SUMMER OF INNOVATION

LUNAR VISION

SLEEPYBOIS

Varshith M Suhas RP Prajwal K

Approach

The goal of this project was to develop an Object Detection System capable of accurately using CNNs identifying and localizing lunar craters from high-resolution satellite imagery. These craters vary significantly in size, lighting conditions, and appearance due to shadows, making this a challenging real-world computer vision problem.

We approached this using Ultralytics YOLO model, aiming for both high recall (not missing craters) and reasonable precision (avoiding too many false positives).

Implementation

- Data Preparation & Preprocessing:

  The dataset consisted of high-resolution **satellite images of the lunar surface** (.png format) and **YOLO-style annotation files** (.txt format) containing bounding box labels in normalized format.

  For the first round of training, we went with Yolov8's smaller model **yolov8n**(nano) since the dataset was too large for higher models to digest. We went with training on Google Colab which makes things lighter on system but it came with its own problems of Computation limit with GPUs.

- Model Training:

  Our training involved using a custom configuration with an image size of 416x416 and training for 50 epochs, adjusting batch size and confidence thresholds for optimal results.

  It took multiple attempts to train the model without losing compute limits or the runtime reset. The final training took around 3 hours and 18 minutes, convergining the box losses to a reasonable number.

After training completion we save the best weights and moved on for the validation set.

Custom Evaluation Pipeline:

While YOLO provides standard metrics like Precision, Recall, and mAP, we built a **custom Python evaluation script** to dig deeper into our model's performance.

Our evaluation pipeline:

- Loaded both images and their corresponding label files.
- Converted YOLO’s normalized box coordinates to pixel format.
- Ran inference using our trained model and filtered predictions based on a **confidence threshold** (we tested values like 0.1, 0.25, and 0.4).
- Calculated **IoU (Intersection over Union)** between predicted and ground truth boxes.

Counted:

- **True Positives (TP)** – correct detections
- **False Positives (FP)** – incorrect extra detections
- **False Negatives (FN)** – missed detections
- **True Negatives (TN)** – estimated using areas without craters and predictions

Computed:

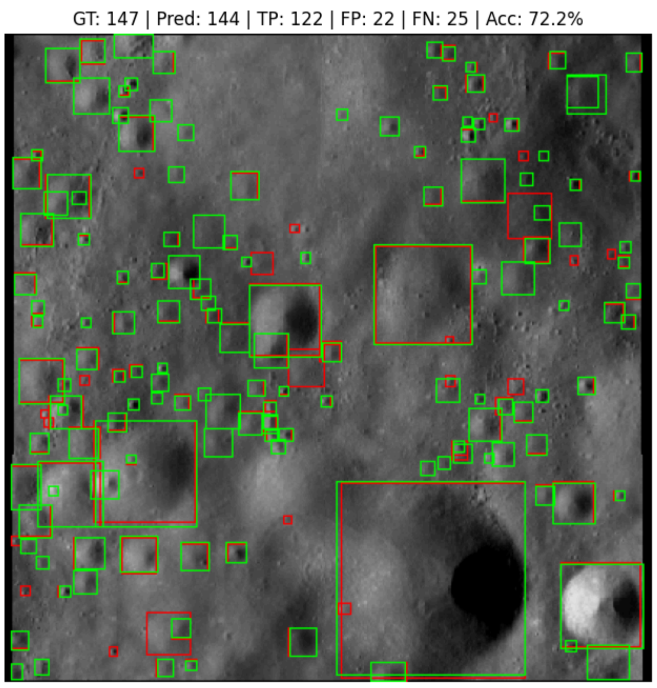
- **Precision** = TP / (TP + FP)
- **Recall** = TP / (TP + FN)
- **F1 Score** = 2 × (Precision × Recall) / (Precision + Recall)
- **Accuracy** = (TP + TN) / (TP + TN + FP + FN)

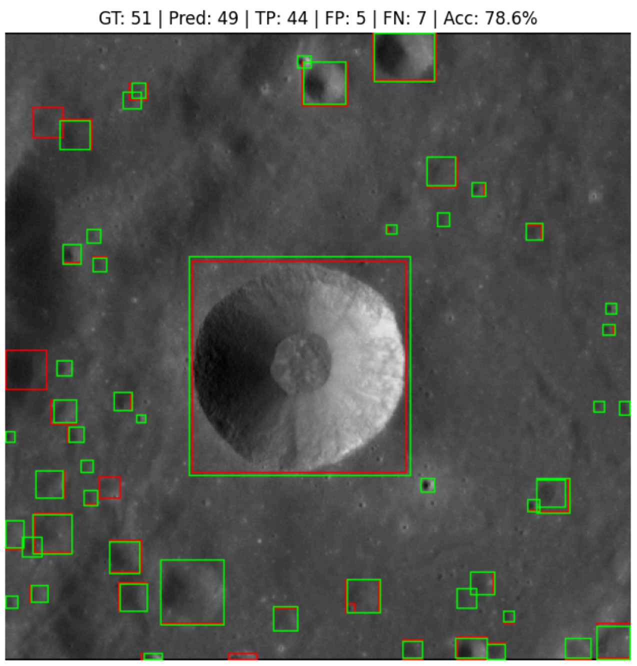
Visual Analysis & Metrics Logging:

To understand how the model was behaving on individual images, we:

- Plotted bounding boxes: **Green for ground truth**, **Red for predictions**
- Displayed image-wise metrics with detailed overlays (TP, FP, FN, Accuracy)
- Logged and plotted trends across multiple images (Precision, Recall, Accuracy)
- Created summary bar charts to compare total predictions and ground truths

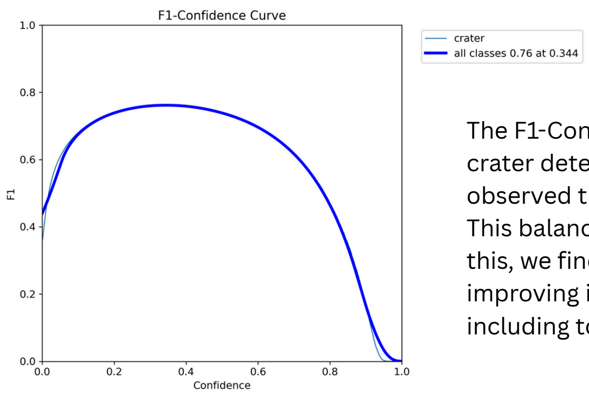
Here are some samples of the test:


GT: 147 | Pred: 144 | TP: 122 | FP: 22 | FN: 25 | Acc: 72.2%



Red- Our Predictions Green- Labels

Experimentation and Tuining



The F1-Confidence Curve helped us find the optimal confidence threshold for crater detection. By plotting F1 scores across different confidence levels, we observed that the score peaked at 0.76 when the confidence was set to 0.344. This balance gave us the best trade-off between precision and recall. Based on this, we fine-tuned our model's confidence threshold to around 0.34, improving its overall detection reliability without missing too many craters or including too many false positives.

We experimented with different **confidence thresholds** and **IoU thresholds** to find the right trade-off between precision and recall. For example:

- Lower confidence increased recall but introduced more false positives.
- Higher thresholds made predictions cleaner but at the risk of missing faint or small craters.

We ultimately settled on a **confidence threshold of 0.323** and **IoU threshold of 0.5** as our optimal evaluation setting.

The Precision came out to be **0.797** and Recall is **0.731** on final completion of training. We thought of improving on this by hyperparameterizing but then came up with **Transfer Learning** as a method of upgrading the model further.

Innovation

Well we can't say this as innovation but we explored **Transfer learning** to enhance the model's performance. We initialized a larger **YOLOv8m** model using pre-trained weights from **YOLOv8n**, a smaller and faster version. This gave us a solid starting point while allowing the model to adapt to our specific crater dataset.

We trained this setup for **20 epochs**, and the results were promising. On the validation set of **3740 images** with nearly **250K crater instances**, the model achieved a **precision of 0.828**, **recall of 0.748**, and **mAP@0.5 of 0.849**. This showed a noticeable boost in both localization accuracy and general detection performance, especially due to the added capacity of the medium-sized model while still benefiting from the learned features of the nano version.

After completing our model tuning, we went for making a custom UI for users to test our model with ease. For this we used **Roboflow** to host our model on cloud with roboflow's serverless service. We uploaded the exported weights and created a simple **Web App** using NextJs and React and calling the Roboflow API in the app to run the inference.

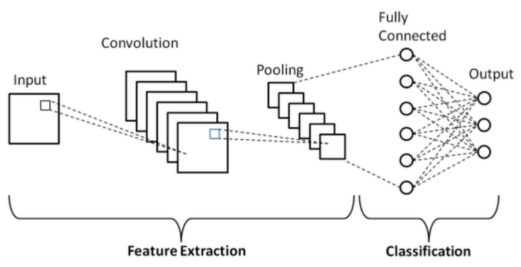
Website link: https://render-6lpb.onrender.com

GitHub link: GitHub - Varshith-syren/Lunar-Vision

How model works:

To identify craters, our model learns to detect specific visual patterns that are characteristic of lunar craters — such as circular or elliptical depressions, shadow gradients, and texture differences from surrounding terrain. During training, we provided the model with thousands of labeled images where crater locations were annotated using bounding boxes. The YOLOv8 model processes each image as a grid and predicts potential object locations (bounding boxes), confidence scores, and class probabilities for each grid cell.

As the model sees more examples, it starts associating certain spatial and textural cues — like rim shadows, bright inner walls, and central darkness — with crater presence.
Thanks to its convolutional layers, it can capture both fine details and broader spatial patterns. During inference, it uses this learned understanding to scan new lunar images and confidently locate craters even if they vary in size, brightness, or terrain.
This ability to generalize across different types of crater appearances is what makes the model effective for real-world lunar surface analysis.



Problems we faced

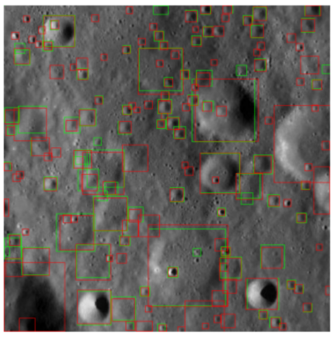
Firstly, the training part was a hassel, taking too long to run and if something goes wrong, go all the way back and start again. This becomes a drag but is the most neccessary part of the process so we went through it.

Another problem being “Predicting more craters than present”, which leads to more False Positives(FP) in the tests. When plotting these on the images, we found out that there are indeed craters and just that labels had not identifying them and our model was outperforming in those instances.

Another problem being “Predicting more craters than present”, which leads to more False Positives(FP) in the tests. When plotting these on the images, we found out that there are indeed craters and just that labels had not identifying them and our model was outperforming in those instances.

Deploying the website was also a problem as free services are not that great for these applications or not that we knew about all of them at the moment

Red-Our Predictions
Green- Labels



Instance where Predictions are more than Ground Truth

Real-World Use Cases

A promising real-world extension of this model is pothole detection on roads. Just like lunar craters, potholes exhibit distinct depressions and edge patterns in imagery, making them visually similar in structure.

By fine-tuning the same YOLO-based architecture on annotated street images, we can train the model to accurately detect potholes in real-time. This can be highly beneficial for automated road inspections, smart city infrastructure, and safety monitoring—reducing manual effort and enabling quicker repairs to prevent accidents and improve road quality.

Potholes detection with our model with altered parameters gives decent results

Conclusion

In conclusion, our project successfully developed a robust crater detection model using CNNs, achieving strong performance through careful training, evaluation, and optimization. This experience not only deepened our understanding of object detection in scientific imaging but also opened doors for practical applications in planetary science and beyond. We thank the organizers for this opportunity and for providing such a meaningful and challenging platform.