Digital Imaging Systems - Fall 2024
Project 3 (Option 2: Implementation of Laplacian Blob Detector)

Shahriar Mohammad Sajid, Halim Mondol, Prasun Datta

Contents

[0] **Our team wants to participate in the competition for bonus points.**

Contributions:
a) Shahriar Mohammad Sajid (Email: mshahri@ncsu.edu)
b) Halim Mondol Vai (Email: mmondol@ncsu.edu)
c) Prasun Datta (Email: pdatta2@ncsu.edu)

Contribution	**Author**
Problem formulation and algorithm development	All
Implementation of LoG	Shahriar Mohammad Sajid, Halim Mondol
Implementation of Non-max Suppression	All
Visualizing blobs and intermediate outputs	Halim Mondol, Prasun Datta
Result analysis and testing	All
Report writing	Prasun Datta, Shahriar Mohammad Sajid

1 Explanation of Algorithm

The Laplacian Blob Detector is a fundamental technique in computer vision designed to locate regions of interest, known as blobs, in images. The detector relies on constructing a scale space, applying the Laplacian of Gaussian (LoG) operator, and utilizing non-maximum suppression to pinpoint blob centers at multiple scales.

1.1 Laplacian of Gaussian (LoG) Filter Generation

The algorithm begins by generating a Laplacian of Gaussian (LoG) filter, which combines Gaussian smoothing with the Laplacian operator to effectively highlight regions with rapid intensity changes in the image. The LoG function in two dimensions is given by:

$$LoG(x, y, \sigma) = \frac{(x^2 + y^2 - 2\sigma^2)}{\sigma^4} \cdot \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right) / (2\pi) \tag{1}$$

where x and y represent spatial coordinates, and σ is the scale parameter that controls the extent of smoothing.

Filter Generation Across Scales. The first step involves creating a series of LoG filters across different scales to capture blobs of varying sizes. These filters are generated iteratively, starting with an initial scale σ_{init} and successively increasing the scale by a factor k for each level:

$$\sigma_i = \sigma_{init} \cdot k^i, \quad \text{for } i = 0, 1, \ldots, n-1 \tag{2}$$

The kernel size is set as a function of the current scale to ensure adequate coverage of the spatial domain:

$$K_i = 2 \cdot \lfloor 3\sigma_i \rfloor + 1 \tag{3}$$

This ensures that the filter extends far enough to account for the Gaussian's spread, effectively covering a range of 3 standard deviations from the mean.

1.2 Laplacian Scale Space Construction

Once the LoG filters are generated, they are convolved with the input image to build the Laplacian scale space.

Convolution with LoG Filters. Let the input image be I. For each level in the scale space, the convolution operation is performed:

$$R_i = I * LoG_i \tag{4}$$

where $*$ denotes the 2D convolution, and R_i represents the response at scale i. The convolution operation highlights regions with maximum variance, such as edges and blob centers. The convolution operation can be mathematically expressed as:

$$R_i(x, y) = \sum_{u=-\lfloor K_i/2 \rfloor}^{\lfloor K_i/2 \rfloor} \sum_{v=-\lfloor K_i/2 \rfloor}^{\lfloor K_i/2 \rfloor} I(x-u, y-v) \cdot LoG_i(u, v) \tag{5}$$

Saving the Response. The squared response values (R_i^2) are saved for each scale, preserving the positive and negative variations. This helps robustly identify blob-like structures in later steps. The response strength indicates the degree of similarity of the region to a blob, with higher values corresponding to higher likelihoods of blob presence.

1.3 Non-Maximum Suppression in Scale Space

Non-maximum suppression is applied across the generated scale space to extract blob locations at different characteristic scales.

Three-Dimensional Non-Maximum Suppression. The scale space forms a three-dimensional matrix of convolution responses with dimensions corresponding to the height, width, and scale level. To locate blobs, the response at each pixel is compared against its neighbors in the 3D neighborhood across spatial and scale dimensions. Specifically:

1. For each level i, compare each pixel's value $R_i(x, y)$ with its neighbors in the same level as well as adjacent levels $(i-1, i+1)$.

2. A pixel is identified as a potential blob if it is the local maximum among all its neighbors and surpasses a given threshold value TT. Mathematically:

$$R_i(x, y) > T \quad \text{and} \quad R_i(x, y) > R_{neighbor}(x', y') \quad \forall (x', y') \in \mathcal{N}(x, y) \tag{6}$$

Here, $\mathcal{N}(x, y)$ represents the set of 8 spatial neighbors in the same scale and 18 neighbors in adjacent scales, resulting in a comprehensive 3D suppression process.

The radius of each blob is calculated using the formula:

$$r_i = \sqrt{2} \cdot \sigma_i \tag{7}$$

This radius corresponds to the scale at which the blob response is maximized, giving an accurate representation of the size of the detected blob.

1.4 Result Display with Blobs Marked

After detecting all possible blobs across different scales, the resulting regions of interest are plotted as circles on the original image.

Visualization. To represent blobs, the centers and corresponding radii are used to draw circles on the image using the OpenCV function '*cv2.circle*. This visualization clearly demonstrates how blobs of different sizes are captured by the Laplacian Blob Detector, providing intuitive insight into the effectiveness of the multi-scale approach.

The final visualization can be represented mathematically as plotting circles with centers (x_i, y_i) and radii r_i for each detected blob:

$$(x_i, y_i, r_i) \quad \text{for all detected blobs} \tag{8}$$

In summary, the algorithm can be presented in the following way.

2 Implementation

In this section, I explain how each code chunk implements the corresponding steps of the algorithm.

2.1 Code Chunk: LoG Function

Algorithm 1 Laplacian Blob Detector Algorithm

Input: Image I, Initial scale σ_{init}, Scale factor k, Number of levels n, Threshold T
Output: List of blobs with centers (x_i, y_i) and radii r_i
Initialization:
Generate Laplacian of Gaussian (LoG) filters for multiple scales:
for $i = 0$ to $n - 1$ **do**
    $\sigma_i \leftarrow \sigma_{init} \cdot k^i$
    $K_i \leftarrow 2 \cdot \lfloor 3\sigma_i \rfloor + 1$ ▷ Kernel size
    Generate $LoG_i(x, y, \sigma_i)$ using:

$$LoG(x, y, \sigma_i) = \frac{(x^2 + y^2 - 2\sigma_i^2)}{\sigma_i^4} \cdot \exp\left(-\frac{x^2 + y^2}{2\sigma_i^2}\right) / (2\pi)$$

end for
Build Laplacian Scale Space:
for $i = 0$ to $n - 1$ **do**
    Compute response $R_i = I * LoG_i$, where $*$ denotes 2D convolution:

$$R_i(x, y) = \sum_{u=-\lfloor K_i/2 \rfloor}^{\lfloor K_i/2 \rfloor} \sum_{v=-\lfloor K_i/2 \rfloor}^{\lfloor K_i/2 \rfloor} I(x - u, y - v) \cdot LoG_i(u, v)$$

    Save R_i^2 for non-maximum suppression.
end for
Non-Maximum Suppression in Scale Space:
for $i = 0$ to $n - 1$ **do**
    for all (x, y) in R_i **do**
        Compare $R_i(x, y)$ with neighbors in spatial and scale dimensions.
        if $R_i(x, y) > T$ **and** $R_i(x, y)$ is local maximum **then**
            Calculate radius $r_i = \sqrt{2} \cdot \sigma_i$
            Append (x, y, r_i) to the list of blobs.
        end if
    end for
end for
Return: List of blobs (x_i, y_i, r_i)

```
    def LoG(level, factor, sigma_init):
    """
    Generate Laplacian of Gaussian (LoG) filters for multiple scales.
    """
    log_kernels = []
    sigma_values = []
    for i in range(level):
        sigma = sigma_init * (factor ** i)
        sigma_values.append(sigma)
        kernel_size = 2 * int(3 * sigma) + 1
        ax = np.arange(-int(kernel_size / 2), int(kernel_size / 2) + 1)
        xx, yy = np.meshgrid(ax, ax)
        norm = (xx**2 + yy**2) / (2 * sigma**2)
        log_kernel = ((xx**2 + yy**2 - 2 * sigma**2) / (sigma**4))
        * np.exp(-norm) / (2 * math.pi)
        log_kernels.append(log_kernel)
    # Visualization of LoG Filters

    plt.figure(figsize=(15, 5))
    for idx, kernel in enumerate(log_kernels):
        plt.subplot(1, level, idx + 1)
        plt.imshow(kernel, cmap='gray')
        plt.title(f'sigma={sigma_values[idx]:.2f})')
        plt.axis('off')
    plt.suptitle('Laplacian of Gaussian (LoG) Filters')
    plt.tight_layout()  # Add proper spacing between plots
    plt.show()
    return log_kernels, sigma_values
```

Explanation.

- The LoG function generates Laplacian of Gaussian filters at multiple scales.
- The for loop iteratively generates filters for different values of σ, which start from an initial scale σ_{init} and are increased by a factor k in each iteration.
- The kernel size is set to $3\sigma\rfloor+1$, ensuring a sufficient filter window to capture Gaussian spread effectively.
- This step directly implements the **Filter Generation Across Scales** as explained in the algorithm, allowing the capture of blobs of different sizes.

2.2 Code Chunk: conv2d and build_scale_space Fucntions

```
def conv2d(f, w):
    ...
    # Custom convolution operation with zero padding
    for channel in range(c):
        for i in range(h):
            for j in range(w_img):
                region = padded_img[i:i + kernel_h, j:j + kernel_w, channel]
                output[i + pad_size, j + pad_size, channel] = np.sum(region * w)
```

```
    ...
    return output_cropped.squeeze() if c == 1 else output_cropped

def build_scale_space(img, log_kernels):
    scale_space = []
    for kernel in log_kernels:
        response = conv2d(img, kernel)
        scale_space.append(response)
    return scale_space

```

Explanation.

- The conv2d function performs a 2D convolution of the image with the LoG filters. This implementation provides explicit control over the convolution process.
- The build_scale_space function then calls conv2d iteratively for each filter to build the Laplacian Scale Space.
- Each $R_i = I * LoG_i$ represents the response at a specific scale. These responses are stored to form a 3D matrix of convolution outputs, reflecting the intensity changes over different scales.

2.3 Code Chunk: nonMaxSup Function

```
def nonMaxSup(convout, sigma, level, threshold):
    ...
    for scale_idx in range(level):
        radius = int(math.sqrt(2) * sigma[scale_idx])
        current_scale = convout[scale_idx]
        ...
        maxima_mask = (current_scale == max_in_neighbors) & (current_scale > threshold)
        for row, col in np.argwhere(maxima_mask):
            blobs.append((col, row, radius))
    return blobs
```

Explanation.

- The 'nonMaxSup' function applies **Non-Maximum Suppression** to find local maxima in the 3D scale space.
- It identifies points that are stronger than their 3D neighbors, effectively isolating the blob centers.
- The condition $R_i(x, y) > T$ and $R_i(x, y) > R_{neighbor}(x', y')$ is used to determine whether a point is a local maximum.
- The radius for each detected blob is computed as $r_i = \sqrt{2} \cdot \sigma_i$, ensuring that the blob size correlates to the scale at which it was detected.

2.4 Code Chunk; plot_blobs Function

```
def plot_blobs(img, blobs):
    for x, y, r in blobs:
        cv2.circle(img, (x, y), r, (255, 0, 0), 2)
    plt.figure(figsize=(10, 10))
    plt.imshow(img)
    plt.title('Detected Blobs')
    plt.axis('off')
    plt.show()

```

Explanation.

- The 'plot_blobs' function visualizes the final results by drawing circles around each detected blob.
- Using the computed coordinates (x_i, y_i) and radii r_i, each detected feature is highlighted, making it easy to verify that the blob detector is functioning correctly.

3 Result Analysis

In this section, we will visualize and analyze outputs at each stage of the algorithm.

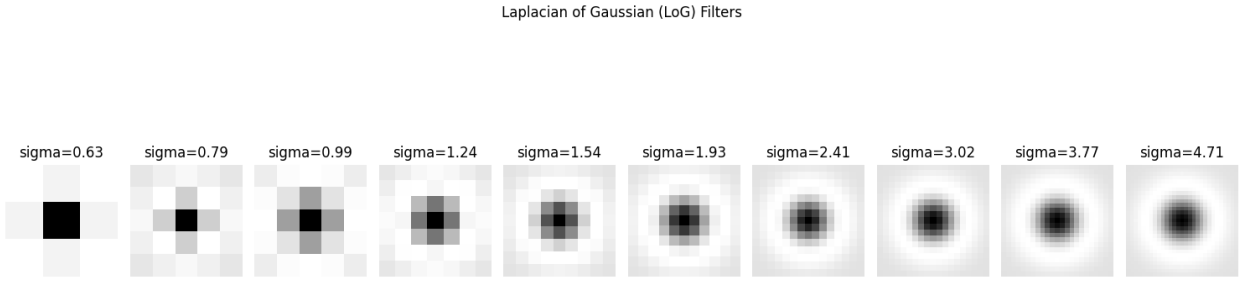


Figure 1: Output of the Laplacian of Gaussian (LoG) Filter Generation. The filters show different degrees of Gaussian smoothing.

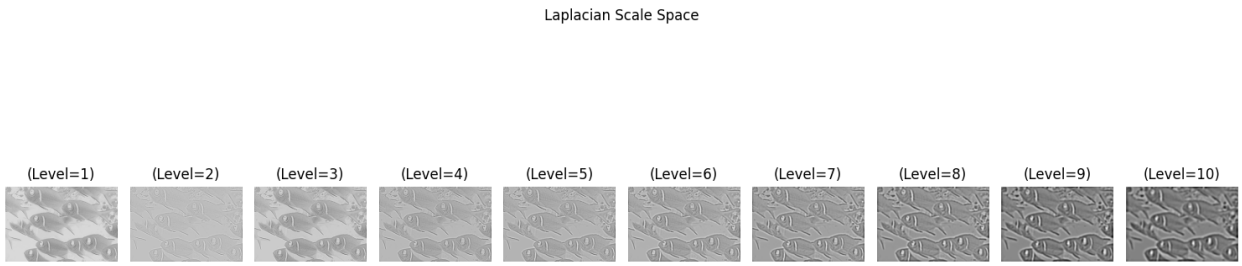
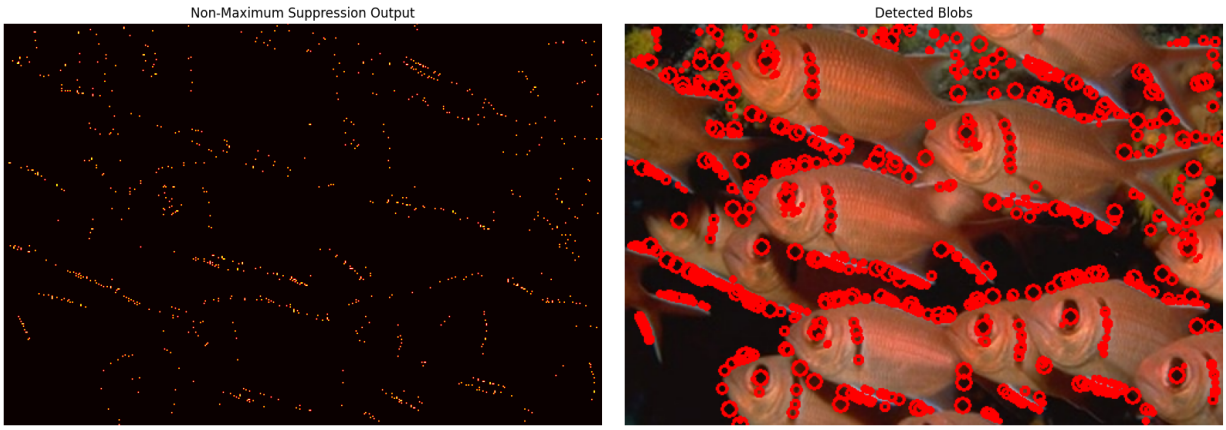


Figure 2: Output of the Laplacian Scale Space Construction. As the level increases, we can see larger features are enhanced.

We use the provided *fishes. jpg* to generate these outputs. We generate filters at 10 different scales (level = 10). starting with an initial standard deviation ($\sigma_{init} = 0.632$). we increment σ by



(a) (b)

Figure 3: Output of (a) the non-max suppression and (b) blob detection. The output (a) after non-max suppression provides a binary map highlighting potential blob centers. The final output (b) visualizes the detected blobs by drawing circles around them.

factor of 1.25. As a threshold for non-max suppression, we use 0.05. Fig. 1 shows the filters with different degrees of Gaussian smoothing. Smaller scales focus on fine features, while larger scales capture coarser structures. By applying filters of increasing sizes, the algorithm ensures that it can detect both small blobs (such as finer details like small fish eyes) and larger blobs (e.g., the broader features of larger fish).

Fig. 2 visualizes each level of scale space. We can observe how different features become prominent as the scale increases. As shown in Fig. 2 for smaller values of σ, the convolution primarily highlights smaller features, such as the sharp edges of the fish eyes. For larger values of σ (higher levels of the scale space), larger features are enhanced, with the responses becoming more generalized, capturing broader aspects of the fish's shape. This allows for smoother detection of larger fish or areas where several fish overlap.

Fig. 3(a) shows the output after non-maximum suppression. The **non-maximum** suppression step ensures that only the most significant points representing potential blob centers are retained. The threshold value 0.05 serves as a filter to eliminate spurious responses that might arise due to noise. This helps in ensuring that only the most prominent features are retained, reducing false positives. As shown in Fig. 3(a), non-maximum suppression is likely to isolate the dark, circular fish eyes as they are prominent features against the fish body, which has a relatively lower variance.

Fig. 3(b) shows the detected blobs. The circles are centered on the fish eyes, indicating high accuracy in blob localization and effective non-max suppression.

4 Factor Analysis

In this section, we will conduct factor analysis by changing the values of some factors that impact the blob detection quality and the algorithm runtime. We will vary the following factors.

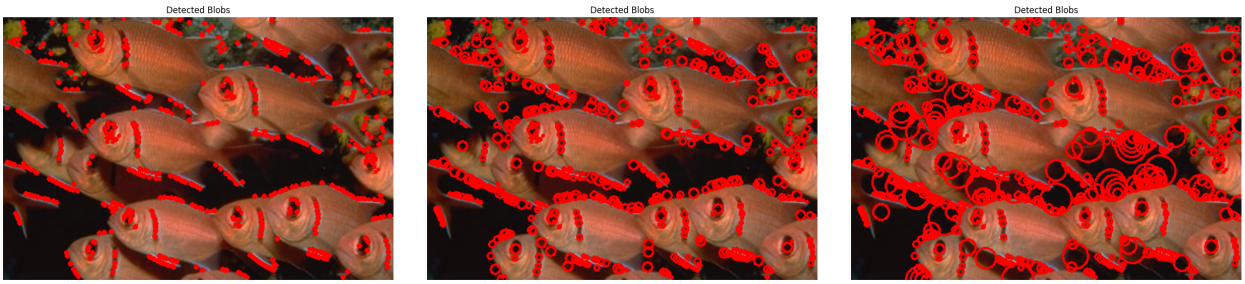
1. Number of Levels
2. Scale Factor
3. Initial Scale (σ_{init})

4. Threshold Value

We note that our optimized setting for blob detection is given by the following: number of levels = 10, scale factor = 1.25, initial scale = 0.63, and threshold value = 0.05. We only vary one factor at a time to observe the effects of that factor.

4.1 Number of Levels

For analyzing the impact of the number of levels, we will try three different values, levels $\in \{5, 10, 15\}$.



(a) (b) (c)

Figure 4: Detected blobs for a number of levels (a) 5, (b) 10, and (c) 15. Fewer levels (< 10) (a) cause reduced granularity in scale space and may miss blobs that do not fit available scales. More levels (> 10) (c) show higher sensitivity, capturing blobs of various sizes. This may detect potential redundant blobs.

Impact on Blob Detection Quality. With fewer levels (level = 5), the *scale space* becomes less granular, meaning that only a limited range of feature sizes is effectively captured. The distance between consecutive scales becomes larger, leading to possible *gaps* in feature detection. Blobs that fall between these fewer scales may be missed entirely because the scales are not fine enough to capture all possible feature sizes. As shown in Fig. 4(a), smaller fish eyes or smaller fishes may be overlooked if the specific scale necessary to capture them does not exist. As can be seen in Fig. 4(c), increasing the number of levels (level = 15) results in a *more detailed scale* space with finer increments between scales. More levels lead to increased sensitivity in blob detection, as features of different sizes are more likely to align closely with one of the available scales. This is particularly beneficial for an image like the fish image, where the eyes of the fish may vary in size due to perspective and proximity to the camera. From Fig. 4(c), we can see that with more levels, there is a higher chance of redundant detections for the same blob at nearby scales.

Impact on Runtime. We can see from TABLE 1, increasing the number of levels increases the runtime. Reducing the number of levels (level = 5) decreases the computational load. Each

Table 1: Runtime for Different Number of Levels.

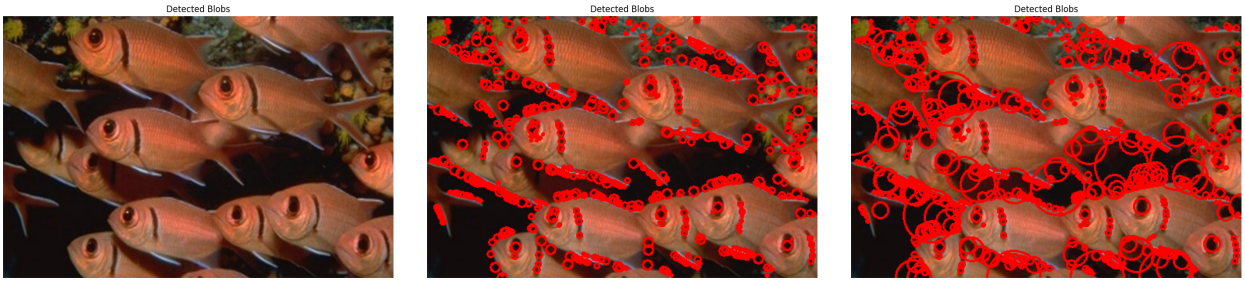
Number of Levels	Runtime (second)
5	4.51
10	9.99
15	18.11

level requires a convolution operation with the image, and fewer levels mean fewer convolutions are

needed. With fewer levels, the memory required to store the *scale space* is reduced. Although the runtime is significantly reduced, this speed improvement comes at the cost of *detection quality*. Important features may be missed, leading to incomplete or inaccurate blob detection, as shown in Fig. 4(a). Adding more levels (level = 15) means that more convolution operations need to be performed, directly increasing the overall runtime. While the runtime increases, the benefit is a more comprehensive scale space, leading to higher detection accuracy. This is evident from Fig. 4(c).

4.2 Scale Factor

We have used three different scale factorsuch as $1.00, 1.25, 1.50$.



(a) (b) (c)

Figure 5: Detected blobs for scale factor (a) 1, (b) 1.25, and (c) 1.50. Increasing the scale factor increases the blob detection accuracy.

Impact on Blob Detection Quality. From Fig. 5(a), it is evident that a smaller scale factor (factor = 1.00) leads to more widely spaced scales, which makes the algorithm more computationally efficient but less effective at capturing blobs that fall between scales. This could result in *missed detections*, especially for smaller or less distinct blobs. A larger scale factor (factor = 1.50) provides finer scale resolution, which means better adaptability to subtle variations in blob size. This helps detect blobs that vary slightly in size, such as fish eyes of different sizes or orientations. However, this may lead to *redundant detections* of the same feature across multiple scales.

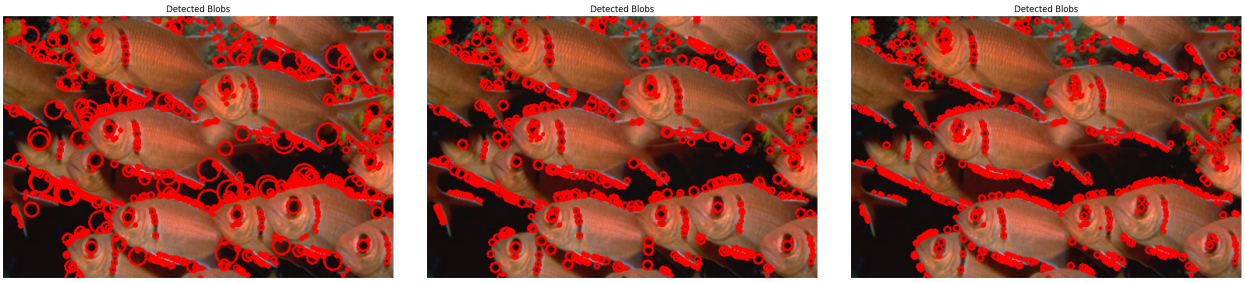
Table 2: Runtime for Different Scale Factors.

Scale Factors	Runtime (second)
1	9.13
1.25	9.61
1.50	15.69

Impact on Runtime. As shown in TABLE 2, a smaller scale factor leads to shorter runtime and more efficient memory usage but potentially compromises the detection quality for blobs that fall between scales. A larger scale factor results in longer runtime and higher memory usage due to more convolution operations and more detailed scale space construction.

4.3 Initial Scale (σ_{init})

We have used three different values of σ_{init}, such as $\frac{1}{\sqrt{0.5}}$, $\frac{1}{\sqrt{2.5}}$, and $\frac{1}{\sqrt{4.5}}$.



(a) (b) (c)

Figure 6: Detected blobs for initial scale (a) $1/\sqrt{0.5}$, (b) $1/\sqrt{2.5}$, and (c) $1/\sqrt{4.5}$. Decreasing the σ_{init} increases the blob detection accuracy.

Impact on Blob Detection Quality. As shown in Fig 6(a), larger initial scale ($\sigma_{init} = 1/\sqrt{0.5}$) increases the sensitivity to small-scale features, resulting in more detections. It is useful for detecting finer details but might lead to cluttered results. From Fig. 6(c), we can see that smaller initial scale ($\sigma_{init} = 1/\sqrt{4.5}$) results in the detection of larger blobs, such as the overall shapes of fish bodies, while smaller details, like tiny fish eyes or small spots, may be smoothed out and missed. This helps in reducing false positives, particularly those caused by small random variations.

Table 3: Runtime for Different Initial Scale (σ_{init}).

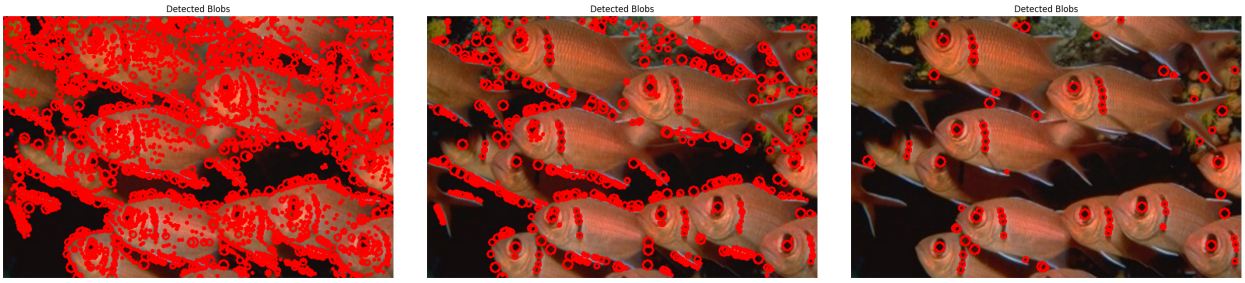
σ_{init}	**Runtime (second)**
$1/\sqrt{0.5}$	11.69
$1/\sqrt{2.5}$	9.27
$1/\sqrt{4.5}$	8.84

Impact on Runtime. TABLE 3 shows that a larger initial scale requires a longer runtime and also results in a larger filter kernel size. This means that each convolution operation requires more computations, as the filter covers a larger area of the image. This generally increases the runtime per convolution. With a larger initial scale, the feature sizes are already relatively broad, meaning fewer levels are needed to span the range of scales. This reduces the total number of convolution operations, thus partially compensating for the increased computational cost of each individual convolution. A smaller initial scale results in a smaller filter kernel, which in turn requires fewer computations for each convolution. Since the algorithm starts with a smaller σ_{init}, more levels are needed to reach the desired range of blob sizes, especially if the objective is to cover large blobs as well. This can increase the overall number of convolutions, which affects runtime. However, the individual convolutions will be relatively fast due to the smaller kernel size.

4.4 Threshold Value

We will use three threshold values, such as $0.01, 0.05$, and 0.1.

Impact on Blob Detection Quality. As shown in Fig. 7(a), lower threshold leads to increased sensitivity, resulting in more blobs being detected, including weak and small features. However, this comes at the cost of increased false positives and a more cluttered detection result. Fig. 7(c) shows the result of a higher threshold. Higher thresholds focus the detection on strong responses, resulting in fewer blobs being detected but with a lower false positive rate. This provides a cleaner



(a) (b) (c)

Figure 7: Detected blobs for threshold values of (a) 0.01, (b) 0.05, and (c) 0.1, used in non-max suppression. Increasing the threshold value decreases the blob detection accuracy.

result but may miss some genuine features that do not generate a strong response, especially if they are not well contrasted.

Table 4: Runtime for Different Threshold Value.

Threshold Value	**Runtime (second)**
0.01	9.18
0.05	9.28
0.1	9.19

Impact on Runtime. TABLE 4 shows that threshold value does not significantly impact runtime.

5 Visualization & Runtime of All Images

In this section, we visualize outputs for all instructor-provided images and our chosen images. We will also measure the runtime for each of them. We use the optimum parameters to visualize the outputs and measure runtime for all images.

Output Visualization of Sunflower

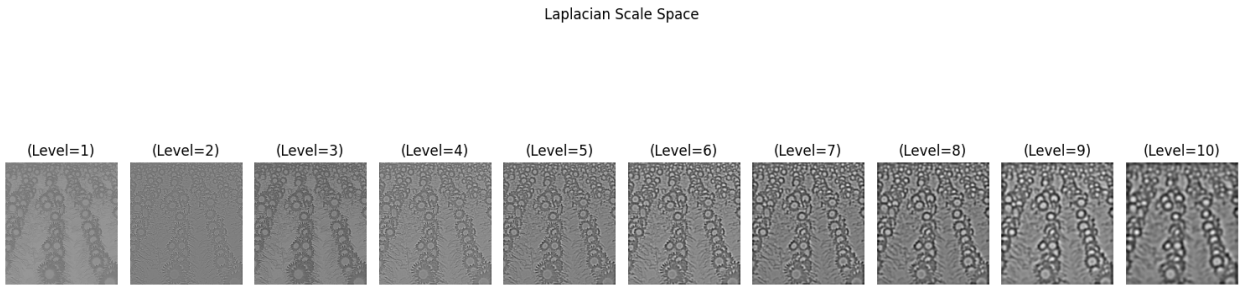
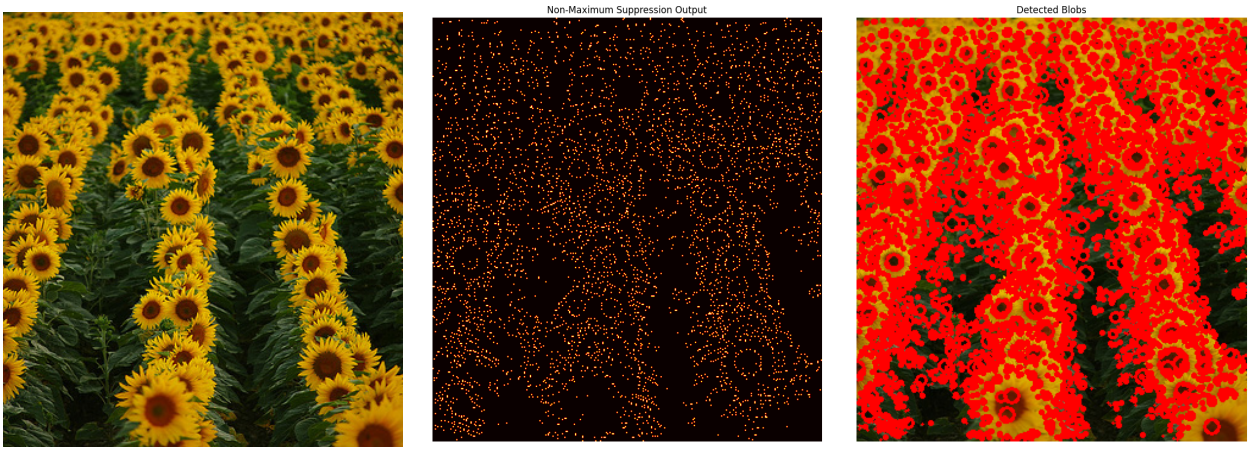


Figure 8: Output of the Laplacian Scale Space Construction. As the level increases, we can see larger features are enhanced.



(a) (b) (c)

Figure 9: (a) Plotting the original sunflower image. Output of (b) the non-max suppression and (b) blob detection. The output (c) after non-max suppression provides a binary map highlighting potential blob centers. The final output (b) visualizes the detected blobs by drawing circles around them.

Output Visualization of Einstein

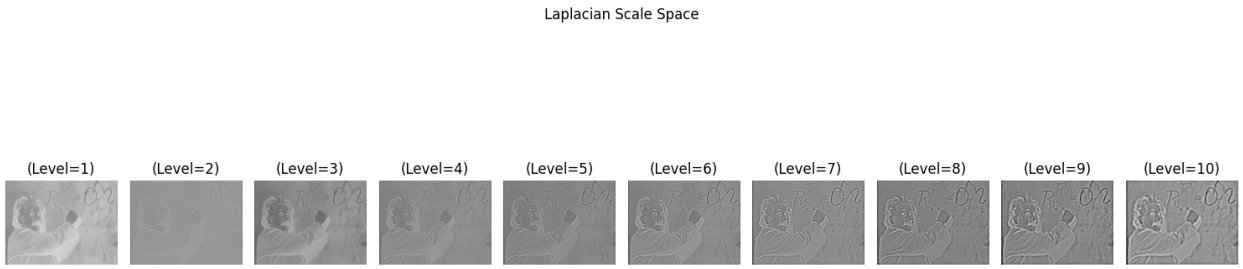


Figure 10: Output of the Laplacian Scale Space Construction. As the level increases, we can see larger features are enhanced.

(a) (b) (c)

Figure 11: (a) Plotting the original Einstein image. Output of (b) the non-max suppression and (b) blob detection. The output (c) after non-max suppression provides a binary map highlighting potential blob centers. The final output (b) visualizes the detected blobs by drawing circles around them.

Output Visualization of Butterfly

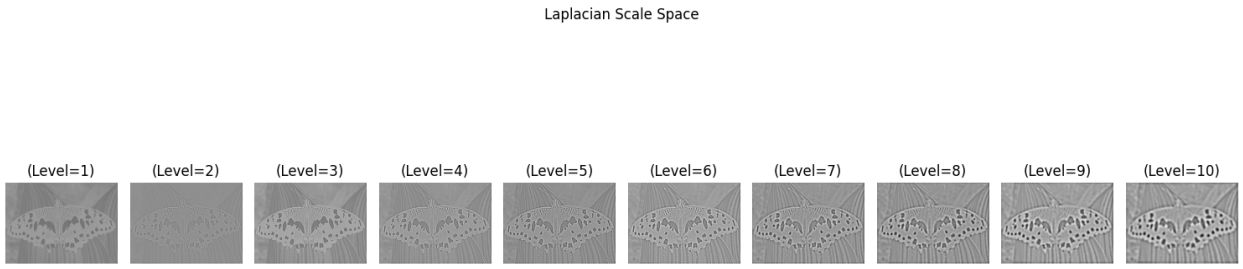
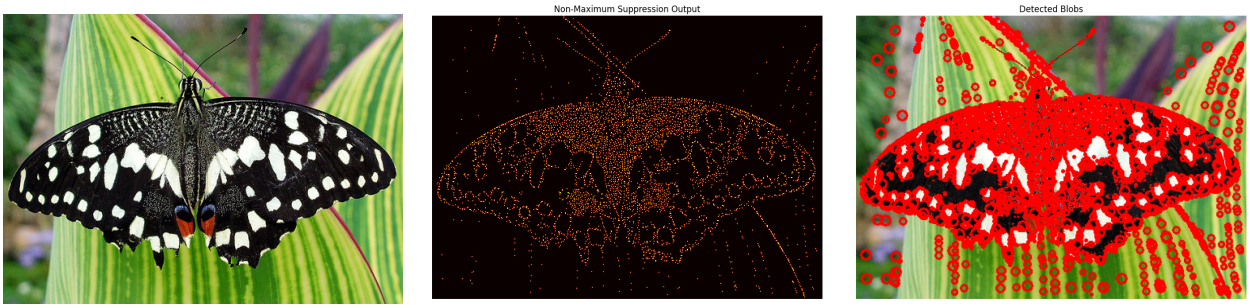


Figure 12: Output of the Laplacian Scale Space Construction. As the level increases, we can see larger features are enhanced.



(a) (b) (c)

Figure 13: (a) Plotting the original butterfly image. Output of (b) the non-max suppression and (b) blob detection. The output (c) after non-max suppression provides a binary map highlighting potential blob centers. The final output (b) visualizes the detected blobs by drawing circles around them.

Output Visualization of Blood Cells

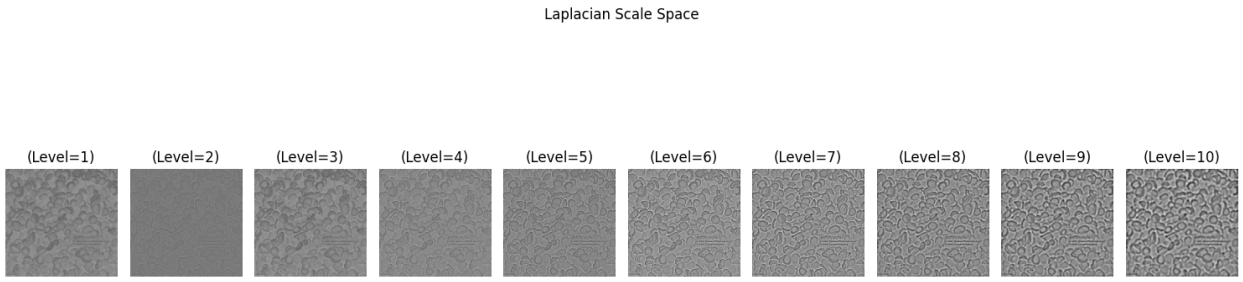
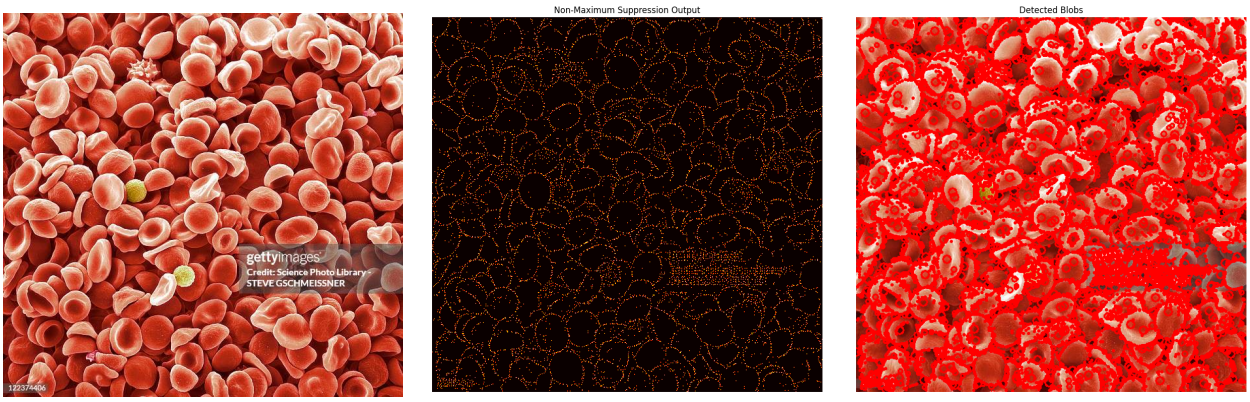


Figure 14: Output of the Laplacian Scale Space Construction. As the level increases, we can see larger features are enhanced.



(a) (b) (c)

Figure 15: (a) Plotting the original bloodcell image. Output of (b) the non-max suppression and (b) blob detection. The output (c) after non-max suppression provides a binary map highlighting potential blob centers. The final output (b) visualizes the detected blobs by drawing circles around them.

Output Visualization of Roses

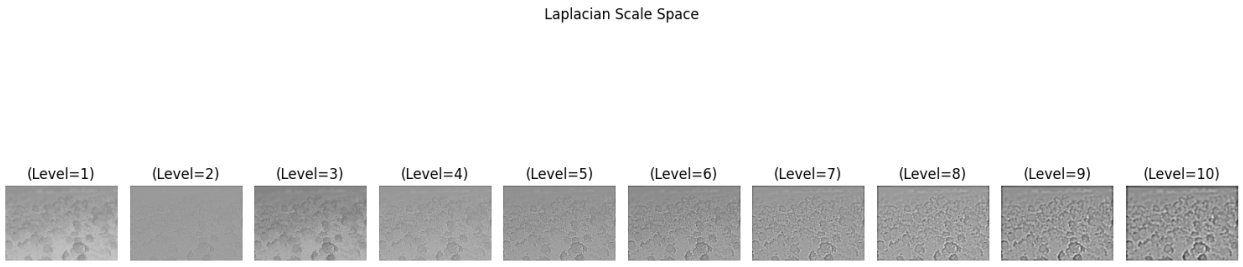
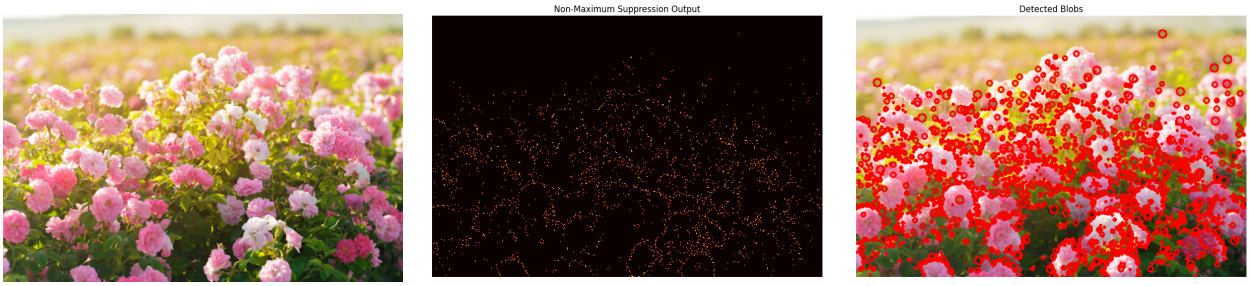


Figure 16: Output of the Laplacian Scale Space Construction. As the level increases, we can see larger features are enhanced.



(a) (b) (c)

Figure 17: (a) Plotting the original rose image. Output of (b) the non-max suppression and (b) blob detection. The output (c) after non-max suppression provides a binary map highlighting potential blob centers. The final output (b) visualizes the detected blobs by drawing circles around them.

Visualization of Geometric Shapes

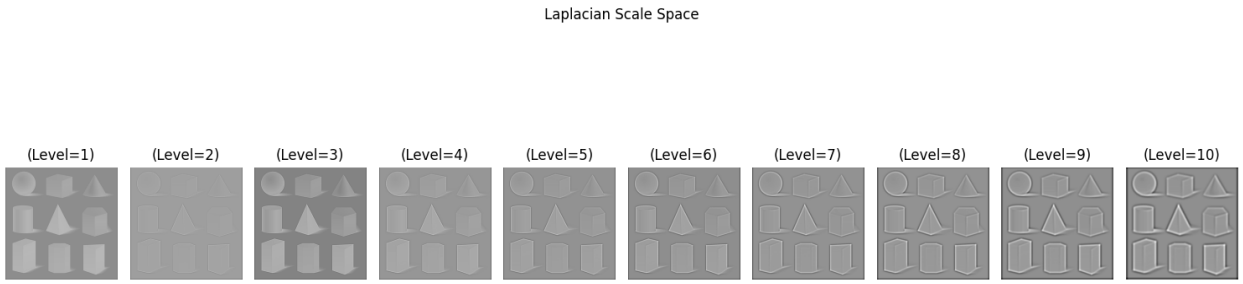
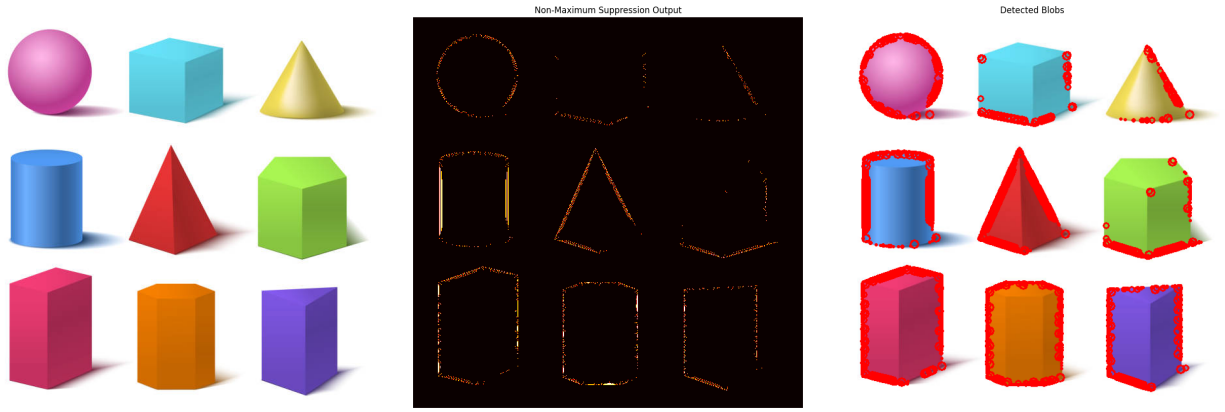


Figure 18: Output of the Laplacian Scale Space Construction. As the level increases, we can see larger features are enhanced.



(a) (b) (c)

Figure 19: (a) Plotting the original geometric shape image. Output of (b) the non-max suppression and (b) blob detection. The output (c) after non-max suppression provides a binary map highlighting potential blob centers. The final output (b) visualizes the detected blobs by drawing circles around them.

Output Visualization of Dogs

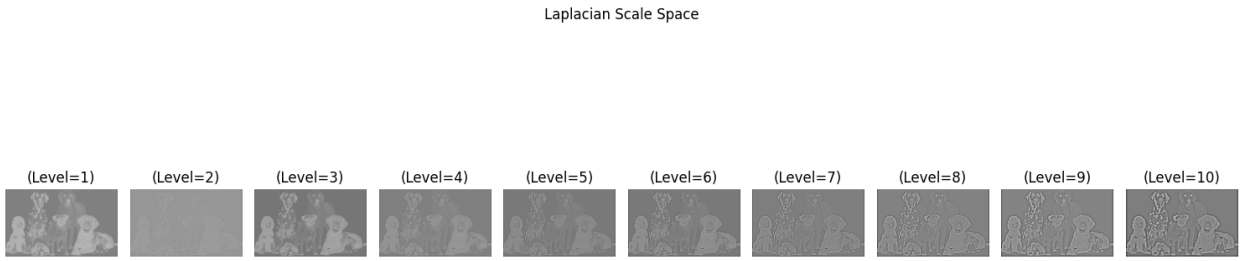
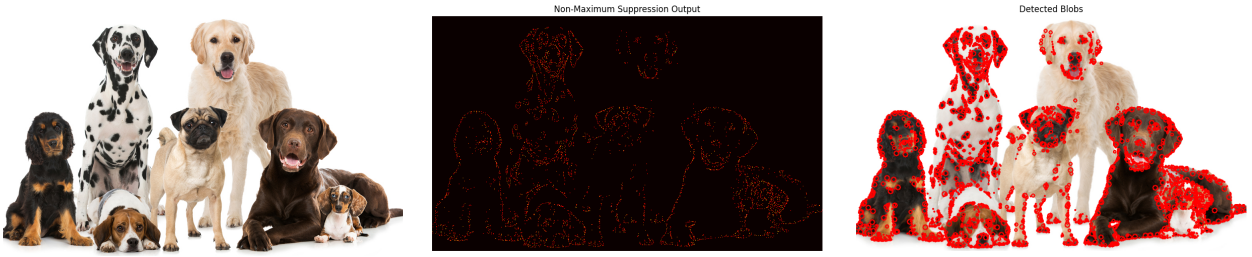


Figure 20: Output of the Laplacian Scale Space Construction. As the level increases, we can see larger features are enhanced.



(a) (b) (c)

Figure 21: (a) Plotting the original dog image. Output of (b) the non-max suppression and (b) blob detection. The output (c) after non-max suppression provides a binary map highlighting potential blob centers. The final output (b) visualizes the detected blobs by drawing circles around them.

Measuring Runtime. To ensure fair comparison of runtime, we use the same parameter setting for all images. The parameter setting is: number of levels = 10, scale factor = 1.25, initial scale = 0.63, and threshold value = 0.05.

Table 5: Runtime for Different Images.

Image	**Runtime (second)**
Fish	9.80
Sunflower	7.61
Einstein	19.21
Butterfly	10.67
Blood Cells	22.11
Rose	14.46
Geometric Shape	22.75
Dog	33.54