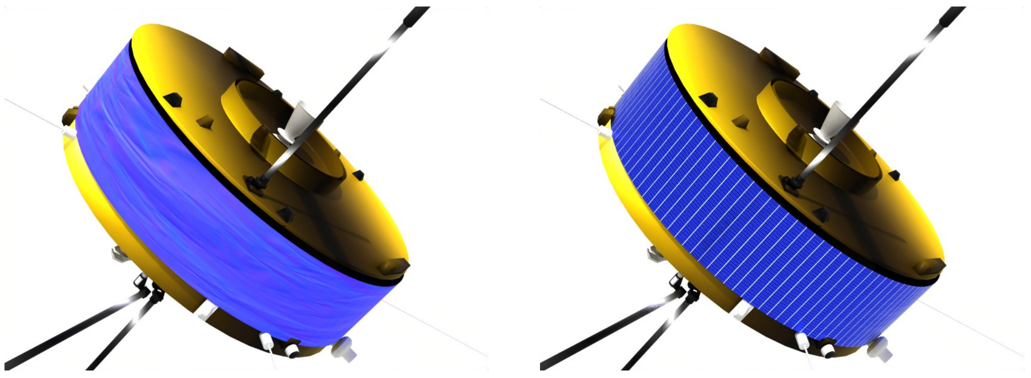

(a) Original solar panel. **(b)** Modified solar panel.

Figure 3.8: Cluster solar panel.

Finally, the modified version of the user-desired spacecraft, based on the keyword input is loaded from the blender libraries of the *'spacecraft_name'.blend* onto the main scene i.e., *sc_collection* as a single mesh *Object*. The spacecraft rotation mode is adjusted from 'XYZ Euler' to 'Quaternion (WXYZ)' which indicates the scalar first quaternion.

[6]https://docs.blender.org/manual/en/latest/scene_layout/object/introduction.html
[7]https://docs.blender.org/manual/en/latest/editors/shader_editor.html

The origin point of the Cartesian world is aligned with the CoM of the spacecraft by performing the translation operation between world origin and CoM.

Camera

The main scene perspective-camera is created both as a sensor and an *Object*, and placed at the origin initially with Quaternion as their rotational parameter; with the start of simulation the camera location is updated accordingly. Local axis of the blender camera follows an unconventional way from the typical pinhole camera model, with the negative z-axis representing the optical-axis as in Figure 3.9. The camera sensor properties are defined as per the model described in Table 3.5; another important property of the sensor to render the actual scale model of the Earth is the camera *clipping* parameter such that, the *clip_ start* and *clip_ end* are set to a value of 0.1122 and an astronomical unit respectively. The values are defined based on the various simulation trails and the maximum unit involved in the scene.

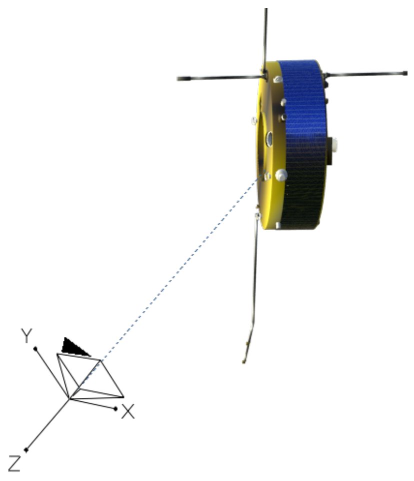


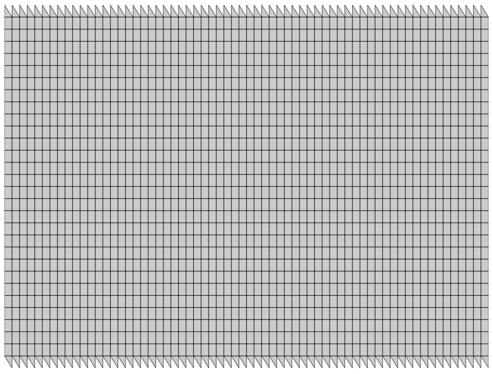
Figure 3.9: Third-person Point-of-View of the Camera and Cluster.

Sunlight

Sun is considered as a light source alone in the scene. This assumption could be inaccurate since the sun is not a mesh object that geometrically appears in the render. Apart from that, the light properties such as sun color with the RGB value of (0.9922, 0.9843, 0.8275) on the scale of 1, and the angular diameter of the Sun seen from Earth as 0.0092 radians are noted. In addition, the location and intensity flux of the sun are real to its units, and are constantly updated from the propagator as discussed in Section 3.2.3.

Earth

Unlike the approaches followed in the literature [3, 22] either by randomising the Earth images or placing a high textured polygonal sphere in the scene background, in POSSE, the Earth model is built with the lowest-minimum number of UV coordinates possible (i.e., 7552 points) for the UV sphere, of-which the image texturing method is possible without producing distorted views on an actual scale. The UV layout utilized is shown in Figure 3.10a; another reason for using fewer points is to reduce the computational effort whilst creating the scene. The realistic views of the Earth are produced with *Shader Editor*; High-resolution images are taken from NASA [51] in which the colour (Albedo) map is diffused with the land-bump map resulting in Earth's surface. Later, this layer is masked with the transparent cloud bump along with the ocean mask consisting Fresnel refraction index of 1.3. A final layer of atmospheric ozone is included with a transparent blue colour resulting in the Earth model as in Figure 3.10b. The Earth model is appended to the scene and rotated at a rate of 15° per hour during the simulation. The Section 3.2.3 discusses the radiation flux contribution from Albedo, which implies the Earth to be modelled as another light source in the scene pointing towards the spacecraft, that shares similar properties with the Sun.


(a) UV Map layout.

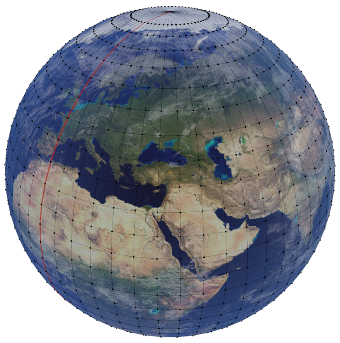

(b) Background Earth model.

Figure 3.10: Earth.

Starry-world

The term 'World' defines the environment of the scene itself and holds rigid in space. The usual Space environment contains celestial bodies including stars all the time. Henceforth, the stars are designed with the *Shader Editor* again and classified into three sizes: faint, small and medium. Each sized star is further classified into three common colours based on appearance that are orange, yellow and white as seen in Figure 3.11. The Voronoi Texture node is used to randomise the vertices in 3D space based on the Euclidean distances. The node tree is programmed in a way that, perhaps the size of the stars could be scaled by changing the functional input parameter. For the simulation, the scale is set to the value

Figure 3.11: Starry background.

of 20, 15 and 10. Likewise, the strength of the stars is adjusted accordingly in the range between 0-4 with no bloom effect.

The discussion up to this point clarifies the elements of the scene, especially about two things: what are the elements and how they are made. However, there is no evidence in the `propagator_script.m` model of Section 3.2, demonstrating the attitude of the camera that points to the target spacecraft. The reason is to avoid the round-off uncertainty during the reference frame transformation and importing the data into the simulator. The unknown parameter is resolved by extracting the camera's pose directly from the rendering event, thanks to the *Object Constraint Properties*.

The *track_ to* constraint demonstrates the possibility of tracking another object relative to the camera or the light source, which is controlled by the *influence* parameter such that the value of 1 resembles the spacecraft CoM point precisely at the centre of the image plane, but in reality, this fails. Therefore, a random value between 0.98 to 1 is set out as an influencing parameter such that, the spacecraft is not always at the centre. In addition, the same constraint is once again imposed on the light sources along with the eclipse switch to turn off the illumination at the eclipse event.

3.4.2. Image rendering

Image rendering is defined as a process to generate photo-realistic images. This process is mainly categorized into two: real-time rendering and pre-rendering based on their generation time. The former process employs the rasterization method, which works on approximating the 3D effects in the scene to the 2D image plane as seen from the viewpoint. This is highly used for interactive graphics and video games. Whereas, the latter uses a computational technique named Ray-tracing that is capable of simulating various optical effects such as reflection, refraction, shadows and occlusion. The sampling of light from the illuminating sources over the target domain, and capturing the reflected photons over the scene objects into the image plane explain the working principle. As one can imagine the cost of such an operation would be proportional to the amount of transporting rays in question from the source, most of the rays do not even intersect the object in the target domain or reach the image plane.

Henceforth, the principle of backward Ray-tracing is introduced in which the rays are optimised and transported from the imaging sensor to the target domain as shown in Figure 3.12, in which only the rays intersected with the scene object's surface are further evaluated over the incoming light bounces and its resulting shadows at that instance to calculate the final colour of the image pixel. In this way, the rendering operation is better optimised whilst producing an image with greater photo-realism accuracy, of course at a higher cost when compared to the real-time rendering.

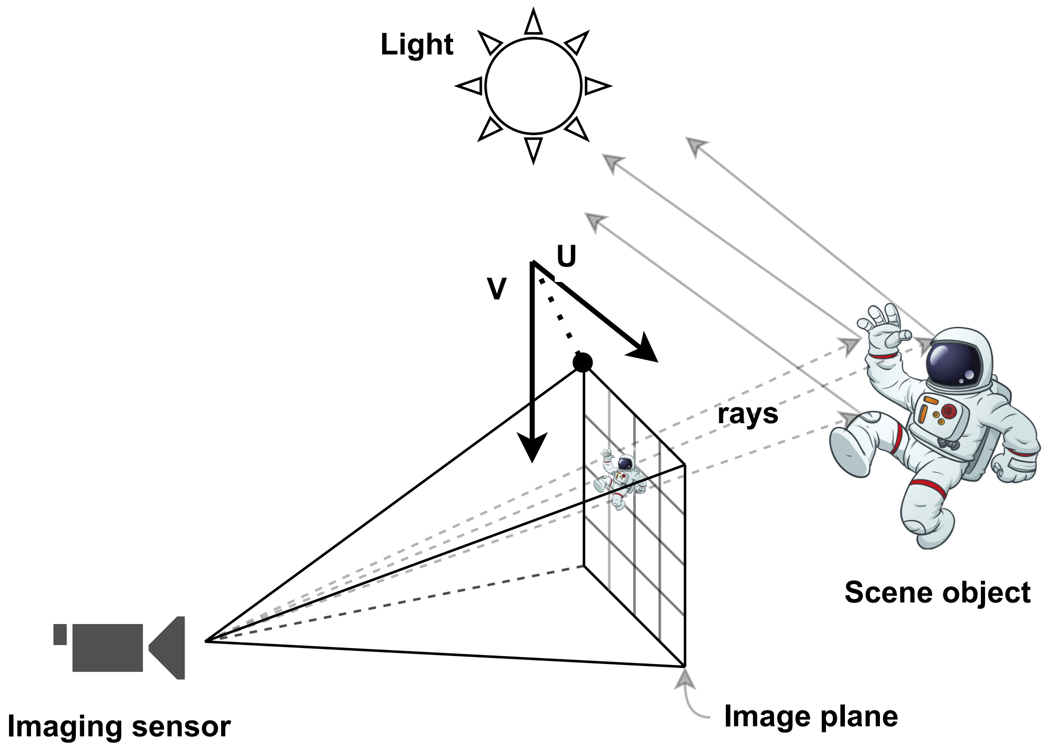


Figure 3.12: Backward Ray Tracing

In Blender, such processes are available as a part of *Render Engine*[8], of which *Cycles* that work on backward ray tracing technique is preferred for the dataset generation due to its high accurate rendering capabilities. The Google Colab is used for accelerated rendering with the GPUs available on the platform. Rendering setup properties are modifiable in the function `render_engine`, among them 'samples' is one key parameter that defines the amount of traceable rays per pixel to be evaluated by the integrator; a decent value of 200 samples are chosen for this operation. The function also accounts for maximum light bounces, clamping, and caustic parameters with an option to induce a Gaussian filter for rendering. Similarly, the output image properties are defined by the function `render_params`.

Lastly, the image rendering pipeline is constructed to generate the final spacecraft dataset with the selected background (i.e., each spacecraft will have two sets of datasets: one is dark starry and the other with Earth appended in the background). The output file `points.csv`, from the Section 3.2 provides necessary input data for setting up the scene objects and render conditions. A flowchart representing the operational sequence of the

[8]Introduction: `https://docs.blender.org/manual/en/latest/render/introduction.html`

pipeline is presented in Figure 3.13. The resulting location and quaternion of the camera at each frame are noted in a '`spacecraft_name + background + pose_data.csv`' file.

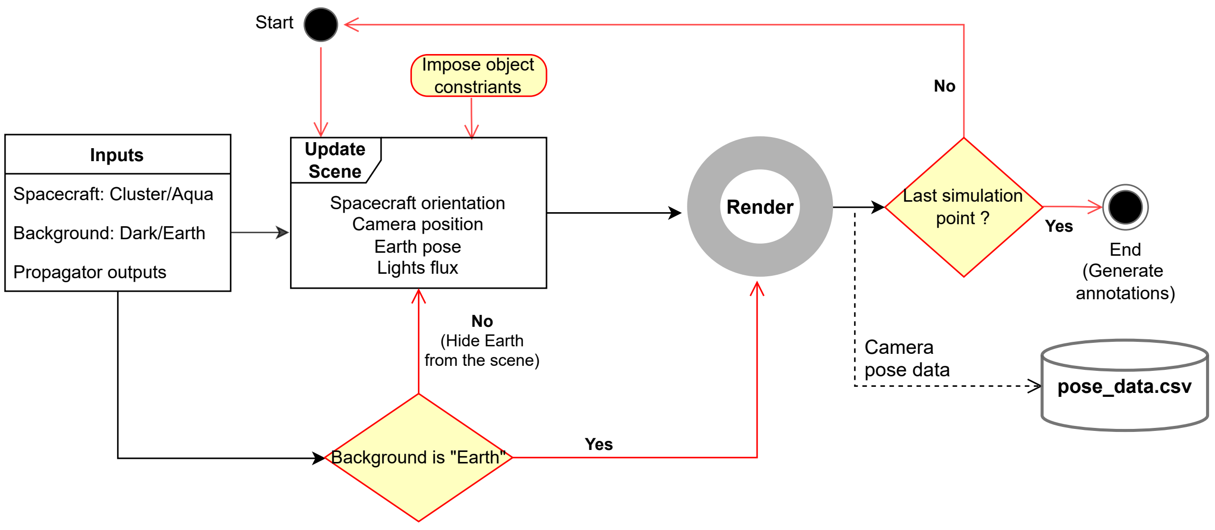


Figure 3.13: Image rendering pipeline

3.4.3. Annotations

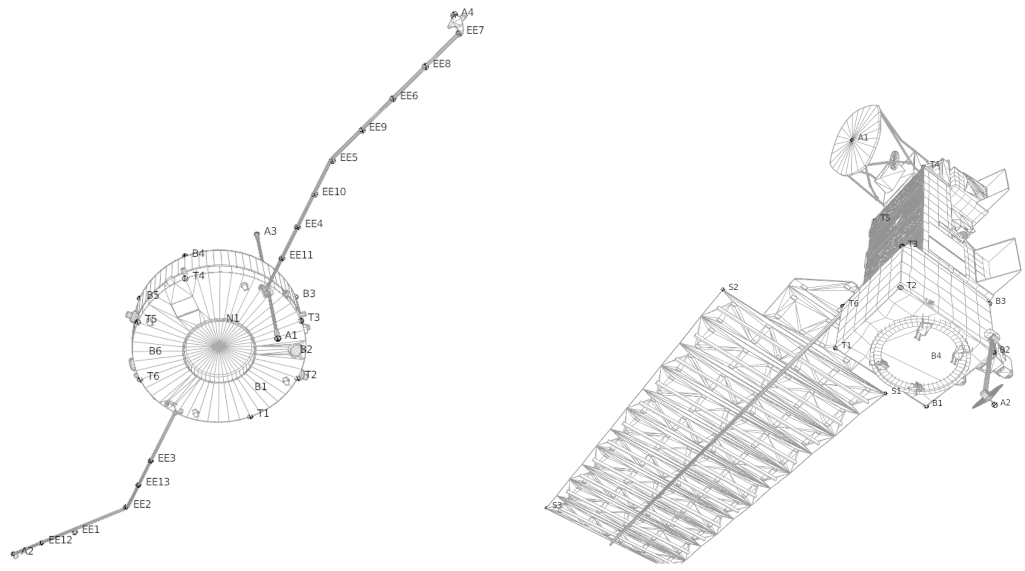
The labelled dataset must contain the synthetic images along with the encoded ground truth data. In general, this data is commonly referred to as image coordinates consisting of information on the segmentation maps, keypoints, boundary box, etc.; these are collectively termed "Annotations". The supervised learning approach requires the annotations to train and validate the neural model. This section is mainly focused on the generation of these annotations (i.e., bounding box, keypoints and pose data). However, the keypoints and pose data are not required for the Spacecraft Detection Network in Chapter 4, but they contribute to the strategy for generating bounding box annotations. Moreover, the annotations are generated by keeping in mind the applicability of this work to perform keypoint extraction and pose estimation in the future.

Category	Prefix
Main body (top)	T
Main body (bottom)	B
Antenna	A
Solar panel	S
Extra Empties	EE

Table 3.6: Categories of labelled keypoints

As regards the study, keypoints are identified as the labelled pre-defined points on the 3D object as shown in Figure 3.14, that may resemble any of the edges or faces. Annotations in the present dataset generation process include only the coordinates of these keypoints and the boundary box derived from the respective keypoints in the image. The keypoints are categorised as in Table 3.6 based on the generically possible spacecraft components i.e., main body, antenna, solar panel and nozzle.

The annotations are primarily stored with the use of *.json* file format, into two dictionaries: '*keypoints*' and '*bounding_ box*', and each dictionary contain another sub-dictionary with a keyword (i.e., category 'prefix' code as in Table 3.6, followed by a sequential number) representing the keypoint. By utilizing the concept of *Empties*[9], the predefined points are modelled as the one-time edits on the final model and saved as *'space-craft_ name'_ annotate.blend.* The 3D positional data of these *empties* are recorded with the function *object_ coordinates* and exported to the `annotations_raw.json`, which is put along with their respective *.blend* files. Whereas, the image-specific annotations of the *empties* are also registered to '`spacecraft_name+background+annotations_w.json`', irrespective of their visibility from the viewpoint during the rendering pipeline execution.



(a) Cluster. **(b)** Aqua.

Figure 3.14: Labelled keypoints.

Later, the keypoint visibility is considered and validated for all the rendered images with the use of blender's internal ray casting technique[10], resulting in the 3D hit point location and a logical value indicating the occurrence of the hit. The *error* (i.e., $keypoint_{actual}$ −

[9]The "empty" is a single coordinate point with no additional geometry (i.e., no volume and surface).
[10]The projection of rays into the viewing volume from the camera, and the particles that hit the first identified mesh validates the keypoint visibility.

$keypoint_{ray\_cast}$) is computed and a tolerance of 0.055 m is applied for the $||error||$, just to avoid the false negatives that occur when the face hits first although the point is right below.

Finally, the *point_ to_ pixel* function is built to solve the *Perspective projection* problem as demonstrated in Section 2.6.1, which transforms the 3D keypoint coordinates in the scene to a 2D-pixel coordinate on the image plane. The dictionary value of the keypoints in the annotations file indicates the (u_{coord}, v_{coord}, *visibility*). The bounding box is defined by two anchor points that is one representing the top-left and the other indicating the bottom-right of the box. Anchor points are computed based on the minimum and maximum pixel coordinate value of the visible keypoints along the UV-axes (i.e., left anchor $= (V_{max}, U_{min})$. From this, it is clear that the presence of visible keypoints is significant to determine the anchor points. Keypoints over the extreme corners of the spacecraft components are not good enough to precisely design the bounding box. Therefore, redundant 'Extra Empties (EE)' are densely populated (see Figure 3.14); such that they would assist in computing the anchor points and are further discarded in the annotations. The dataset annotations are stored in a file '`spacecraft_name+background+annotations.json`.

3.5. POSSE Dataset

All the previous sections clearly explain the process of generating the synthetic images. To this point, four individual datasets are Cluster-Dark, Cluster-Earth, Aqua-Dark, and Aqua-Earth based on the user choice, of which each consists of $(N + 1) \times H$ images. The `generator.py` will collect the individual datasets including the annotations, and generate a single multi-spacecraft dataset named 'POSSE Dataset'. The code will evaluate the dataset and eliminate the redundant high-contrast images that occur due to the following scenario through image processing.

1. Target spacecraft is in eclipse with the light source by the Earth.

2. Camera is in eclipse with the light source by the Earth/Target spacecraft.

In addition, it is possible to add the Gaussian noise and filtering to the images with `generator.py`, if the *variance* and *std deviation* parameters are known. Lastly, the code will generate two files: `dataset.csv` and `annotations.json`, in which the former contains the *filename*, *spacecraft name* and the camera *pose_ data*. Whereas, the latter holds the ground truth data of the *keypoints* and the *bounding box*.

3.5.1. Re-partitioning

The standard procedure of training a Neural Network is to perform dataset partitions, that are usually called 'train', 'valid' and 'test' datasets. To perform this operation, the `partition.py` is utilized at which the *image filenames* and the *pose_ data* of the dataset are randomly shuffled and distributed into six data files with suffix `images.csv` and `pose.csv` for each subset. The partition ratio is 80, 10 and 10 percent indicating *train*, *valid* and *test* datasets respectively.

4 | Spacecraft detection

This chapter discusses the network proposed for the detection of the spacecraft in the multi-class datasets consisting of grayscale images. From the SOTA and the KPEC outcomes mentioned in Chapter 1, it is clear that Convolution Neural Networks are well-advanced in such detections. These neural models are carefully considered in the proposed network named Spacecraft Detection Network (SDN). Similar to the dataset generation algorithm introduced in Chapter 3, the SDN is also coded in Python using Keras, which is a high-level API of the TensorFlow platform for model development.

The schema diagram shown in Figure 4.1 illustrates a simplified SDN architecture highlighting its key components. To outline the architecture, SDN's backbone consists of ResNet architectures discussed in Section 2.4.4 along with pre-trained weights learned on the ImageNet dataset. This is further extended with the Feature Pyramid Networks (see Section 2.5.2) to achieve multi-scale invariance. Later, the proposals are generated by another simple convolution-based Region Proposal Network that is followed by the two fully connected layers to predict the *classification* and the *bounding box* regression. The development of the SDN model and its training are detailed in the following sections.

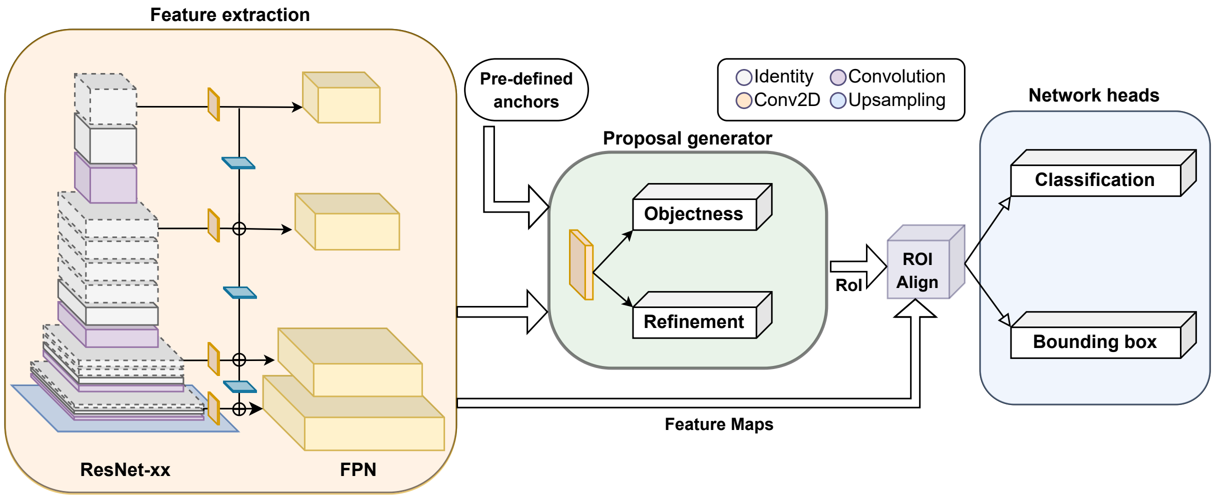


Figure 4.1: End-to-End Architecture: Spacecraft Detection Network

Spacecraft detection network

At first, it is thought to propose a simple unified network similar to Proença and Gao [3]. Later this baseline network was enhanced to perform RoI checks such that, a reduction in search regions eventually reduces the computation power on unwanted regions along with the advantage of detecting multiple objects in the input image. For this, the two-stage detectors are agreeable for the reasons stated at the end of Section 2.5. Among the two-stage detectors discussed in Section 2.5, Mask R-CNN is the latest method which modifies the Faster R-CNN by including the Feature Pyramid Networks (FPN).

Therefore, the modified ResNet-50 backbone in [3] is extended with the FPN (see [42] and Section 2.5.2). The fully-connected layer and the global average pooling layers are removed from the baseline's CNN backbone. It is also evident that this backbone setup, which is also implemented in the Faster R-CNN, performed better with FPN in its architecture with a positive increment on AP by 7.6 points. Hence, ResNet-xx with FPN is considered as the backbone for SDN.

Furthermore, the potential candidate regions are generated by the Region Proposal Network (RPN) model discussed in Section 2.5.1. With this, implementing the SDN on our novel dataset (i.e., POSSE) requires certain quantitative analysis to be performed, to decide the RPN's adaptability for the study. For this reason, the RPN's detection accuracy on the candidate regions' proposal is validated. On this account, similar to the ablation experiment in [42] is performed in which, the COCO-style Average Recall (AR) for the top 100 and 1000 proposals at a set IoU threshold of 0.5 is evaluated as a performance parameter. The implementation details are shared in the following Section 5.2 in Table 5.2. As a result, the RPN with NMS filtering performed well with an $AR_{0.5}^{100}$ and $AR_{0.5}^{1k}$ of 98.42% and 100%.

Overall, to describe the proposed SDN, the backbone module consists of both ResNet-18 and ResNet-50 architecture for feature extraction depending on the complexity of the dataset. In the Figure 4.1, the bottom-up pyramid represents a blend of both ResNet architectures where the solids and dotted blocks resemble ResNet-18 and ResNet-50 respectively. The decision to change the backbone between the above two architectures is due to the lesser amount of training samples available when compared to the other datasets used in the baselines. In addition, this step improves the performance, and computation efficiency with fewer trainable parameters and also prevents the overfitting problem. Along with the backbone, the SDN is later extended to employ FPN to perform scale-invariant object detection. Once the Region Proposal Networks (RPN) generate proposals for the candidate regions, the feature maps are hand-picked from certain levels of the FPN pyramid as explained in Section 2.5, based on the area of the proposed candidate regions (i.e., RoI). Further, the FPN maps over these regions are cropped, aligned and fed to the network heads module with fully connected layers to classify the spacecraft and regress its bounding box.

The size of the SDN's input image is also another pivotal requirement because the rendered grayscale images of Cluster and Aqua in the POSSE dataset have vertical and horizontal dimensions of 960×1280 pixels. The baselines require the shorter side to be ≈ 600 pixels. Hence, the input is resized to match the requirements. Another constraint on the image size is that it must be divided by 2 at least 6 times to avoid fractions. Therefore, the dimension of the pipeline's input image is set to 640×640 pixels. Along with this a unique mean pixel value computed on the entire images in the dataset is removed from the image as a data pre-processing step known as mean normalization.

In total, the key modifications over the typical baseline architecture are the following: At first, the CNN's inputs such as image dimensions are set to 640×640 pixels and the higher-level stages (i.e., $c5$ and $p6$ stages) are included. Secondly, the batch normalization layers are trained conditionally, only when the batch size > 32. Lastly, the loss functions strategy on model convergence is modified such that, even the training can be accurate with a lower batch size. The problem related to the model convergence is explained below.

Model training

Similar to the Faster R-CNN, our object detection model also outputs four predictions from the network heads: two from the RPN (i.e., *rpn_ class*, *rpn_ bbox*) and, two for the spacecraft classification (*class*) and the bounding box (*bbox*). To discuss the strategy for building the prediction targets, at first, the targets for the branches *rpn_ class* and *rpn_ bbox* strictly follow the approach mentioned in the baseline Ren et al. [36] (see Sections 3.1.2 and 3.1.3).

In a nutshell, for *rpn_ class* target, each anchor box is labelled as positive, negative or neutral. The anchor box of the highest IoU overlap or IoU ≥ 0.7 with a ground-truth box is considered positive, and the anchor box is assigned negative when the IoU < 0.3. Optimizing all anchors will be biased towards negative samples as they are dominant. Therefore, only N_{rpn} anchors are trained per image, of which the ratio of positive and negative samples is 1:1. The excess positive or negative samples are marked as neutral samples, and are ignored in loss computation.

The binary cross entropy loss function is used to compute *rpn_ class_ loss*, where only the positive and negative samples are considered. Similarly, the *rpn_ bbox_ loss* utilizes robust loss function (i.e., smooth L1 a.k.a Huber loss with $\delta = 1$) to perform regression of the four coordinate vector $t = \begin{bmatrix} t_y & t_y & t_h & t_w \end{bmatrix}$,

$$\begin{aligned} t_y &= \frac{Y - y_a}{h_a} & t_x &= \frac{X - x_a}{w_a} \\ t_h &= \log\left(\frac{H}{h_a}\right) & t_w &= \log\left(\frac{W}{w_a}\right) \end{aligned} \tag{4.1}$$

where, based on the choice of computation (i.e., target/prediction vector), the (Y, X) and (H, W) denote the centre coordinates and height/width of the ground truth/estimated box. Similarly, the subscript 'a' denotes the corresponding dimensions of the anchor box.

Secondly, for building the *class* and *bbox* targets, the $\sim$ 2000 rois from the proposal generator are labelled as positive and negative rois in the paper [36, 37]. These are classified based on the IoU overlap with the ground truth such that, IoU ≥ 0.5 are positive and < 0.5 are negative. The *bbox_ loss* follows the same approach as *rpn_ bbox_ loss* computation by evaluating positive rois alone. Whereas, the target class ids of the negative rois are assigned as zeros. Similar to the RPN training, here only N_{odn} rois are trained per image. The ratio of positive to negative rois is 1:3, in case of shortcomings, the targets are zero-padded to preserve the tensor dimension.

In [35–37] *class_ loss* computation is a two-step strategy. First, the sparse softmax cross entropy loss function is used between the *class* target labels and predicted logits, and later, the prediction losses associated with the inactive classes of the dataset, and in the image are erased. As a consequence, the loss function is prone towards prediction errors, which eventually happen either in training with a smaller dataset or at the initial steps/epochs. For instance, when all the predictions over the image are inaccurate and belong to the inactive classes, the correctly penalized losses from the first step are ignored in the second. In the end, the prediction losses mean, which accounts for the active predictions alone, results in *nan*. As a result, the false predictions are not penalized correctly and the weights do not optimize at the step-end.

To overcome this limitation, the strategy is slightly modified such that, the losses of both positive and negative rois are considered, as in *rpn_ class_ loss* computation, since the rois with IoU < 0.5 implies the background. To distinguish the zero-padded and true negative roi targets, the class ids of the negative rois are assigned to the background class, and the tensor is padded with '-2' instead of zeros. Later, the losses associated with inactive target classes are removed, and the background is considered active. Thus, false predictions are accurately penalized and this strategy proved to be efficient in our study.

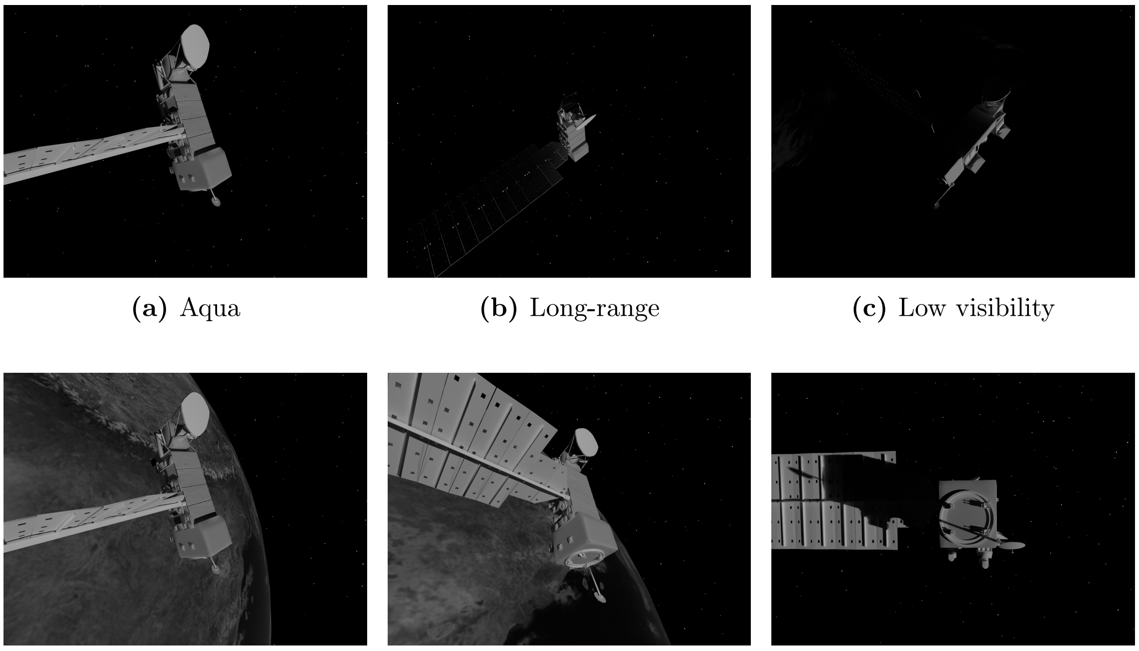
5 | Results

This chapter discusses the performance of the tools implemented in this research (i.e., Chapters 3 and 4). To present these results in an orderly manner, it is thought to break down the chapter into two sections. At first, the Section 5.1 discusses the results from the image rendering pipeline, and later comes the discussion on the results achieved through the Spacecraft Detection Network in Section 5.2.

5.1. Spacecraft renders

The propagator described in the Section 3.2 is set to generate the pose data, for the objects in the Blender scene. The arbitrary value for the number of instances N is set to 100, at which the spacecraft's orbit is propagated. Later, the number of camera's perspective views (i.e., H) recorded per instance is 25. Therefore, this results in 2525 or $(N+1) \times H$ frames/images in one run.

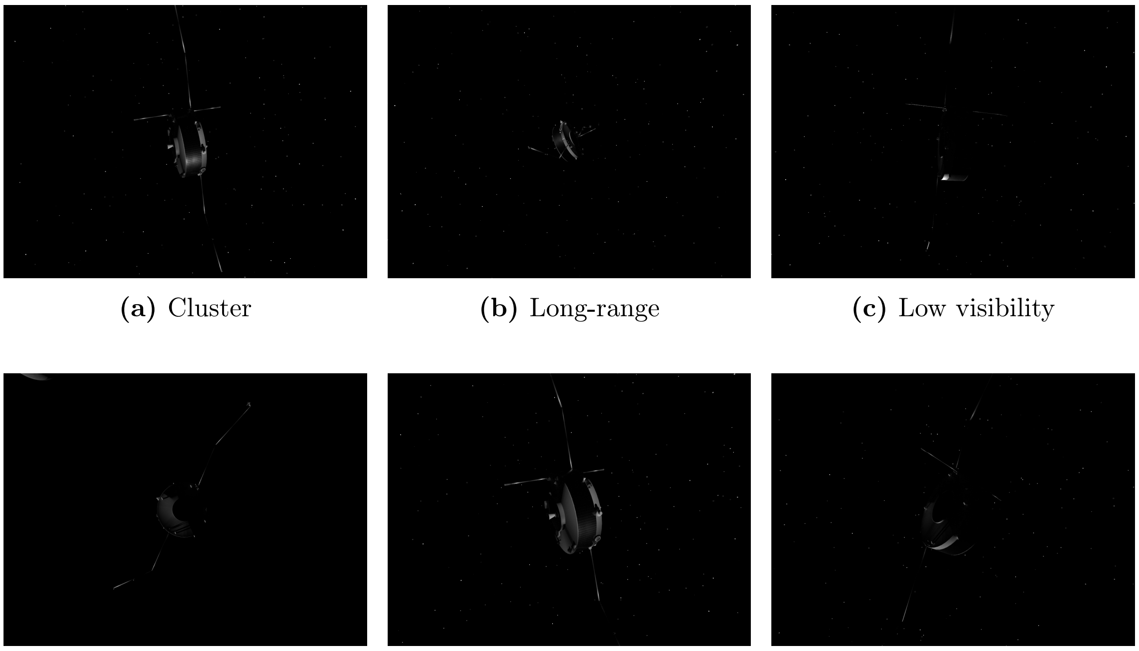
The current state-of-the-art datasets [22, 23] lack the noisy stars in the background. POSSE dataset is introduced also to include challenging conditions for the DL techniques to estimate the pose, and having a starry background is one of them. The decision on the presence of Earth in the background is also considered a challenge of robustness, henceforth, the selection of an image background type (i.e., w/ Earth or w/o Earth) is set as a simulator's input parameter. For instance, in the first run, the spacecraft images are rendered with Earth, and in the following run without it. So far, the POSSE include a starry background in all 4 variants, which are, Cluster w/o Earth, Cluster w/ Earth, Aqua w/o Earth and Aqua w/ Earth. In total, by including m no. of spacecraft and p background variants in the dataset, there exist $m \times p \times ((N+1) \times H)$ images in the dataset. For us, $m = 2$ (i.e., Cluster and Aqua) and $p = 2$ (i.e., w/ Earth and w/o Earth), which results in 10100 images in the POSSE dataset. Later, the images that are pitch-dark and consist of less than three visible keypoints are filtered out from the POSSE dataset, because they do not contribute to the model training and the final pose estimation, therefore, resulting in 8186 images. A few sample images of Aqua and Cluster, as a result of the image rendering pipeline execution, are shown in Figures 5.1 and 5.2.



(a) Aqua **(b)** Long-range **(c)** Low visibility

(d) Earth background **(e)** Partial view at close-range **(f)** Strong shadow regions

Figure 5.1: Sample rendered views of Aqua in challenging conditions with applied Gaussian noise and blurring



(a) Cluster **(b)** Long-range **(c)** Low visibility

(d) Earth background **(e)** Partial view at close-range **(f)** Strong shadow regions

Figure 5.2: Sample rendered views of Cluster in challenging conditions with applied Gaussian noise and blurring

In addition to the main dataset POSSE, it is intended also to render the scene without a starry background for validating the robustness and performance of the pose estimation in comparison. For this purpose, the POSSE dataset is further subdivided into 4 versions, which are detailed in the following Section 5.2. The Figure 5.3 shows the render of the Cluster and Aqua spacecraft with a dark background.

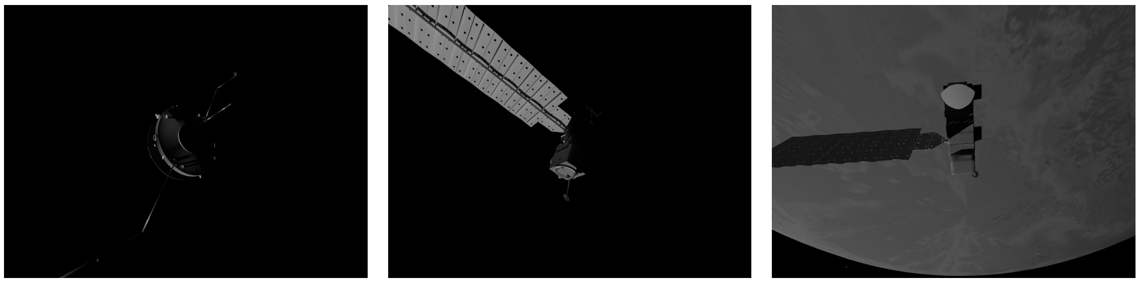

(a) Cluster (b) Aqua (c) Aqua + Earth background

Figure 5.3: Images from POSSE dataset without starry background, and applied Gaussian noise and blurring

5.2. Multi spacecraft detection

Over the decoupled approach, it has been proven by researchers that the accurate prediction of high-confidence keypoints, dictates the pose measurement accuracy, with an exception in the case of symmetric ambiguities. Henceforth, the research focuses on delivering a detection network that can assist further in high-confidence keypoint extraction and pose estimation. On this basis, the proposed SDN described in Chapter 4 is set to demonstrate the robustness of our network over the SOTA datasets, it is thought to perform five experiments in which all five are set to detect and classify the given spacecraft at varying complexities with regard to the background. The first two experiments are on a single-instance and single-class problem performed over Cluster and Aqua separately, at a very simplified level with a dark background. The main focus is to test the performance of the proposed network in the multi-classification task over the results achieved in the SOTA detection methods.

Later, the following two experiments will perform a single-instance, multi-spacecraft problem with and without the presence of Earth in the background. Although the robustness with respect to Earth has already been published, this study brings novelty in validating the CNNs in the spacecraft domain for solving a multi-classification problem, where the neural network is trained and tested with the dataset consisting of both Cluster and Aqua.

The final experiment is over the POSSE dataset, which consists of Cluster, Aqua, Earth and Stars in the image. These experiments are performed on Google Colab. The comparison between the last two experiments will depict the impact of the noisy stars in the background, and implicitly the robustness of the proposed network on the multi-spacecraft detection, since all the above-mentioned experiments require dedicated datasets to train and test the proposed network. These datasets are created by partitioning the main POSSE dataset in the case-by-case scenario. The details are tabulated in Table 5.1.

Dataset	Target	Background	#images
`POSSE_v1`	Cluster	Dark	2525
`POSSE_v2`	Aqua	Dark	1568
`POSSE_v3`	Cluster, Aqua	Dark	4093
`POSSE_v4`	Cluster, Aqua	Earth	4051
`POSSE`	Cluster, Aqua	Earth, Stars	8237

Table 5.1: Categorization of datasets for the experiments

An ablation experiment over the adaptability of transfer learning and RPN in our proposed network is carried out before performing the object detection task over any of the above-mentioned datasets. The reason is that the RPN manages to output the candidate regions and the post-processing module searches for the spacecraft in these regions alone to classify and regress the bounding box. What if the RPN proposals are inaccurate? Hence, at first, the network consisting of the backbone and proposal generator module (i.e., ResNet-18, FPN and RPN) is trained and tested on the `POSSE_v1` dataset. In addition, the study on the impact of NMS filtering in RPN is also carried out. The results are shown in Table 5.2.

The implementation details of the ablation experiment follow the paper [42] (see Section 5.1 of [42]), except for the batch size and learning rate. The adoption of ResNet-18 unveils the possibility of having a larger batch size computation, hence the batch size is increased from 16 to 48. Whereas the learning rate is slowed down and fixed at 0.001 instead of a varied value after certain mini-batches. This is because the training size is much smaller

Dataset	Filtering	ImageNet	$AR_{0.5}^{100}$	$AR_{0.5}^{1k}$	t_{GPU}	#params	FLOPs
`POSSE_v1`	Top-K	✓	83.4	98.8	186	14.98M	152.63B
		×	94.46	97.23	177	14.98M	152.63B
	Top-K, NMS	✓	91.7	100.0	185	14.98M	152.63B
		×	98.42	100.0	203	14.98M	152.63B

Table 5.2: Ablation experiment for RPN, Transfer learning adaptability

compared to the COCO dataset. Higher learning rates overshoot the optimal point when tested on the `POSSE_v1` dataset.

In the above Table 5.2, the $AR_{IoU}^{\#proposals}$ measures the ability of the model to detect objects irrespective of the class match. This is an aggregate metric that is calculated for the entire dataset. A value closer to 100 denotes the model accuracy to find objects and is computed on a per-image basis over the $\#proposals$ that fulfil the set IoU threshold condition.

The model is either fine-tuned or fully trained on the first two cases in the Table 5.2 (i.e., w/ and w/o adapting ImageNet weights) since the only add-on for the last two cases is the NMS filtering operation. The corresponding weights from the first two cases are adapted during the model inference for the third and fourth cases. It is observed that the results achieved in all four cases are coherent, as the results achieved by the fully trained method are relatively higher than the fine-tuning approach. Whereas, upgrading the network with an NMS filtering operation (i.e., Case 3 and 4) resulted in a relatively higher $AR_{IoU}^{\#proposals}$, and the computational cost for including the NMS is negligible. In addition, the NMS increased the $AR_{0.5}^{100}$, which keeps the best proposals at the top and filters the redundant.

5.2.1. Experiment on `POSSE_v1` and `POSSE_v2`

The `POSSE_v1` is a single instance dataset consisting of 2020 train, 252 val and 253 test images of Cluster. Training and testing are primarily performed on the `POSSE_v1` for Object detection. The pre-trained ImageNet weights are adapted for the backbone. For this, the SDN explained in Chapter 4 is trained for 60 epochs with the #anchors and #RoIs trained per image (i.e., N_{rpn}, N_{SDN}) are 256 and 512. The remaining setup of the hyperparameters follows the RPN ablation experiment from the previous section. Performance indicators for this single-class object detection task are evaluated at 10 different thresholds (i.e., $0.5 - 0.95$) as in Section 2.5.3, resulting in Table 5.3.

Method	TL	$AR_{0.5}^{100}$	$AR_{0.5}^{1k}$	mAR	$mAP_{0.5}^{0.95}$	t_{GPU}	#params	FLOPs
SDN	ImageNet	99.6	99.6	99.21	78.03	224	28.9M	152.63B

Table 5.3: Comparison of detection results on `POSSE_v1` test set with SDN and baseline

From the above table, the SDN has clearly shown better results with an $mAP@[.5, .95]$ and mAR of 78.03%, and 99.21% respectively. The precision-and-recall curves indicate the performance of the SDN model on `POSSE_v1` dataset. Following the concept introduced in Section 2.5.3, the 10 P-R curves corresponding to different IoU thresholds are plotted on the right in Figure 5.4. On the left, a few inference detection samples are shown, it can be seen that the SDN identifies the clear images perfectly. Whereas, the partial/less visible images are a little complex to overlap with the ground truth completely. Here, the reason for the high-precision thresholds such as $AP_{0.85}^{0.95}$ resulted in a relatively lower precision than $AP_{0.5}^{0.8}$, and it is anticipated due to the bounding box annotations' inaccuracy in the trained data.

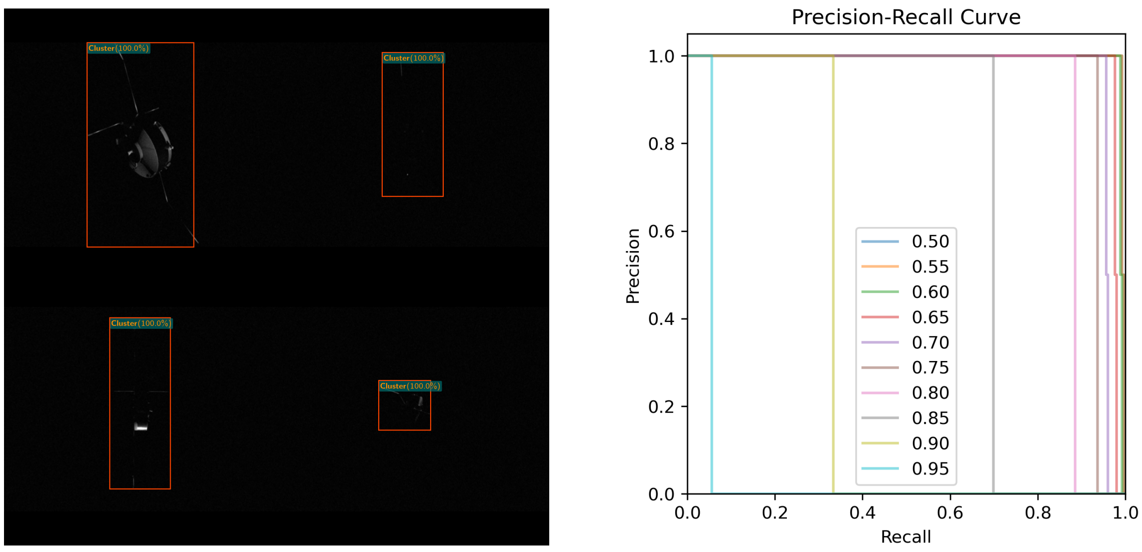


Figure 5.4: Performance prediction of SDN on `POSSE_v1`

Similarly, the SDN is trained and tested on another single instance `POSSE_v2` dataset with 1254 train, 157 val and 157 test images of Aqua. The pre-trained ImageNet weights are adopted for the backbone, and trained for 60 epochs with $N_{rpn} = 256$ and $N_{odn} = 512$. The hyperparameters follow the same as the above `POSSE_v1` experiment. Performance indicators for the detection of Aqua spacecraft are tabulated in Table 5.4.

Method	TL	$AR_{0.5}^{100}$	$AR_{0.5}^{1k}$	mAR	$mAP_{0.5}^{0.95}$	t_{GPU}	#params	FLOPs
SDN	ImageNet	98.72	98.72	96.17	60.93	222	28.9M	152.63B

Table 5.4: Detection results on `POSSE_v2` test set with SDN

Where the `POSSE_v2` experiment yielded slightly lower performance with $mAP@[.5, .95]$ and mAR of 60.93%, and 96.17% respectively. From the Figure 5.5, the high-precision losses observable on the right are again reasoned on the training annotations' inaccuracies. However, the recall performance remains closer to the previous experiment. The sample images from the `POSSE_v2` inference that are presented on the left in Figure 5.5, clearly represent the troubles in bounding box predictions for partial/less visible images.

To conclude the experiments on a single-instance and single-class category, upon reviewing the high-recall parameter (i.e., mAR), one can substantiate the effectiveness of the proposed SDN network in detecting the spacecraft over the dark background.

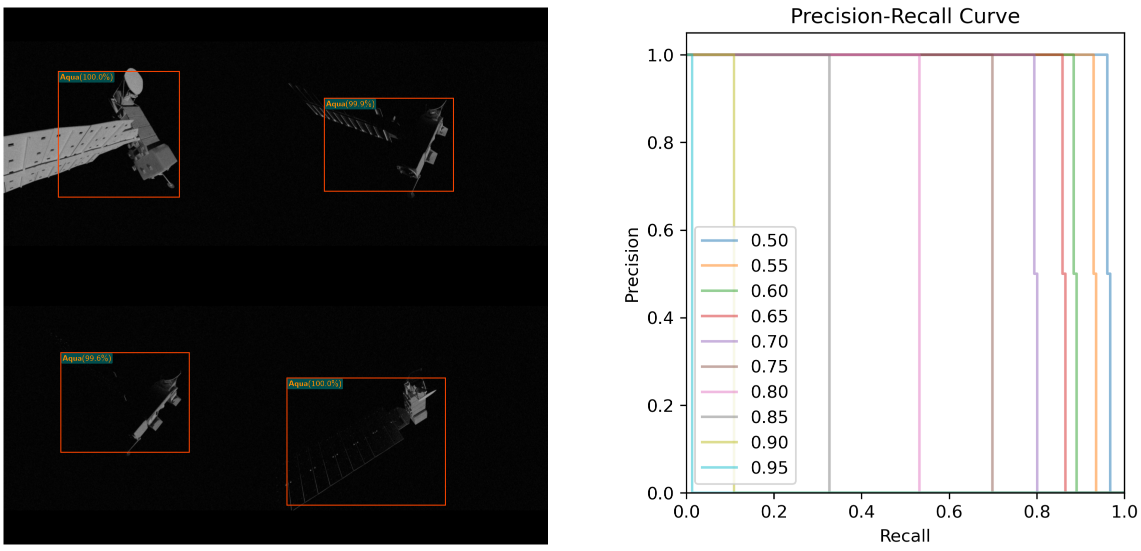


Figure 5.5: Performance prediction of ODN on `POSSE_v2`

In addition, to discuss the precision factor, which influences both the accurate class match and bounding box predictions, the SDN's performance with the ResNet18 architecture relies on the spacecraft's illumination conditions (whether it's clear or partially visible) and the inaccuracies present in the annotations of the trained dataset. Overall, the SDN network must be acknowledged for its inference time of approx. 223 ms (i.e., $\approx$ 5 FPS).

5.2.2. Experiment on `POSSE_v3` and `POSSE_v4`

The `POSSE_v3` and `POSSE_v4` datasets contain single-instance, multi-spacecraft images, and the experiments mentioned in Section 5.2.1 demonstrated their effectiveness for spacecraft detection. Therefore, it is expected that employing the identical SDN with ResNet18 will yield improved results on the `POSSE_v3` dataset with 3274 train, 410 val and 409 test images of both Cluster and Aqua. The hyperparameters are similar to the above two experiments, and the ImageNet weights are used to train the network for 60 epochs with $N_{rpn} = 256$ and $N_{odn} = 512$ resulting in Table 5.5. It is to be noted that the parameter mAR in the Table 5.5 corresponds to the mean recall value of Aqua (49.37%) and Cluster (98.4%). In Figure 5.6 (on the right), the 10 P-R curves corresponding to different IoU thresholds are plotted separately for each class, such that the plot demonstrates if the multi-class predictions are biased to a specific class in the detection task.

Method	TL	$AR_{0.5}^{100}$	$AR_{0.5}^{1k}$	mAR	$mAP_{0.5}^{0.95}$	t_{GPU}	#params	FLOPs
SDN	ImageNet	99.75	99.75	73.88	50.6	241	28.9M	152.63B

Table 5.5: Detection results on `POSSE_v3` test set with SDN

This plotting method is used to display all the results of the multi-classification problems in this paper. The mean precision metric (i.e., mAP) is notably reduced to 50.6%, and the predictions are greatly influenced to predict Cluster than Aqua. A few sample predictions are shown in Figure 5.6 (on the left), where even a clear image of visible Aqua spacecraft remains undetected whereas, a complex partial/less visible image of Cluster is detected perfectly by the SDN.

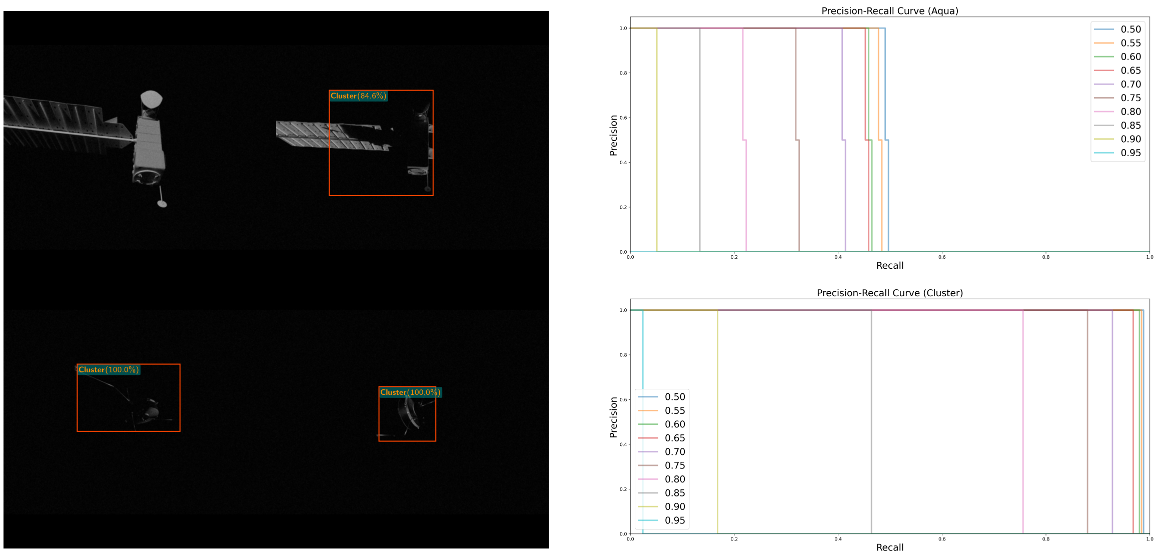


Figure 5.6: Performance prediction of SDN on `POSSE_v3`

That said, the very same SDN with ResNet18 is trained and tested on the `POSSE_v4` dataset with 3241 train, 405 val and 405 test images for 60 epochs. In these images, the detection complexity is increased by adding Earth in the background. Consequently, the metrics are substantially diminished to result in $mAP@[.5, .95]$ and mAR of 12.31%, and 26.08% respectively. In this case, it is evident that the increase in image complexity directly impacts the network's performance. However, to ensure the previous statement is valid, the `POSSE_v4` dataset is again trained and tested for 60 epochs by enhancing the SDN's backbone from ResNet18 to ResNet32, which resulted in $mAP@[.5, .95]$ and mAR of 11.74%, and 35.66%. To overcome the poor results, the network is enhanced further from ResNet32 to ResNet50 and the training is extended from 60 to 100 epochs. Overall, this resulted in slightly better performance with $mAP@[.5, .95]$ and mAR of 33.21%, and 73.96% respectively as reported in Table 5.6. In addition, there has been an adverse effect

Method	TL	$AR_{0.5}^{100}$	$AR_{0.5}^{1k}$	mAR	$mAP_{0.5}^{0.95}$	t_{GPU}	#params	FLOPs
SDN	ImageNet	98.76	99.5	73.96	33.21	241	42.04M	190.57B

Table 5.6: Detection results on `POSSE_v4` test set with SDN

on inference times and computational cost by an increment of 68ms of t_{GPU} on average and 37.94B FLOPs. Similar to the `POSSE_v3` predictions, from the Figure 5.7 (right), it is observed that this experiment is also more likely to predict Cluster than Aqua. From the right-top image in the Figure 5.7 (left), it is noticed that the network is capable of addressing the challenge with Earth in the background. However, the other two images of Aqua in the Figure 5.7 (left) indicate the detection failure and network's drawbacks in falsely detecting Aqua as Cluster at partial/low visibility.

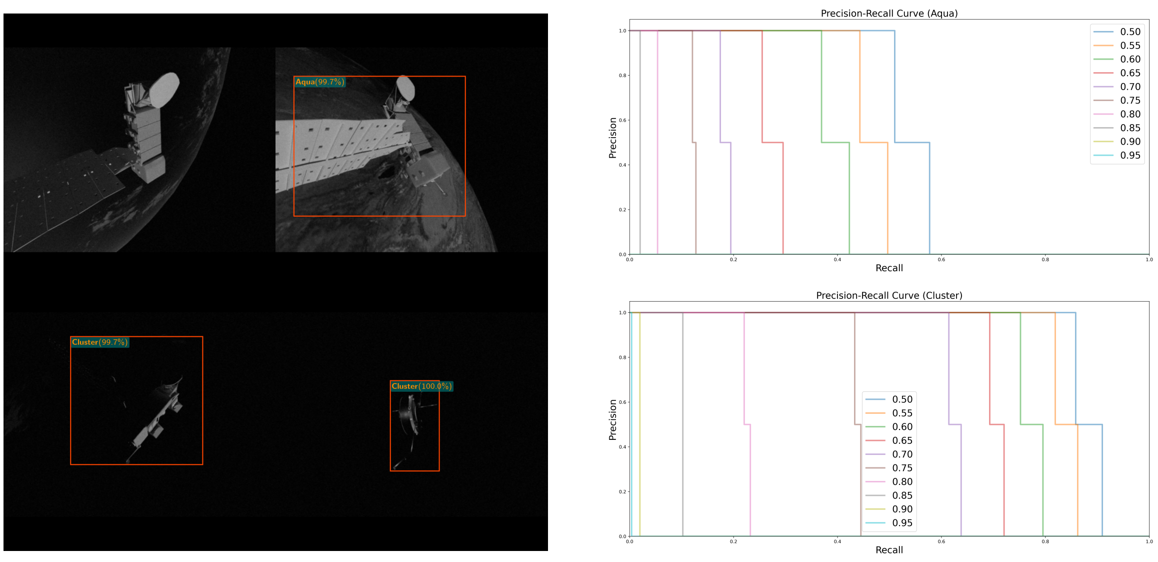


Figure 5.7: Performance prediction of SDN on `POSSE_v4`

To summarize the single-instance and multi-class experiments, the decision on the depth of the SDN is directly influenced by an increase in image complexity, in our case, resulting in an enhancement from ResNet18 to ResNet50. As a result, the enhanced SDN performs well in the detection and classification tasks, both with and without the Earth in the background, yet with a lower precision metric.

Overall, the SDN's performance between the single-class and multi-class classification is significantly lowered by mean recall of 23.77% and mean precision of 27.57% on average. However, the SDN's ability to detect and classify the spacecraft in challenging conditions occurred at an increased computation cost eventually reducing the inference times from 5 FPS to 4 FPS. It is anticipated that the proportion of images used for multi-class training could be another factor affecting the biased predictions because the network is trained with a higher amount of Cluster images than Aqua.

5.2.3. Experiment on `POSSE`

The POSSE dataset includes noisy stars and Earth in the background. In total, the dataset contains 3242 Cluster and 4995 Aqua images of which the dataset is partitioned into 6590 train, 823 val and 824 test samples. Similar to the previous experiments the hyperparameters are constant, and the transfer learning approach with the ImageNet weights is adopted for training the SDN up to 100 epochs. Considering the previous experiment outcomes, the SDN with ResNet50 architecture is implemented as the backbone for this multi-classification problem. The performance metrics in the Table 5.7 closely resemble the performance of `POSSE_v4` experiment with $mAP@[.5, .95]$ and mAR of 36.57%, and 74.92% respectively.

Method	TL	$AR_{0.5}^{100}$	$AR_{0.5}^{1k}$	mAR	$mAP_{0.5}^{0.95}$	t_{GPU}	#params	FLOPs
SDN	ImageNet	98.66	98.66	74.92	36.57	245	42.04M	190.57B

Table 5.7: Detection results on `POSSE` test set with SDN

In the Figure 5.8 (right), the P-R curves state that the predictions are relatively more accurate for Cluster than Aqua. Nevertheless, from the Figure 5.8 (left), the SDN has proven to detect and classify the spacecraft even in noisy conditions.

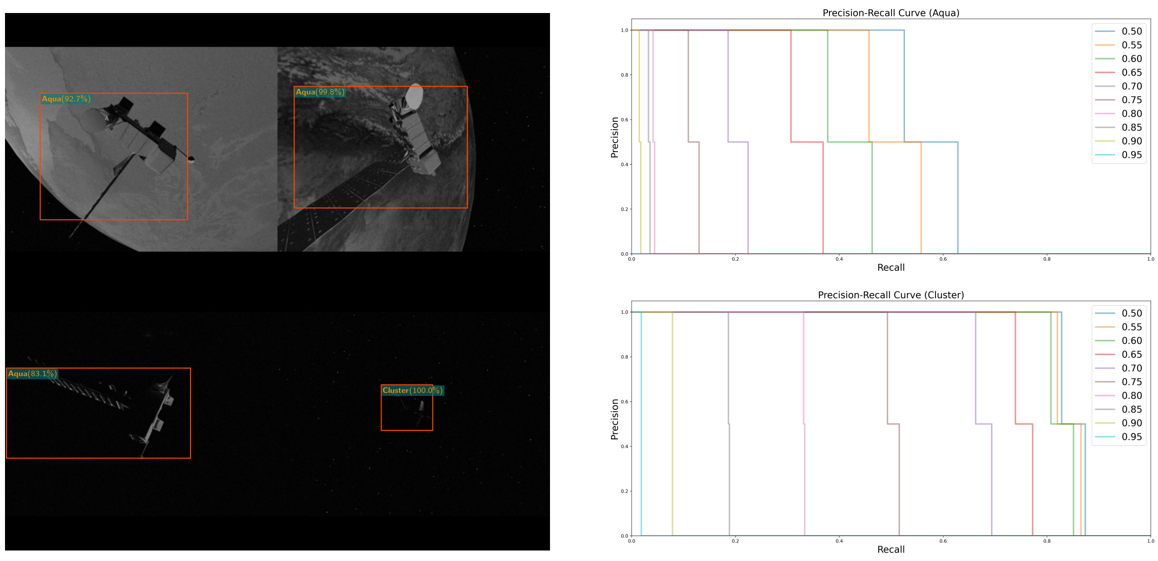


Figure 5.8: Performance prediction of SDN on `POSSE`

To conclude, the performance of SDN shows potential in spacecraft classification and detection. The lower performance has resulted from the failure of precise overlap of the bounding box. However, in most of the test images, the SDN has proven to detect the main body perfectly. Given that a considerable number of keypoints are always on the main body, the use of SDN in pose estimation is reliable.

6 | Conclusions and future developments

This work primarily aimed to understand the applicability and robustness of deep learning techniques for spacecraft detection over multi-spacecraft datasets with disparate features such that, the research outcomes further contribute to the development of keypoint extraction and pose estimation algorithms on a multi-spacecraft dataset.

In this context, a multi-spacecraft dataset generation algorithm is introduced to overcome the shortage of publicly available pose estimation datasets. The key components of the algorithm include the orbital propagator for defining the pose of the scene objects and a physical-based rendering simulator to generate realistic synthetic images. The camera path trajectory relative to the spacecraft was designed using the ROEs. This algorithm requires only a 3D model and its Epoch with 3D keypoints being labelled in the global space as an input and offers significant autonomy for the user to select both spacecraft and background.

The Ren et al. and Proença and Gao's architectures are modified to perform the multi-spacecraft detection based on the image complexity and the number of available training samples. A few modifications regarding the loss functions' strategy and the input dimensions resulted in successful training with a smaller batch size and increased detection performance. Moreover, the current version of the CNN code is updated to run with the latest API releases of TensorFlow and Keras. In addition, the baseline code is restructured into modules depending on their tasks. This step clearly offers an edge to further expand the Faster R-CNN for other tasks and also to merge the SDN with additional networks dedicated to the keypoint extraction or pose estimation.

Apart from the main contributions mentioned above, this document also discusses the current state-of-the-art with regard to deep learning techniques in detail. Moreover, the coordinate frames needed for orbital propagation, Keplerian motion along with the relative dynamics are studied in Chapter 2. In addition to that, the mathematical concepts of the CNN operations and a few typical CNN architectures that had a significant impact on image classification and object detection architectures are also elaborated in this chapter. In Chapter 3, the assumptions and simplifications adopted for the 3D models, celestial bodies and unperturbed dynamics of both the spacecraft and camera are determined. The modifications applied to the scene objects and the working principle of the rendering techniques implemented in the POSSE Simulator are also mentioned. While generating the POSSE dataset and its annotations, the removal of keypoints from the occluded regions is considered a challenge as it implies false positives. This is resolved with pixel-level accuracy by re-projecting the 3D point onto the PCS frame and further evaluating the pixel value. The proposal of SDN in Chapter 4 is simple and it directly modifies the baseline to predict the spacecraft.

To quickly outline the results, the dataset generation algorithm introduced in Chapter 3 resulted in 8237 novel synthetic images of Aqua and Cluster that consist of sharp and smooth features along with their annotations for a bounding box, keypoints and pose data, at challenging conditions. Later, the SDN is evaluated for its applicability and robustness over the partitioned datasets that are categorized based on image complexity. Wherein, the SDN is proven to be efficient in detecting a single spacecraft with a dark background at a mean Average Precision (mAP) of 78.03% and 5 FPS, and this performance is further reduced when the complexity is induced by adding the Earth, Stars and multiple spacecraft in the dataset. In the end, the SDN resulted in 36.57% of mAP and 4 FPS over the complexities included in the POSSE Dataset.

Furthermore, the research outcomes can be improved on both contributions. The first is multi-spacecraft dataset generation, where the current approach for generating annotations is based on the visibility of manually annotated 3D keypoints in the image frame. Yet this approach is accurate only when the entire spacecraft is in the FOV. Therefore, a tangible approach independent of such manual annotations is recommended. Later, the biased predictions on multi-spacecraft detections can be further verified by creating a dataset with an equal number of images from each class so that the performance can be improved. The propagator and POSSE simulator can be extended with the addition of perturbations and control dynamics to the propagator, which can unleash the potential to replicate the RPO scenario in the simulator. Finally, it is nice to have the Sun to physically appear in the render. On the other end, the implementation of image augmentation techniques in the pre-processing steps has the potential to improve robustness and detection performance. To attain empirical evidence, the proposed SDN can be validated either on publicly available single-spacecraft datasets or by creating a combination of datasets to evaluate multi-spacecraft detection performance. Ultimately, the model's onboard implementation can assess the real-time performance and network efficiency.

Bibliography

[1] Jianing Song, Duarte Rondao, and Nabil Aouf. Deep learning-based spacecraft relative navigation methods: A survey. *Acta Astronautica*, 191:22–40, 2022. ISSN 0094-5765. doi: 10.1016/j.actaastro.2021.10.025.

[2] Sumant Sharma and Simone D'Amico. Pose Estimation for Non-Cooperative Rendezvous Using Neural Networks. *CoRR*, Jun 2019. doi: 10.48550/ARXIV.1906.09868.

[3] Pedro F. Proença and Yang Gao. Deep Learning for Spacecraft Pose Estimation from Photorealistic Rendering. *CoRR*, 2019. doi: 10.48550/ARXIV.1907.04298.

[4] Tae Ha Park, Sumant Sharma, and Simone D'Amico. Towards Robust Learning-Based Pose Estimation of Noncooperative Spacecraft. *CoRR*, Sep 2019. doi: 10.48550/ARXIV.1909.00392.

[5] Simone D'Amico. *Autonomous Formation Flying in Low Earth Orbit*. PhD thesis, Delft University of Technology, Ridderprint BV, Alblasserdam, Mar 2010.

[6] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. *Communications of the ACM*, 60(6): 84–90, May 2017. ISSN 0001-0782. doi: 10.1145/3065386.

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, Dec 2015. doi: 10.48550/arXiv.1512.03385.

[8] MathWorks. Implement position and velocity of astronomical objects - simulink. `https://www.mathworks.com/help/aeroblks/planetaryephemeris.html`, Nov 2023.

[9] Portuguese Space Agency. ADRIOS (Active Debris Removal/ In-Orbit Servicing) - Portugal Space. `https://ptspace.pt/adrios/`, Sep 2023.

[10] Vaios J. Lappas, Jason L. Forshaw, and Lourens Visagie. RemoveDEBRIS: An EU Low Cost Demonstration Mission to test ADR technologies. In *65th International*

Astronautical Congress 2014 (IAC 2014), volume 3, page 1612. International Astronautical Federation (IAF), Sep 2014.

[11] Thomas Chabot, Keyvan Kanani, Alexandre Pollini, François Chaumette, Eric Marchand, and Jason Forshaw. Vision-based navigation experiment onboard the RemoveDEBRIS mission. In *GNC 2017 - 10th International ESA Conference on Guidance, Navigation & Control Systems*, pages 1–23, Salzburg, Austria, May 2017. URL `https://hal.inria.fr/hal-01784234`.

[12] Matt Pyrak and Joseph Anderson. Performance of Northrop Grumman's Mission Extension Vehicle (MEV) RPO Imagers at GEO. In *2021 Technical Papers*. AMOS Conference, 2021. URL `https://amostech.com/TechnicalPapers/2021/Poster/Pyrak.pdf`.

[13] Matthew A. Vavrina, C. Eugene Skelton, Keith D. Deweese, Bo J. Naasz, David E. Gaylor, and Christopher D'souza. Safe rendezvous trajectory design for the RESTORE-L mission. In *Spaceflight Mechanics 2019*, Advances in the Astronautical Sciences, pages 3649–3668. Univelt Inc., Jan 2019. ISBN 9780877036593.

[14] Robin Biesbroek, Sarmad Aziz, Andrew Wolahan, Stefano Cipolla, Muriel Richard-noca, and Luc Piguet. The ClearSpace-1 mission: ESA and ClearSpace team up to remove debris. In *8th European Conference on Space Debris*, volume 8 of *1*. ESA Space Debris Office, Apr 2021.

[15] Richard O. Duda and Peter E. Hart. Use of the Hough Transformation to Detect Lines and Curves in Pictures. *Communications of the ACM*, 15(1):11–15, Jan 1972. ISSN 0001-0782. doi: 10.1145/361237.361242.

[16] Irwin Sobel and Gary Feldman. A 3×3 isotropic gradient operator for image processing. *Pattern Classification and Scene Analysis*, pages 271–272, Jan 1973.

[17] John Canny. A Computational Approach to Edge Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(6):679–698, 1986. doi: 10.1109/TPAMI.1986.4767851.

[18] D G Lowe. Three-Dimensional Object Recognition from Single Two-Dimensional Images. *Artificial Intelligence*, 31(3):355–395, Mar 1987. ISSN 0004-3702. doi: 10.1016/0004-3702(87)90070-1.

[19] Radu Horaud. New Methods for Matching 3-D Objects with Single Perspective Views. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-9(3):401–412, 1987. doi: 10.1109/TPAMI.1987.4767922.

[20] M. Dhome, M. Richetin, J.-T. Lapreste, and G. Rives. Determination of the attitude of 3D objects from a single perspective view. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(12):1265–1278, 1989. doi: 10.1109/34.41365.

[21] P. Wlczek, A. Maccato, and R.J.P. Defigueiredo. Pose Estimation of Three-Dimensional Objects from Single Camera Images. *Digital Signal Processing*, 5(3): 176–183, 1995. ISSN 1051-2004. doi: 10.1006/dspr.1995.1018.

[22] Mate Kisantal, Sumant Sharma, Tae Ha Park, Dario Izzo, Marcus Märtens, and Simone D'Amico. Spacecraft Pose Estimation Dataset (SPEED), 2019. URL `https://doi.org/10.5281/zenodo.6327546`.

[23] Tae Ha Park, Marcus Märtens, Gurvan Lecuyer, Dario Izzo, and Simone D'Amico. Next Generation Spacecraft Pose Estimation Dataset (SPEED+), 2021. URL `https://doi.org/10.25740/wv398fc4383`.

[24] Sumant Sharma, Connor Beierle, and Simone D'Amico. Pose estimation for non-cooperative spacecraft rendezvous using convolutional neural networks. In *2018 IEEE Aerospace Conference*, pages 1–12, Jun 2018. doi: 10.1109/AERO.2018.8396425.

[25] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, 115(3):211–252, Dec 2015. ISSN 1573-1405. doi: 10.1007/s11263-015-0816-y.

[26] Sumant Sharma and Simone D'Amico. Neural Network-Based Pose Estimation for Noncooperative Spacecraft Rendezvous. *IEEE Transactions on Aerospace and Electronic Systems*, 56(6):4638–4658, Dec 2020. doi: 10.1109/TAES.2020.2999148.

[27] Lisa Jonsson. Simulations of satellite attitude maneuvers : Detumbling and pointing. Master's thesis, Luleå University of Technology, Space Technology, May 2019.

[28] Gabriella Gaias and Marco Lovera. Trajectory design for proximity operations: The relative orbital elements' perspective. *Journal of Guidance, Control, and Dynamics*, 44(12):2294–2302, Sep 2021. doi: 10.2514/1.G006175.

[29] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, Apr 2015. doi: 10.48550/arXiv.1409.1556.

[30] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. *IEEE*, pages 248–255, 2009. doi: 10.1109/CVPR.2009.5206848. URL `https://image-net.org/index.php`.

[31] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: common objects in context. *CoRR*, abs/1405.0312, 2014. URL `https://cocodataset.org/#home`.

[32] Alina Kuznetsova, Hassan Rom, Neil Alldrin, Jasper R. R. Uijlings, Ivan Krasin, Jordi Pont-Tuset, Shahab Kamali, Stefan Popov, Matteo Malloci, Tom Duerig, and Vittorio Ferrari. The open images dataset V4: unified image classification, object detection, and visual relationship detection at scale. *CoRR*, abs/1811.00982, 2018. URL `https://storage.googleapis.com/openimages/web/index.html`.

[33] ML Glossary. Loss Functions — ML Glossary documentation. `https://ml-cheatsheet.readthedocs.io/en/latest/loss_functions.html`, July 2023.

[34] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *CVPR*, pages 580–587, 2014. doi: 10.1109/CVPR.2014.81.

[35] Ross Girshick. Fast R-CNN. *CV*, pages 1440–1448, 2015. doi: 10.1109/ICCV.2015.169.

[36] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. *PAMI*, 39(6):1137–1149, 2017. doi: 10.1109/TPAMI.2016.2577031.

[37] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask R-CNN. *PAMI*, pages 2980–2988, 2017. doi: 10.1109/ICCV.2017.322.

[38] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You Only Look Once: Unified, real-time object detection. *CVPR*, pages 779–788, 2016. doi: 10.1109/CVPR.2016.91.

[39] Joseph Redmon and Ali Farhadi. YOLO9000: Better, Faster, Stronger. *CoRR*, abs/1612.08242, 2016. doi: 10.48550/arXiv.1612.08242.

[40] Joseph Redmon and Ali Farhadi. YOLOv3: An Incremental Improvement. *CoRR*, abs/1804.02767, 2018. doi: 10.48550/arXiv.1804.02767.

[41] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. YOLOv4: Optimal Speed and Accuracy of Object Detection. *CoRR*, abs/2004.10934, 2020. doi: 10.48550/arXiv.2004.10934.

[42] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and

Serge Belongie. Feature pyramid networks for object detection. *CVPR*, pages 936–944, 2017. doi: 10.1109/CVPR.2017.106.

[43] Martin A. Fischler and Robert C. Bolles. Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. *Communications of the ACM*, 24(6):381–395, Jun 1981. ISSN 0001-0782. doi: 10.1145/358669.358692.

[44] Cluster - Satellite Missions - eoPortal Directory. `https://www.eoportal.org/satellite-missions/cluster#Cluster.html.8`, May 2012.

[45] CelesTrak. CelesTrak: Search Satellite Catalog. `https://celestrak.org/satcat/search.php`, Aug 2022.

[46] Aqua brochure. `https://www.nasa.gov/pdf/151986main_Aqua_brochure.pdf`, Mar 2002.

[47] SPENVIS. Spenvis - space environment, effects, and education system. `https://www.spenvis.oma.be/`, Aug 2022.

[48] Lorenzo Pasqualetto Cassinis, Robert Fonod, and Eberhard Gill. Review of the robustness and applicability of monocular pose estimation systems for relative navigation with an uncooperative spacecraft. *Progress in Aerospace Sciences*, 110:100548, 2019. ISSN 0376-0421. doi: 10.1016/j.paerosci.2019.05.008.

[49] Roberto Opromolla, Giancarmine Fasano, Giancarlo Rufino, and Michele Grassi. A review of cooperative and uncooperative spacecraft pose determination techniques for close-proximity operations. *Progress in Aerospace Sciences*, 93:53–72, 2017. ISSN 0376-0421. doi: 10.1016/j.paerosci.2017.07.001.

[50] Blender Online Community. *Blender - a 3D modelling and rendering package*. Blender Foundation, Stichting Blender Foundation, Amsterdam, 2018. URL `http://www.blender.org`.

[51] NASA. NASA Visible Earth. `https://www.visibleearth.nasa.gov/`, 2022.

[52] Google Colab. Welcome To Colaboratory - Colaboratory. `https://colab.research.google.com/`, 2017.

Acknowledgements

I am extremely grateful to my advisor, Prof. Mauro Massari for his unwavering support and belief in my ability to tackle this advanced topic. I am grateful for his suggestions and the freedom of choice in the research, which helped me understand the importance of making sound decisions in the future.

Thanks to all the faculty members for sharing their knowledge. Their expertise made my Space Engineering degree more meaningful. Special thanks to friends of Zingarata and the loved ones implicitly, for making me feel special with all their love and affection. Without them, my abroad education in Milan would not be such a memorable, fun, and delightful experience.

Family members should also be thanked for their positive encouragement, understanding, and support in completing the master's degree. I hope they will encourage me, in the same way, to pursue my dreams and support the decisions I make in the future.

I would like to express my gratitude to my roommates for their moral and tangible support from the beginning of this journey with things related to settling in Milan, during the stay and the exams, and also during the stressful times.