# Lunar-Vision Progress Report

BABLUBADMOSH
Soumya Basuli
L Shreya
Pritika Paripelly

## OBJECTIVE AND DATASET

Lunar crater detection is essential for understanding the Moon's geological history and planning safe landings. Traditional manual or semi-automated methods lack scalability across large volumes of high-resolution imagery. In this project, we fine-tuned YOLOv5, a real-time, single-shot CNN-based object detector, on a custom dataset of grayscale lunar images annotated in YOLO format. The dataset's diversity in illumination, terrain, and crater scale provided a robust benchmark for evaluating model generalization. Our pipeline includes model training, optimized inference, and a lightweight web interface for visualization—enabling scalable, automated crater detection on lunar surfaces.

## MODEL SELECTION AND SETUP

The task was to detect lunar craters using a CNN-based object detection model, helping us understand how Convolutional Neural Networks (CNNs) perform on real-world datasets. We chose YOLOv5, a fast and accurate CNN model for real-time object detection.

### YOLOv5 Key Features:

- CSPDarknet53 backbone
- PANet for path aggregation
- Focus and Conv layers for image encoding
- Anchor-based multi-scale predictions

### Modifications:

- Changed the number of classes to **1** (crater)
- Reconfigured anchor boxes to suit crater size distribution

The dataset contained grayscale lunar surface images with bounding box annotations for craters. To manage compute limitations on Google Colab, we:

- **Split the dataset into subsets of 100 images.**
- **Trained each subset for 100 epochs on YOLOv5s using separate Google accounts in parallel (up to 15 at a time).**
- **Extracted and compared metrics like precision, recall, and mAP from each subset.**

## DATA SUBSET STRUCTURE:

```
subset_X/
├── images/
│   ├── train/
│   └── val/
├── labels/
│   ├── train/
│   └── val/
└── data_subset_X.yaml
```

Each folder will have 100 image-label pairs.

We ensured:

- Each .jpg image has a corresponding .txt file in YOLO format
- Exact filename matching (case-sensitive)
- All bounding boxes are inside image bounds
- Normalization of all coordinates

After evaluating all subsets, we selected the **7 best-performing ones** based on **precision** and retrained them on a **larger YOLO model** to improve overall performance beyond the limits of individual subset training.

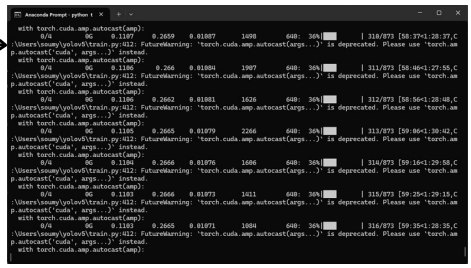
### Subset-Based Training Strategy:

- Split the dataset into subsets of 100 images each.
- Trained each subset independently for 100 epochs.
- Saved final training metrics from results.csv into .json for centralized comparison.
- Compared metrics such as precision, recall, and mAP across subsets.

# TRAINING AND EVALUATION STRATEGY

## Key Steps Taken:

- Attempted initial training on CPU (inefficient).
- Tried to load the entire dataset at once, which exceeded RAM limits.
- Switched to **subset training** (100 images per subset).
- Used **Google Colab GPU** to speed up the training process.
- Found that only **one Colab session/account could run at a time.**
- Leveraged **multiple Google accounts** (team + family + friends) to **parallelize training**:
  a. Each member ran 5 subsets simultaneously using 5 accounts.
  b. Effectively trained 15 subsets in the time it would take to train 1.
- Eventually, we ran out of phone numbers for account verifications.
- To bypass this, we used **VPN services** to connect to different country servers and create more Google accounts to scale up.



### Observations:

- Subsets with relatively balanced recall and mAP performed better.
- Subsets 1–5 had precision values in the 0.61–0.63 range, but lower mAP.
- Some later subsets had high object loss or unstable class loss, suggesting potential annotation noise or harder images.

# Challenges and Learnings

| Challenges: | Learnings and Solutions: |
| --- | --- |
| ● **Colab's compute limitations and account limits.**<br>● **Dataset too large to be trained as a whole.**<br>● **Slow training on CPU made iterative debugging difficult.**<br>● **Frequent disconnections in Colab meant losing training sessions.** | ● **Modular training using subsets helped avoid GPU/RAM limits.**<br>● **Distributing workload among multiple Colab accounts saved significant time.**<br>● **VPN usage allowed us to bypass regional account limits and continue progress.**<br>● **Exporting results.csv into .json helped with organized metric comparisons.**<br>● **Ensured consistent formatting of predictions to match required test output.** |

# TRAINING PIPELINE

Training was done on **Google Colab Pro** using **Tesla T4 GPU**.

## Steps Followed:

1. Mounted Google Drive
2. Cloned Ultralytics YOLOv5
3. Created a custom YAML file for each subset (data_subsetx.yaml) :

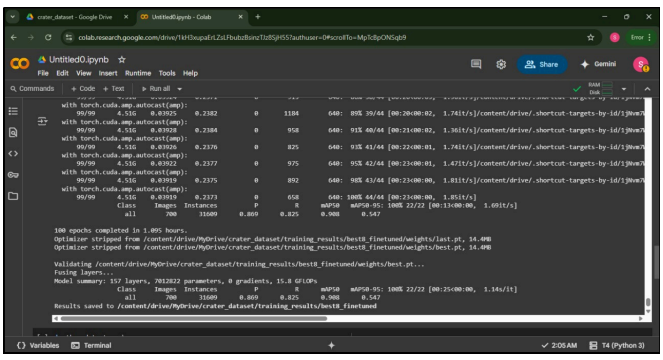
```
train: /content/drive/MyDrive/YOLO_subsets/subset_x/images/train
val: /content/drive/MyDrive/YOLO_subsets/subset_x/images/val

nc: 2
names: ["crater", "boulder"]
```

4.Executed training:

```
!python train.py --img 640 --batch 16 --epochs 100 \
--data /content/drive/MyDrive/YOLO_subsets/subset_X/data_subset_X.yaml \
--weights yolov5s.pt --name subsetX_yolov5
```

**Observations:**

- Loss converged well by epoch ~80
- Post finetuning Precision: ~0.87, Recall: ~0.83
- Best weights saved as best.pt



# INFERENCE AND OUTPUT

We created a script generate_predictions.py to:

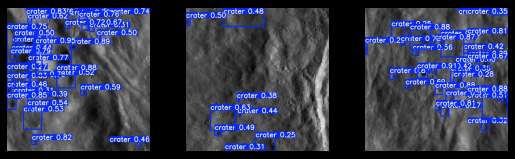
- Load best.pt
- Run inference on images in test dataset
- Save YOLO-formatted outputs in predictions/labels/

### SAMPLE CODE SNIPPET:

```
model = torch.hub.load('ultralytics/yolov5', 'custom',
path='best.pt')
for img_path in test_images:
    results = model(img_path)
    results.save_txt('predictions/labels/')
```

### OUTPUT STRUCTURE:

```
test_results.zip/
├── labels/
└── test_images/
```



# WEB UI WITH GRADIO

We built a Gradio-based interface (app.py) for live crater detection.

## Final Feature List of the Web UI for Lunar Crater Detection

1. **Manual Image Upload Options:**
   - Drag-and-drop image upload
   - Upload from local folders
   - **Copy-paste image from clipboard** (e.g., screenshots)
2. **Webcam Upload Functionality:**
   - Live image capture via webcam
   - Useful in simulations of real-time detection during space expeditions
3. **Brightness and Contrast Adjustment:**
   - Slider-based controls to adjust input image parameters
   - Helps improve crater visibility under varied lighting conditions
4. **Automatic Crater Detection:**
   - YOLOv5-based model instantly detects and annotates craters in the uploaded image
5. **Crater Count Display:**
   - Real-time count of detected craters displayed to the user
6. **Individual Crater Cropping:**
   - All detected craters are cropped and displayed as individual thumbnails for closer inspection
7. **Download Options for All Outputs:**
   - Annotated image download
   - Cropped craters download (individually or all at once)
   - Downloadable label files (YOLO-format)
8. **Confidence Score Histogram:**
   - Visual histogram showing distribution of confidence scores across detections
   - Useful for evaluating model certainty and setting detection thresholds
9. **User-Friendly Interface (UI):**

- Minimalistic, responsive, and clean layout
- All tools clearly labeled with intuitive interactions
- Dark-mode compatible (if added)

10. **Real-time Inference Speed:**
- Fast prediction times with on-the-fly annotation
- Optimized to run on both CPU and GPU machines

11. **Input Image Preview and Post-Detection Comparison:**
- Side-by-side display: Original vs Annotated image
- Helps users visually validate model accuracy

## TERMINAL COMMANDS:

```
cd %USERPROFILE%\OneDrive\Desktop\moon_crater_detection
git clone https://github.com/ultralytics/yolov5
pip install -r requirements.txt
#If that fails or is incomplete, then run:
pip install torch torchvision torchaudio --index-url https://download.pytorch.org/whl/cpu
pip install gradio opencv-python matplotlib seaborn pillow
python app.py
```

```
The working directory was organized as follows:
moon_crater_detection/
├── app.py          # Gradio-based frontend interface
├── best.pt         # Fine-tuned YOLOv5 model weights
├── requirements.txt    # List of required Python dependencies
└── yolov5/         # YOLOv5 repository (cloned from Ultralytics)
```

# PROBLEMS ENCOUNTERED & SOLUTIONS

| Issue | Description | Solution |
|---|---|---|
| Dataset Mismatch | Label files had inconsistent normalization | Applied pre-validation script |
| Slow inference | Original test set took long on CPU | Batched inference with GPU |
| Label saving errors | Initial outputs were saved in JSON | Converted to YOLO format via .save_txt() |
| UI Latency | Gradio app took long for large images | Resized inputs to 640x640 before inference |

# CONCLUSION AND REAL-WORLD USES

This project demonstrates the effectiveness of YOLOv5 in the domain of planetary data science. Our model generalizes well to unseen lunar images, providing fast and accurate crater detection. The modular design—training, inference, and GUI—ensures it can be easily adapted for more classes (e.g., boulders) or planetary bodies (e.g., Mars). The UI has been designed in such a way it provides features that facilitate real-time analysis and detection during space expeditions through cams. We hope this contributes to scalable lunar mapping pipelines and further applications in space exploration and geology.

# CREDITS AND CONTRIBUTIONS

| SOUMYA | SHREYA | PRITIKA |
|---|---|---|
| • Found the pre-trained model and identified the training procedure<br>• Trained subsets: 1–5, 6, 7, 9, 10, 16–20<br>• Fine tuned obtained sorted subsets and established the final model<br>• Designed and managed the website<br>• Compiled and prepared the final report | • Handled training for subsets: 8, 21–23<br>• Assisted in report preparation | • Coordinated and managed emails related to subset model training |