detector.py - Solar Panel Detection Engine

Overview

This module implements the core solar panel detection using SAHI (Slicing Aided Hyper Inference) with a YOLOv8 model. It handles high-resolution image analysis by processing image slices and merging results.

Logic

Class: SolarPanelDetector

Method	Purpose
__init__()	Configure detection parameters
initialize()	Load the YOLOv8 model into memory
detect()	Run detection on a satellite image
_extract_polygon()	Convert detection to Shapely polygon
_get_device()	Determine CPU/GPU availability

Key Dependencies

```python
from sahi import AutoDetectionModel
from sahi.predict import get_sliced_prediction
from shapely.geometry import Polygon
```

How It Works

1. Model Initialization

```python
def initialize(self):
    device = self._get_device()  # "cuda:0" or "cpu"

    self.model = AutoDetectionModel.from_pretrained(
        model_type='yolov8',
        model_path=self.model_path,
        confidence_threshold=self.confidence_threshold,
        device=device
    )
```

SAHI wraps the YOLOv8 model to enable sliced inference - critical for high-resolution satellite imagery.

2. Sliced Prediction (SAHI)

```
┌─────────────────────────────────────┐
│         1024x1024 Image             │
│  ┌──────┬──────┬──────┬──────┐      │
│  │ 640  │ 640  │ 640  │ 640  │      │
│  │  1   │  2   │  3   │  4   │      │
│  ├──────┼──────┼──────┼──────┤      │
│  │  5   │  6   │  7   │  8   │      │
│  │      │      │      │      │      │
│  └──────┴──────┴──────┴──────┘      │
│         20% overlap                 │
└─────────────────────────────────────┘
```

- **Slice size**: 640x640 pixels (YOLOv8 optimal)
- **Overlap ratio**: 0.2 (20%) - prevents edge detection failures

- **Merging**: SAHI handles duplicate detection consolidation

3. Detection Flow

```
result = get_sliced_prediction(
    str(image_path),
    self.model,
    slice_height=self.slice_height,      # 640
    slice_width=self.slice_width,        # 640
    overlap_height_ratio=self.overlap_ratio,  # 0.2
    overlap_width_ratio=self.overlap_ratio    # 0.2
)
```

4. Polygon Extraction

```
def _extract_polygon(self, prediction):
    # Try mask-based (more accurate)
    if prediction.mask is not None:
        mask_points = prediction.mask.to_coco_segmentation()[0]
        coords = [(points[i], points[i+1]) for i in range(0, len(points), 2)]
        return Polygon(coords)

    # Fallback to bounding box
    bbox = prediction.bbox
    return Polygon([(x1,y1), (x2,y1), (x2,y2), (x1,y2)])
```

Priority: Segmentation mask → Bounding box

Why It Works

SAHI for High-Resolution Images

Standard YOLO struggles with large images because:

- Fixed input size (typically 640x640)
- Downscaling loses small object detail
- Solar panels become too small to detect

SAHI solves this by:

1. **Slicing** the image into manageable chunks
2. **Running inference** on each slice
3. **Merging** results with NMS (Non-Maximum Suppression)

Overlap Prevents Edge Cases

Without overlap, panels on slice boundaries get:

- Cut in half → missed detection
- Partial detection → low confidence

20% overlap ensures every panel is fully contained in at least one slice.

Mask vs. Bounding Box

- **Masks**: Pixel-precise panel boundaries (if model supports)
- **Bboxes**: Rectangular approximation (guaranteed)

The fallback ensures reliability even with varying model outputs.

GPU Detection

```
@staticmethod
def _get_device():
    if os.system("nvidia-smi > /dev/null 2>&1") == 0:
        return "cuda:0"
    return "cpu"
```

Simple heuristic - if nvidia-smi runs successfully, GPU is available.

Usage in Main Pipeline

Initialization

```
# In pipeline.py
self.detector = SolarPanelDetector(
    model_path=Config.get_model_path(),
    confidence_threshold=Config.CONFIDENCE_THRESHOLD,  # 0.25
    slice_size=Config.SLICE_HEIGHT,  # 640
    overlap_ratio=Config.OVERLAP_RATIO  # 0.2
)
```

Model Loading (Once)

```
def run(self):
    ...
    self.detector.initialize()  # Load model once

    for row in df.iterrows():
        result = self._process_sample(row, ...)
```

Per-Sample Detection

```python
def _process_sample(self, row, ...):
    ...
    all_polygons = self.detector.detect(image_path)
    # Returns: [(Polygon, confidence), (Polygon, confidence), ...]
```

Output Format

Field	Type	Description
polygon	Shapely Polygon	Panel boundary coordinates
confidence	float	Detection confidence (0.0-1.0)

Integration with Geometry

```python
# Pass detections to buffer zone matching
poly_1200, conf_1200, overlap = find_best_panel(
    all_polygons,  # From detector
    center_px,     # Image center
    radius_1_px,   # 1200 sq.ft buffer
    Config.MIN_OVERLAP_AREA
)
```