Automated Recovery of Bolides Photometric Profiles

Not Confidential

Baptiste Guivarch
FISE25-SOIA

Under the supervision of Josep M.Trigo Rodríguez (ICE-CSIC), Fabrice Comblet (ENSTA)

1 Keywords

Convolutional Neural Networks, Star identification, Star detection, Bolide segmentation U-Net

Résumé

Ce projet présente un processus permettant d'estimer la magnitude apparente des bolides dans des vidéos du ciel nocturne enregistrées par le Réseau Espagnol d'Observation des Météores (Spanish Meteor Network). Les étoiles sont d'abord détectées à l'aide d'un réseau de neurones convolutifs (CNN) basé sur l'architecture U-Net, puis identifiées automatiquement via Astrometry.net ou manuellement. Le mouvement du bolide est localisé grâce à une analyse par flot optique, suivie d'une segmentation image par image de sa trajectoire. Les mesures photométriques des étoiles et des bolides sont effectuées par ajustement d'une gaussienne bidimensionnelle, ce qui permet d'estimer le flux intégré. En utilisant des étoiles de référence dont la magnitude est connue, la magnitude apparente du bolide est déterminée par étalonnage des rapports de flux. Cette approche intégrée combine apprentissage profond, vision par ordinateur et photométrie classique pour fournir une estimation automatisée de la magnitude des bolides.

Abstract

This project presents a photometric pipeline for estimating the apparent magnitude of bolides in night-sky videos recorded by the Spanish Meteor Network. Stellar sources are first detected using a convolutional neural network (CNN) based on the U-Net architecture and subsequently identified either automatically via Astrometry.net or manually. The bolide's motion is localized through optical flow analysis, followed by frame-by-frame segmentation of the bolide region. Photometric measurements for both stars and bolides are performed using two-dimensional Gaussian fitting, enabling the estimation of integrated flux. By referencing known stellar magnitudes, the bolide's apparent magnitude is derived through calibrated flux ratios. This integrated approach combines deep learning, computer vision, and classical photometry to achieve automated magnitude estimation of bolides.

Acknowledgements

I would like to express my sincere gratitude to all those who supported and guided me throughout this project. I especially thank my supervisor, Dr. Josep M. Trigo Rodríguez, for his valuable advice, rigorous guidance, and availability throughout this work. I also extend my thanks to all the members of the Institute for their warm welcome. Finally, I am grateful to my colleagues, instructors, and family for their constant moral support.

Contents

2 Introduction

2.1 Internship framework

During five months, I had the opportunity to work at the Institute of Space Sciences (ICE-CSIC) as part of my end-of-studies internship. The institute is located on the campus of the Autonomous University of Barcelona. It is part of the Higher Council for Scientific Research (CSIC), which is the largest institution dedicated to research in Spain.

The Institute of Space Sciences is structured around five main research departments.
- The Star Formation, Stellar and High-Energy Astrophysics Department focuses on the physical processes associated with stars, from their formation to their death.
- The Nuclear, Gravitational Wave, and Multi-messenger Astrophysics Department studies compact objects such as neutron stars and black holes, as well as gravitational interactions.
- The Extragalactic Astrophysics and Cosmology Group investigates the fundamental laws and origins of the Universe by studying supernovae and the impact of massive black holes in shaping galaxies.
- An Advanced Engineering Unit supports the development of technologies for new missions and astronomical observatories.
- Finally, the Planetary Systems and Earth Observation Department, which I joined, aims at improving the knowledge of our planet. Through the characterization of planetary systems around other stars and the study of asteroids, comets, and meteorites, the research group gains insight into the processes that occurred in our planetary system.

I worked under the supervision of Dr. Josep M. Trigo Rodríguez. As part of his work, Dr. Josep M. Trigo Rodríguez coordinates the Spanish Fireball and Meteorite Recovery Network (SPMN).

The Spanish Meteorite Recovery Network (SPMN) studies interplanetary matter penetrating the Earth's atmosphere and producing meteors, or those brighter than Venus that are called fireballs. From the astrometric trajectory reconstruction of the fireball obtained from the monitoring stations, an atmospheric trajectory is computed solving the method of intersection of planes. Then, taking into account the effect of the atmospheric wind on the final dark flight, the possible falling areas are computed with the goal to recover fresh meteorites for direct study.

Amateur astronomers are also participating in the SPMN network coverage with specific observing programs. The main aim is to increase the knowledge on meteoroid interaction with the atmosphere, learn more about the dynamical mechanisms of meteoroid delivery to the Earth, and about the role that primitive bodies had on the origin of life in the Earth.

The SPMN has promoted the recovery of three meteorites: the L6 chondrite Villalbeto de la Peña (2004), the eucrite Puerto Lapice (2007) and L5 chondrite Traspena (2022). Dr. Josep M. Trigo Rodríguez is assisted in his work by Dr.Eloy Peña-Asensio en Ph.D student Pau Grèbol.

2.2 Objectives

Even though bolides do not always produce meteorites, it is possible to gain insights into their physical properties by studying variations in their luminosity (photometric profiles). When a bolide enters the field of view of a camera, it appears with a certain brightness that evolves as it travels through the atmosphere. As it descends, increasing air density enhances friction and therefore temperature. The various materials composing the bolides have different melting or vaporization points leading to diverse patterns in luminosity. Sharp increases in brightness can often indicate fragmentation events, while smoother variations may reflect continuous ablation of heterogeneous materials. Studying the photometric profile can help determine whether a bolide is likely to produce a meteorite that could possibly be recovered or not.

Cameras do not directly measure brightness, they record pixel values or intensities. Therefore to recover the photometric profile of a bolide it is essential to compare its apparent brightness to some reference point within the image. This is the reason why stars' magnitudes are used. The magnitude of a star is a measure of its brightness. The apparent magnitude defines how bright a star appears in the sky to an observer on Earth while the absolute magnitude refers to the brightness of a star seen from a distance of 10 parsecs (32.6 light-years) from Earth. The lower the magnitude, the brighter the object. For instance, the Sun has an apparent magnitude of -26.74, while Sirius, the brightest star in the night sky has a magnitude of -1.46. The naked eye-limit in a dark sky is around +6.

Using stars magnitude as reference also allows to correct atmospheric effects. Stars in the same frame as the bolide experience the same atmospheric conditions. Using them as reference helps correct those effects, making the bolide's magnitude estimation more accurate.

I was assigned the task of exploring the use of machine learning techniques to automate the recovery of photometric profiles of bolides.

The project can be divided into several key stages. The first step involves detecting stars in the images, which presents notable challenges. Given the small apparent size of stars (often only a few pixels in images with a resolution of 3840×2160), their detection is non-trivial. Furthermore, varying background conditions such as buildings, ambient lighting, and weather add additional complexity to this task.

Once detected, each star must be accurately identified, as every star has a known, fixed magnitude. Incorrect identification would introduce significant errors in the calibration of the bolide's photometric profile.

The following phase involves detecting and tracking the bolide across video frames. This requires robust motion analysis and segmentation techniques to isolate the bolide and extract its light curve reliably over time.

3 Bibliography

3.1 Stars Segmentation

Recent works have explored deep learning approaches for detecting and segmenting stars, each focusing on different datasets and application contexts.

In their 2025 study, Zhao et al. [1] introduced a convolutional neural network (CNN)-based method for real-time star detection and centroiding onboard CubeSats. Their system was trained and tested on star tracker imagery captured by CubeSat optical sensors, characterized by sparse star fields, high contrast, and minimal background clutter.

Mastrofini et al. [2] developed a U-Net-based segmentation system for star sensor images. Their dataset consisted of simulated and real star sensor captures, including scenarios with optical noise such as ghost reflections and lens flares, representative of satellite-based navigation instruments.

Burke et al. [3] proposed the Astro R-CNN framework, which adapts Mask R-CNN for the deblending and classification of overlapping astronomical sources. Their method was evaluated on wide-field astronomical survey data, such as that from ground-based telescopes, containing both stars and galaxies densely distributed across large portions of the sky.

Hausen and Robertson [4] presented the Morpheus framework, designed for pixel-level segmentation of deep space imagery. Their dataset consisted of high-resolution astronomical images from the Hubble Space Telescope, featuring detailed galactic and extragalactic structures.

These studies demonstrate the versatility of deep learning for astronomical image analysis, addressing a variety of datasets ranging from CubeSat star tracker images to space-based deep field observations. In contrast, the present work focuses on SPMN (Spanish Meteor Network) imagery, which often contains complex scenes with not only stars, but also terrestrial elements such as buildings, urban lighting, and varied atmospheric conditions. This requires segmentation models to handle background variability and noise.

3.2 Meteor Detection and Segmentation

In recent years, deep learning has become a pivotal tool in meteor detection, driving automation and enhancing accuracy in meteor observation networks worldwide. Various studies have explored CNN-based architectures, often leveraging transfer learning and feature localization techniques to address classification, detection, and tracking challenges.

Peña-Asensio et al. [5] made a significant contribution by applying ResNet-34 combined with Grad-CAM for meteor classification and motion localization using data from the Spanish Meteor Network (SPMN). Their approach demonstrates that transfer learning, coupled with attention-based localization, can achieve high classification accuracy even with limited labeled data.

Other research efforts have targeted specific meteor networks and datasets. Sennlaub et al. [6] employed GRU-enhanced CNNs on the AllSky7 Fireball Network, achieving robust detection performance on structured datasets characterized by low background complexity. Cecil and Campbell-Brown [7] developed CNN-based models for meteor tracking within the CAMO system, while Gural [8] introduced MeteorNet, an LSTM-RNN framework designed for trajectory tracking in the CAMS dataset.

These models generally report strong performance metrics, often surpassing 98% accuracy in classification tasks. Their training, however, primarily relies on curated datasets that are preprocessed to reduce background clutter, normalize image scales, or isolate motion frames. In contrast, our methodology operates directly on raw video data, which includes challenges such as atmospheric disturbances, urban structures, clouds, and partial occlusions of bolides. Additionally, our focus extends beyond detection to the spatially precise segmentation of bolide trails on a frame-by-frame basis.

4 Stars Detection

4.1 Dataset creation

Neural Networks require a large and diverse dataset to perform effectively in real world conditions. The SPMN provides a substantial collection of images, captured in various locations, featuring diverse backgrounds and lightning conditions. Yet, these images are not directly usable to train a model.

To train a model effectively, the images must be labelled. In other words, for each input image, the expected result must be known. In our case, when an image is fed to the model, the desired output is a binary image in which the stars appear white and the background is black.

One possible approach would be to manually label each image. Since some images contain hundreds of stars this process would be extremely time-consuming. To address this problem, the idea of generating labels automatically was adopted.

To preserve the diversity of backgrounds, stars are automatically removed from the SPMN images using morphological opening, resulting in clean, star-free backgrounds (Figure 2). The final step involves reintroducing stars into the images.

To ensure accuracy, a few star patterns were extracted from real images. These patterns are then overlaid onto the star-free backgrounds (Figure 3). Since the star patterns and their placement are known and controlled, the corresponding labels can be automatically generated.

Figure 1 – Original Image

Figure 2 – Star free image after morphological opening

Figure 3 – Image with synthetic stars

The models trained on the first dataset delivered promising results but lacked robustness and quickly reached a performance plateau. Neither architectural changes nor hyperparameter adjustments led to meaningful improvements. This limitation arose because the artificial stars, although reasonably realistic, allowed the model to capture only a narrow range of patterns. More importantly, the model was never exposed to real stellar data, which was its actual target. Therefore, further progress depended on improving the dataset itself.

To enhance the dataset, supervised annotation was employed. This approach involves using the output of one model to generate training data for another. While studies have shown that this method can degrade performance in the context of large language models (LLMS) [9], it has proven effective in the field of computer vision.

In this work, the model initially trained on synthetic data was used to construct a new dataset. For each image, if the model's output met expectations, it was included in the new dataset. Rather than labelling each star individually, the only required action was to accept or reject each image based on the quality of the model's output.

To reduce false positives, the dataset also included images in which no detection was expected. SPMN images contain overlaid text displaying metadata such as the date, time, and location of the recording. This text is typically white and certain characters such as the colon in the time display can closely resemble stars. Additionally, some images include bolides that may also be mistakenly identified as stars (Figure 1).

To mitigate the risk of overfitting, data augmentation techniques were applied to the training dataset. Overfitting occurs when a model memorizes the learning data rather than learning to generalize from it. In such cases, the model performs extremely well on training data but fails to deliver accurate predictions when confronted with data it has never seen before. This issue is particularly pronounced when working with small datasets.

Data augmentation helps mitigate this by artificially increasing the variability of the training data through transformations such as rotations, flips, and brightness adjustments. As a result, the model is exposed to a broader range of scenarios, which improves its generalization capabilities and robustness.

4.2 Model architecture

Most segmentation models are based on an encoder-decoder architecture. The encoder progressively reduces the spatial resolution of the input image while increasing the depth of the feature maps, allowing the model to capture high-level semantic information.

The decoder then upsamples the encoded representation to produce a segmentation map that matches the original image size.

While this structure is effective for detecting large or well-defined objects, it presents a challenge when dealing with smaller features such as stars, which occupy only few pixels. During the encoding phase, little structures can be blurred or lost. This is particularly problematic in our case, where preserving fine spatial details is crucial for identifying stars.

This is why the model used to detect stars is based on the U-Net architecture [10]. U-Net takes its name from its characteristic U-shape, formed by the symmetrical structure of the encoder on one side and the decoder on the other (Figure 4). The specificity of the U-Net architecture lies in its use of skip connections: feature maps from the encoder are directly combined with those in the decoder. These skip connections help mitigate the loss of spatial information that can occur during the downsampling process in the encoder.

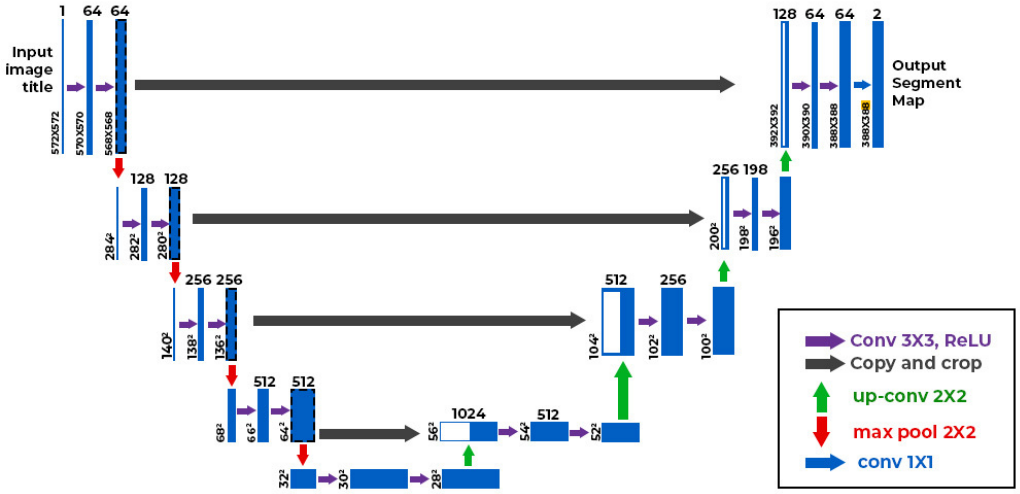

Figure 4 – U-Net architecture.
https://datasciencetoday.net/images/unet.png

4.3 Transfer learning

4.3.1 Fine tuning

To enhance the model's performance, fine tuning was employed. In classical transfer learning, a model trained for a specific task with a large dataset is adapted to a new task. Typically, the weights of the early layers of the model are frozen, and the last layers are replaced or retrained to fulfill the requirements of the new task. The main goal of transfer learning is to leverage the knowledge acquired by the base model, during training on large datasets. Since the early layers often capture general features like edges and textures, they can be reused across different tasks. Transfer learning thus speeds up training and provides a better starting point.

Fine tuning is a variant of transfer learning, where some of the previously frozen early layers are unfrozen and retrained alongside the last layers. This allows the model to adjust more finely to the specifics of the new task. Usually, a smaller learning rate is used during fine tuning, as the goal is to refine rather than learn from scratch.

Two of the most popular encoders were tested, ResNet [11], and EfficientNet [12]

4.3.2 ResNet

The ResNet architecture was designed to address the problem of performance degradation in very deep neural networks. As network's grew deeper, their training and validation accuracies began to deteriorate, and this was not due to overfitting but rather to optimization difficulties. Deeper networks often performed worse than smaller ones, even on training set. To investigate this issue, the authors proposed the use of identity shortcut connections which allow gradients to propagate more easily through the network.

Mathematically, the output of a residual block can be written as:

$$\text{Output} = \text{ReLU}(F(x) + x) \tag{1}$$

where:

— x is the input to the block,
— $F(x)$ is the transformation applied by two or more stacked convolutional layers,
— the addition $F(x) + x$ represents the residual connection,
— and ReLU is the activation function applied element-wise.

This formulation allows the network to preserve the input when needed, and only learn residual corrections through as illustrated in Figure 5. The residual blocks led to significantly better performance compared to plain architectures.

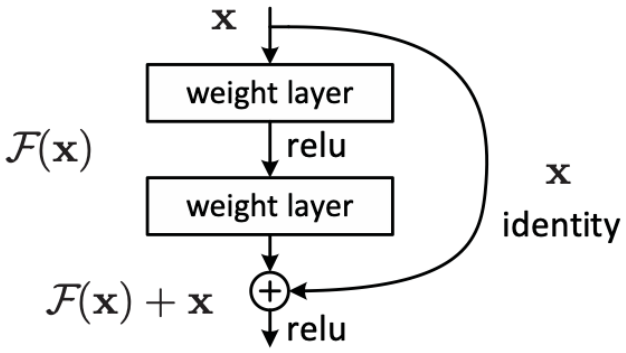

Figure 5 – Residual Block
https://arxiv.org/abs/1512.03385

To address the growing computational cost associated with very deep networks, the ResNet architecture introduced bottleneck blocks. These blocks are designed to reduce the number of parameters and operations without sacrificing model performance. Instead of relying solely on repeated 3×3 convolutions, bottleneck blocks use a more compact structure. Table 1 compares a standard residual block with its bottleneck counterpart showing a drastic reduction in parameter count.

Block Type	Convolution Layers	Channels	Parameters
Plain Block	3×3 conv	256 input, 256 output	590,080
	3×3 conv	256 input, 256 output	590,080
	–	–	**Total: 1,180,160**
Bottleneck Block	1×1 conv (reduce)	256 input, 64 output	16,448
	3×3 conv	64 input, 64 output	36,928
	1×1 conv (expand)	64 input, 256 output	16,448
	–	–	**Total: 69,824**

Table 1 – Comparison between Plain and Bottleneck Residual Blocks

4.3.3 EfficientNet

While ResNet improves performance by increasing the depth of the network (stacking more layers), EfficientNet introduces a more balanced strategy. It scales the network across three dimensions simultaneously:

— Depth: the number of layers.
— Width: the number of channels in each layer.
— Resolution: the input image size.

Scaling any of these individually can lead to suboptimal results or inefficient models. EfficientNet proposes a coumpound scaling method which uniformly scales all three dimensions using a fixed set of coefficients:

$$\begin{cases} \text{depth} = \alpha^{\phi} \\ \text{width} = \beta^{\phi} \\ \text{resolution} = \gamma^{\phi} \\ \alpha \cdot \beta^2 \cdot \gamma^2 \approx 2 \end{cases}$$

Here:

— ϕ is a user-defined compound coefficient that controls overall model scaling,
— α, β, γ are constants determined by grid search,
— The constraint ensures the total computational cost grows roughly by a factor of 2 when ϕ increases by 1.

The authors first used Neural Architecture Search (NAS) to find an efficient base model under Floating Points Operation per Second (FLOPS) and accuracy constraints. This base model is known as EfficientNet-B0. Then, using grid search with $\phi = 1$, they selected optimal values for α, β, γ to guide how the architecture should grow. The other versions from EfficientNet-B1 to B7 were found using the compound scaling rule with increasing values of ϕ

Despite its strong performance, the original EfficientNet architecture often exhibited slow training times, particularly on large-scale datasets. To address this limitation, a new version was introduced: EfficientNetV2 [13]. As illustrated in Figure 6, EfficientNetV2 outperforms its predecessor on ImageNet Top-1 Accuracy and training time.

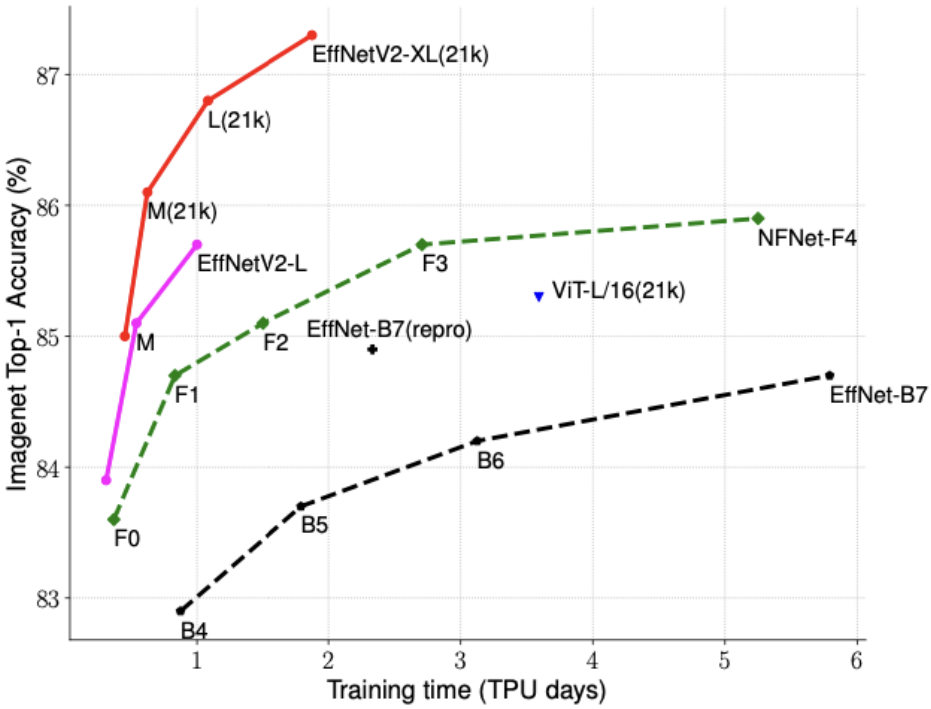

Figure 6 – Comparison of various models ImageNet Top-1 Accuracy and training time
https://arxiv.org/abs/2104.00298

Unlike the original model, which used a uniform compound scaling strategy across all layers, EfficientNetV2 adopts a progressive scaling approach applying different scaling strategies to early and deeper stages of the network. To do so, the architecture incorporates a combination of traditional MBConv blocks and Fused-MBConv blocks. The original EfficientNet used only MBConv blocks, which consist of an expansion layer, a depthwise convolution and a projection layer. While MBConvs are efficient in deeper layers where the model learns complex features, they are relatively slow to train, especially in the early stages of the network where high-resolution feature maps are processed.

In the early layers, the features are more general and the spatial resolution is higher, meaning that simpler operations can be just as effective and more efficient. To take advantage of this, EfficientNetV2 replaces MBConv blocks with Fused-MBConv blocks in the initial stages. A Fused-MBConv block removes the depthwise convolution and merges the expansion and the spatial convolution in a single regular convolution, followed by a projection layer. This results in faster training.

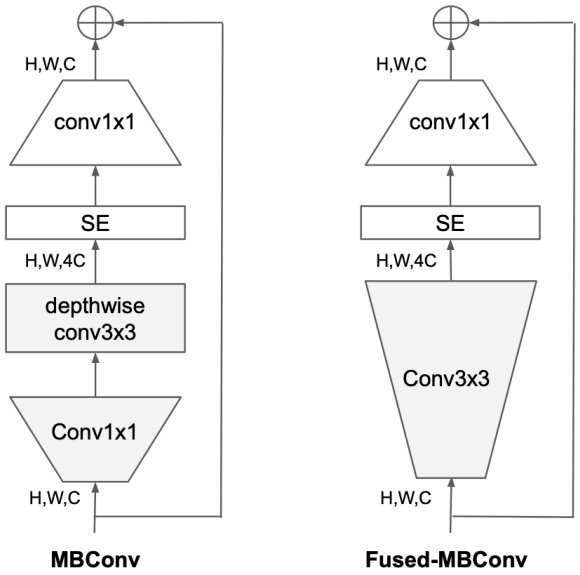

Figure 7 – MBConv and Fused-MBConv blocks
https://arxiv.org/abs/2104.00298

4.4 Optimizer

An optimizer is an algorithm used to update the weights of a neural network during training in order to minimize a given loss function. It plays a central role in the learning process by determining how the model adapts based on the computed errors.

Most optimizers are based on the principle of gradient descent. At each training step, the optimizer computes the gradient of the loss function with respect to the model's parameters. These gradients indicate the direction and rate of change needed to reduce the loss.

The fundamental idea of gradient descent is to update the weights in the opposite direction of the gradient. The basic update rule is:

$$\theta_{t+1} = \theta_t - \eta \cdot \nabla_\theta \mathcal{L}(\theta_t) \tag{2}$$

where:
— θ represents the model's parameters (weights),
— η is the learning rate,
— $\nabla_\theta \mathcal{L}(\theta_t)$ is the gradient of the loss function $\mathcal{L}$ with respect to θ.

If the gradient is positive : the loss increases when the weight increases and thus, the new weight should be lower hence the negative sign in (2). Conversly, if the gradient is negative, the loss increases when the weight decreases, so the next weight should be higher

More advanced versions of gradient descent incorporate additional techniques to improve stability and convergence. The optimizer used to train the model is Adam [14] which combines Momentum and RMSProp.

Momentum accelerates gradient descent by considering an exponentially decaying moving average of past gradients. Instead of relying only on the current gradient, it accumulates a velocity vector that helps smooth the updates:

$$v_t = \gamma v_{t-1} + \eta \cdot \nabla_\theta \mathcal{L}(\theta_t) \tag{3}$$
$$\theta_{t+1} = \theta_t - v_t \tag{4}$$

where:
— v_t is the velocity at step t,
— $\gamma \in [0, 1]$ is the momentum coefficient
Since v_{t-1} is multiplied by γ at each step, its influence decays exponentially, which allows older gradients to have diminishing impact. This smooths out oscillations, and allows faster convergence.

RMSProp adapts the learning rate for each parameter individually by using a moving average of squared gradients. RMSProp aims to address the problem of the learning rate being too large or too small for certain parameters by normalizing the gradient step with a root mean square of recent gradient magnitudes.

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1 - \gamma) \left(\nabla_\theta \mathcal{L}(\theta_t)\right)^2 \tag{5}$$
$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} \cdot \nabla_\theta \mathcal{L}(\theta_t) \tag{6}$$

where:
— $E[g^2]_t$ is the exponentially weighted moving average of the squared gradients at time t,
— $\gamma \in [0, 1]$ is the decay rate controlling how fast past gradients are forgotten,
— ϵ is a small constant added to the denominator for numerical stability,
— η is the base learning rate.

If the gradient is large in some region, it means the slope is steep in that direction, suggesting a potentially interesting local minima to explore. Since the learning rate is inversely proportional to the gradient magnitude, it will be smaller to avoid overshooting this region. Conversely, if the gradient magnitude is small, the loss landscape is relatively flat in that zone, so using a larger learning rate helps the optimizer escape and explore other regions.

4.5 Loss functions

The choice of loss function is a critical factor in training a deep learning model. It defines the objective that the optimizer seeks to minimize, making model performance highly dependent on this function. In the context of segmenting small objects such as stars, selecting an appropriate loss function becomes even more crucial due to their limited size and subtle features. This is the reason why four different losses were tested during training.

4.5.1 Binary Cross-Entropy (BCE) Loss

The Binary Cross-Entropy (BCE) loss is a widely used loss function for binary classification tasks. It measures the difference between two probability distributions: the predicted probabilities and the true binary labels.

Given a predicted probability $p \in [0, 1]$ and a true binary label $y \in \{0, 1\}$, the BCE loss for a single prediction is defined as:

$$\mathcal{L}_{BCE}(p, y) = -\left[y \log(p) + (1 - y) \log(1 - p)\right]$$

where log is the natural logarithm. Intuitively, the loss penalizes the prediction p when it diverges from the true label y.

For a dataset of N samples, the average BCE loss is:

$$\mathcal{L}_{BCE} = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(p_i) + (1-y_i)\log(1-p_i)]$$

The BCE loss is particularly well suited for segmentation tasks with pixel-wise binary classification, where each pixel is classified as foreground or background.

4.5.2 Dice Coefficient

The Dice coefficient, also known as the Dice-Sørensen coefficient, is a measure of similarity between two sets. Mathematically, it is defined as follows:

Let A and B be two sets. The Dice coefficient is given by:

$$DSC(A,B) = \frac{2|A \cap B|}{|A| + |B|}$$

where $|A|$ and $|B|$ represent the sizes of sets A and B respectively, and $|A \cap B|$ represents the size of the intersection of sets A and B.

A Dice coefficient of 1 indicates perfect overlap between the two sets, whereas a score of 0 indicates no intersection.

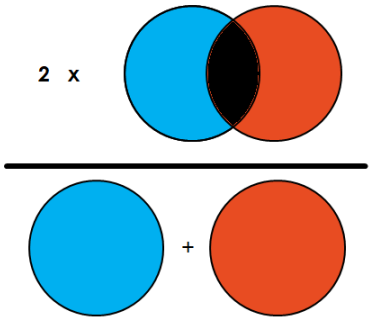

Figure 8 – Dice Coefficient
*https://miro.medium.com/max/429/1*yUd5ckecHjWZf6hGrdlwzA.png*

From the Dice coefficient, the Dice loss function is defined as:

$$\mathcal{L}_{Dice} : D \to [0,1]$$

such that for any pair of sets $(A,B) \in D$,

$$\mathcal{L}_{Dice}(A,B) = 1 - DSC(A,B)$$

4.5.3 Jaccard Coefficient

The Jaccard coefficient is also a similarity measure between two sets, closely related to the Dice coefficient. Mathematically, it is defined as follows:

Let A and B be two sets. The Jaccard coefficient is given by:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

where $|A|$ and $|B|$ represent the sizes (cardinalities) of sets A and B respectively, and $|A \cap B|$ is the size of their intersection.

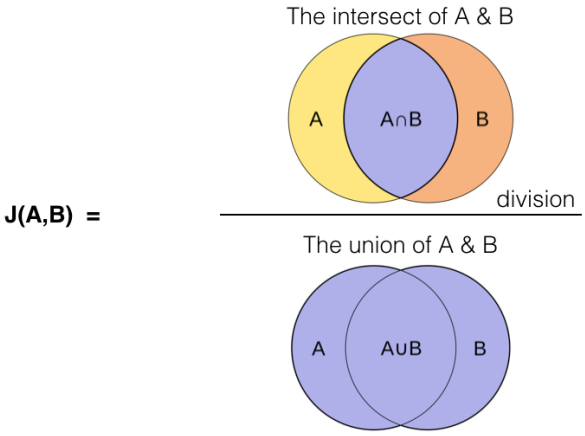

Figure 9 – Jaccard Coefficient
*https://miro.medium.com/v2/resize:fit:744/1*XiLRKr_Bo-VdgqVI-SvSQg.png*

From the Jaccard coefficient, the Jaccard loss function is defined as:

$$\mathcal{L}_{Jaccard} : D \to [0, 1]$$

such that for any pair of sets $(A, B) \in D$,

$$\mathcal{L}_{Jaccard}(A, B) = 1 - J(A, B)$$

4.5.4 Tversky Coefficient

A third loss function used for training the segmentation model is the Tversky coefficient. Let A and B be two sets. The Tversky coefficient is given by:

$$T(A, B) = \frac{|A \cap B|}{|A \cap B| + \alpha \cdot |\text{FP}| + \beta \cdot |\text{FN}|}$$

where:
— $|A \cap B|$ represents the size of the intersection of sets A and B,
— $|\text{FP}|$ is the number of false positives,
— $|\text{FN}|$ is the number of false negatives,
— α and β are scalar weights that adjust the relative importance of false positives and false negatives.

From this definition, the loss function to minimize becomes

$$\mathcal{L}_{Tversky} : D \to [0, 1]$$

such that for any pair of sets $(A, B) \in D$,

$$\mathcal{L}_{Tversky}(A, B) = 1 - T(A, B)$$

4.6 Learning rate

One of the most critical hyperparameter for the model's performance is the learning rate. This parameter determines how quickly the model's weights are updated during training. A small learning rate results in slow parameter updates, while a large one leads to faster updates .

If the learning rate is too low, the model may take an excessively long time to converge to a local minimum of the loss function. Conversely, if the learning rate is too high, the model may fail to converge altogether (Figure 10). This highlights the importance of choosing an appropriate learning rate.

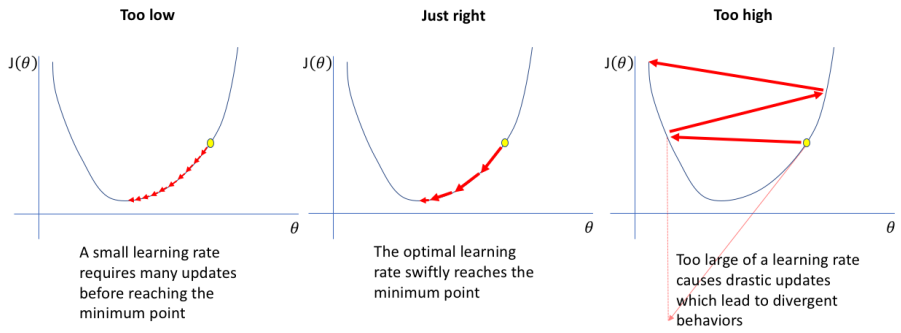

Figure 10 – Learning rate
https://www.jeremyjordan.me/content/images/2018/02/Screen-Shot-2018-02-24-at-11.47.09-AM.png

Choosing an appropriate learning rate is crucial, yet there is no method that guarantees finding the optimal value. Instead, it must be determined empirically by testing different values. However, training deep learning models is time-consuming and repeating the training process multiple times to fine-tune the learning rate can be computationally expensive.

4.6.1 ReduceLROnPlateau

One commonly used approach to address this issue is the ReduceLROnPlateau strategy. When the validation loss stops improving for a certain number of epochs defined by the user, the learning rate is automatically reduced by a predefined factor. The rationale is that if the model reaches a plateau, reducing the learning rate enables finer adjustments to the parameters, potentially allowing the model to escape the plateau and improve further.

4.6.2 Cyclical Learning Rate

Another approach is the Cyclical Learning Rate (CLR) method, proposed by Leslie N.Smith [15]. In this strategy, the learning rate varies periodically between two user-defined bounds : a minimum and a maximum. These values can be estimated using a learning rate range test conducted over a few epochs. Instead of maintaining a constant or monotonically decreasing learning rate, CLR introduces controlled oscillations. This helps the model to explore different regions of the loss landscape more effectively and prevents it from getting trapped in a suboptimal local minima, while also allowing for finer convergence during the downward phases of the cycle.

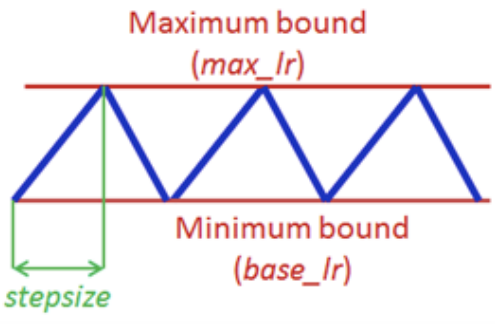

Figure 11 – Cyclical Learning Rate
https://arxiv.org/pdf/1506.01186

4.7 Image Preprocessing

To accelerate training, the model was trained on 224×224 images. However, some SPMN images have resolutions as high as 3840×2160 , with stars represented by only a few pixels. Simply resizing such large images to 224×224 would cause the stars to disappear. Instead of downscaling, the original image is divided into 224×224 pixel patches which are then fed individually to the model. This way, stars remain visible.

The main drawback of this approach is the loss of contextual information. In the full image, features like urban zones are easily recognizable, but when isolated into small patches, it becomes much harder to identify whether a patch corresponds to an urban area or not. The model did not confuse bolides or text colon with stars, largely because such examples were included in the dataset, and remain relatively easy to recognize, even on 224×224 patches. However, simply adding background patches to the dataset proved insufficient to fully address false detections (Figure 12). Even worse, attempts to reduce false positives by penalizing them more heavily resulted in fewer stars being detected overall, yet false positives persisted.

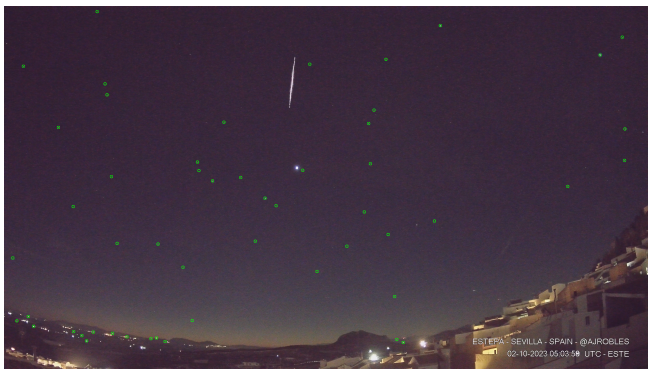

Figure 12 – Detected stars highlighted in green, including false positives

To address this issue, it was decided to detect urban zones beforehand. Urban zone detection was performed using a convolutional neural network that outputs a binary mask. The urban zone appears white, the other regions black. Although the model still requires 224×224 inputs, the prominence of urban zones ensures they remain visible even after resizing.

The dataset for urban zone segmentation was created using the open-source annotation tool CVAT. Segmentation masks were generated for 10 different observation stations. Due to variations in camera

orientation at each station, this resulted in 32 unique segmentation masks covering a total of 369 images.

When an image is passed through the urban zone segmentation model, it ouputs a 224×224 binary mask where pixels corresponding to urban areas are set to 1 and all other regions to 0. This mask is then inverted and multiplied with the input, effectively masking out the urban zones by setting them to black. The resulting image is then resized back to its original dimensions. This processed image (Figure 14) is subsequently divided into 224×224 patches and used as input for the star detection model.

Figure 13 – Image from test set before preprocessing

Figure 14 – Image from test set after preprocessing

4.8 Model training

Star segmentation remains a difficult task due to their size and the diversity of backgrounds. To guide the model during its learning process, an initial training phase was carried out on a simplified dataset. This preliminary dataset consisted of synthetic images containing artificially generated white dots on black backgrounds.

The purpose of this step was to orient the model toward detecting luminous sources in a clean and controlled setting. By learning to focus on such features early on, the model is better prepared to identify real stars during the subsequent training phase on more complex, real-world data.

During the second training phase, the models were trained on a dataset containing synthetic stars, bolides, and text patches. This dataset consisted of 221 bolide images, 545 text images, and 1313 star images. For each category, 20% of the data was set aside for validation.

In the third training phase, 160 supervised annotated images were added to the existing dataset. The purpose of introducing this dataset at the final stage was to leverage the learning from the previous phases and adapt the models to real-world data. The training process employed EarlyStopping if the validation loss did not improve for 20 consecutive epochs, training was stopped to prevent overfitting.

Initial results showed that the EfficientNetV2L and EfficientNetV2M architectures offered the best performances in terms of recall and precision (Figure 15). Yet, when testing the models on the test dataset, the models that detected the most stars were the EfficientNetV2S and EfficientNetV2M architectures. The ResNet and EfficientNetV2L models showed poor performances on the test dataset (Figure 16), likely due to their important number of parameters compared to the little size of the dataset (Figure 17).

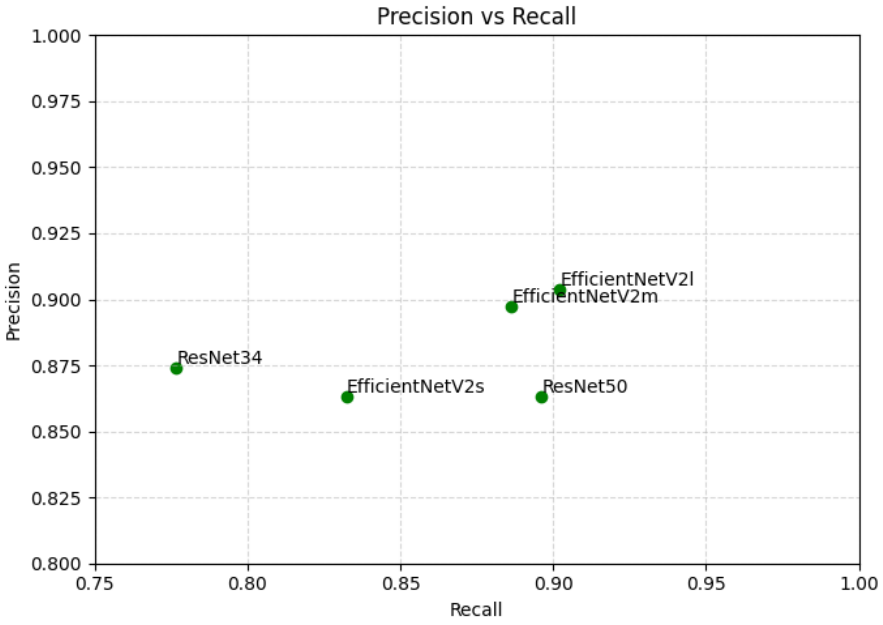

Figure 15 – Recall vs Precision

Figure 16 – Comparison of different models on a test image

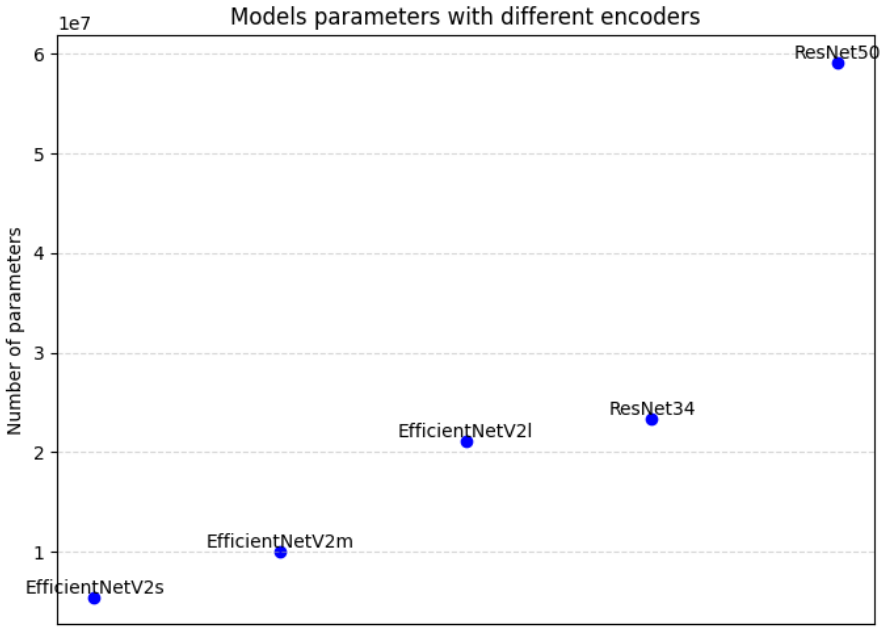

Figure 17 – Number of parameters in different encoder architectures

While EfficientNetV2S and EfficientNetV2M deliver comparable performance on most images, the EfficientNetV2M architecture consistently outperforms its counterpart when dealing with images featuring brighter backgrounds (Figure 18), likely due to its larger capacity and deeper feature representations, which enable better discrimination between stars and bright background noise.

Figure 18 – Comparison of EfficientNetV2S and EfficientNetV2M on bright test images

Based on these findings, the focus shifted to the EfficientNetV2M architecture, which yielded the best performance. The model was trained using four different loss functions: Dice, Jaccard, Binary

Cross-Entropy (BCE), and Tversky. As shown in Figure 19, the performance was similar across losses; however, testing on the test dataset revealed that the model trained with BCE detected substantially more stars.

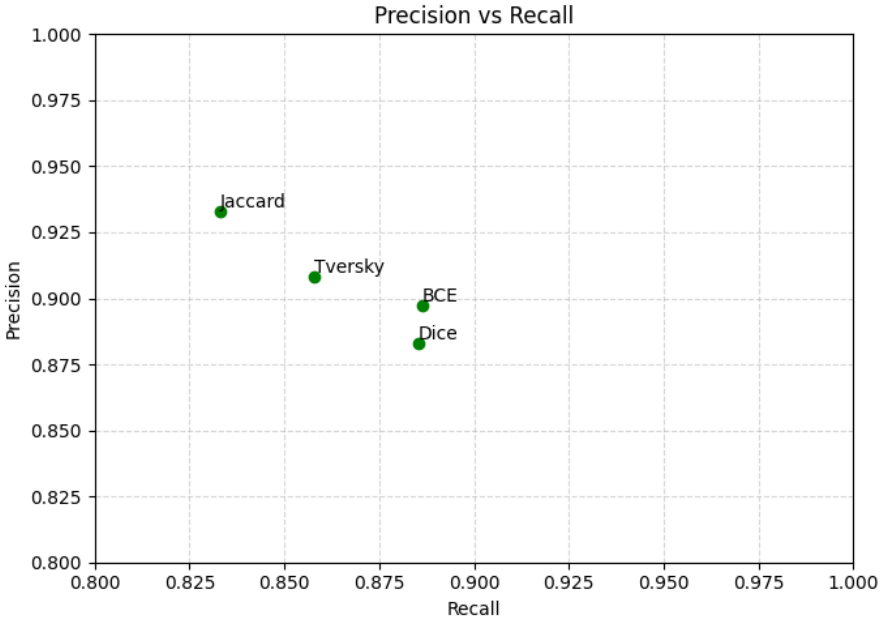

Figure 19 – Recall vs Precision for different loss functions

This behavior can be explained by the way each loss function penalizes errors. BCE treats each pixel independently and penalizes false negatives heavily, which encourages the model to label more pixels as positive to avoid missing potential stars. Dice loss, although based on overlap, shares this tendency since missing positives significantly reduces the intersection score, leading the model to be more liberal in its predictions. As a result, both losses tend to yield higher recall but also more false positives, lowering precision. In contrast, Tversky and Jaccard losses place stronger penalties on false positives, pushing the model toward more conservative segmentations. This often results in cleaner predictions with higher precision, but also means that some faint or small stars may be missed, lowering recall. Given the priority of maximizing star detection, the model trained with BCE was ultimately selected

Eventually, two learning rate schedulers were compared: ReduceOnPlateau and CyclicalLearningRate. For phase 1, the starting learning rate (LR) was set to 1e-3, and if the validation loss remained unchanged for 5 epochs, the LR was halved. In phase 2, the LR started at 1e-4. With the cyclical scheduler, the base LR ranged between 1e-3 and 1e-5 for phase 1 and between 1e-4 and 1e-6 for phase 2.

Figure 20 shows a phenomenon of overfitting: the training loss keeps decreasing while the validation loss increases. It can also be seen that adding the supervised images reduced the model's performance, as these images were harder to predict since they were not artificially generated. The same observation applies to Figure 21. However, in this case, no overfitting appears on the training curves.

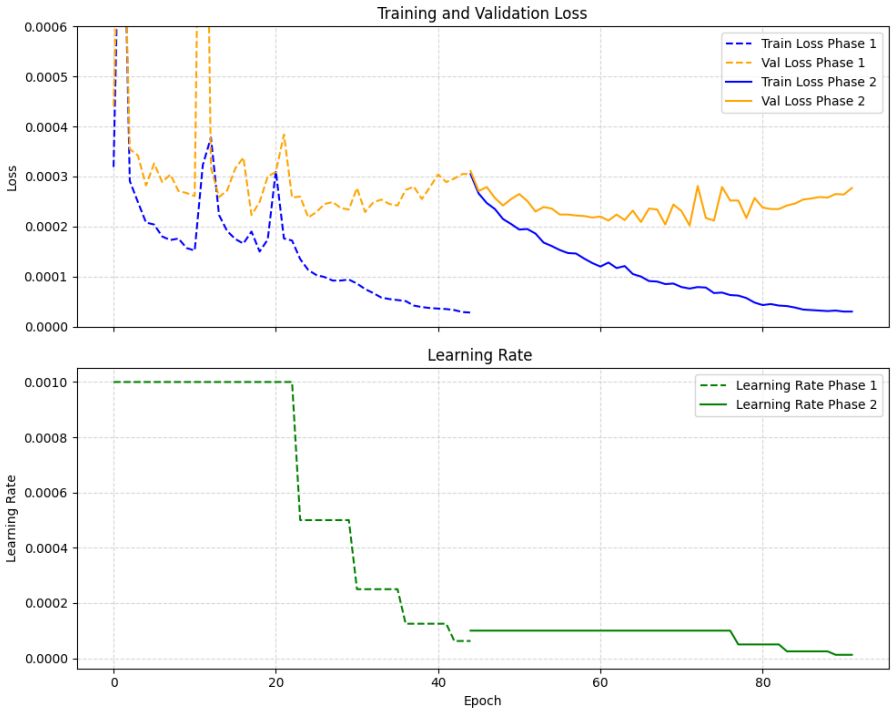

Figure 20 – Training and Validation Loss with ReduceOnPlateau

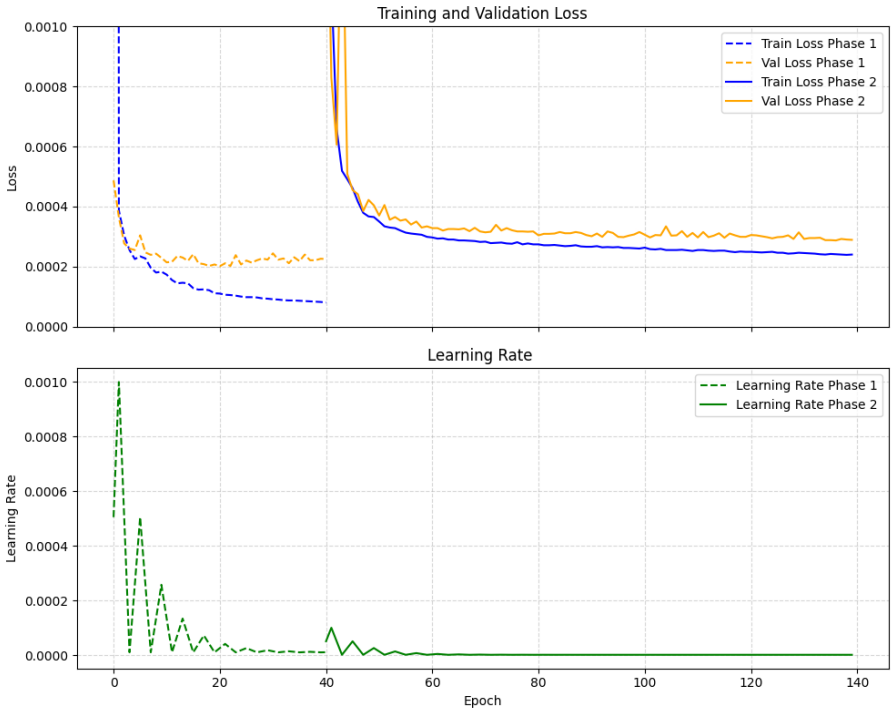

Figure 21 – Training and Validation Loss with CyclicalLearningRate

The model trained with ReduceOnPlateau outperforms its counterpart in terms of recall and precision (Table 2). The same observations can be made when testing the models on bright images (Figure 22).

Model	Precision	Recall
CyclicalLearningRate	0.8491	0.8261
ReduceOnPlateau	0.8975	0.8864

Table 2 – Model performance in terms of precision and recall

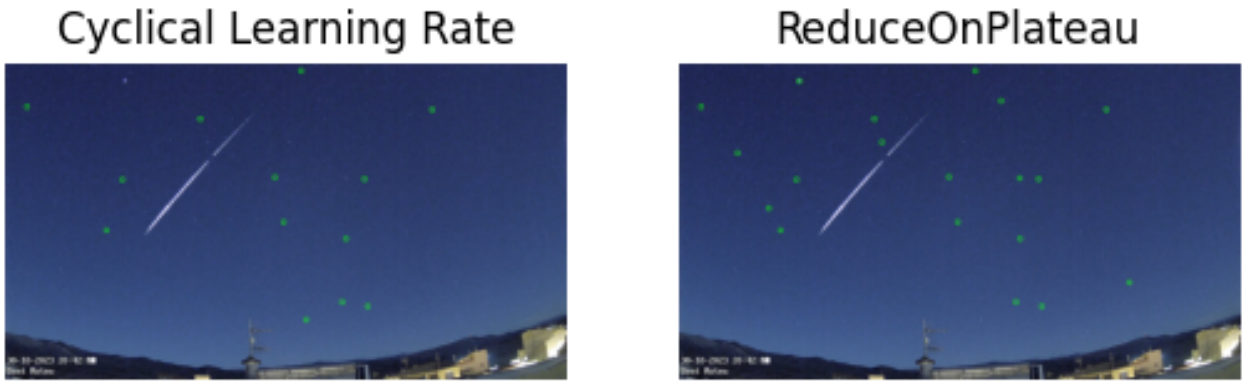

Figure 22 – Comparison of stars detected on a test image using the two models

4.9 Model Compression

4.9.1 Model Pruning

Model pruning is a widely used technique to reduce the size and complexity of deep neural networks by removing less important parameters. There are two main types of pruning:

— Structured pruning removes entire units such as channels, filters, or layers. This leads to a smaller, denser model with a simpler architecture and typically results in faster inference and reduced memory footprint. However, implementing structured pruning often requires careful redesign of the model and more complex re-training procedures.

— Unstructured pruning removes individual weights based on certain criteria (usually magnitude) without regard to their location in the network. This creates sparse weight matrices but retains the original model architecture.

In this work, due to time constraints only unstructured pruning was applied.

Specifically, we employed magnitude-based pruning, which removes weights with the smallest absolute values, under the assumption that these weights contribute least to the model's output. This was done iteratively: after each pruning step, the model was fine-tuned to recover accuracy, and pruning continued until the target sparsity was reached.

Formally, given a weight tensor W, a binary mask M is computed such that

$$W_{\text{pruned}} = W \odot M,$$

where

$$M_{i,j} = \begin{cases} 0 & \text{if } |W_{i,j}| < \tau, \\ 1 & \text{otherwise,} \end{cases}$$

and the threshold τ corresponds to the pruning percentile defining the fraction of weights to prune.

The pruned model demonstrates better overall efficiency compared to the original unpruned version. Although there is a slight decrease in recall and precision, the pruning process significantly reduces the number of non-zero parameters, which leads to a smaller model size and faster inference time.

Table 3 – Comparison of model performance and characteristics

Model	Precision	Recall	Inference Time (s)	Model Size (MB)
Baseline model	0.8975	0.8864	4.5	126
pruned model	0.8907	0.8862	3.8	42.3

Pruning helps improve the model's performance by eliminating less significant weights, which reduces model complexity and helps prevent overfitting. The removal of these small-magnitude weights acts as an implicit regularization, enabling the network to focus on more important features.

4.9.2 Half Precision Training

To optimize training and inference efficiency, the model parameters and inputs were converted to half precision (16-bit floating point). This reduces memory consumption and bandwidth requirements, which enables faster computation and inference time.

Table 4 summarizes the comparison between models trained and evaluated in full precision (FP32) and half precision (FP16).

Table 4 – Comparison between full precision and half precision training

Model	Precision	Recall	Inference Time (s)	Model Size (MB)
Full Precision (FP32)	0.8907	0.8862	3.8	42.3
Half Precision (FP16)	0.8904	0.8867	3	21.15

As observed, half precision training yields comparable performance metrics while significantly reducing model size and inference time.

5 Stars Identification

Once the stars were detected, the next step involved identifying them. As explained earlier, the identification part is crucial as each star has its own magnitude. The latter will be used to later recover the bolides magnitude. To identify stars, it is necessary to first introduce celestial coordinates.

5.1 Celestial coordinates

To identify any place on earth, we use its latitude and longitude, corresponding respectively to the distance in degrees to the equator and the distance in degrees to the prime meridian.

Likewise, in the sky, each object can be located by two numbers Right Ascension and Declination. Declination expressed in degrees measures the angular distance north or south of the celestial equator which is an imaginary projection of Earth's equator onto the celestial sphere.

Ascension on the other hand measures the angular distances eastward along the celestial equator from the vernal equinox and is expressed in hours, minutes, seconds.

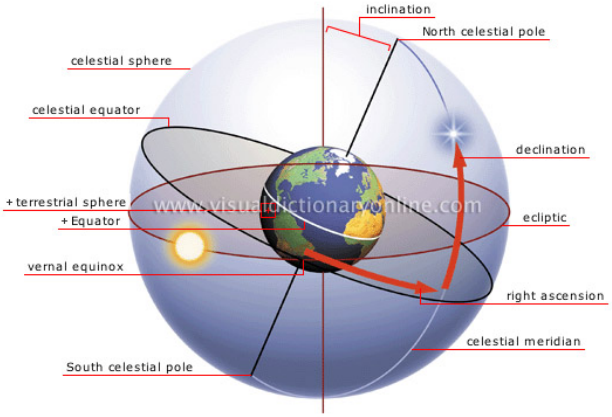

Figure 23 – Celestial coordinates
https://www.visualdictionaryonline.com/images/astronomy/astronomical-observation/celestial-coordinate-system.jpg

5.2 Automated identification

5.2.1 Resolution using Astrometry

The first method explored is a fully automated one using Astrometry.net [16]. Given an astronomical image, the software can identify stars and if the resolution is successfull produce a World Coordinate System object (WCS). It is a standard for mapping pixel coordinates in an astronomical image to celestial coordinates.

When used on raw SPMN images, astrometry struggles to detect the stars and hence the astronomical resolution fails. However, when provided with a noiseless image, a binary mask where stars are represented by white dots, the software is sometimes able to produce a WCS.

To do so, the stars detected by Astrometry.net are grouped into sets of four, called quads. For each quad, a geometric hashing process is applied to compute a unique, invariant descriptor. Specifically, given four stars A, B, C, D, the stars A and B define a local coordinate frame where A is set at the origin and B at position $(1, 1)$. The relative positions of C and D in this frame define the hash code:

$$\text{hash} = (x_C, y_C, x_D, y_D)$$

This code is invariant to translation, rotation, and scale, and is used to describe the quad's geometry robustly.

The computed hash code is then compared to a precomputed index containing millions of reference quads. The system searches for reference codes close to the query code in a four-dimensional hash space. Each matching entry proposes a hypothesis about the image's astrometric calibration, including its position, orientation, and scale on the sky.

Each hypothesis is evaluated using a Bayesian decision test. Let F denote the model where the alignment is correct (foreground), and B the model where it is not (background). The system computes the Bayes factor:

$$K = \frac{P(D \mid F)}{P(D \mid B)}$$

where D is the observed data (the star positions). The hypothesis is accepted if:

$$K > \frac{P(B)}{P(F)} \cdot \frac{u(TN) - u(FP)}{u(TP) - u(FN)}$$

Using conservative priors $P(F)/P(B) = 10^{-6}$ and utility values such that false positives are heavily penalized, the acceptance threshold becomes:

$$K > 10^9$$

If a hypothesis passes this threshold, meaning it predicts the positions of other stars in the image accurately, it is retained as the correct astrometric calibration.

If the astrometric resolution proves to be successfull, each detected star is projected to its celestial coordinates using the WCS created by Astrometry. These celestial coordinates are then compared to the celestial coordinates of stars having a magnitude inferior to 4 in the Hipparcos catalog. The detected star in the image is associated to its closest match.

In some cases however, the astrometric resolution fails, likely due to image distorsions that alter the relative positions of stars.

5.2.2 Image Distortion in Wide-Angle Cameras

Projecting a 3D point from the real world onto a 2D image plane involves several transformations. Initially, the point is converted from the world coordinate system to the camera coordinate system. This transformation preserves information and is achieved by applying a rotation and translation:

$$\mathbf{P}_{\text{camera}} = \mathbf{R} \cdot \mathbf{P}_{\text{world}} + \mathbf{t}$$

where:
— $\mathbf{P}_{\text{world}}$ is the 3D point in world coordinates,
— $\mathbf{R} \in \mathbb{R}^{3\times3}$ is the rotation matrix,
— $\mathbf{t} \in \mathbb{R}^3$ is the translation vector,
— $\mathbf{P}_{\text{camera}}$ is the 3D point in the camera frame.

In an ideal pinhole model, a 3D point (X, Y, Z) is projected onto the image plane as follows:

$$x = f_x \cdot \frac{X}{Z}, \quad y = f_y \cdot \frac{Y}{Z}$$

where f_x and f_y are the focal lengths in pixel units along the x and y directions.

However, real lenses deviate from this ideal behavior, especially when the field of view (FOV) is wide.

Due to imperfections in real lenses, especially wide-angle lenses, the projection of 3D points deviates from the ideal model. This deviation is known as distortion, and it can be categorized into two types: radial and tangential distortion.

Radial distortion occurs when light rays bend differently based on their distance from the optical center. It causes straight lines to appear curved. The distorted coordinates (x_d, y_d) are related to the undistorted normalized coordinates (x_u, y_u) by:

$$r^2 = x_u^2 + y_u^2$$

$$x_d = x_u \cdot (1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$

$$y_d = y_u \cdot (1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$

where k_1, k_2, k_3 are the radial distortion coefficients.

Tangential distortion arises when the lens is not perfectly parallel to the image plane, often due to misalignment during manufacturing or mounting. It is modeled as:

$$x_d = x_u + [2p_1 x_u y_u + p_2(r^2 + 2x_u^2)]$$

$$y_d = y_u + [p_1(r^2 + 2y_u^2) + 2p_2 x_u y_u]$$

where p_1 and p_2 are tangential distortion coefficients.

Distortion can significantly affect the precision of the observed positions. This makes astrometric solutions more challenging, especially in wide-field images. Implementing distortion correction could be a way to facilitate astrometric resolution for Astrometry.net.

5.3 Manual identification

When automatic star identification fails, the process is performed manually. The user specifies the SPMN station from which the video was recorded along with the corresponding date and time. This piece of information is used to generate a binary image showing the stars that could have been visible in the camera's field of view at that moment (Figure 24). The user then manually associates some detected stars in the image with projected catalog stars by clicking on the matching pairs.

Figure 24 – Manual Calibration

6 Bolide segmentation

6.1 Dataset

The bolide segmentation dataset was created using the open-source annotation tool CVAT (Computer Vision Annotation Tool). This software facilitates efficient and user-friendly annotation, including a feature that allows applying a YOLO model to a selected region of the image—in our case, the region containing the bolide. This method enabled the generation of segmentation patches with minimal manual effort.

Figure 25 – Original image and corresponding segmentation mask

To improve model robustness and reduce the risk of false positives, additional patches were manually annotated and included in the dataset, specifically covering regions containing stars and overlaid text. The final dataset was divided into three subsets: training, validation, and test, ensuring a balanced and representative distribution for performance evaluation.

6.2 Inference Strategy and Motion Zone Detection

As described previously, SPMN images have very high resolution, and downscaling them for faster processing would result in a significant loss of detail. To preserve spatial information, the bolide segmentation model, like the star segmentation network is applied to patches of size 224×224 extracted from the original image.

However, unlike star detection, which only requires identifying stars once per image, bolide detection must be performed across every frame of the video. For an input resolution of 3840×2180, approximately 180 patches are needed to cover the entire image. For a video containing 30 frames and an inference time of 0.5 seconds per patch, processing the entire video would take roughly 45 minutes.

To address this limitation, a Region of Interest (ROI) is first identified to localize the bolide's motion within the video. The segmentation model is then applied only within this ROI, significantly reducing the computational cost.

To determine the motion zone, we rely on the strong assumption that the bolide moves rapidly across frames, and that its trajectory can be inferred from this movement. To extract this motion information, optical flow is computed across consecutive frames, allowing us to detect regions where significant motion occurs, presumably corresponding to the bolide path.

The optical flow represents the apparent motion of pixel intensities between two consecutive frames of a video. In the implementation, the Farnebäck method is used to estimate a dense optical flow field between the previous grayscale frame $I_{t-1}(x, y)$ and the current frame $I_t(x, y)$.

The fundamental assumption is that the intensity of a pixel remains constant over time as it moves:

$$I(x, y, t) = I(x + u, y + v, t + 1)$$

By applying a first-order Taylor expansion, we obtain the optical flow constraint equation:

$$\frac{\partial I}{\partial x} \cdot u + \frac{\partial I}{\partial y} \cdot v + \frac{\partial I}{\partial t} = 0$$

where $u(x, y)$ and $v(x, y)$ represent the horizontal and vertical components of the motion vector at pixel (x, y).

The Farnebäck algorithm estimates this dense flow field by approximating the neighborhood of each pixel with a quadratic polynomial:

$$f(x) \approx x^\top Ax + b^\top x + c$$

This local polynomial model allows the method to capture smooth motion variations and improve robustness to noise.

The result is a dense flow field $\vec{v}(x, y) = (u(x, y), v(x, y))$ that describes the displacement vector for each pixel in the image.

To quantify the amount of motion in each frame, we convert the optical flow vectors from Cartesian to polar coordinates. Given a dense optical flow field $\vec{v}(x, y) = (u(x, y), v(x, y))$, this transformation computes the magnitude of motion at each pixel as:

$$\text{mag}(x, y) = \sqrt{u(x, y)^2 + v(x, y)^2}$$

The direction of motion is ignored in this step, as we are only interested in the intensity of movement.

Next, a thresholding operation is applied:

$$\text{mag}(x, y) = 0 \quad \text{if} \quad \text{mag}(x, y) < \theta$$

where θ is a predefined magnitude threshold. This step filters out low-amplitude motion, often caused by noise or minor pixel fluctuations, preserving only meaningful motion patterns.

Finally, the resulting magnitude map is accumulated across frames:

$$M(x, y) \mathrel{+}= \text{mag}(x, y)$$

where $M(x, y)$ represents the cumulative motion magnitude at pixel (x, y). This produces a motion heatmap that highlights regions with consistent and significant movement over time.

Figure 26 – Bolide Motion Zone

The bolide motion zone is subsequently divided into 224×224 patches, which are then fed to the segmentation model. This approach significantly reduces the number of model inferences required, thereby improving computational efficiency.

6.3 Model Training

The architecture used to segment the bolides is based on a U-Net-like design, similar to the network used for star segmentation. This choice is motivated by the fact that, in some video frames, bolides may appear or disappear very briefly and occupy only a small region of the image. In such cases, preserving fine spatial details is crucial for accurate detection. The skip connections inherent to the U-Net architecture effectively address this by allowing high-resolution features from the encoder to be directly combined with the decoder layers, enhancing localization accuracy. This makes U-Net particularly well suited for segmenting small and transient objects like bolides.

Given the results of the star segmentation experiments, and in order to reduce the number of training runs, only the models using EfficientNetV2S and EfficientNetV2M encoders were retained for further testing. The ResNet34, ResNet50, and EfficientNetV2L models were excluded, as they had shown poor performance in star segmentation, likely due to the dataset being too small to effectively train such large architectures. Finally, the same four loss functions and the same two learning rate schedulers were used in the subsequent experiments. The BCE loss once again yielded the best performance in terms of recall, while the Tversky loss achieved higher precision (Figure 27). However, as before, the model with the higher recall was selected to minimize the risk of missing bolides.

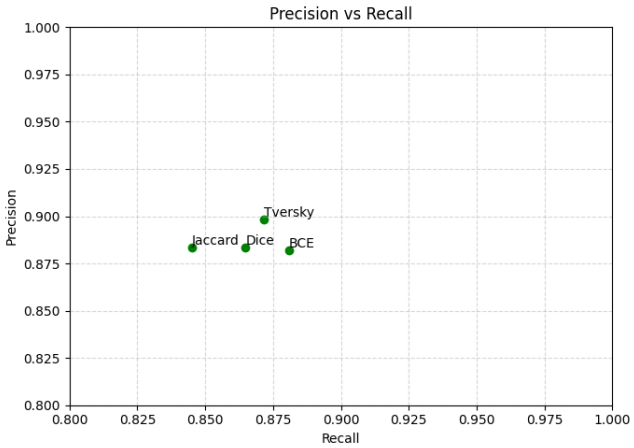

Figure 27 – Comparison of model performances using different loss functions

Both learning rate schedulers, ReduceOnPlateau and Cyclical Learning Rate, were evaluated. This time, the cyclical learning rate introduced instability during training (Figure 21), whereas the ReduceOnPlateau scheduler resulted in limited overfitting.

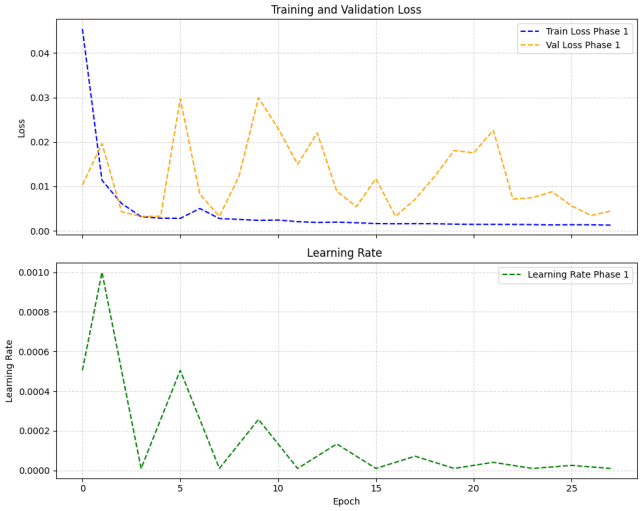

Figure 28 – Training curves with Cyclical Learning Rate

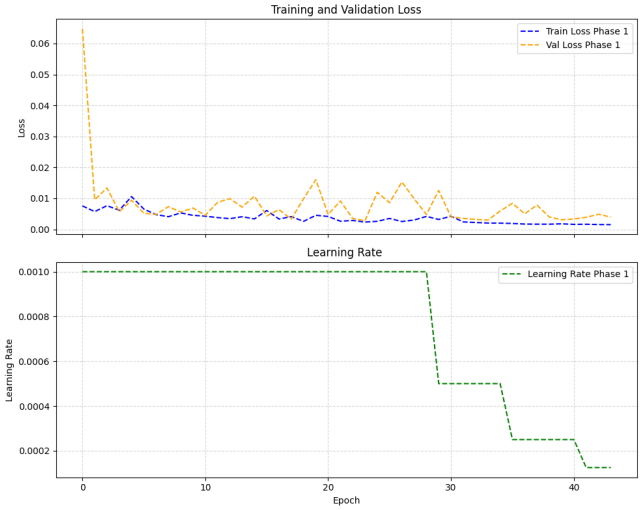

Figure 29 – Training curves with ReduceOnPlateau

It should be noted that very small bolides are sometimes not detected, suggesting that further improvements could focus on enhancing sensitivity to smaller objects.

6.4 Model Compression

Again, the model was pruned and half precision was used to reduce inference time and storage.

Table 5 – Comparison of model performance and characteristics

Model	Precision	Recall	Model Size (MB)
Baseline model	0.8822	0.8808	126
pruned model	0.8836	0.8797	21.15

7 Interface

When the program is launched, a user interface is displayed that allows the user to select an image file for calibration and a video file for analysis.

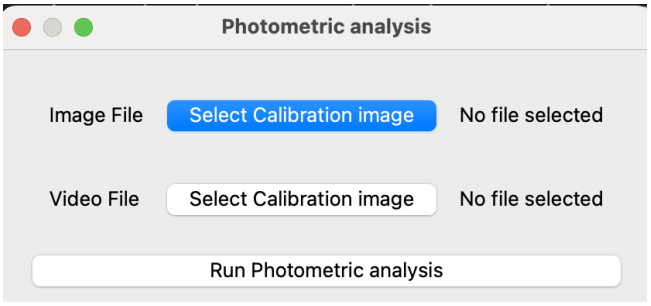

Figure 30 – User interface for selecting the calibration image

Upon clicking the *Run Photometric Analysis* button, the processing pipeline is initiated.

7.1 Calibration

During the calibration phase, the program first attempts automated calibration. If this step fails, manual calibration is triggered, as described earlier. Once the stars are successfully identified, their luminosities are extracted for further processing.

To accurately estimate the luminosity of stars in a grayscale astronomical image, a two-dimensional (2D) Gaussian model is fitted to image patches centered around detected star coordinates. The 2D Gaussian function is defined as

$$G(x, y) = A \exp\left(-\left(\frac{(x - x_0)^2}{2\sigma_x^2} + \frac{(y - y_0)^2}{2\sigma_y^2}\right)\right) + C,$$

where A is the amplitude, (x_0, y_0) denotes the center, σ_x and σ_y are the standard deviations along the x and y axes respectively, and C is the background offset. The fitting is performed using non-linear least squares optimization on local patches extracted from the image. The integrated luminosity is computed as $A \cdot 2\pi\sigma_x\sigma_y$. Optionally, subpixel recentering is performed by locating the pixel of maximum intensity within the patch before fitting, thereby improving accuracy in the presence of centroiding errors.

A dataset containing among others stars magnitude and luminosities is returned.

7.2 Bolide Segmentation

The next phase is bolide segmentation. First, all frames of the video are analyzed to detect the bolide motion zone and determine the region of interest (ROI). The model makes predictions only when movement is detected within this motion zone using optical flow. Prediction stops when the bolide head is not going forward anymore. For each processed frame, the bolide segmentation mask, the bolide's head, and the motion zone can be visualized (see Figure 31).

Figure 31 – Visualization available for each frame

The magnitude of the bolide is estimated by photometric comparison with reference stars of known magnitudes. Given the integrated luminosity L_{bolide} obtained via 2D Gaussian fitting, and the luminosity L_{ref} of a reference star with known magnitude m_{ref}, the bolide's magnitude m_{bolide} is computed using the relation

$$m_{\text{bolide}} = m_{\text{ref}} - 2.5 \log_{10} \left(\frac{L_{\text{bolide}}}{L_{\text{ref}}} \right).$$

The photometric profile of the bolide is plotted using this formula.

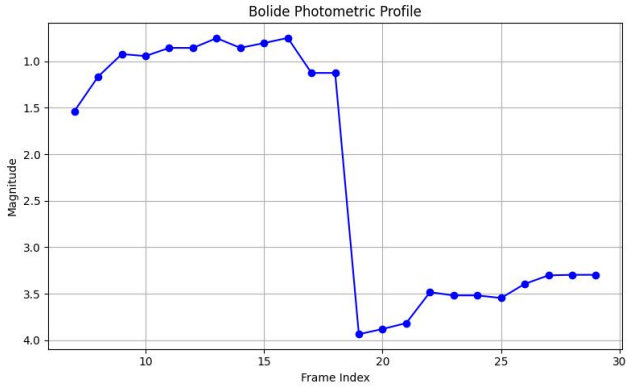

Figure 32 – Bolide Photometric Profile

8 Conclusion

This report presented the development of a pipeline dedicated to the detection and photometric analysis of bolides observed by the SPMN network. The approach combines deep learning-based segmentation techniques with astrophysical calibration methods to extract reliable photometric profiles from high-resolution video recordings.

The system includes two major components: automatic star detection and identification, and bolide segmentation across frames. The star detection module identifies stars in calibration images and retrieves their magnitudes, allowing for photometric calibration using catalogs. The bolide segmentation network, based on a U-Net architecture, is applied to patches of the video frames, with a region of interest determined via optical flow to minimize unnecessary computations and drastically reduce inference time.

An interactive interface enables users to select input data, monitor the process, and visualize intermediate results, such as star matches, bolide masks, and extracted profiles.

Despite the promising results, several improvements could be made to enhance performance and robustness:

- Lens distortion correction: Applying geometric distortion correction during the calibration phase would improve astrometric precision and facilitate the use of astrometry tools such as Astrometry.net.
- Improved detection of small bolides: In some cases, bolides appear extremely small in the image and escape detection. Dataset augmentation could address this issue.
- Star detection in complex scenes: Star detection remains challenging when noise interferes with the background.

9 Appendix

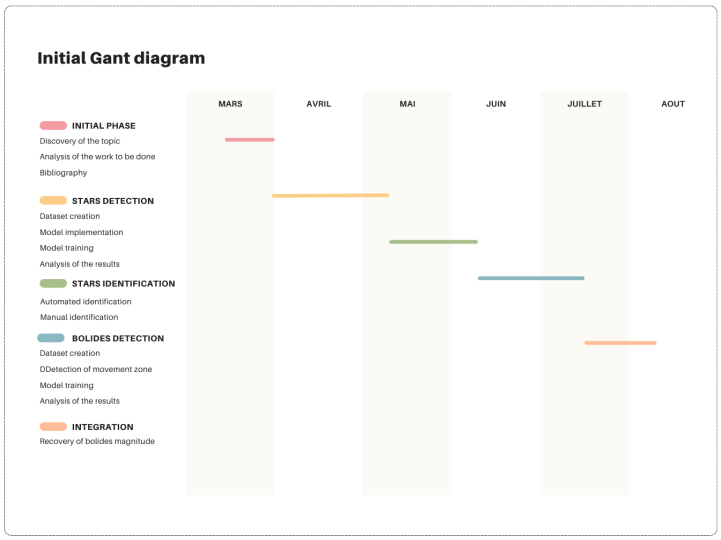

Figure 33 – Initial Gant Diagram

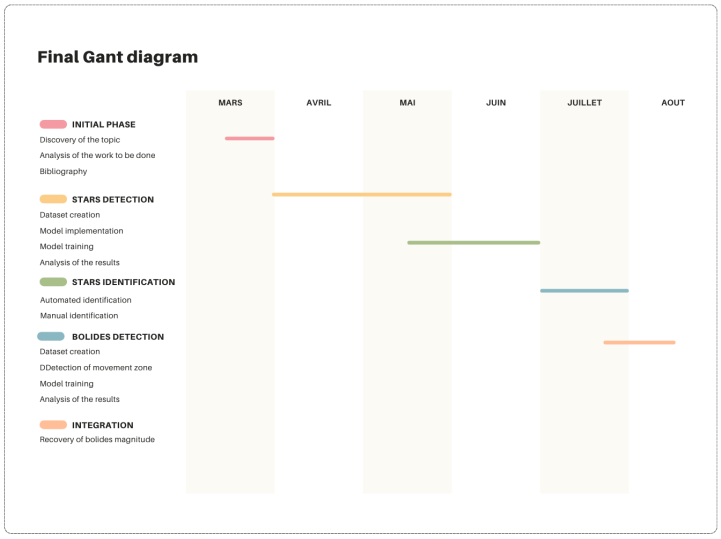

Figure 34 – Final Gant Diagram

List of Figures

List of Tables

References

[1] Hongrui Zhao, Michael F. Lembeck, Adrian Zhuang, Riya Shah, and Jesse Wei. Real-time convolutional neural network-based star detection and centroiding method for cubesat star tracker. *IEEE Transactions on Aerospace and Electronic Systems*, page 1–13, 2025. 7

[2] Marco Mastrofini, Ivan Agostinelli, and Fabio Curti. Design and validation of a u-net-based algorithm for star sensor image segmentation. *Applied Sciences*, 13(3), 2023. 7

[3] Colin J Burke, Patrick D Aleo, Yu-Ching Chen, Xin Liu, John R Peterson, Glenn H Sembroski, and Joshua Yao-Yu Lin. Deblending and classifying astronomical sources with mask r-cnn deep learning. *Monthly Notices of the Royal Astronomical Society*, 490(3):3952–3965, October 2019. 7

[4] Ryan Hausen and Brant E. Robertson. Morpheus: A deep learning framework for the pixel-level analysis of astronomical image data. *The Astrophysical Journal Supplement Series*, 248(1):20, May 2020. 7

[5] Eloy Peña-Asensio, Josep M. Trigo-Rodríguez, Pau Grèbol-Tomàs, David Regordosa-Avellana, and Albert Rimola. Deep machine learning for meteor monitoring: Advances with transfer learning and gradient-weighted class activation mapping. *Planetary and Space Science*, 238:105802, 2023. 7

[6] T. Sennlaub et al. Machine learning classification of meteor candidates using the allsky7 fireball network. *arXiv preprint arXiv:2206.03115*, 2022. 7

[7] D. Cecil and M. Campbell-Brown. The application of convolutional neural networks to the automation of a meteor detection pipeline. *Planetary and Space Science*, 186:104920, 2020. 7

[8] Peter S Gural. Deep learning algorithms applied to the classification of video meteor detections. *Monthly Notices of the Royal Astronomical Society*, 489(4):5109–5118, 09 2019. 7

[9] Shumaylov-Z. Zhao Y. et al Shumailov, I. Ai models collapse when trained on recursively generated data. *Nature 631, 755-759*, 2024. 9

[10] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015. 10

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015. 10

[12] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks, 2020. 10

[13] Mingxing Tan and Quoc V. Le. Efficientnetv2: Smaller models and faster training, 2021. 12

[14] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017. 14

[15] Leslie N. Smith. Cyclical learning rates for training neural networks, 2017. 18

[16] Dustin Lang, David W. Hogg, Keir Mierle, Michael Blanton, and Sam Roweis. Astrometry.net: Blind astrometric calibration of arbitrary astronomical images. *The Astronomical Journal*, 139(5):1782–1800, March 2010. 28