Integral Verification Algorithm

Visual explanation

Based on Dey et al. (2016) Methods

2025-06-18

Introduction

An Integral image, also known as Summed Area Table, is data structure used in computer vision, where each pixel represents the cumulative sum of a corresponding input pixel with all pixels above and left of the input pixel. It enables rapid calculation of summations over image sub-regions. Any rectangular subset of such sub-region can be evaluated in constant time. This concept was introduced by Viola & Jones and allows fast computation of rectangular image features since they enable the summation of image values over any rectangle image region in constant time i.e. computational complexity of O(1) instead of O(n).

The algorithm works like this

- The integral image is the same size as the original image.
- Each pixel in the integral image stores the sum of all pixels above and to the left of it in the original image.
- The sum of pixel values within any rectangular region can be calculated using only four array lookups in the integral image.
  - Let the integral image be denoted by ($ii(x, y)$).
  - Let the original image be ($img(x, y)$).
  - Let the top-left corner of the rectangle be ((x_1, y_1)) and the bottom-right corner be ((x_2, y_2)). The sum of pixels within the rectangle is then calculated as: ($S = ii(x_2, y_2) - ii(x_1 - 1, y_2) - ii(x_2, y_1 - 1) + ii(x_1 - 1, y_1 - 1)$).

Python Implementation

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib.patches import Rectangle
import cv2

class IntegralImage:
    """
    Implementation of Integral Image (Summed Area Table) algorithm
    as described by Viola & Jones
    """

    def __init__(self, image):
        """
        Initialize with input image and compute integral image

        Args:
            image: 2D numpy array representing the input image
        """
        self.original_image = image.astype(np.float64)
        self.rows, self.cols = image.shape

        # Create integral image with padding (one extra row and column of zeros)
        self.integral_image = np.zeros((self.rows + 1, self.cols + 1), dtype=np.float64)

        # Compute integral image
        self._compute_integral()

    def _compute_integral(self):
        """
        Compute the integral image using dynamic programming approach
        Each pixel (i,j) contains sum of all pixels from (0,0) to (i,j)

        Formula: I(x,y) = I(x-1,y) + I(x,y-1) - I(x-1,y-1) + img(x,y)
        """
        for i in range(1, self.rows + 1):
            for j in range(1, self.cols + 1):
                self.integral_image[i, j] = (
                    self.integral_image[i-1, j] +      # Sum above
                    self.integral_image[i, j-1] -      # Sum to left
```

```python
                    self.integral_image[i-1, j-1] +    # Remove double counted
                    self.original_image[i-1, j-1]      # Add current pixel
                )

    def sum_region(self, row1, col1, row2, col2):
        """
        Calculate sum of rectangular region in O(1) time

        Args:
            row1, col1: Top-left corner (inclusive)
            row2, col2: Bottom-right corner (inclusive)

        Returns:
            Sum of all pixels in the rectangular region

        Formula: Sum = D - B - C + A
        where A, B, C, D are integral values at corners
        """
        # Convert to integral image coordinates (add 1 due to padding)
        A = self.integral_image[row1, col1]           # Top-left
        B = self.integral_image[row1, col2 + 1]       # Top-right
        C = self.integral_image[row2 + 1, col1]       # Bottom-left
        D = self.integral_image[row2 + 1, col2 + 1]   # Bottom-right

        return D - B - C + A

# Create a sample image for demonstration
def create_sample_image():
    """Create a simple 6x6 sample image for clear visualization"""
    np.random.seed(42)
    image = np.random.randint(1, 10, (6, 6))
    return image

# Demonstrate the algorithm step by step
sample_img = create_sample_image()
integral_calc = IntegralImage(sample_img)

print("=== INTEGRAL IMAGE ALGORITHM DEMONSTRATION ===\n")
print("Original Image:")
print(sample_img)
print("\nIntegral Image (with padding):")
print(integral_calc.integral_image.astype(int))
```

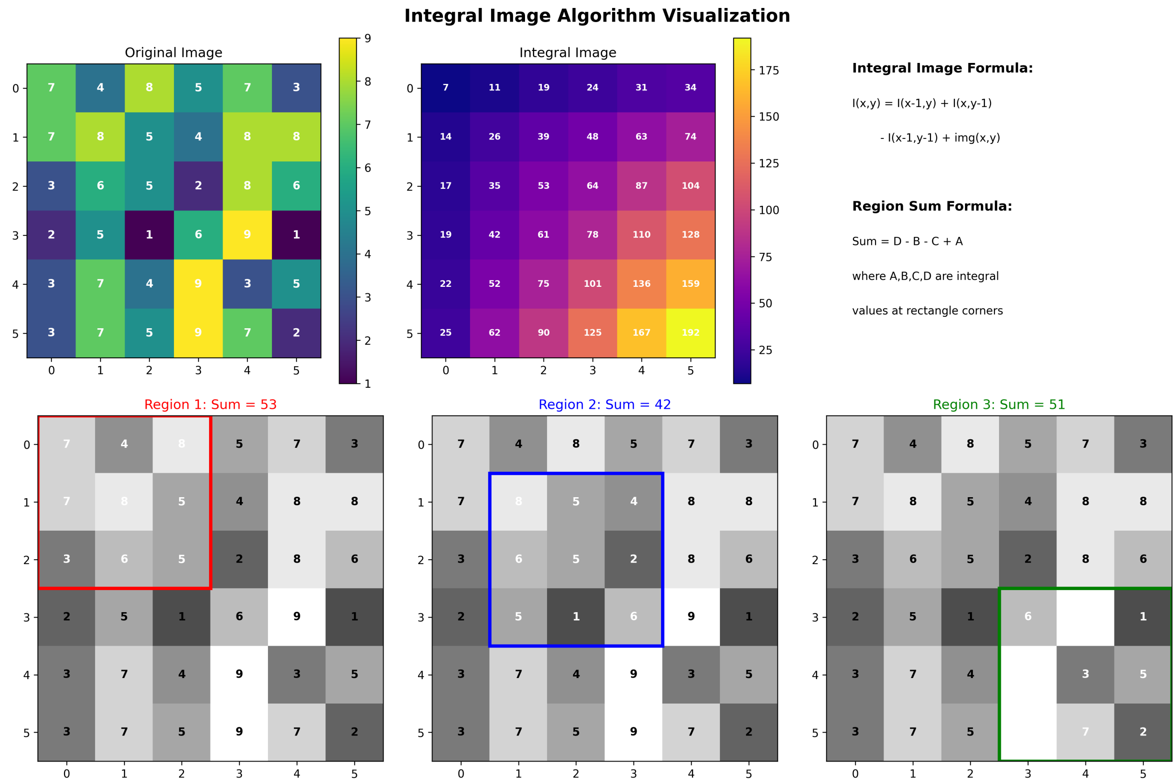
```python
# Test region sum calculation
test_regions = [
    (0, 0, 2, 2),  # 3x3 region from top-left
    (1, 1, 3, 3),  # 3x3 region from center
    (3, 3, 5, 5),  # 3x3 region from bottom-right
]

print("\n=== REGION SUM CALCULATIONS ===")
for i, (r1, c1, r2, c2) in enumerate(test_regions):
    region_sum = integral_calc.sum_region(r1, c1, r2, c2)

    # Verify with direct calculation
    direct_sum = np.sum(sample_img[r1:r2+1, c1:c2+1])

    print(f"\nRegion {i+1}: ({r1},{c1}) to ({r2},{c2})")
    print(f"Integral method: {region_sum}")
    print(f"Direct method: {direct_sum}")
    print(f"Match: {region_sum == direct_sum}")

# Create visualizations
fig, axes = plt.subplots(2, 3, figsize=(15, 10))
fig.suptitle('Integral Image Algorithm Visualization', fontsize=16, fontweight='bold')

# 1. Original Image
im1 = axes[0, 0].imshow(sample_img, cmap='viridis', interpolation='nearest')
axes[0, 0].set_title('Original Image')
for i in range(sample_img.shape[0]):
    for j in range(sample_img.shape[1]):
        axes[0, 0].text(j, i, str(sample_img[i, j]),
                       ha='center', va='center', color='white', fontweight='bold')
plt.colorbar(im1, ax=axes[0, 0])

# 2. Integral Image (without padding for clarity)
integral_display = integral_calc.integral_image[1:, 1:]  # Remove padding for display
im2 = axes[0, 1].imshow(integral_display, cmap='plasma', interpolation='nearest')
axes[0, 1].set_title('Integral Image')
for i in range(integral_display.shape[0]):
    for j in range(integral_display.shape[1]):
        axes[0, 1].text(j, i, str(int(integral_display[i, j])),
                       ha='center', va='center', color='white', fontweight='bold', fontsize=8
plt.colorbar(im2, ax=axes[0, 1])
```

```
# 3. Step-by-step calculation visualization
axes[0, 2].text(0.1, 0.9, 'Integral Image Formula:', fontsize=12, fontweight='bold', transfo
axes[0, 2].text(0.1, 0.8, 'I(x,y) = I(x-1,y) + I(x,y-1)', fontsize=10, transform=axes[0, 2].t
axes[0, 2].text(0.1, 0.7, '         - I(x-1,y-1) + img(x,y)', fontsize=10, transform=axes[0, 2
axes[0, 2].text(0.1, 0.5, 'Region Sum Formula:', fontsize=12, fontweight='bold', transform=ax
axes[0, 2].text(0.1, 0.4, 'Sum = D - B - C + A', fontsize=10, transform=axes[0, 2].transAxes)
axes[0, 2].text(0.1, 0.3, 'where A,B,C,D are integral', fontsize=10, transform=axes[0, 2].tra
axes[0, 2].text(0.1, 0.2, 'values at rectangle corners', fontsize=10, transform=axes[0, 2].tr
axes[0, 2].axis('off')

# 4-6. Region sum demonstrations
colors = ['red', 'blue', 'green']
for idx, (r1, c1, r2, c2) in enumerate(test_regions):
    ax = axes[1, idx]

    # Show original image with highlighted region
    im = ax.imshow(sample_img, cmap='gray', alpha=0.7, interpolation='nearest')

    # Highlight the region
    rect = Rectangle((c1-0.5, r1-0.5), c2-c1+1, r2-r1+1,
                     linewidth=3, edgecolor=colors[idx], facecolor='none')
    ax.add_patch(rect)

    # Add text annotations
    for i in range(sample_img.shape[0]):
        for j in range(sample_img.shape[1]):
            color = 'white' if r1 <= i <= r2 and c1 <= j <= c2 else 'black'
            ax.text(j, i, str(sample_img[i, j]),
                   ha='center', va='center', color=color, fontweight='bold')

    region_sum = integral_calc.sum_region(r1, c1, r2, c2)
    ax.set_title(f'Region {idx+1}: Sum = {int(region_sum)}', color=colors[idx])
    ax.set_xticks(range(sample_img.shape[1]))
    ax.set_yticks(range(sample_img.shape[0]))

plt.tight_layout()
plt.show()

# Performance comparison demonstration
print("\n=== PERFORMANCE ANALYSIS ===")

def naive_region_sum(image, r1, c1, r2, c2):
```

```
    """Naive O(n) approach for comparison"""
    return np.sum(image[r1:r2+1, c1:c2+1])

# Create larger image for performance testing
large_img = np.random.randint(0, 255, (100, 100))
large_integral = IntegralImage(large_img)

import time

# Test multiple regions
test_coords = [(10, 10, 50, 50), (20, 20, 80, 80), (0, 0, 99, 99)]

print("Performance comparison on 100x100 image:")
print("Region\t\tIntegral Time\tNaive Time\tSpeedup")
print("-" * 55)

for r1, c1, r2, c2 in test_coords:
    # Time integral method
    start = time.time()
    for _ in range(1000):
        integral_sum = large_integral.sum_region(r1, c1, r2, c2)
    integral_time = time.time() - start

    # Time naive method
    start = time.time()
    for _ in range(1000):
        naive_sum = naive_region_sum(large_img, r1, c1, r2, c2)
    naive_time = time.time() - start

    speedup = naive_time / integral_time
    print(f"({r1},{c1})-({r2},{c2})\t{integral_time:.6f}s\t{naive_time:.6f}s\t{speedup:.1f}x

print(f"\nSpace Complexity: O(M×N) = O({large_img.shape[0]}×{large_img.shape[1]}) = O({large_
print("Time Complexity for Range Sum Query: O(1)")
print("Time Complexity to Build Integral Image: O(M×N)")
```

```
=== INTEGRAL IMAGE ALGORITHM DEMONSTRATION ===

Original Image:
[[7 4 8 5 7 3]
 [7 8 5 4 8 8]
 [3 6 5 2 8 6]]
```

```
 [2 5 1 6 9 1]
 [3 7 4 9 3 5]
 [3 7 5 9 7 2]]

Integral Image (with padding):
[[  0   0   0   0   0   0   0]
 [  0   7  11  19  24  31  34]
 [  0  14  26  39  48  63  74]
 [  0  17  35  53  64  87 104]
 [  0  19  42  61  78 110 128]
 [  0  22  52  75 101 136 159]
 [  0  25  62  90 125 167 192]]

=== REGION SUM CALCULATIONS ===

Region 1: (0,0) to (2,2)
Integral method: 53.0
Direct method: 53
Match: True

Region 2: (1,1) to (3,3)
Integral method: 42.0
Direct method: 42
Match: True

Region 3: (3,3) to (5,5)
Integral method: 51.0
Direct method: 51
Match: True
```




```
=== PERFORMANCE ANALYSIS ===
Performance comparison on 100x100 image:
Region        Integral Time   Naive Time  Speedup
-------------------------------------------------------
(10,10)-(50,50) 0.000430s   0.003139s   7.3x
(20,20)-(80,80) 0.000432s   0.003606s   8.3x
(0,0)-(99,99)   0.000435s   0.003536s   8.1x

Space Complexity: O(M×N) = O(100×100) = O(10000)
Time Complexity for Range Sum Query: O(1)
Time Complexity to Build Integral Image: O(M×N)
```

References & Resources

Primary References:

- Rohatgi, A. Integral Image. https://medium.com/@anubhavroh/integral-image-141f6181db5e