Detailed Explanation of Gaussian Pyramid Construction in SIFT Algorithm

Deep Amish Shah

April 9, 2025

1 Introduction

This document provides a detailed step-by-step breakdown of the Gaussian pyramid construction used in the SIFT (Scale-Invariant Feature Transform) algorithm. The function is implemented in Python and explained thoroughly, with each line analyzed in depth.

2 Python Code for Gaussian Pyramid

```python
def build_gaussian_pyramid(image, num_octaves=4, num_scales=5, sigma=1.6):
    """
    Build a Gaussian pyramid for the given image.

    Parameters:
      image       - Input grayscale image as a NumPy array.
      num_octaves - Number of octaves to generate.
      num_scales  - Number of scale levels per octave.
      sigma       - Base sigma value for Gaussian blurring.

    Returns:
      pyramid - A list of octaves, each containing a list of blurred images.
    """
    pyramid = []  # Initialize an empty list to hold the pyramid structure
    k = 2 ** (1.0 / (num_scales - 3))  # Compute the scale factor
    current_image = image.copy()  # Copy the original image to avoid modifications

    for octave in range(num_octaves):
        scales = []  # List to store different scales for the current octave
        for s in range(num_scales):
            sigma_total = sigma * (k ** s)  # Compute total sigma for this scale
            blurred = gaussian_filter(current_image, sigma_total)  # Apply Gaussian blur
```

```python
            scales.append(blurred)  # Store the blurred image
        pyramid.append(scales)  # Append the scales to the pyramid
        current_image = current_image[::2, ::2]  # Downsample for
    the next octave

    return pyramid
```

Listing 1: Gaussian Pyramid Construction

3 Step-by-Step Explanation

3.1 Initializing an Empty Pyramid List

```
pyramid = []
```

This initializes an empty list that will later store different octaves. Each octave will contain images at various scales (blur levels).

3.2 Computing the Scale Factor k

```
k = 2 ** (1.0 / (num_scales - 3))
```

The scale factor k is computed such that the total blur doubles after (num$_scales-3)steps.The subtraction of 3 follows from the SIFT design, ensuring additional scale images to detect keypoints rel$

3.3 Copying the Original Image

```
current_image = image.copy()
```

A copy of the original image is created so that modifications (downsampling) do not affect the original input image.

3.4 Looping Over Octaves

```
for octave in range(num_octaves):
```

Each octave contains a set of images that progressively get more blurred.

3.5 Creating a List for Scale Images

```
scales = []
```

This list will store different blurred versions of the image for the current octave.

3.6 Looping Over Scale Levels

```
for s in range(num_scales):
```

This loop creates different blurred versions by adjusting the sigma value.

3.7 Computing Sigma for Each Scale Level

```
sigma_total = sigma * (k ** s)
```

The total sigma is calculated dynamically for each scale level to progressively increase the blur.

3.8 Applying Gaussian Blur

```
blurred = gaussian_filter(current_image, sigma_total)
```

The Gaussian filter is applied to the image to create a blurred version.

3.9 Storing the Blurred Image

```
scales.append(blurred)
```

Each blurred image is appended to the list of scale images.

3.10 Adding the Octave to the Pyramid

```
pyramid.append(scales)
```

Once all scale images for an octave are generated, they are added to the pyramid.

3.11 Downsampling for the Next Octave

```
current_image = current_image[::2, ::2]
```

The image is downsampled by taking every other pixel to prepare for the next octave.

3.12 Returning the Complete Pyramid

```
return pyramid
```

Finally, the pyramid containing multiple octaves with different blur levels is returned.

4 Mathematical Explanation of Scale Factor k

The factor k ensures that the total sigma doubles after a set number of steps:

$$k^{(num\_scales-3)} = 2$$

Solving for k:

$$k = 2^{\frac{1}{num\_scales-3}}$$

For example, if $num\_scales = 5$:

$$k = 2^{\frac{1}{2}} = \sqrt{2} \approx 1.414$$

This ensures a smooth transition in scale-space.