CostalSeg User Guide

Overview

CostalSeg is a machine learning system for coastal erosion monitoring that provides multi-class image segmentation and analysis of coastal scenes. The system can identify seven different terrain types: background, cobbles, dry sand, plants, sky, water, and wet sand.

The project supports two coastal environments out of the box - Metal Marcy and Silhouette Jaenette - but can be extended to work with other coastal types through custom training.

Getting Started

System Requirements

For basic usage, you'll need Python 3.8+ and at least 8GB of RAM. If you plan to train models, a CUDA-compatible GPU with 6GB+ VRAM is strongly recommended. The system works on Windows, macOS, and Linux.

Installation

Clone the repository and set up the environment:

```bash
git clone https://github.com/cxh42/CostalSeg.git
cd CostalSeg
conda create -n CostalSeg python=3.12
conda activate CostalSeg
pip install -r requirements.txt
```

Verify the installation by checking if PyTorch can detect your GPU:

```bash
python -c "import torch; print(torch.cuda.is_available())"
```

Quick Start

Download the pretrained models from the provided link and place them in the `models/` directory. Then launch the web interface:

```bash
python app.py
```

The interface will open in your browser where you can upload images for segmentation analysis.

Project Structure

The codebase is organized into several key components:

- `app.py` - Main web interface using Gradio
- `models/` - Directory for trained model files and reference vectors
- `SegmentModelTraining/` - Training scripts and data preparation tools
- `pipeline/` - Core algorithms for outlier detection, image alignment, and preprocessing
- `reference_images/` - Reference image sets for outlier detection

Each coastal type (MM for Metal Marcy, SJ for Silhouette Jaenette) has its own model file, reference vector, and reference image directory.

Training Your Own Models

Data Preparation

The training process requires properly annotated segmentation data. We recommend using Roboflow for annotation, which provides efficient tools for semantic segmentation tasks.

Create a new semantic segmentation project in Roboflow and define these seven classes:

- Background (black: 0,0,0)
- Cobbles (dark gray: 139,137,137)
- Dry sand (light yellow: 255,228,181)
- Plant (green: 0,128,0)
- Sky (sky blue: 135,206,235)
- Water (blue: 0,0,255)
- Wet sand (sandy brown: 194,178,128)

For best results, collect at least 200 training images with good variety in lighting conditions, weather, tidal states, and viewing angles. Use the Smart Polygon tool in Roboflow for efficient annotation.

Export your dataset at 1024x1024 resolution with a 70/15/15 train/validation/test split. Apply reasonable augmentations like rotation (±15°), brightness/contrast adjustments (±15%), and horizontal flipping.

Running Training

Place your exported dataset in the appropriate training directory:

```bash
cd SegmentModelTraining/MetalMarcy  # or SilhouetteJaenette
mkdir dataset
# Extract your Roboflow export here
```

The training script uses DeepLabV3Plus with an EfficientNet-B6 encoder. Key parameters can be adjusted in `train.py`:

```python
BATCH_SIZE = 2          # Adjust based on GPU memory
IMAGE_SIZE = 1024       # Input image resolution
EPOCHS = 100            # Maximum training epochs
LEARNING_RATE = 8e-5    # Learning rate
```

Start training:

```bash
python train.py
```

Training will automatically stop early if validation performance stops improving. The best model will be saved as a `.pth` file. Good models typically achieve IoU scores above 0.8 and F1 scores above 0.85.

Model Deployment

After training, copy your model to the main models directory and update the file paths in `app.py` if needed:

```bash
cp best_model.pth ../../models/MY_best_model.pth
```

Advanced Features

Outlier Detection

The system includes automatic outlier detection using cosine similarity comparison against reference images. This helps identify images that may not be suitable for analysis.

To set up outlier detection for a new coastal type, collect 20-50 high-quality reference images and place them in `reference_images/YOUR_TYPE/`. Then generate a reference vector:

```bash
python create_reference_vector.py
```

The detection threshold can be adjusted in `pipeline/ImgOutlier.py`. Lower values (closer to 0) are more permissive, while higher values (closer to 1) are more restrictive.

Spatial Alignment

For comparing images taken from different perspectives of the same coastal area, the system provides automatic spatial alignment using SuperGlue feature matching. This is particularly useful for time-series analysis or multi-camera setups.

Access this feature through the "Spatial Alignment Segmentation" tab in the web interface, where you can upload a reference image and a target image for alignment and analysis.

Performance Monitoring

Use the enhanced interface (`app_with_computing_analysis.py`) to get detailed performance metrics including processing times, memory usage, and throughput rates. For comprehensive benchmarking, run:

```bash
python model_performance_test.py
```

This generates detailed reports on inference performance, batch processing efficiency, and hardware utilization.

Using the API

For programmatic access, you can import the core functions directly:

```python
from app import load_model, perform_segmentation
import cv2

# Load your model
model = load_model("models/MM_best_model.pth")

# Process an image
image = cv2.imread("coastal_image.jpg")
seg_map, overlay, analysis = perform_segmentation(model, image)

# Save results
cv2.imwrite("segmentation_result.jpg", seg_map)
```

For batch processing multiple images, create a simple loop with progress tracking using `tqdm`.

Troubleshooting

Out of memory errors: Reduce batch size or image resolution. For inference, try using CPU mode by setting `device="cpu"`.

Model loading issues: Ensure your model file path is correct and the file isn't corrupted. Try loading with `weights_only=False` if you encounter serialization errors.

Poor segmentation quality: Check that your input images are similar to the training data. If working with a new coastal environment, you'll likely need to train a custom model.

Slow performance: Verify GPU acceleration is working. On CPU-only systems, expect significantly longer processing times.

Best Practices

Keep your training data well-organized and maintain consistent annotation quality. When collecting reference images for outlier detection, ensure they represent the typical conditions you expect to encounter.

For production deployments, consider implementing proper input validation, logging, and error handling around the core segmentation functions.

Regular model evaluation on new data helps identify when retraining might be beneficial. The system is designed to be extensible, so adding new coastal types or segmentation classes is straightforward with proper training data.

Support

For technical issues, submit detailed bug reports to the GitHub repository. Include your system specifications, error messages, and steps to reproduce any problems.

Project repository: https://github.com/cxh42/CostalSeg

Demo applications:

- Metal Marcy: https://huggingface.co/spaces/AveMujica/MetalMarcy
- Silhouette Jaenette: https://huggingface.co/spaces/AveMujica/SilhouetteJaenette