Hit-or-Miss Morphological Transformations

May 11, 2025

Abstract

This document demonstrates the implementation of hit-or-miss morphological transformations using NumPy. The technique identifies specific patterns in binary images using directional kernels.

1 Libraries Used

- `numpy`: For numerical operations and array manipulation (essential for image processing)

2 Step-by-Step Process

2.1 Step 1: Import Libraries

```python
import numpy as np
```

2.2 Step 2: Define Binary Image

Create a 14x14 binary image with specific patterns to detect:

```python
image = np.array([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
                  [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
                  [0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0],
                  [0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0],
                  [0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0],
                  [0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0],
                  [0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0],
                  [0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0],
                  [0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0],
                  [0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0],
                  [0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0],
                  [0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0],
                  [0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
                  [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]], dtype
                     =np.int8)
```

2.3 Step 3: Define Directional Kernels

Create 8 specialized kernels for detecting different corner and edge patterns:

```
# Corner kernels
nw_kernel = np.array([[-1, -1,  0],
                      [-1,  1,  1],
                      [ 0,  1,  0]], dtype=np.int8)

ne_kernel = np.array([[ 0, -1, -1],
                      [ 1,  1, -1],
                      [ 0,  1,  0]], dtype=np.int8)

se_kernel = np.array([[ 0,  1,  0],
                      [ 1,  1, -1],
                      [ 0, -1, -1]], dtype=np.int8)

sw_kernel = np.array([[ 0,  1,  0],
                      [-1,  1,  1],
                      [-1, -1,  0]], dtype=np.int8)

# Edge kernels
n_kernel = np.array([[-1, -1,  0],
                     [ 1,  1,  0],
                     [ 0,  0,  0]], dtype=np.int8)

e_kernel = np.array([[ 0,  1, -1],
                     [ 0,  1, -1],
                     [ 0,  0,  0]], dtype=np.int8)

s_kernel = np.array([[ 0,  0,  0],
                     [ 0,  1,  1],
                     [ 0, -1, -1]], dtype=np.int8)

w_kernel = np.array([[ 0,  0,  0],
                     [-1,  1,  0],
                     [-1,  1,  0]], dtype=np.int8)
```

2.4 Step 4: Hit-or-Miss Transformation

Implement the hit-or-miss operation to detect specific patterns:

```
def calculate_hit_or_miss(image, kernel, condition_sum):
    converted_image = np.where(image == 1, 1, -1).astype(np.int8)
    height, width = converted_image.shape
    matrix = np.zeros((height, width), dtype='int8')
    for i in range(1, height-2):
        for j in range(1, width-2):
            result = converted_image[i-1:i+2, j-1:j+2] * kernel
            result = result.flatten().tolist()
            if sum(result) == condition_sum:
                matrix[i, j] = 1
    return matrix

matrix1 = calculate_hit_or_miss(image, nw_kernel, 6)
matrix2 = calculate_hit_or_miss(image, ne_kernel, 6)
matrix3 = calculate_hit_or_miss(image, sw_kernel, 6)
matrix4 = calculate_hit_or_miss(image, se_kernel, 6)
matrix5 = calculate_hit_or_miss(image, n_kernel, 4)
matrix6 = calculate_hit_or_miss(image, e_kernel, 4)
matrix7 = calculate_hit_or_miss(image, s_kernel, 4)
matrix8 = calculate_hit_or_miss(image, w_kernel, 4)
```

2.5 Step 5: Combine Results

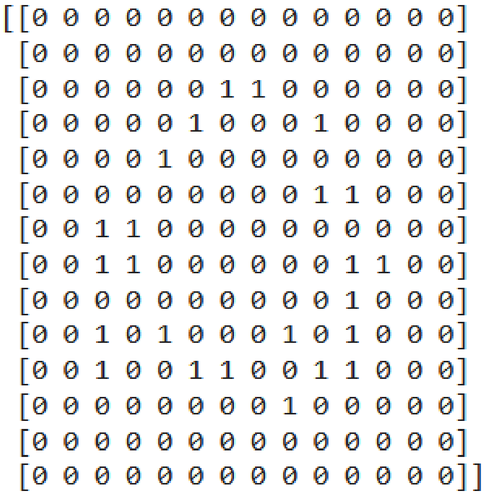
Combine all detection matrices using logical OR:

```
matrices_list = [matrix1, matrix2, matrix3, matrix4,
                 matrix5, matrix6, matrix7, matrix8]
final_matrix = matrices_list[0].copy()

for mat in matrices_list[1:]:
    final_matrix = np.logical_or(final_matrix, mat)

final_matrix = final_matrix.astype(np.int8)
```

2.6 Print Output Section



```
[[0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 1 1 0 0 0 0 0 0]
 [0 0 0 0 0 1 0 0 0 1 0 0 0 0]
 [0 0 0 0 1 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 1 1 0 0 0]
 [0 0 1 1 0 0 0 0 0 0 0 0 0 0]
 [0 0 1 1 0 0 0 0 0 0 1 1 0 0]
 [0 0 0 0 0 0 0 0 0 0 1 0 0 0]
 [0 0 1 0 1 0 0 0 1 0 1 0 0 0]
 [0 0 1 0 0 1 1 0 0 1 1 0 0 0]
 [0 0 0 0 0 0 0 0 1 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0]]
```

Figure 1: Final hit-or-miss transformation result (14×14 array)

3 Technical Explanations

3.1 Hit-or-Miss Transform

- **Purpose**: Detects specific patterns in binary images
- **Kernel Types**: Uses both positive (1) and negative (-1) values to match patterns
- **Condition Sum**: Threshold for successful pattern detection
- **Combination**: Results from multiple kernels are combined using logical OR

3.2 Implementation Notes

- Image converted to (-1, 1) values for pattern matching
- Each kernel detects different corner/edge configurations
- Boundary pixels are ignored in processing
- Final result shows all detected pattern locations

`https://github.com/AsadiAhmad/Hit-and-Miss`