Distinctive Image Features from Scale-Invariant Keypoints

Project ID - 3 | Team name - Team Agile

Samruddhi Shastri (2019111039)
Pranjali Pathre (2019112002)
Anandhini Rajendran (2019101055)
Ayush Goyal (2019111026)

Github link

https://github.com/pranjali-pathre/SMAI-Project

OBJECTIVES TILL MID-EVALUATION:

- Paper and relevant work reading
- Implementing Scale Space and Image Pyramids
- Finding Scale Space Extrema
- Localizing Extrema

Until mid-evaluation we have implemented the topics mentioned above and have verified its working. The results obtained for each part and its working are documented below. The sample image used is:

Implementing Scale Space and Image Pyramids

Theory:

The concept of Scale Space deals with the application of a continuous range of Gaussian Filters to the target image such that the chosen Gaussian have differing values of the sigma parameter.

1. We use the Gaussian function as a scale-space kernel. So the scale space of an image is defined as a result of convolution of a variable-scale Gaussian, *G(x, y, σ)* with an input image.
$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y),$$
2. We use scale-space extrema, i.e., the convolved Gaussian difference to detect the stable points.
$$D(x, y, \sigma) = (G(x, y, k\sigma) - G(x, y, \sigma)) * I(x, y)$$
The maxima and the minima $\sigma^2\nabla^2 G$ produce the most stable image features.
$$G(x, y, k\sigma) - G(x, y, \sigma) \approx (k - 1)\sigma^2\nabla^2 G$$

The above process is implemented in the code as below.

Procedure:

This section primarily included the implementation of setting the number of octaves, generating the Gaussian kernels and images and Difference of Gaussian.

1. Setting the number of octaves:
Number of octaves in an image pyramid is a function of the shape of the base image. It represents the number of times we can repeatedly halve an image until it becomes too small. If *l* is the shorter side length of the image, then we have the

following relation between the number of octaves and the length of the shortest side.

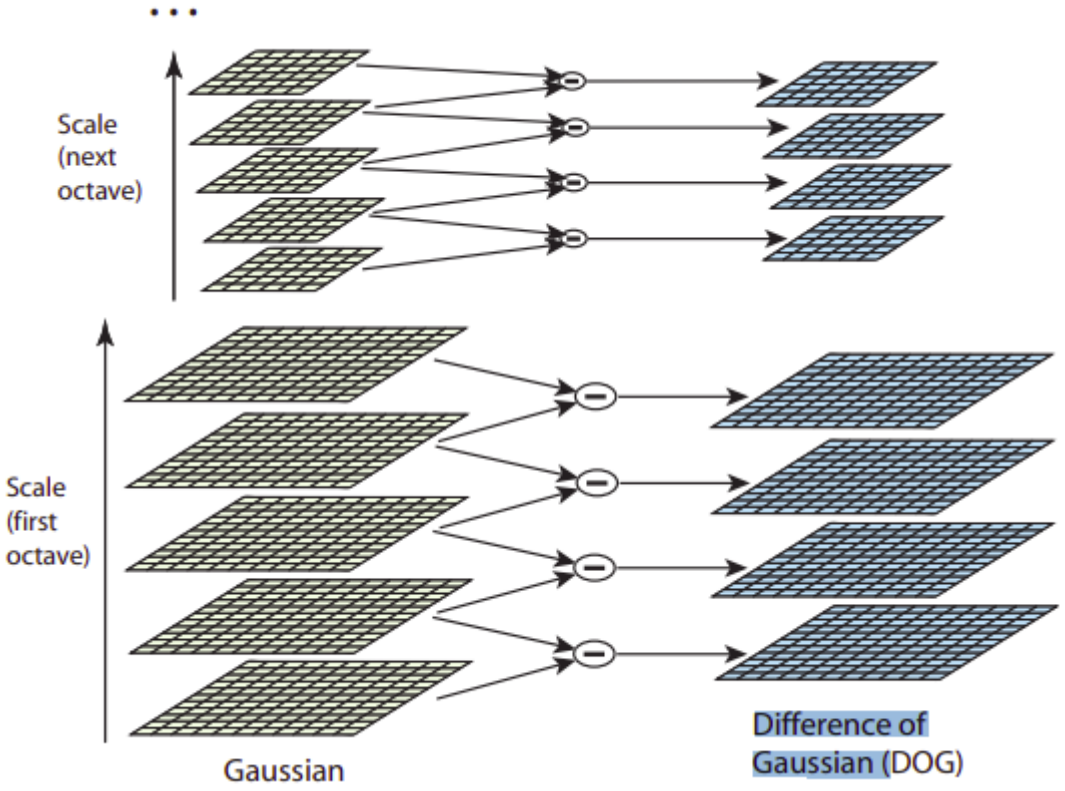
$$l/2^x = 1$$

Here *x* is the number of times we can halve the base image. We can take the logarithm of both sides and solve for *x* to obtain the following:

$$x = log(y)/log(2).$$

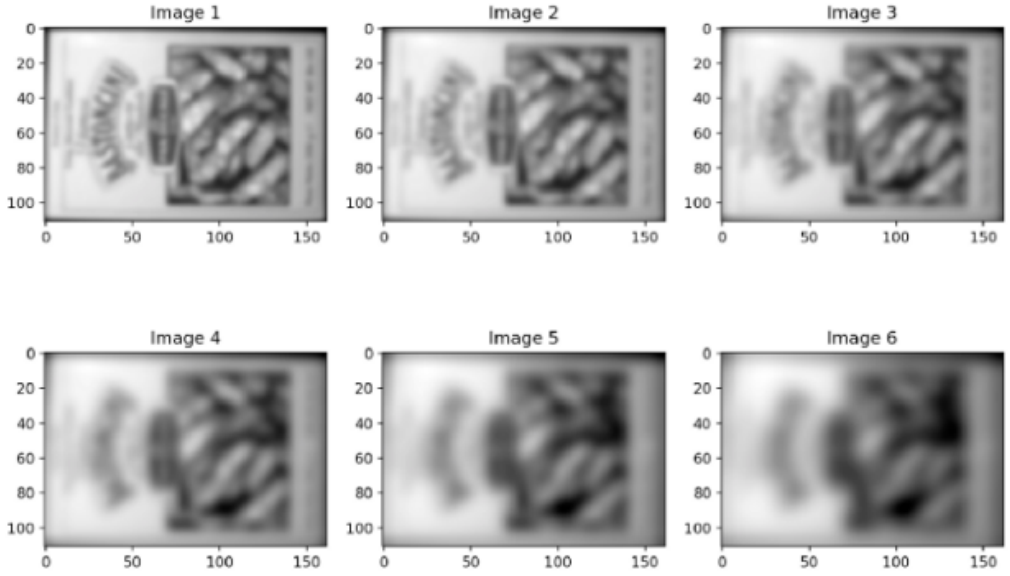
We round *x* down to the nearest integer (floor(x)) to have an integer number of layers in our image pyramid.

2. <u>Generating the Gaussian kernels and images :</u>
   The initial image is incrementally convolved with Gaussians to produce images separated by a constant factor k in scale space, shown stacked in the left column. We choose to divide each octave of scale space (i.e., doubling of σ) into an integer number, s, of intervals, so $k = 2^{1/s}$. We must produce s + 3 images in the stack of blurred images for each octave, so that final extrema detection covers a complete octave.



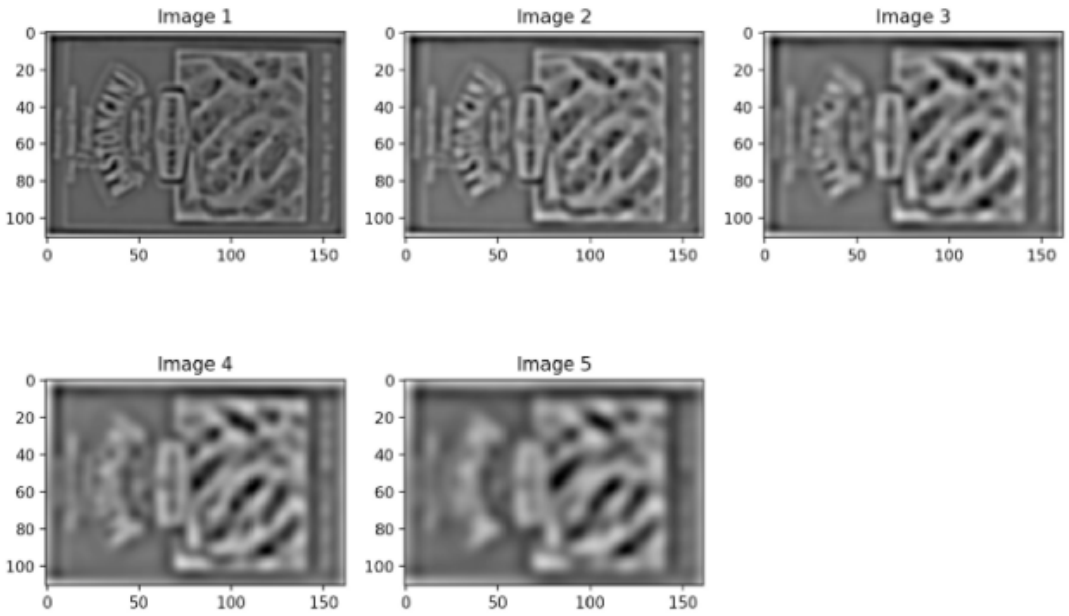
Generating the Gaussian kernels sets the amount of blur for each image belonging to an octave. Each octave has a (number of intervals + 3) images. All the images in an octave have the same dimensions but varying amounts of blur added to each image. We add two extra images at the beginning and the end to

have a total number of intervals + 3 when the (number of intervals +1) images to cover. We start with the initial value set for sigma to blur the image.



3. <u>Generating the Gaussian images difference of Gaussian:</u>
   Next for generating Gaussian images, we blur each image according to the value set in the Gaussian kernel.
   Now that we have Gaussian images, for generating the difference of Gaussian images, we subtract two successive pairs of blurred images.

   The output of the sample image used for a particular octave is displayed below.



Scale-space extrema detection

AIM: Detecting locations that are invariant to scale change of the image.
METHOD:
1.Can be accomplished by searching for stable features across all possible scales, using a continuous function of scale known as scale space.

```
scale space: L(x, y, σ) = G(x, y, σ) * I(x, y)
where G: gaussian kernel G(x, y, σ) = (1/2πσ^2)* e ^(-(x^2+y^2 )/2σ^2): gaussian_images
```

2. These stable keypoint locations can be calculated using scale-space extrema in the difference-of-Gaussian function convolved with the image, D(x, y, σ), using formula:

```
D(x, y, σ) = L(x, y, kσ) - L(x, y, σ): dog_images
```

Why we should choose Difference-of-Gaussian (Dog) : The difference-of-Gaussian function provides a close approximation to the scale-normalized Laplacian of Gaussian.

```
G(x, y, kσ) - G(x, y, σ) ≈ (k - 1)σ^2∇^2 G
```

EFFICIENT computation of D(x, y, σ):

1. convolve the initial image by gaussian incrementally by factor k
2. divide each octave into intervals of s such that k = 2^1/s.
3. Take 3 adjacent images and subtract to get D(x, y, σ)
4. Resample the image using 2σ by taking the second pixel of every row and column.

3. In this function we find the pixel extremas using Difference-of-Gaussian and then add the localized extremas to the key points if they exist.

Finding The Local Extrema

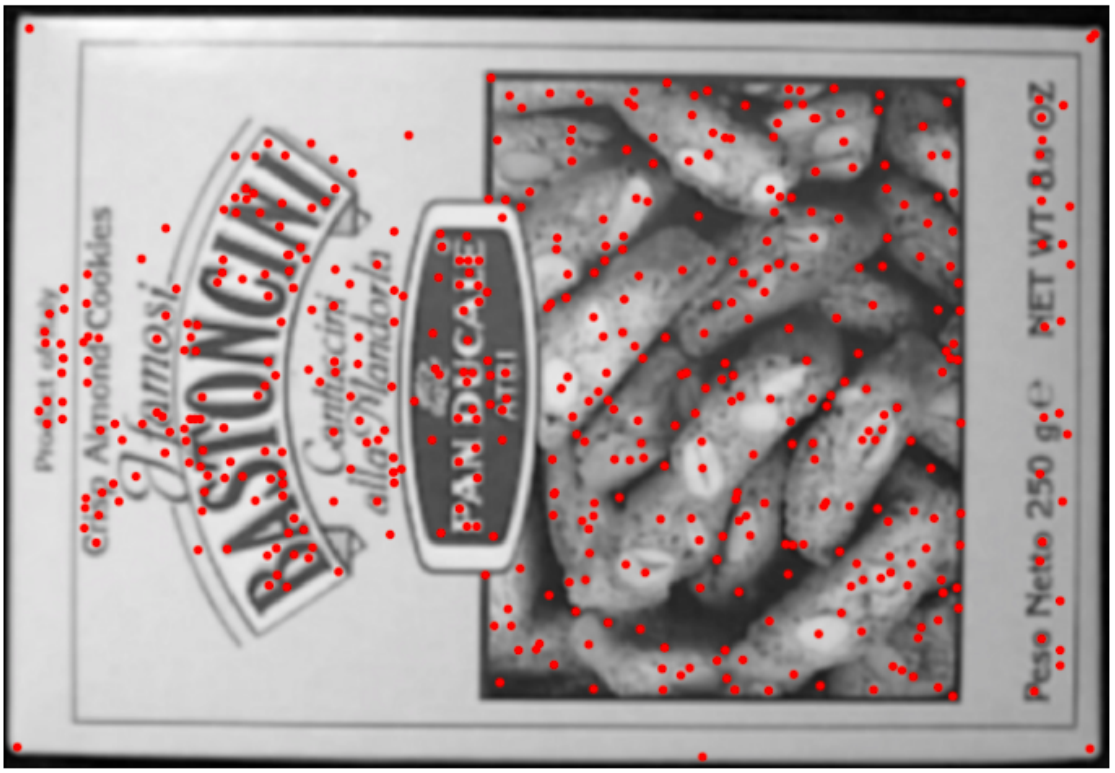
We have to compute the local extrema of `D(x, y, σ)`, to find that we compare each sample point to its 8 neighbours and 9 neighbours in the scale above and below. It is an extreme if the point is smaller or larger than all of these neighbours.

MINIMA:

```
truth_value = all(center_pixel <= sub_img1) and all(center_pixel <=
sub_img3) and all(center_pixel <= sub_img2[0, :]) and all(center_pixel
<= sub_img2[2, :]) and center_pixel <= sub_img2[1, 0] and center_pixel
<= sub_img2[1, 2]
```

MAXIMA:

```
truth_value = all(center_pixel >= sub_img1) and all(center_pixel >=
sub_img3) and all(center_pixel >= sub_img2[0, :]) and all(center_pixel
>= sub_img2[2, :]) and center_pixel >= sub_img2[1, 0] and center_pixel
>= sub_img2[1, 2]
```



Localizing Extrema

A quadratic model is fitted to the input keypoint pixel and all 26 of its neighboring pixels. We update the keypoint's position with the sub pixel-accurate extremum estimated from this model. We iterate at most 5 times until the next update moves the keypoint less than 0.5 in any of the three directions. This means the quadratic model has converged to one pixel location. The two helper functions `centerPixelGradientComputation()` and `centerPixelHessianComputation()` implement second-order central finite difference approximations of the gradients and hessians in all three dimensions.