# Satellite Image Classification System

Aseel Ahmad, Shifaa Khatib, Bashar Naser
Instructor: Dr. Boaz Ben-Moshe
Course: Introduction to Space Engineering – Final Project
Ariel University

June 2025

**Abstract**

This project presents a modular onboard classification system for satellite images. It focuses on three detection tasks: horizon presence, sun flare, and overall image quality. The system integrates individual CNN models into a unified pipeline that also handles compression of useful images, enabling efficient pre-transmission filtering and prioritization in CubeSat systems.

## 1 Introduction

The increasing use of CubeSats and small satellites in Earth observation has brought new challenges related to onboard computing and data transmission. One significant limitation is the restricted bandwidth available for downlinking captured images to ground stations. As a result, there is a need for autonomous onboard systems that can evaluate image utility and reduce the transmission of low-quality or redundant data.

This project proposes a fully automated system to classify and filter satellite images using deep learning models. The system comprises three independent binary classifiers that identify the presence of a visible Earth horizon, detect sun flares or glare, and evaluate the image's visual quality. These classifiers are then combined into a unified pipeline that determines whether an image should be kept, discarded, or compressed for transmission.

By incorporating data preprocessing, model training, image evaluation, and adaptive compression, this project offers a lightweight and practical solution for onboard satellite image filtering. It supports more efficient data transmission and paves the way for intelligent satellite operations in real-time.

## 2 System Architecture

This project implements a **modular and automated pipeline** for onboard satellite image processing, tailored for CubeSats and other small satellites. It focuses on three main detection goals:

- Horizon Detection
- Sun Flare (Glare) Detection
- Image Quality Evaluation

Only images classified as "good" are compressed and sent back to Earth, conserving valuable bandwidth. The system is built around deep learning classifiers, organized as independent modules that are integrated into a unified decision pipeline.

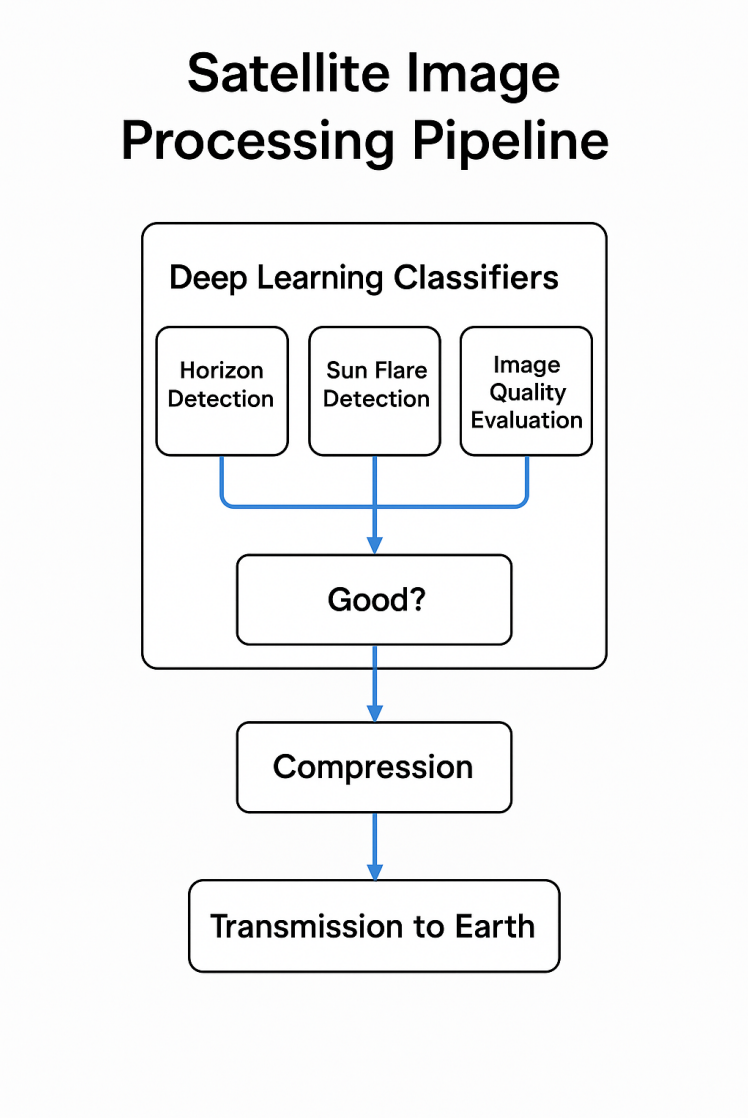


Figure 1: Overview of the Unified Classification and Compression Pipeline

## Three Detection Goals

Each detection task is performed by a binary classifier trained on labeled satellite imagery. Below is a breakdown of each task:

### Horizon Detection

**Goal:** Detect whether the Earth's horizon is visible in the image.

- **Dataset:** `dataset/classification_sets/horizon_detection/`
  - `horizon/` → label = 1
  - `no_horizon/` → label = 0
- **Model:** `src/models/horizon_detector.py`
- **Evaluation Script:** `src/detection/horizon_evaluation.py`

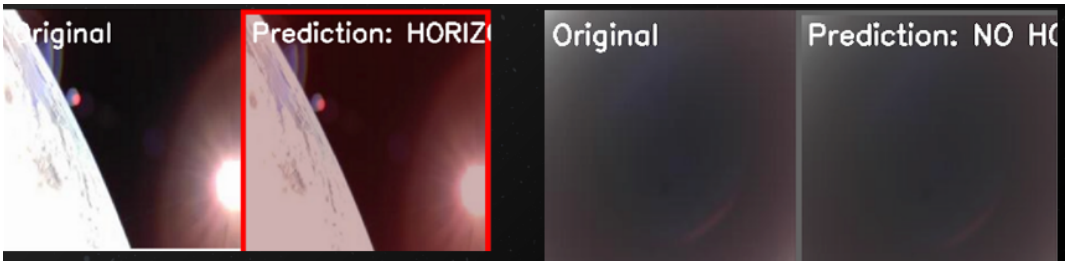


Figure 2: Sample predictions from the Horizon Detector model. The left side shows the original satellite images, and the right side shows the model's predictions: `HORIZON` or `NO HORIZON`.

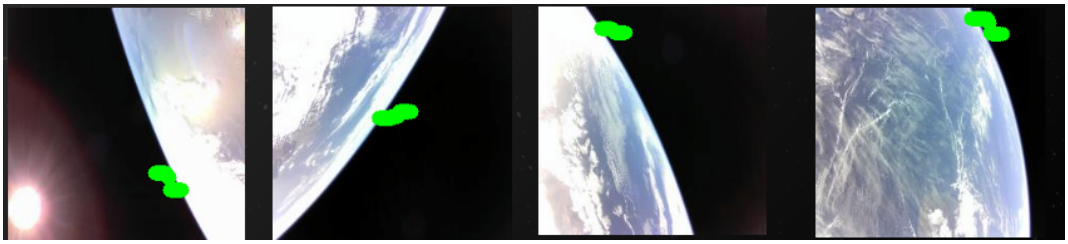

Figure 3: Visual examples of horizon detection on satellite images. Green markers highlight the detected horizon region in each image, demonstrating the model's spatial accuracy.

### Flare (Sun Glare) Detection

**Goal:** Detect overexposed areas caused by direct sunlight or lens flare.

- **Dataset:** `dataset/classification_sets/flare_detection/`
  - `flare/` → label = 1
  - `no_flare/` → label = 0
- **Model:** `src/models/flare_detector.py`

- **Evaluation Script:** `src/detection/flare_evaluation.py`

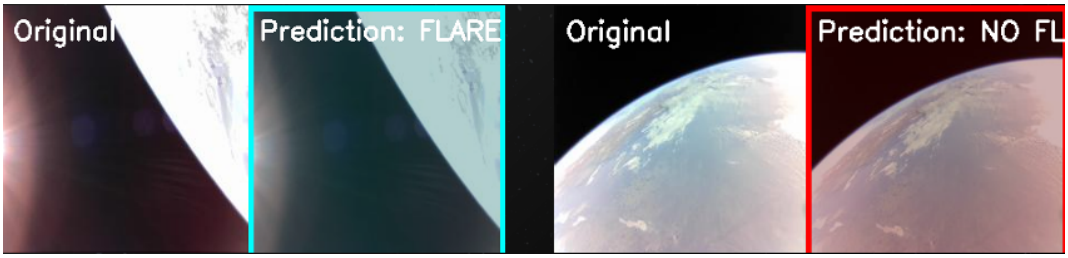


Figure 4: Sample predictions from the Flare Detector model. Left: images with visible sun flare correctly classified as `FLARE`. Right: clear images correctly classified as `NO FLARE`.

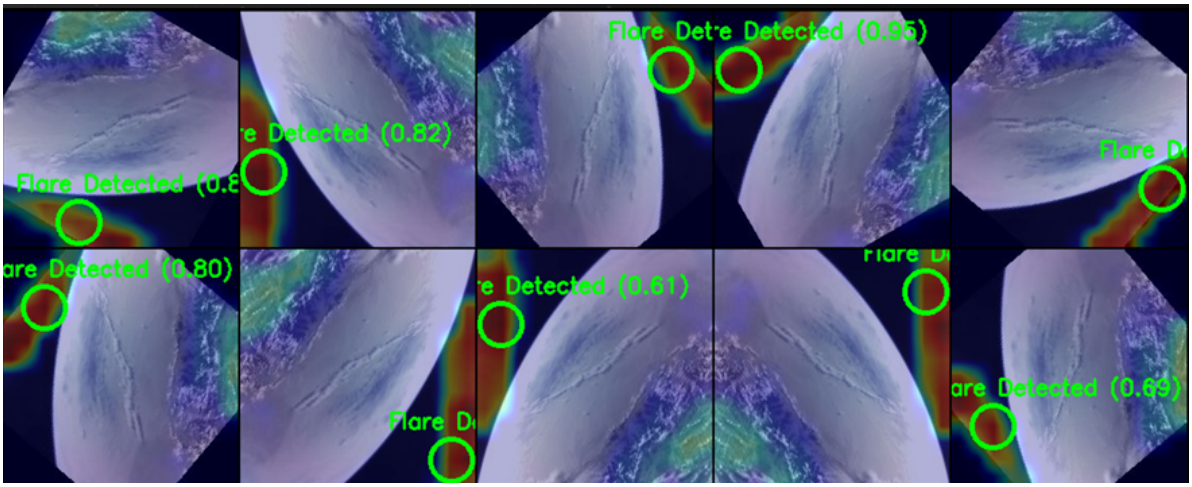


Figure 5: Flare Detector – prediction examples with model confidence. Each green circle indicates a region where the model detected flare presence, along with the predicted probability. These visualizations demonstrate the detector's robustness across varied illumination and angles.

### Image Quality Evaluation

**Goal:** Assess whether the image is sharp and usable, or blurred/overexposed.

- **Dataset:** `dataset/classification_sets/quality_detection/`
  - `good/` → high-quality image
  - `bad/` → unusable image
- **Model:** `src/models/quality_detector.py`
- **Evaluation Script:** `src/detection/quality_evaluation.py`

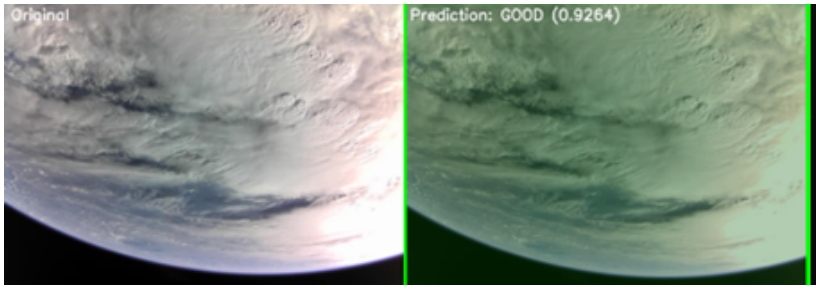


Figure 6: Sample output from the Image Quality Evaluator. The left image is the original satellite input, and the right image shows the model's prediction: `GOOD` with a confidence score of 0.9264.

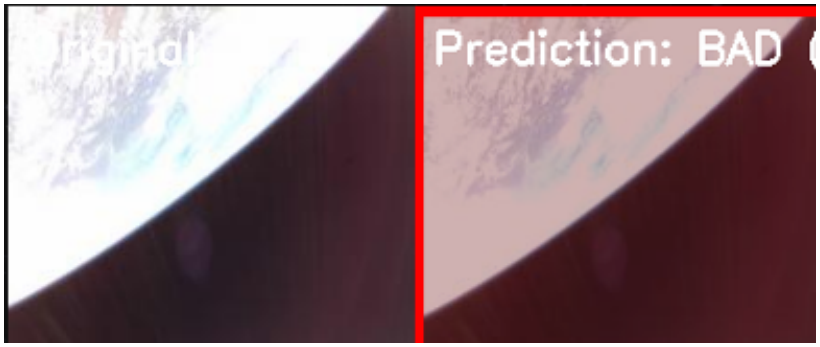


Figure 7: Prediction example from the Quality Evaluator. The image is identified as `BAD` due to overexposure and lack of visible surface detail, with high model confidence.

Each classifier uses pretrained weights stored in the `models/` folder (e.g., `flare_detector_best.pth`).

## Unified Pipeline Explanation

The unified classification and filtering logic is implemented in:

- `src/models/unified_classifier. py`

This script sequentially runs the three independent classifiers and makes a global decision for each image.

### Decision Flow

1. Run Horizon Detector
2. Run Flare Detector
3. Run Quality Detector

### Decision Logic

- If no horizon is detected → **discard image**
- If flare is detected → **discard image**
- If quality is bad → **discard image**
- Otherwise → **compress and transmit image**

### Compression

Compression is handled by:

- `src/compression/compress.py`

  It optimizes the image for bandwidth efficiency while maintaining critical visual features.

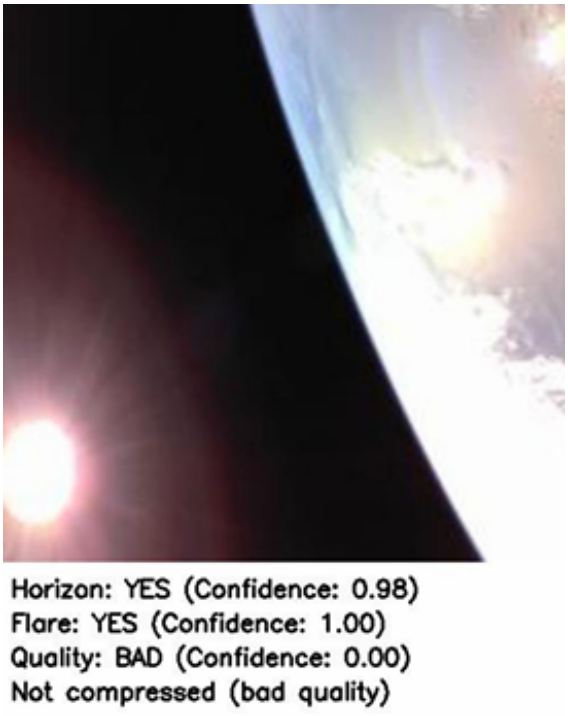


Figure 8: Example output from the unified classification pipeline. The image is correctly identified as containing both a visible horizon and solar flare, but is rejected for transmission due to low visual quality.

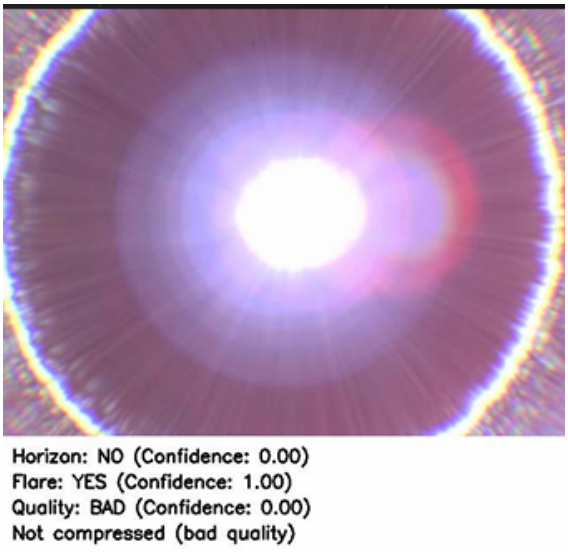


Figure 9: Unified pipeline prediction example. The model detects the absence of a visible horizon, presence of a strong flare, and poor image quality. As a result, the image is discarded and not compressed.


Figure 10: Unified pipeline prediction example. Although the horizon is visible and no flare is detected, the image is flagged as low quality and therefore not compressed. This demonstrates the selective nature of the transmission filter.

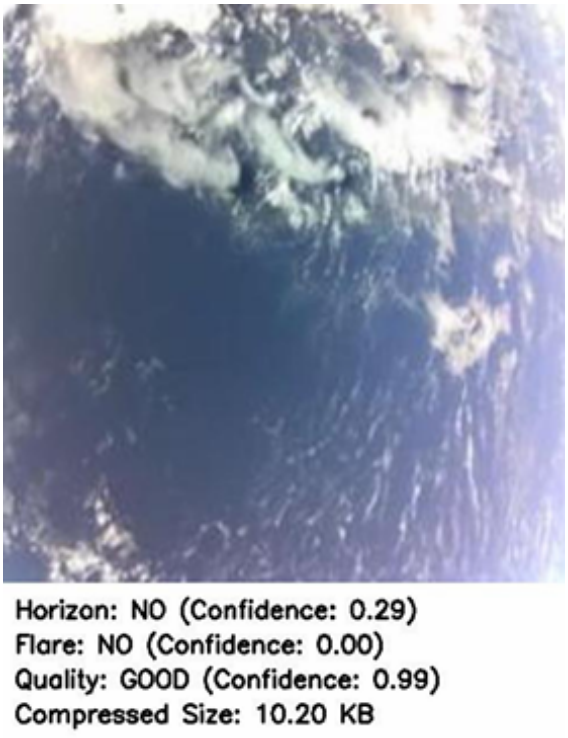


Figure 11: Unified pipeline prediction example. Despite the absence of both horizon and flare, the image is classified as high quality and is therefore compressed and marked for transmission.


Figure 12: Unified pipeline prediction example. The image contains a visible horizon and no flare, but is rejected due to poor visual quality, illustrating the importance of quality filtering in transmission decisions.


Figure 13: Unified pipeline prediction example. The model detects the absence of a horizon, the presence of a flare, and extremely low image quality, resulting in rejection and no compression.

# 3 Dataset

## Source and Origin

The dataset used in this project originates from the **BIRDS-3/4 CubeSat missions**, available via *Mendeley Data*. These missions provide real satellite imagery captured by small-scale satellites in Low Earth Orbit (LEO). The raw data includes a variety of orbital scenes, such as Earth's surface, horizon boundaries, outer space, and solar flares. These images were used to construct task-specific classification sets for training and evaluating onboard models.

## Original Categories

The unprocessed dataset contains the following primary categories:

- `earth/` – General Earth surface imagery.
- `horizon/` – Images with the Earth's horizon visibly present.
- `space/` – Frames containing deep space or star fields.
- `sunburn/` – Images corrupted by sun flares or strong exposure.

## Preprocessing Steps

To ensure uniformity and model readiness, the following preprocessing steps were applied:

1. **Resizing:** All images were resized to `224×224` pixels using bicubic interpolation to standardize input dimensions across tasks.
2. **Image Format Conversion:** All images were saved in '.jpg' format and converted to RGB mode if necessary.
3. **Augmentation (Training Only):**
   - Random horizontal/vertical flips
   - Rotation (±15 degrees)
   - Random brightness enhancements
   - Cropping and padding to preserve content
4. **Normalization:** Pixel values were normalized using standard PyTorch torchvision transforms.
5. **Handling Corrupted Images:** The preprocessing pipeline enables `ImageFile.LOAD_TRUNCATED_IMA = True` to avoid crashes from incomplete images.
6. **Directory Structure Creation:** Helper functions created necessary folders, and classes were mapped explicitly to numeric labels for training consistency.
7. **Train/Validation/Test Splitting:** Each dataset was split into:
   - training set
   - validation set
   - test set

   The split was stratified to ensure balanced class distribution across sets.

## Loading and Usage

The dataset was loaded using a custom PyTorch `Dataset` class defined in `src/data/dataset.py`. It dynamically maps subfolder labels to integer classes and applies real-time transformations during training. It is compatible with PyTorch's `DataLoader` for efficient batch loading.

The dataset plays a critical role in enabling the development of task-specific classifiers for CubeSat operations. Its curated structure, balanced labels, and robust preprocessing make it suitable for real-time onboard filtering and compression of satellite imagery.

# 4 Model Training

Each of the three classification models (Horizon Detector, Flare Detector, and Quality Evaluator) was trained independently using a modular PyTorch-based pipeline. The training code is structured in separate scripts for each task, allowing for flexible experimentation and isolated evaluation.

## Architecture and Initialization

Each model is a convolutional neural network (CNN) tailored for binary classification and defined in the corresponding module under `src/models/`. The training script dynamically loads the model, initializes its weights, and moves it to the appropriate device (CPU or GPU) using:

- `get_device()` – Detects CUDA availability
- `set_seed()` – Ensures reproducible runs

## Training Configuration

The training procedure is encapsulated in a generic function `train_model()`, which handles both training and validation phases per epoch. Key configurations used:

- Optimizer: Adam
- Loss function: CrossEntropyLoss
- Batch size: 32
- Image size: 224×224
- Learning rate: 0.001
- Epochs: 20–25

A progress bar is shown using `tqdm`, and model metrics are printed after each epoch. The model operates in two distinct modes:

- `train()` mode – Enables backpropagation and dropout
- `eval()` mode – Disables gradient updates for validation

## Metrics and Evaluation

For each epoch, the script tracks:

- **Loss:** Averaged for both training and validation sets
- **Accuracy:** Ratio of correct predictions over total samples

The results are stored in a dictionary `history{}` containing four lists: `train_loss`, `val_loss`, `train_acc`, `val_acc`

These metrics can be later plotted or exported for post-analysis.

## Model Checkpointing

At the end of each validation phase, the model's accuracy is compared to the best value so far. If improved, the model's weights are saved as:

- `models/horizon_detector_best.pth`
- `models/flare_detector_best.pth`
- `models/quality_detector_best.pth`

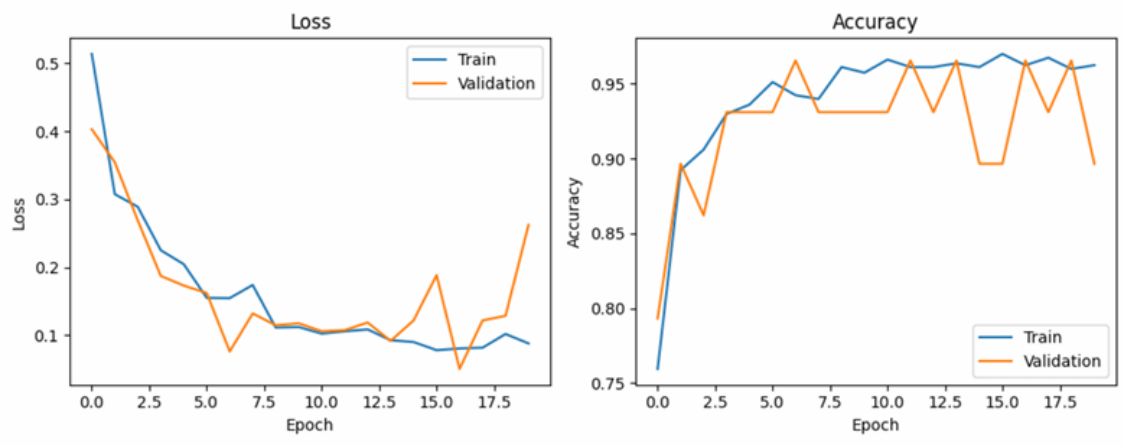


Figure 14: Training and validation loss and accuracy curves for the Horizon Detector model over 20 epochs. The plots show rapid convergence and stable validation performance, indicating effective training.

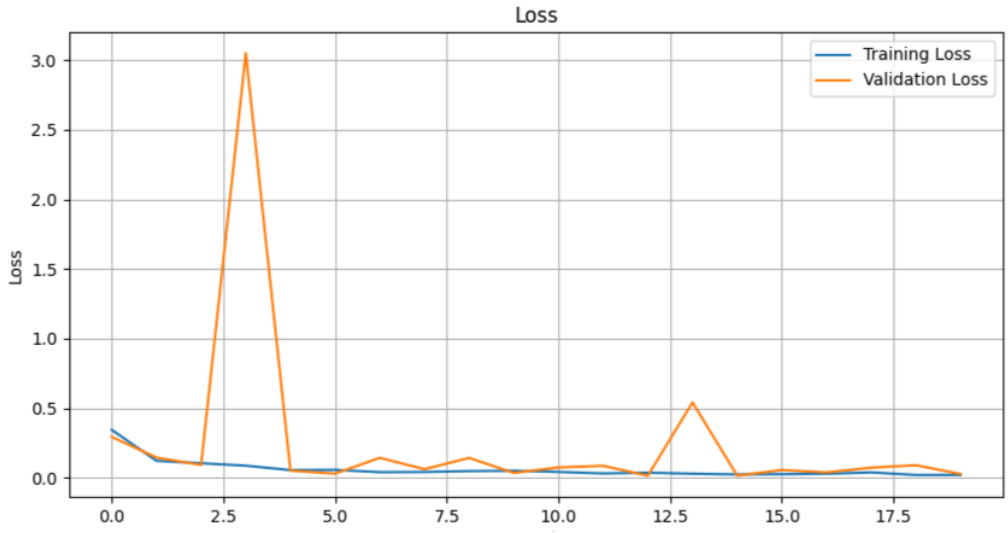


Figure 15: Flare Detector – accuracy over epochs. The model quickly achieves high training and validation accuracy, with minor fluctuations in some epochs due to flare image variance.

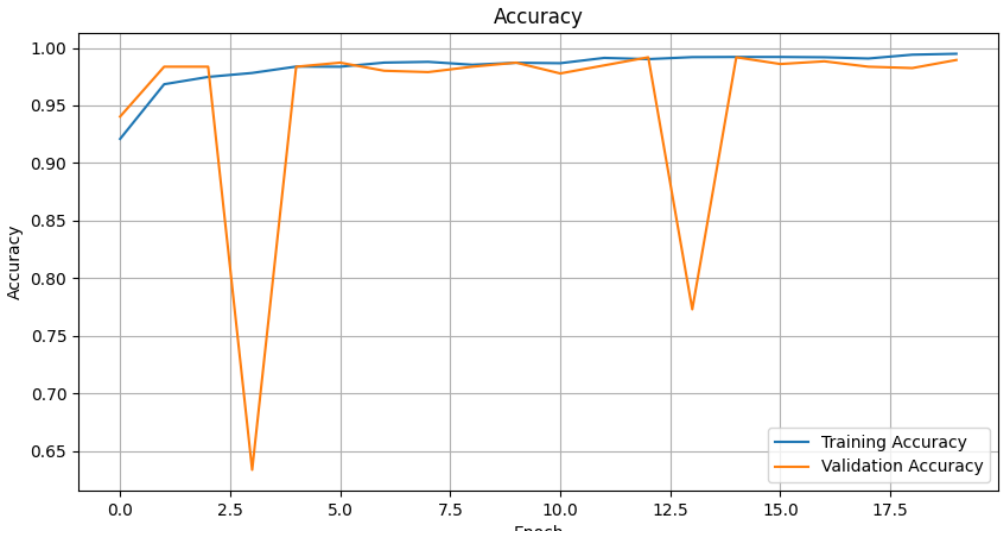


Figure 16: Flare Detector – loss over epochs. The validation loss is generally stable, with a few spikes possibly related to edge cases or lighting artifacts in specific batches.

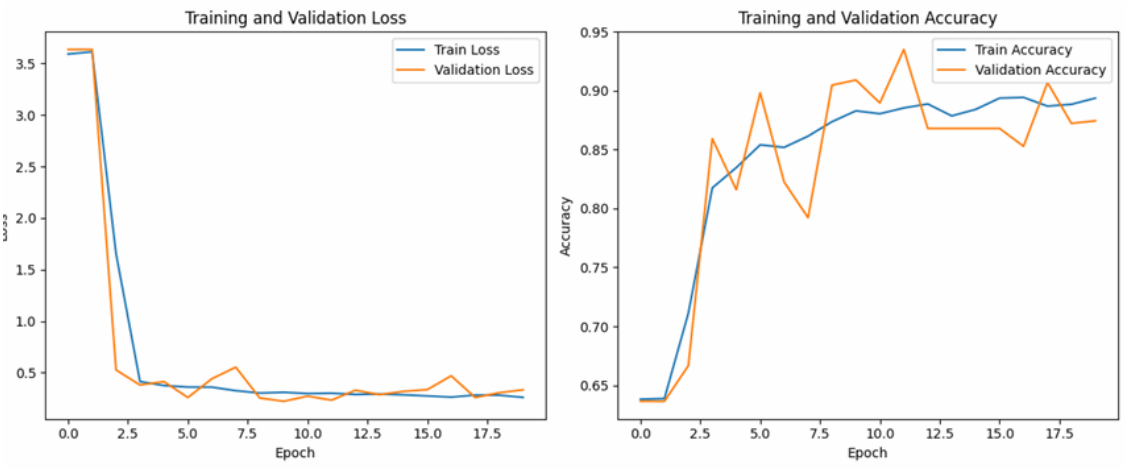


Figure 17: Training and validation performance of the Image Quality Evaluator. The model shows fast convergence and stable validation behavior, with accuracy peaking above 90% after several epochs.

## Command-Line Interface

Each training script accepts configurable parameters via the command line. A sample command to train the horizon detector is:

```
python src/models/train_horizon_detector.py \
    --batch_size 32 \
    --img_size 224 \
    --num_epochs 20 \
    --learning_rate 0.001
```

Similar commands apply for the flare and quality models by invoking the respective training scripts.

### Output and Integration

The final model weights are stored in the `models/` directory and later used during inference by the unified pipeline. The training history and saved checkpoints provide a complete trace of the model's learning process and are critical for reproducibility and future fine-tuning.

## 5 Conclusion and Future Work

This project presents a complete and efficient onboard image classification system for CubeSats. It combines three lightweight binary classifiers to evaluate satellite images for horizon presence, solar flare impact, and visual quality. The unified decision pipeline enables intelligent pre-transmission filtering, ensuring that only meaningful and high-quality images are compressed and downlinked.

### Future Work

- **Multi-Label Prediction:** Replace the three separate classifiers with a unified multi-label model that can jointly predict multiple image attributes (e.g., horizon, flare, and quality) — reducing inference time and improving consistency.
- **Data Expansion and Diversity:** Enrich the training dataset with more challenging satellite images, including nighttime views, cloud cover, extreme lighting, and different Earth regions, to improve generalization and robustness.
- **Semantic Segmentation:** Extend the classification approach to pixel-level segmentation to locate and outline features like Earth's edge, flare regions, or image defects — enabling more precise decisions.

## 6 System Requirements

The system was developed and tested using the following configuration:

- Python 3.9, PyTorch 1.13, torchvision 0.14
- CUDA 11.7 (optional for GPU acceleration)
- Training conducted on NVIDIA RTX 3060 (for experimentation)
- Memory footprint of each model $<$ 2MB (post-training)

For onboard deployment, the system is compatible with lightweight platforms such as Raspberry Pi 4 or Jetson Nano.

## References


[1] BIRDS-3 and BIRDS-4 CubeSat Mission Imagery, Mendeley Data. `https://data.mendeley.com`

[2] Paszke et al., "PyTorch: An Imperative Style, High-Performance Deep Learning Library," NeurIPS 2019.

[3] da Costa-Luis, "tqdm: A Fast, Extensible Progress Bar for Python and CLI," GitHub, 2022.

[4] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," Nature, vol. 521, no. 7553, pp. 436–444, 2015.