Crater Detection Training Data Generator

A Comprehensive Python Toolkit for Lunar Crater Detection

Documentation Generated: November 18, 2025

Table of Contents

1. Overview

The Crater Detection Training Data Generator is a professional-grade Python toolkit designed to facilitate the creation of labeled training datasets for crater detection in lunar images. This tool bridges the gap between planetary science data and modern deep learning frameworks, enabling researchers to quickly prepare datasets for training YOLO, CNN, and DETR models.

The toolkit handles the complexities of geospatial data processing, including coordinate reference system transformations, multiple map projections, and format conversions, allowing researchers to focus on model development rather than data preprocessing.

2. Features

• **Multi-Format Input Support:** Handles GeoTiff and ISIS .cub files seamlessly

• **Projection Flexibility:** Supports equidistant cylindrical, stereographic, and orthographic projections

• **Shapefile Integration:** Reads crater annotations from ESRI shapefiles with automatic CRS reprojection

• **Dual Output Formats:** Generates both YOLO and COCO format labels for different model architectures

• **Topography Processing:** Optional support for topography/DEM rasters for multi-modal learning

• **Intelligent Tiling:** Automatically splits large images into manageable tiles with configurable overlap

• **Visualization:** Creates annotated PNG images showing all labeled craters for quality verification

• **Dataset Management:** Built-in utilities for splitting, validation, statistics, and format conversion

• **Production Ready:** Robust error handling, progress reporting, and comprehensive logging

3. Installation

3.1 System Dependencies

For ISIS .cub file support, install GDAL with ISIS drivers:

```
Ubuntu/Debian:
sudo apt-get install gdal-bin libgdal-dev python3-gdal

macOS (using Homebrew):
brew install gdal
```

3.2 Python Dependencies

```
pip install -r requirements.txt
```

3.3 Required Python Packages

Package	Version	Purpose
rasterio	≥1.3.0	Geospatial raster I/O
gdal	≥3.0.0	ISIS .cub file support
geopandas	≥0.12.0	Shapefile processing
shapely	≥2.0.0	Geometric operations
numpy	≥1.23.0	Array processing
pillow	≥9.0.0	Image handling
matplotlib	≥3.5.0	Visualization
scikit-learn	≥1.0.0	Dataset splitting

4. Quick Start

4.1 Basic Usage

Generate a YOLO format dataset from a single image:

```
python crater_training_data_generator.py \
    --image lunar_image.tif \
    --craters craters.shp \
    --output ./crater_dataset \
    --format yolo
```

4.2 Advanced Usage with Tiling

Generate tiled dataset with topography in both formats:

```
python crater_training_data_generator.py \
    --image lunar_mosaic.cub \
    --craters crater_database.shp \
    --output ./crater_dataset_tiled \
    --format both \
    --tile-size 512 \
    --overlap 64 \
    --topography lunar_dem.tif
```

5. Command Line Usage

Argument	Required	Description
--image	Yes	Path to input image (GeoTiff or .cub)
--craters	Yes	Path to crater shapefile
--output	Yes	Output directory for dataset
--format	No	Output format: yolo, coco, or both (default: yolo)
--topography	No	Optional topography raster file path
--tile-size	No	Tile size for splitting large images (e.g., 512)
--overlap	No	Overlap between tiles in pixels (default: 0)

6. Input Requirements

6.1 Image Files

Supported Formats:

• GeoTiff (.tif, .tiff) with embedded georeferencing

• ISIS cube files (.cub) with label information

Requirements:

• Valid coordinate reference system (CRS) information

• Single-band (grayscale) or multi-band (RGB) supported

• Any bit depth supported by GDAL/rasterio

6.2 Crater Shapefile

Format: ESRI Shapefile (.shp)

Supported Geometry Types:

• **Point:** Center point of crater (generates circular bounding box)

• **Polygon:** Crater outline (generates bounding box from bounds)

• **Circle/Ellipse:** Crater perimeter

Common Attributes (optional but recommended):

• diameter or radius: Crater size in map units

• name or id: Crater identifier

• confidence: Annotation confidence/quality

6.3 Projection Support

The tool automatically handles different map projections commonly used in planetary science. The projection information should be embedded in the GeoTiff metadata or ISIS label. Supported projections include:

• **Equidistant Cylindrical (Equirectangular):** Most common for global mosaics

• **Stereographic:** Often used for polar regions

• **Orthographic:** Used for hemisphere views

7. Output Structure

```
output_directory/
■■■ images/                      # Training images
■   ■■■ image_000.png
■   ■■■ image_001.png
■   ■■■ ...
■■■ labels/                     # YOLO format labels
■   ■■■ image_000.txt
■   ■■■ image_001.txt
■   ■■■ ...
■■■ visualizations/             # Annotated images
■   ■■■ image_000_labeled.png
■   ■■■ ...
■■■ topography_images/       # Topography images (optional)
■   ■■■ ...
■■■ topography_labels/       # Topography labels (optional)
■   ■■■ ...
■■■ annotations_coco.json    # COCO format labels
■■■ dataset.yaml              # YOLO dataset config
```

8. Label Formats

8.1 YOLO Format

Each .txt file contains one line per crater with normalized coordinates:

```
class_id x_center y_center width height
```

Ex

```
0 0.5234 0.7123 0.0456 0.0389
0 0.3421 0.2156 0.0892 0.0734
```

All values except class_id are normalized to [0, 1] range.

8.2 COCO Format

JSON file with three main sections:

```
{
  "images": [
    {"id": 0, "file_name": "image_000.png", "width": 512, "height": 512}
  ],
  "annotations": [
    {"id": 0, "image_id": 0, "category_id": 0, "bbox": [x, y, w, h]}
  ],
  "categories": [
    {"id": 0, "name": "crater"}
  ]
}
```

9. Dataset Utilities

The crater_utils.py module provides essential dataset management functions:

9.1 Split Dataset

```
python crater_utils.py split --dataset ./crater_dataset \
                             --train-ratio 0.8 \
                             --val-ratio 0.1 \
                             --test-ratio 0.1
```

9.2 View Statistics

```
python crater_utils.py stats --dataset ./crater_dataset
```

Output includes:

• Total images and crater counts

• Average craters per image

• Crater size statistics (min, max, mean, median)

9.3 Verify Dataset

```
python crater_utils.py verify --dataset ./crater_dataset
```

Checks for:

• Missing label files

• Missing image files

• Dataset integrity

9.4 Convert Formats

```
python crater_utils.py convert --dataset ./crater_dataset \
                               --output ./annotations_coco.json
```

10. Advanced Examples

10.1 Large Mosaic Processing

For large images (>10,000 pixels), use tiling to manage memory:

```
python crater_training_data_generator.py \
    --image large_mosaic_20000x20000.tif \
    --craters crater_catalog.shp \
    --output ./large_dataset \
    --tile-size 640 \
    --overlap 128 \
    --format both
```

10.2 Multi-Modal Learning

Create datasets with both optical and topographic data:

```
python crater_training_data_generator.py \
    --image lunar_optical.tif \
    --craters craters.shp \
    --topography lunar_slope_map.tif \
    --output ./multimodal_dataset \
    --tile-size 512
```

This creates separate image and topography datasets with matching labels.

10.3 Complete Workflow

End-to-end workflow from generation to training:

```
# Step 1: Generate dataset
python crater_training_data_generator.py \
    --image data.tif --craters craters.shp \
    --output ./dataset --tile-size 512

# Step 2: Verify
python crater_utils.py verify --dataset ./dataset

# Step 3: View statistics
python crater_utils.py stats --dataset ./dataset

# Step 4: Split into train/val/test
python crater_utils.py split --dataset ./dataset
```

11. Troubleshooting

Issue 1: CRS mismatch warning

Solution: The tool automatically reprojects shapefiles. Verify both files have valid CRS: gdalinfo image.tif | grep "Coordinate System" and ogrinfo -al -so craters.shp

Issue 2: Craters not in correct locations

Solution: Check: (1) Projection mismatch, (2) Different coordinate systems, (3) Missing georeferencing. Ensure both image and shapefile have proper CRS defined.

Issue 3: ISIS .cub files not opening

Solution: Install GDAL with ISIS drivers or convert to GeoTiff: gdal_translate input.cub output.tif

Issue 4: Out of memory errors

Solution: Use tiling: --tile-size 512 --overlap 64

Issue 5: Empty visualizations or missing craters

Solution: Check that crater shapefile overlaps with image extent. Use ogrinfo and gdalinfo to verify spatial bounds match.

12. Technical Details

12.1 Coordinate Transformation

The tool uses rasterio's affine transformation to convert between geographic coordinates (from shapefiles) and pixel coordinates (for bounding boxes). The transformation automatically accounts for the map projection, resolution, and extent of the input image.

12.2 Tiling Strategy

When tiling is enabled, the tool uses a sliding window approach with configurable overlap. This ensures craters near tile boundaries are captured completely. The overlap should be set to at least twice the maximum expected crater diameter to avoid splitting craters across tiles.

12.3 Image Normalization

Images are normalized to 0-255 range using percentile-based scaling (2nd and 98th percentiles) to handle outliers and varying dynamic ranges. This approach is more robust than min-max scaling for planetary science data which often contains extreme values.

12.4 Performance Considerations

Image Size	Recommended Approach	Memory Usage
< 5000x5000	Full image processing	Low (< 2GB)
5000-10000	Full or tile (512-1024)	Medium (2-4GB)
> 10000x10000	Tiling required (512-640)	High (4-8GB)
Very large mosaics	Tiling with overlap 64-128	Moderate per tile

13. Code Structure

13.1 Main Components

crater_training_data_generator.py

Main script containing the CraterDatasetGenerator class. Handles image loading, coordinate transformation, tiling, and label generation.

crater_utils.py

Utility functions for dataset management including splitting, statistics, verification, and format conversion.

example_usage.py

Example scripts demonstrating various use cases from simple generation to complete training workflows.

requirements.txt

Python package dependencies with version specifications.

13.2 Key Classes and Methods

CraterDatasetGenerator

• load_data(): Loads image and shapefile with CRS handling

• geometry_to_bbox(): Converts shapefile geometry to pixel bounding boxes

• bbox_to_yolo(): Converts pixel bbox to YOLO normalized format

• generate_full_image_dataset(): Processes entire image

• generate_tiled_dataset(): Splits image into tiles with overlap

14. Best Practices

Data Preparation: Always verify CRS information in both image and shapefile before processing. Use gdalinfo and ogrinfo to check.

Tile Size Selection: Choose tile sizes that match your target model input size (512, 640, or 1024 are common for YOLO).

Overlap Strategy: Use 10-20% overlap (64-128 pixels for 512-640 px tiles) to avoid missing craters on boundaries.

Quality Control: Always review visualizations in the visualizations/ folder to verify correct labeling.

Dataset Balance: Check statistics to ensure good distribution of crater sizes and reasonable craters per image.

Training Split: Use 70-80% training, 10-20% validation, and 10% test for typical datasets. Adjust based on dataset size.

Multi-scale Training: For datasets with large size variation, consider creating separate datasets for different scales.

Memory Management: For large datasets, process in batches or use tiling to avoid memory issues.

15. Training Models

15.1 YOLOv8 Training

```
from ultralytics import YOLO

# Load pretrained model
model = YOLO('yolov8n.pt')

# Train on crater dataset
results = model.train(
    data='./crater_dataset/dataset.yaml',
    epochs=100,
    imgsz=512,
    batch=16,
    patience=20,
    name='crater_detector'
)

# Validate
metrics = model.val()
print(f'mAP50: {metrics.box.map50:.3f}')
```

15.2 Model Selection

Model	Speed	Accuracy	Use Case
YOLOv8n	Fastest	Good	Real-time detection, embedded systems
YOLOv8s	Fast	Better	Balanced speed/accuracy
YOLOv8m	Medium	High	Research applications
YOLOv8l/x	Slow	Highest	Maximum accuracy required
DETR	Slow	High	Transformer-based, research

Appendix A: Example Workflows

A.1 Workflow for Small Dataset (< 100 images)

```
# Generate without tiling
python crater_training_data_generator.py \
    --image small_region.tif \
    --craters craters.shp \
    --output ./small_dataset \
    --format yolo

# Split 70/20/10
python crater_utils.py split --dataset ./small_dataset \
    --train-ratio 0.7 --val-ratio 0.2 --test-ratio 0.1

# Train lightweight model
yolo train data=./small_dataset/dataset.yaml \
    model=yolov8n.pt epochs=50
```

A.2 Workflow for Large Dataset (> 1000 images)

```
# Generate with tiling
python crater_training_data_generator.py \
    --image large_mosaic.cub \
    --craters comprehensive_catalog.shp \
    --output ./large_dataset \
    --format both \
    --tile-size 640 --overlap 96

# Verify and get stats
python crater_utils.py verify --dataset ./large_dataset
python crater_utils.py stats --dataset ./large_dataset

# Split 80/15/5
python crater_utils.py split --dataset ./large_dataset \
    --train-ratio 0.8 --val-ratio 0.15 --test-ratio 0.05

# Train robust model
yolo train data=./large_dataset/dataset.yaml \
    model=yolov8m.pt epochs=150 imgsz=640
```

Summary

The Crater Detection Training Data Generator provides a complete solution for preparing lunar crater detection datasets. By automating the complex processes of geospatial data handling, coordinate transformations, and format conversions, it enables researchers to focus on model development and analysis rather than data preprocessing.

Key Takeaways:

• Supports multiple input formats (GeoTiff, ISIS .cub) and projections

• Generates both YOLO and COCO format labels for flexibility

• Handles large images efficiently through intelligent tiling

• Includes comprehensive utilities for dataset management

• Provides visualization for quality assurance

• Production-ready with robust error handling

• Well-documented with extensive examples

Whether you're working with small annotated regions or large planetary mosaics, this toolkit provides the flexibility and robustness needed for professional crater detection research. The combination of automation, quality control features, and comprehensive documentation makes it an ideal choice for planetary scientists and machine learning researchers alike.