visualizer.py – Detection Visualization

Overview

This module creates visual overlays on satellite images showing detection results. It highlights detected solar panels, buffer zones, and the selected target panel with clear annotations.

Logic

Class: DetectionVisualizer

Method	Purpose
`__init__()`	Configure overlay transparency
`draw_results()`	Create complete visualization
`_draw_buffer_zone()`	Draw circular search area
`_draw_all_panels()`	Outline all detected panels
`_draw_selected_panel()`	Highlight the chosen panel
`_add_annotations()`	Add text labels

Visual Elements

Element	Color	Style
Buffer zone	Cyan (255, 255, 0)	Circle outline, 2px
All panels	Yellow (0, 255, 255)	Polygon outline, 1px
Selected panel	Green (0, 255, 0)	Filled + outline, 3px

Element	Color	Style
Annotations	White/Green/Red	Text labels

How It Works

1. Overlay Blending

```
def __init__(self):
    self.overlay_alpha = 0.4   # Overlay weight
    self.original_alpha = 0.6  # Original image weight

def draw_results(self, ...):
    img = cv2.imread(str(image_path))
    overlay = img.copy()

    # Draw on overlay layer
    self._draw_buffer_zone(overlay, ...)
    self._draw_all_panels(overlay, ...)
    self._draw_selected_panel(overlay, ...)

    # Blend overlay with original
    result = cv2.addWeighted(overlay, 0.4, img, 0.6, 0)
```

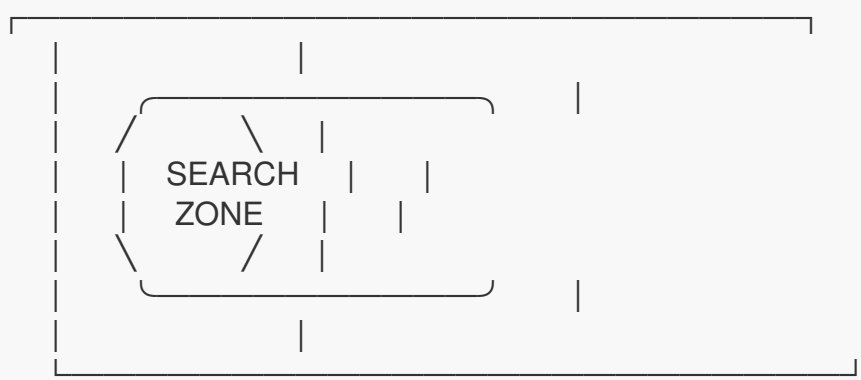
Why blending?

- Pure overlay obscures the underlying image
- 40/60 blend preserves satellite imagery detail
- Detection elements remain visible but non-intrusive

2. Buffer Zone Circle

```python
def _draw_buffer_zone(self, img, center, radius):
    cv2.circle(img,
               (int(center[0]), int(center[1])),  # Center point
               int(radius),                       # Radius in pixels
               (255, 255, 0),                     # Cyan color (BGR)
               2)                                 # Line thickness
```

Visual representation:



3. All Panels Visualization

```python
def _draw_all_panels(self, img, polygons):
    for poly, _ in polygons:
        coords = np.array(poly.exterior.coords, dtype=np.int32)
        cv2.polylines(img, [coords], True, (0, 255, 255), 1)
```

- Iterates through all detected panels
- Converts Shapely polygon to NumPy array
- Draws yellow outline (thin, 1px)

- Shows detection coverage without emphasis

4. Selected Panel Highlight

```python
def _draw_selected_panel(self, img, poly):
    coords = np.array(poly.exterior.coords, dtype=np.int32)

    # Fill with green
    cv2.fillPoly(img, [coords], (0, 255, 0))

    # Strong outline
    cv2.polylines(img, [coords], True, (0, 255, 0), 3)

    # Label at centroid
    centroid = poly.centroid
    cv2.putText(img, "TARGET PANEL",
                (int(centroid.x) - 50, int(centroid.y)),
                cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 0, 0), 2)
```

Visual distinction:

5. Text Annotations

```python
def _add_annotations(self, img, buffer_sqft, has_solar):
    # Buffer info (top-left)
    cv2.putText(img, f"Buffer: {buffer_sqft} sq.ft",
                (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (255, 255, 255), 2)

    # Status (below buffer)
    status_text = "SOLAR CONFIRMED" if has_solar else "NO SOLAR FOUND"
    color = (0, 255, 0) if has_solar else (0, 0, 255)  # Green or Red
    cv2.putText(img, status_text, (10, 60),
                cv2.FONT_HERSHEY_SIMPLEX, 0.8, color, 2)
```

Example output:

```
┌──────────────────────────────────────────────────────────────────────────────┐
│  Buffer: 1200 sq.ft                  |
│  SOLAR CONFIRMED                          |
│                                  |
│      (visualization content)          |
│                                  |
└──────────────────────────────────────────────────────────────────────────────┘
```

Why It Works

OpenCV Coordinate System

Shapely polygons use the same coordinate system, enabling direct conversion.

Color Hierarchy

Color	Meaning	Visibility
Cyan (buffer)	Search area	Medium
Yellow (all)	Detected	Low
Green (selected)	Target	High
Red (no solar)	Alert	High

This creates a natural visual hierarchy from context → detection → result.

Alpha Blending Formula

```
result = overlay × 0.4 + original × 0.6
```

This weighted average:

- Preserves 60% of original image detail
- Adds 40% of detection markings
- Maintains image readability

Coordinate Rounding

```
(int(center[0]), int(center[1]))
np.array(poly.exterior.coords, dtype=np.int32)
```

OpenCV requires integer pixel coordinates. Float coordinates from Shapely are cast to prevent rendering errors.

Usage in Main Pipeline

Initialization

```
# In pipeline.py
self.visualizer = DetectionVisualizer()
```

Per-Sample Visualization

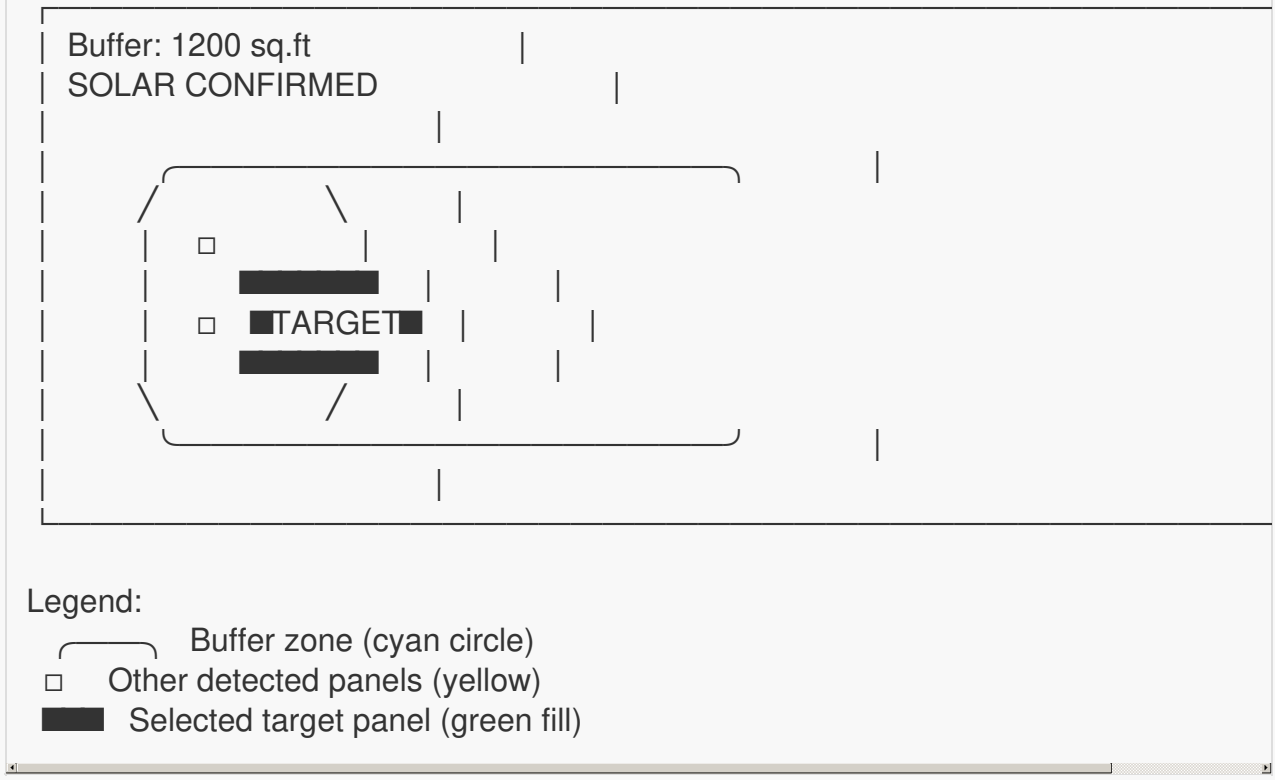
```python
def _process_sample(self, row, ...):
    ...
    overlay_path = sample_folder / f"{sample_id}_overlay.png"

    self.visualizer.draw_results(
        image_path,            # Original satellite image
        all_polygons,          # All detected panels
        result_data['polygon'],  # Selected best panel
        center_px,             # Image center
        result_data['radius_px'],# Buffer radius used
        result_data['buffer_sqft'],  # For annotation
        overlay_path           # Output location
    )
```

Output Files

```
predictions/1001/
 ├─── 1001.jpg          ← Original satellite image
 ├─── 1001_overlay.png   ← Visualization with detections
 └─── 1001.json         ← JSON results
```

Visualization Contents



Error Handling

```python
def draw_results(self, image_path, ...):
    img = cv2.imread(str(image_path))
    if img is None:
        logger.error(f"Cannot load image: {image_path}")
        return  # Silent failure, logged
```

Visualization failures don't crash the pipeline.