CS472-Assignment2

Iason Tzimas - csdp1333

March 28, 2024

1 Edge Detection

For Edge detection, we need the Magnitude of the gradients, as well as the corresponding angles within the image.

The derivatives of the initial image at x, y directions are derived by convolving the input signal with the respective x, y Sobel filters defined by:

$$Sob_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \text{ and } Sob_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \tag{1}$$

The derivative operations magnifies noise and it is high likely that the response of the differentiation at edges does not lead to local maxima. To face this, a two step process of smoothing, followed by differentiation is applied. However, since both filtering and differentiation are convolutions and the associative property holds:

$$\frac{d}{dx}(f * g) = f * \frac{d}{dx} g \tag{2}$$

So, one simply convolves the Gaussian Filter with the Sobel filter to acquire the synthesized filter. The function that generates the Gaussian filter only receives the σ value as input. The kernel size is the dynamically defined as:

$$k = \lceil 3\sigma \rceil \tag{3}$$

And the filter values at every kernel cell are given by:

$$g(x_i, y_i) = \frac{1}{2\pi\sigma^2} e^{-\frac{{x_i}^2 + {y_i}^2}{2\sigma^2}} \tag{4}$$

This leads to the following filters for $\sigma = 3$ Fig-1:

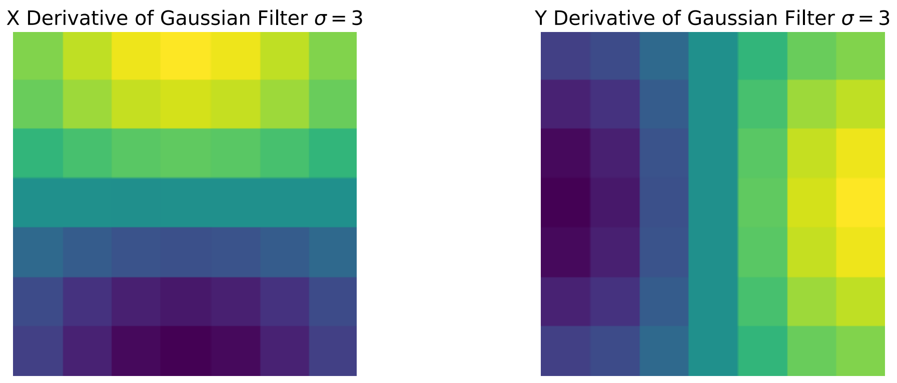

Figure 1: Synthesized filter for $\sigma = 3$

The convolution of the original image with those filters leads to the gradients at directions x and y. The resulting gradients are presented below for values of σ in $(1, 2, 3, 4)$, Fig-2.

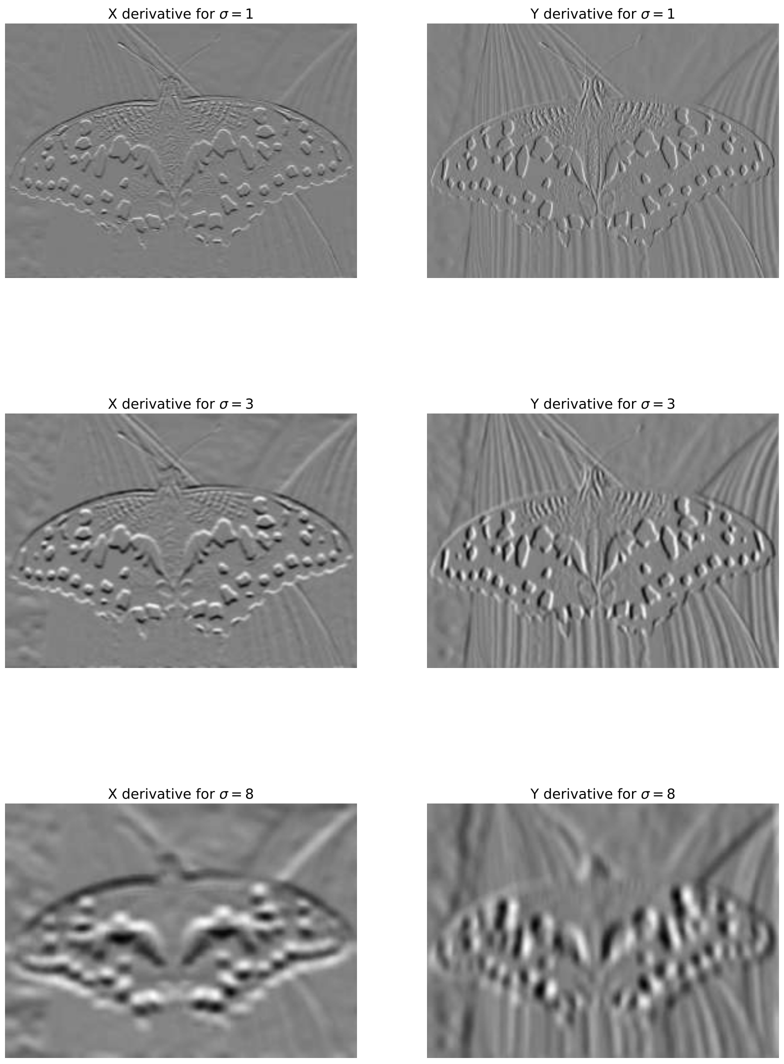
X derivative for $\sigma = 1$
Y derivative for $\sigma = 1$
X derivative for $\sigma = 3$
Y derivative for $\sigma = 3$
X derivative for $\sigma = 8$
Y derivative for $\sigma = 8$

Figure 2: X, Y derivatives for different σ values

The respective Magnitudes are derived by the formula:

$$Magn = \sqrt{dX^2 + dY^2} \tag{5}$$

and are the following Fig-3

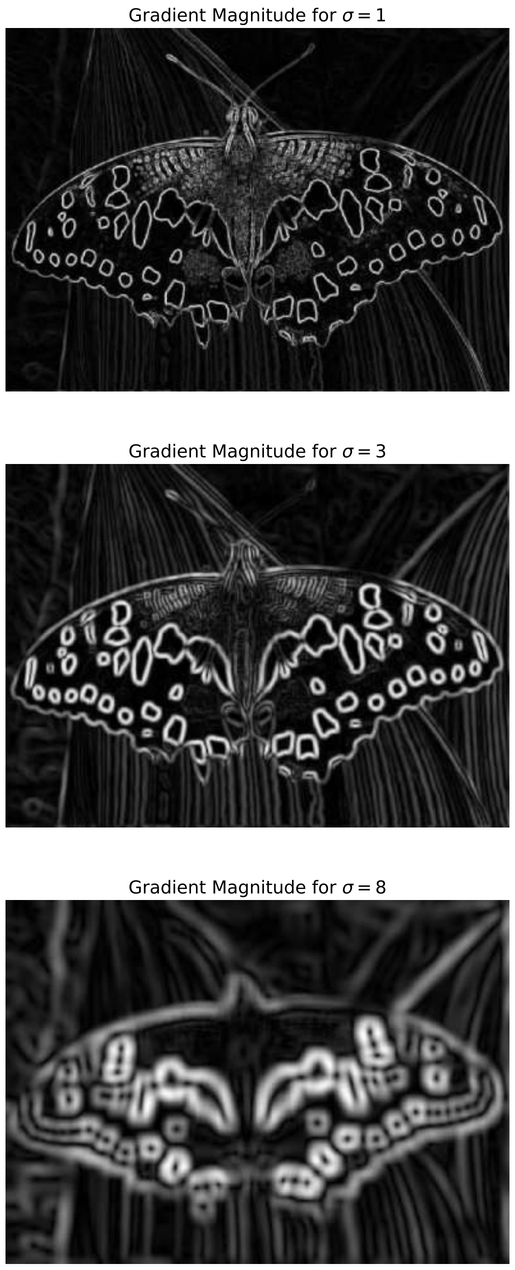
Gradient Magnitude for $\sigma = 1$
Gradient Magnitude for $\sigma = 3$
Gradient Magnitude for $\sigma = 8$

Figure 3: Gradient Magnitude for different σ values

Finally, the gradient angles given by:

$$\theta = tan^{-1}\left(\frac{dY}{dX}\right) \tag{6}$$

are the following Fig-4:

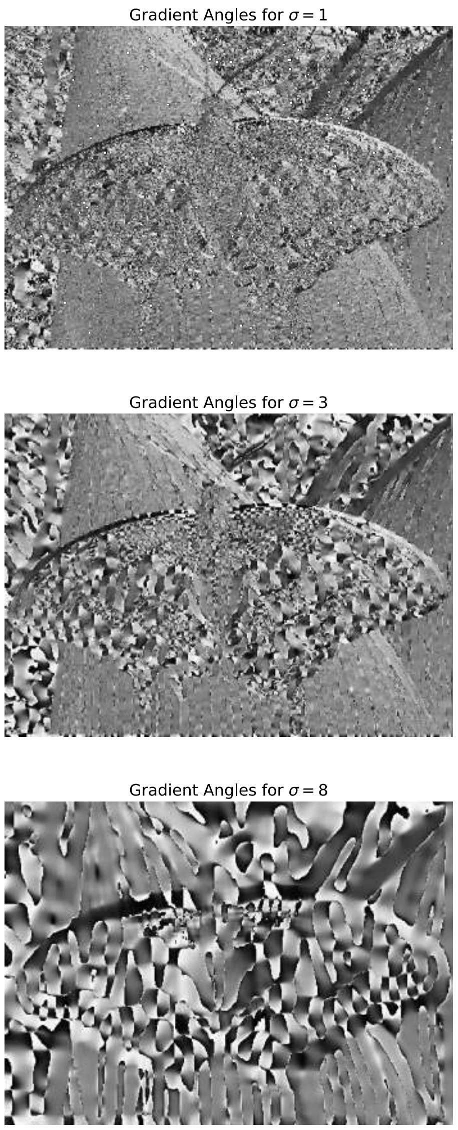
Gradient Angles for $\sigma = 1$
Gradient Angles for $\sigma = 3$
Gradient Angles for $\sigma = 8$

Figure 4: Gradient Agnles for different σ values

The rough look of the Angle Figures are due to the discontinuity of the tan^{-1} function.

1.1 Non-Maximum-Suppression Strategy A

Now, non-maximum-suppression must take place. The first strategy involves checking each Gradient Magnitude pixel against all pixels (8) in a 3×3 neighborhood regardless of the Gradient angle θ. If the pixel is not the largest it is suppressed into the value of 0. This leads to the following result Fig-5:

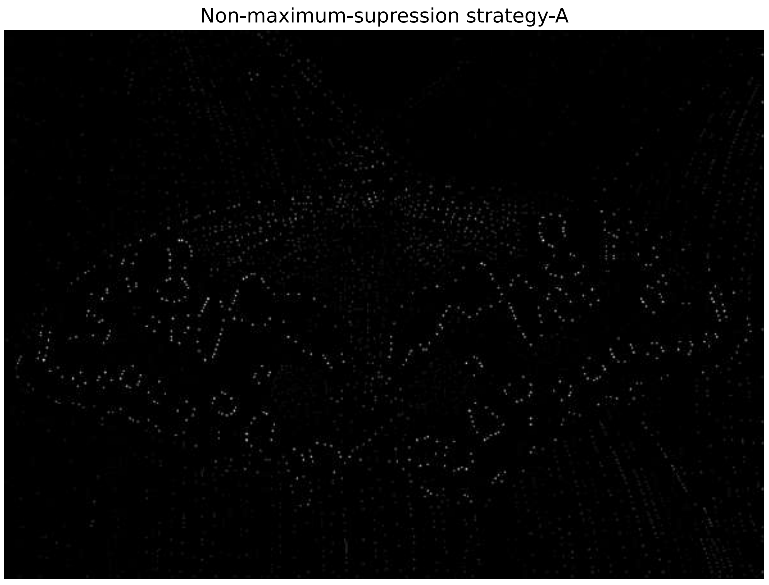

Figure 5: NMS Strategy A

The output is totally suboptimal and cannot serve as part of the edge detection process. Ignoring the orientation of the gradient leads to the rejection of candidate pixels by directly comparing them against neighboring pixels that also belong to the ”saddle” or ridge of the edge and therefore should not get rejected. This leads to the obvious introduction ob strategy B, which incorporates the gradient Angle into the NMS process.

1.2 Non-Maximum-Suppression Strategy B

In this case, the gradient Angle is discretized into bins representing the $0, 45, 90, 135$ degree angles. Then, the candidate pixel is checked against 2 other pixels depending on the bin of the angle it belongs to. Essentially, this serves as a way to not compare an edge pixel with neighboring pixels that also belong to the edge. This leads to the following result Fig-6

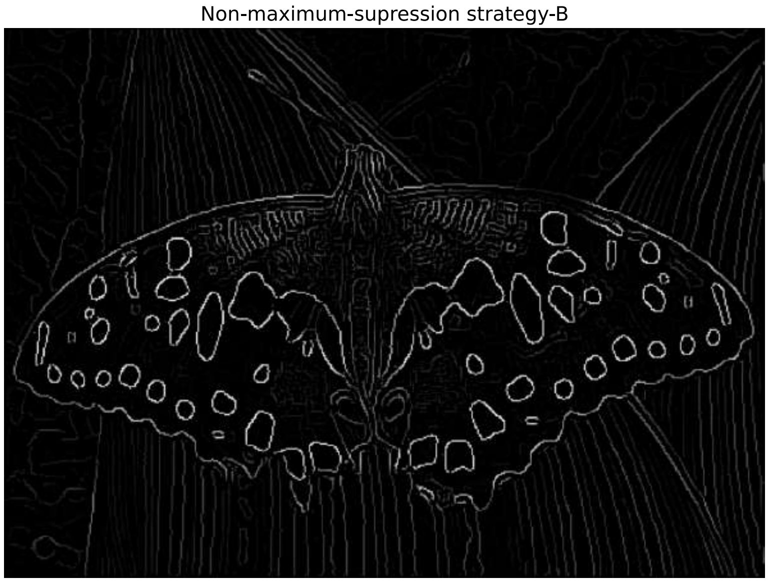

Figure 6: NMS Strategy B

1.3 Non-Maximum-Suppression Strategy C

One can go a step further and not discretize the angle. In that case the 2 neighboring pixel values are chosen to be the interpolated pixel values of the two closest pixels, weighted by the coordinate of the intersection of the gradient line with the pixel boundaries. This leads to the following result Fig-7

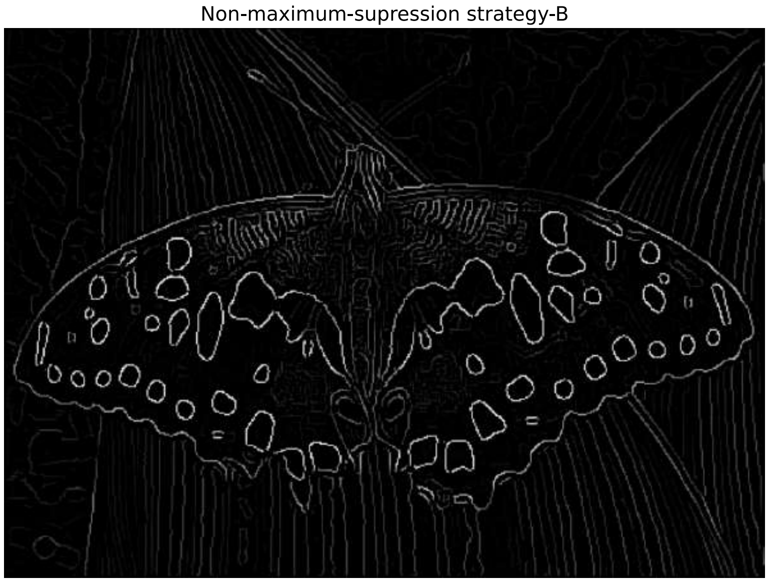

Figure 7: NMS Strategy C

,which is not significantly different than the NMS of strategy B.

1.4 Canny implementation

As a final touch, the Canny algorithm is implemented. This includes the process of hysterisis thresholding by two thresholds t_1 and t_2. Pixels with a gradient magnitude in (t_1, t_2) are edge candidates, pixels with pixel values $p < t_1$ are rejected and pixels $p > t_2$ are considered edges. Then, a revised version of the connected components labeling algorithm implemented in Assignment1 is used to derive the final Canny edge detection outcome. An 8-connectivity is used and a valid region is defined by a set of pixels that are either 1 or 2 and are connected with one-another and there is at least one pixel of value 2 in that region. Below, you can find the Canny Edge detection outputs before and after the application of the connectivity labeling for the provided Images and various values of t_1, t_2 Fig-(8-**??**).

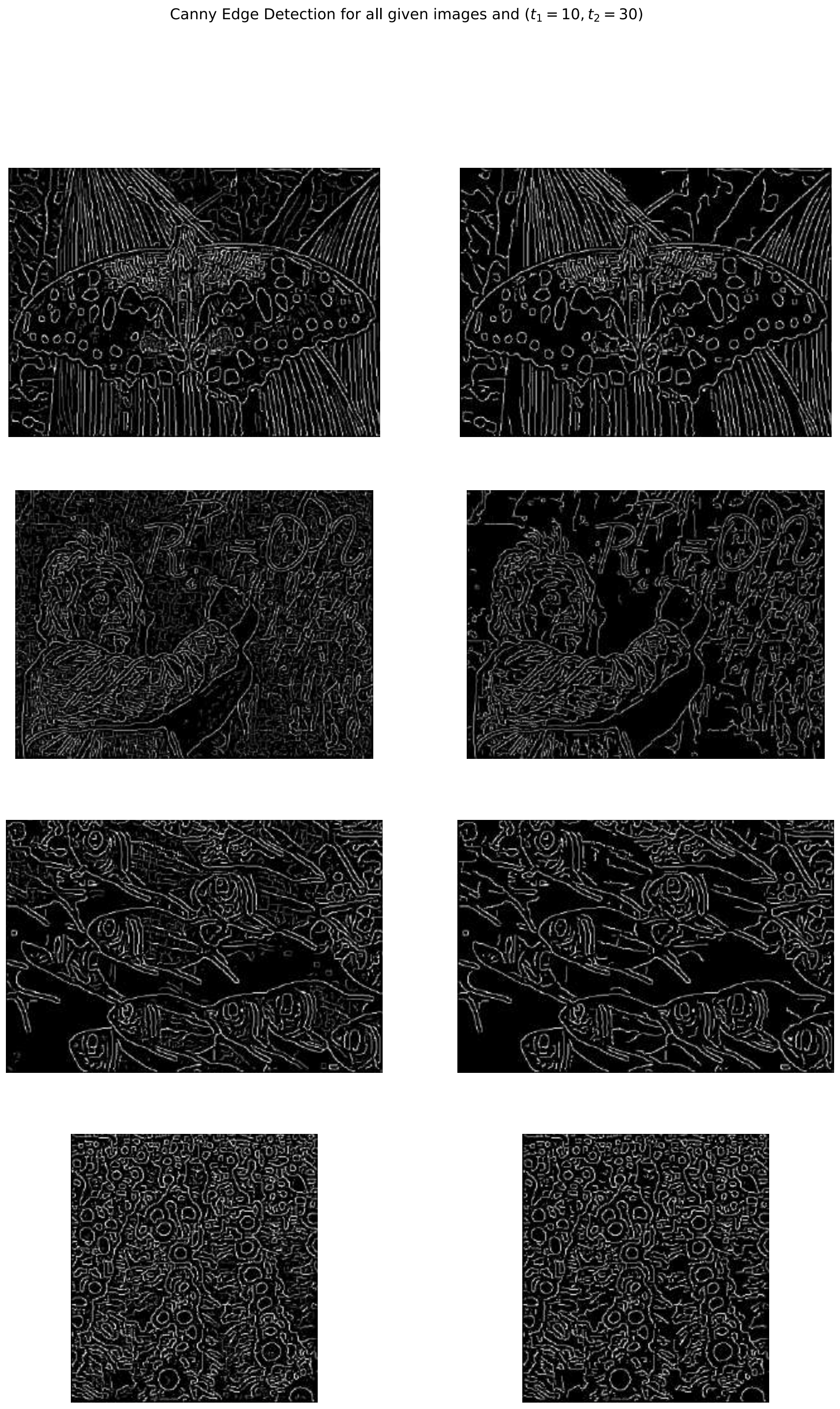
Canny Edge Detection for all given images and ($t_1 = 10, t_2 = 30$)

Figure 8: Canny Edge Detection outputs for $t_1 = 10, t_2 = 30$

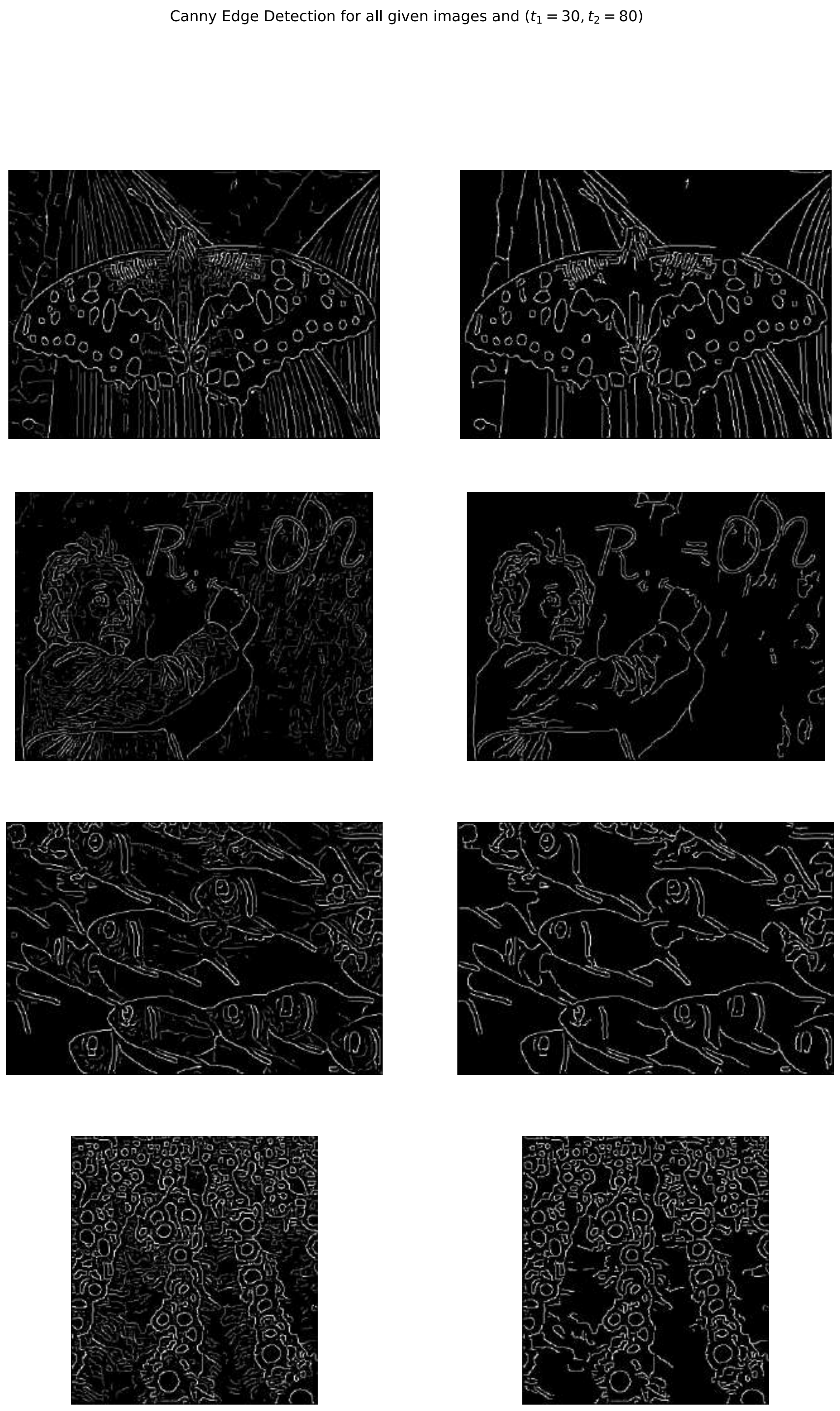
Canny Edge Detection for all given images and ($t_1 = 30, t_2 = 80$)

Figure 9: Canny Edge Detection outputs for $t_1 = 30, t_2 = 80$

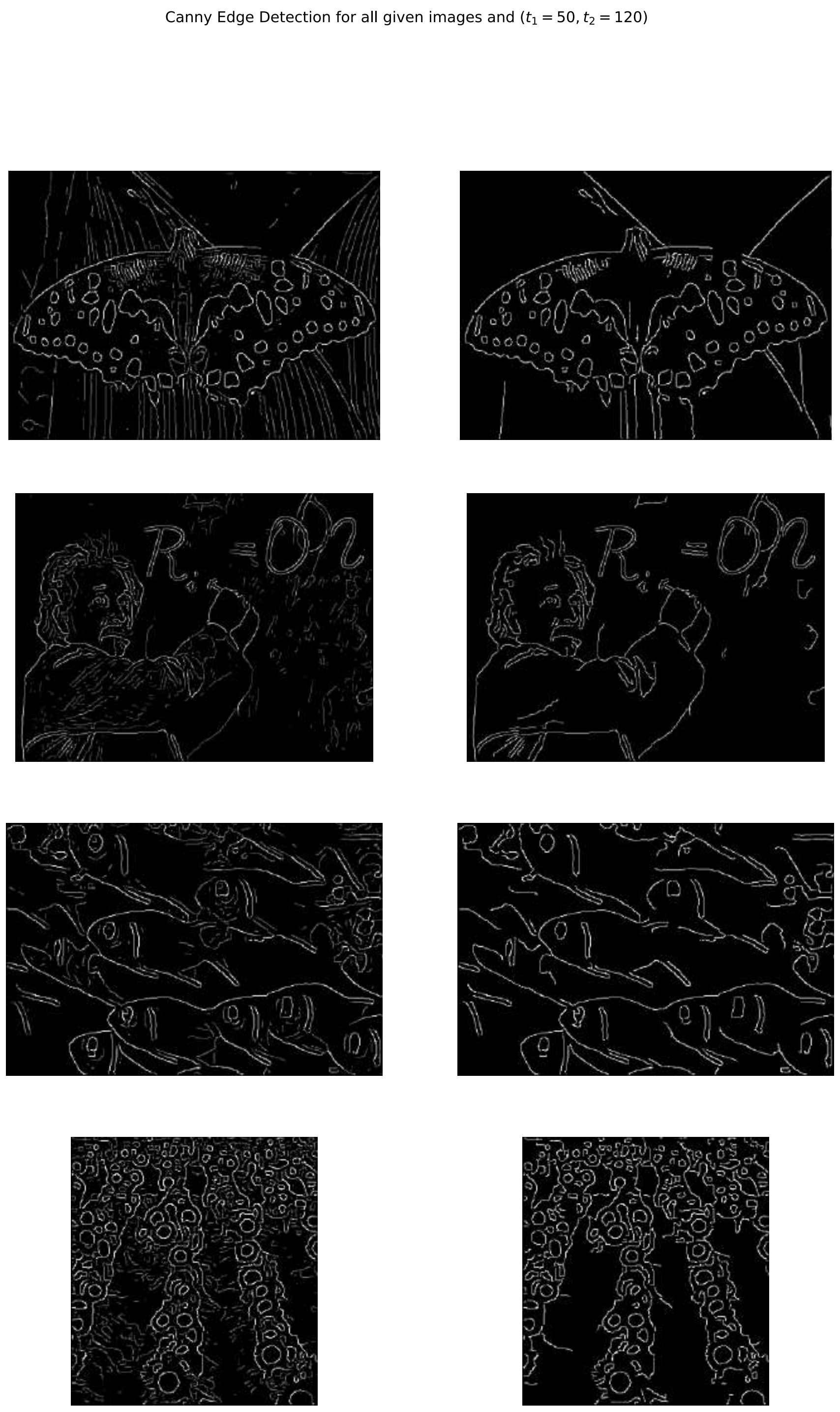
Canny Edge Detection for all given images and ($t_1 = 50, t_2 = 120$)

Figure 10: Canny Edge Detection outputs for $t_1 = 50, t_2 = 120$

2 Blob Detection

For Blob Detection a similar filtering process is applied. In this case however, the need for convolving the Image with differently sized filters arises. The Laplacian of the Gaussian is chosen as a kernel for the efficiency with which it can capture blob-like features.

First of all, the discrete LoG kernel is defined. The kernel size is once-again dynamically defined by the σ value of the underlying Gaussian it refers to. The expression giving the pixel values across the kernel cells is:

$$\nabla^2 G(x_i, y_j) = \left[\frac{x_i^2 + y_j^2 - 2\sigma^2}{\sigma^4}\right] e^{-\frac{x_i^2+y_j^2}{2\sigma^2}} \tag{7}$$

The kernel size relative to σ is given by:

$$k = \lceil 3\sigma \rceil \tag{8}$$

,so as to adequately host all useful features of the Laplacian.

Some Laplacians for varying σ values are presented below Fig-11:

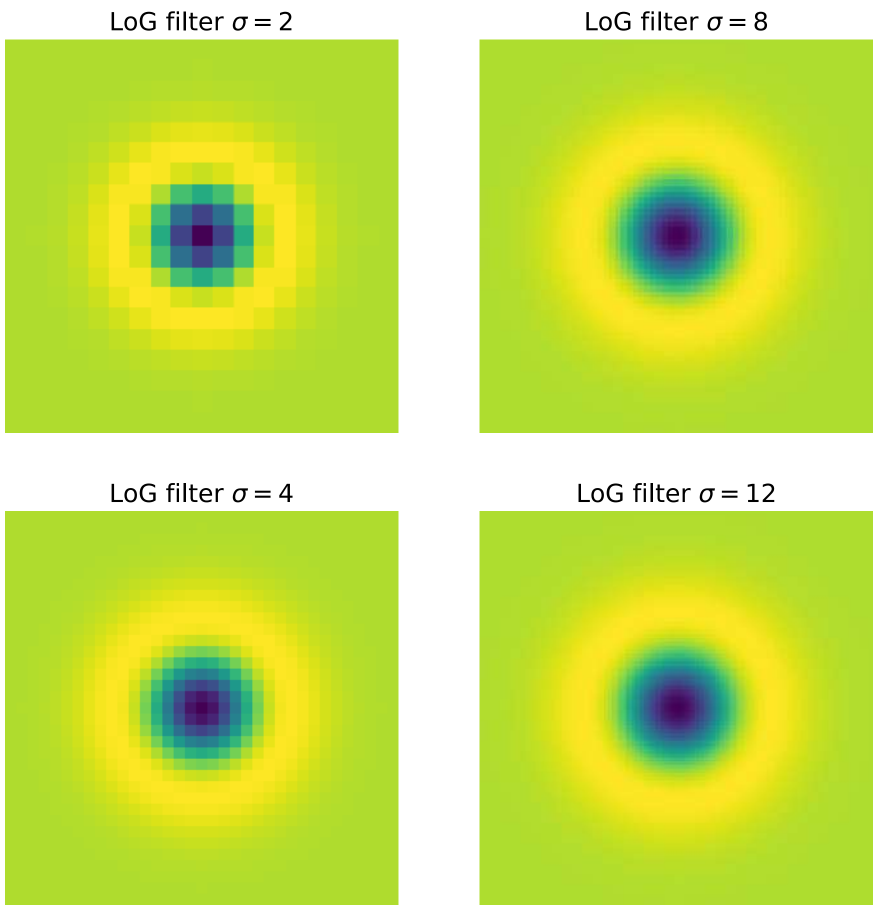

Figure 11: LoG filters for various σ values.

2.1 Strategy A

The first strategy with which to implement Blob Detection is to create a series or LaPlacian filters with different sizes and σ's and convolve the same initial Image with those. Assuming n number of filters, the resulting scale-space Image is a 3D Tensor of dimensions $n \times w \times h$, where h is the Image height and w is the image width. Then Non-Maximum Supression is carried out by sliding a $3 \times 3 \times 3$ cube along the tensor and checking whether or not the respective pixel is the local maximum in this neighborhood of pixels. If it is not the pixel value is set to 0 and is rejected. Then, another thresholding step is implemented which rejects all identified maxima that fall under it. The intuition behind that is that there might indeed local maxima but their magnitude is negligible and thus cannot possibly refer to an actual meaningful blob. Finally, the Harris Corner Detection Transform is applied to the original Image and all identified blobs with coordinates (x, y) are rejected if the corresponding Harris value $H(x, y)$ is less than a large negative threshold. This is due to the fact that the LaPlacian get highly activated in edge regions and the Harris Transform gives large negative values for the same regions. Thus, one can filter out "false alarms" that way. Replicate Padding is performed which replicates multiple copies of the first and last rows/columns along the respective directions. This proves meaningful as it does not form edges between the Image and Padding areas, which would then translate into artifacts when filtered with the LoG filters.

For $\sigma_0 = 1.4$, $k = 1.4$, $n = 12$ values the resulting LoG filtered Images of the given butterfly Image are as follows (Note: The filtered Images are squared) Fig-(12-15):

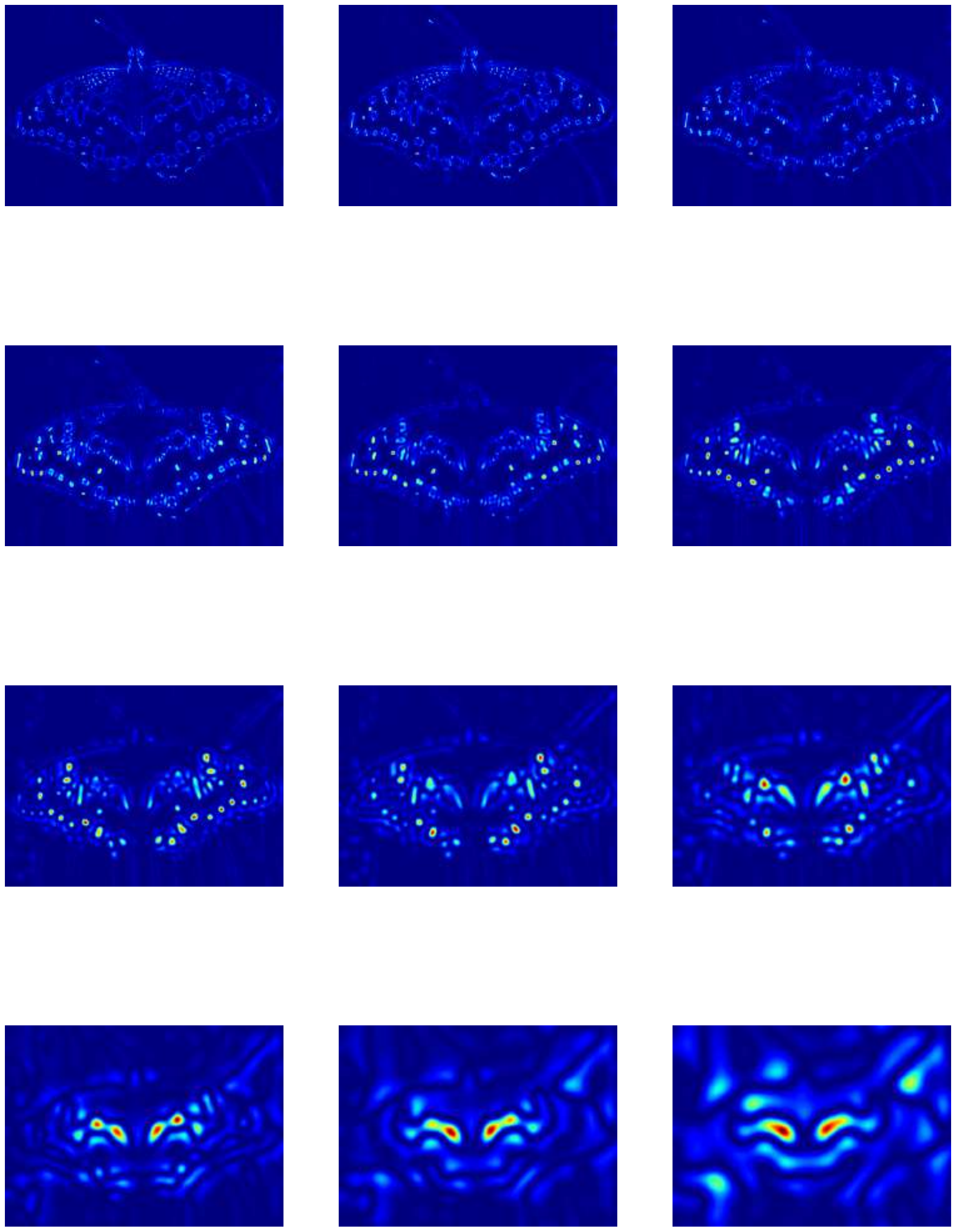

Figure 12: LoG Filtered Butterfly Image

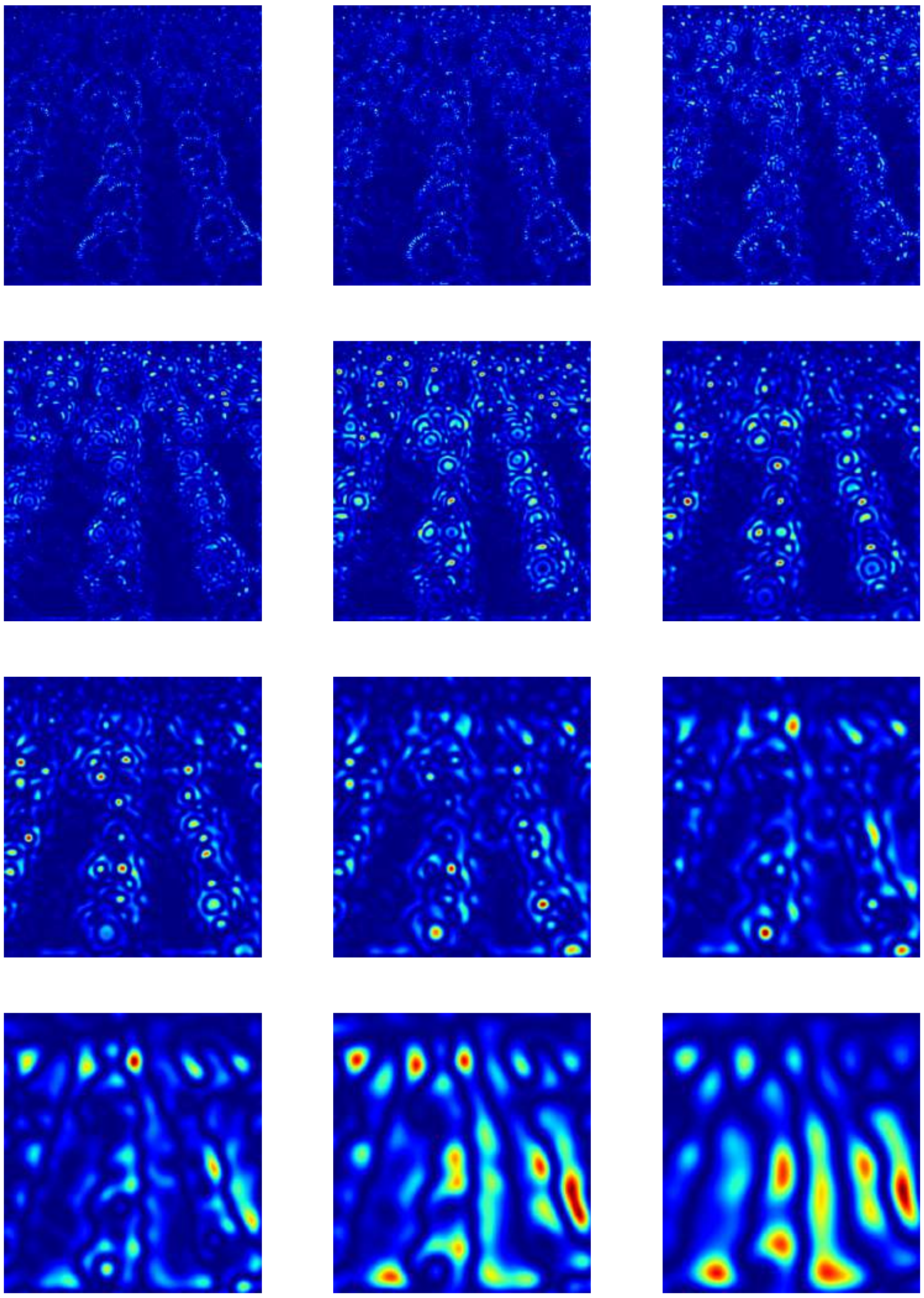

Figure 13: LoG Filtered Sunflower Image

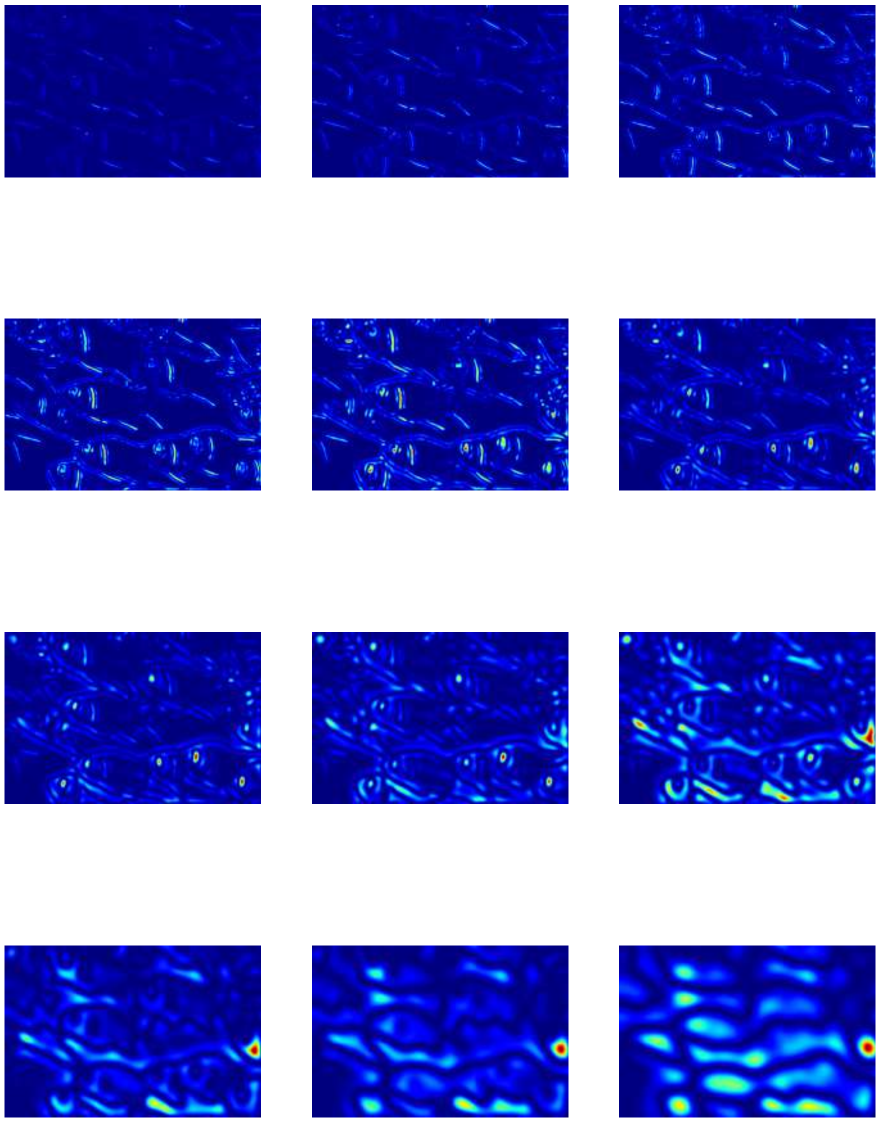

Figure 14: LoG Filtered Fishes Image

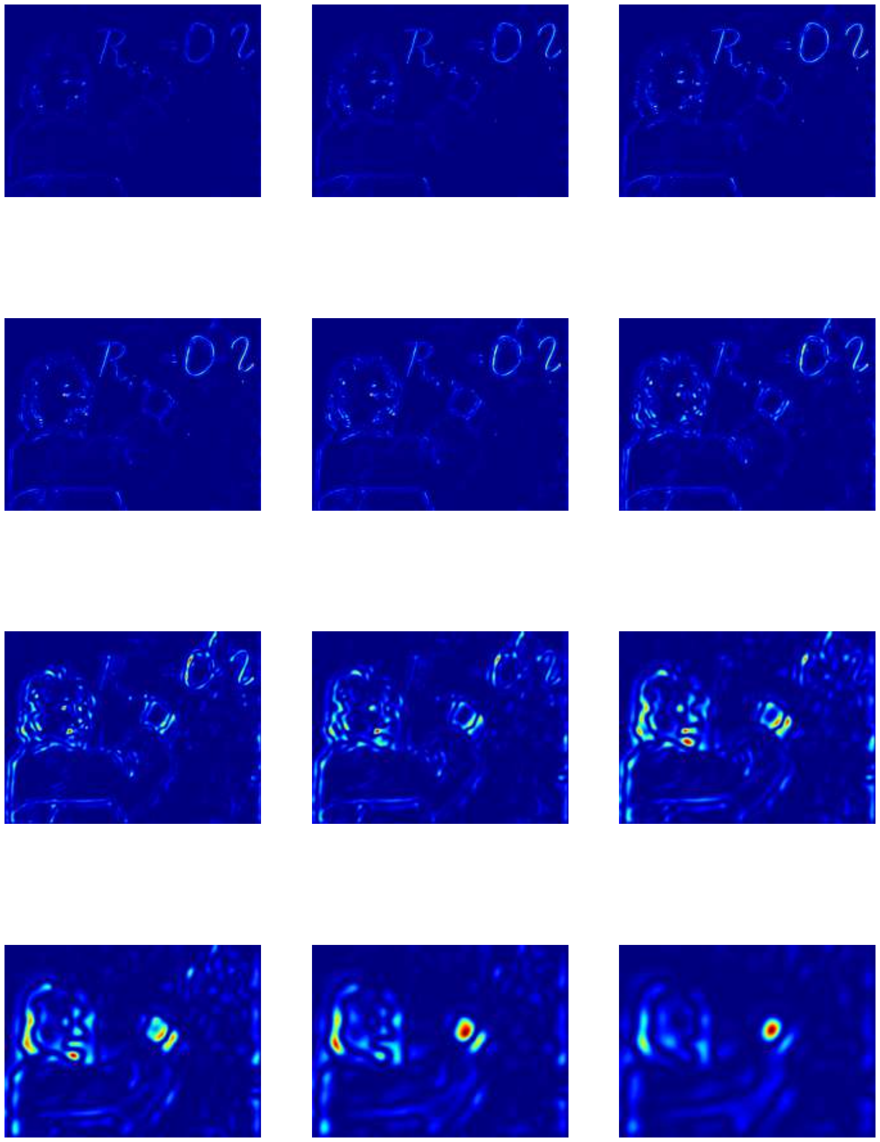

Figure 15: LoG Filtered Fishes Image

The resulting Einstein picture Blob points in the context of their respective scale in which they were detected are the following Fig-16:

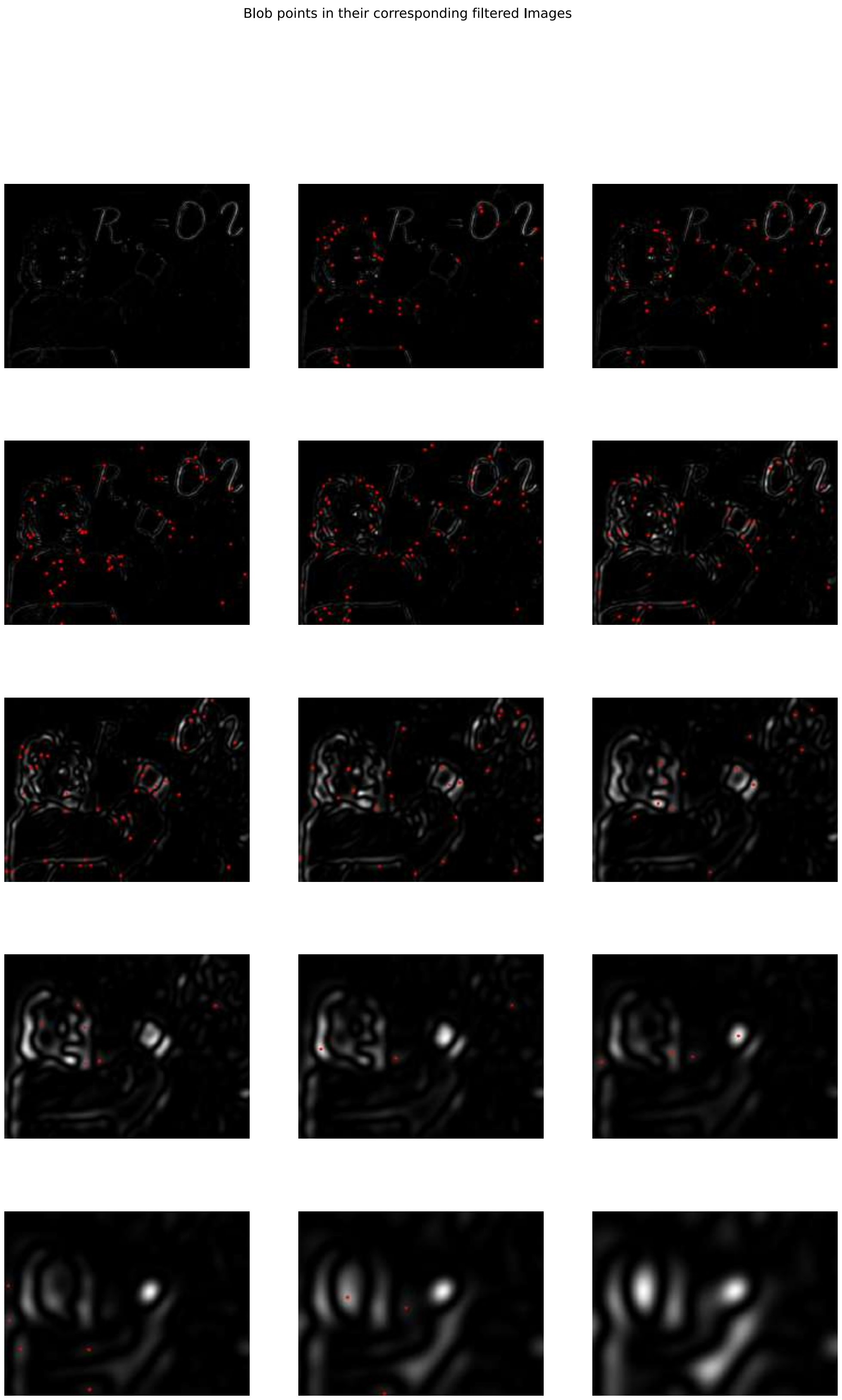

Figure 16: Detected blobs at their respective scales

The points that were detected as blobs but were then rejected as Harris Edges are plotted against the original Gray scale image and are the following Fig-17:

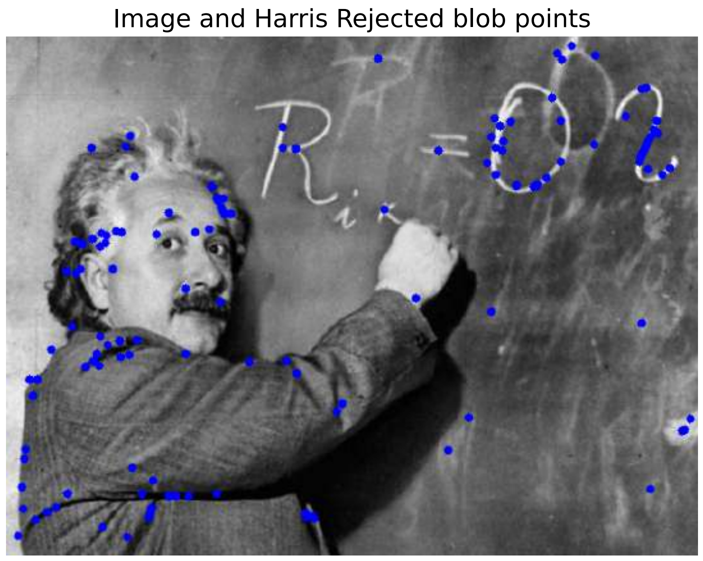

Figure 17: Blobs Rejected as Harris Edges

Note, that these points indeed lie in edge regions of the Image.

The Harris Output of the original Image is the following Fig-18:

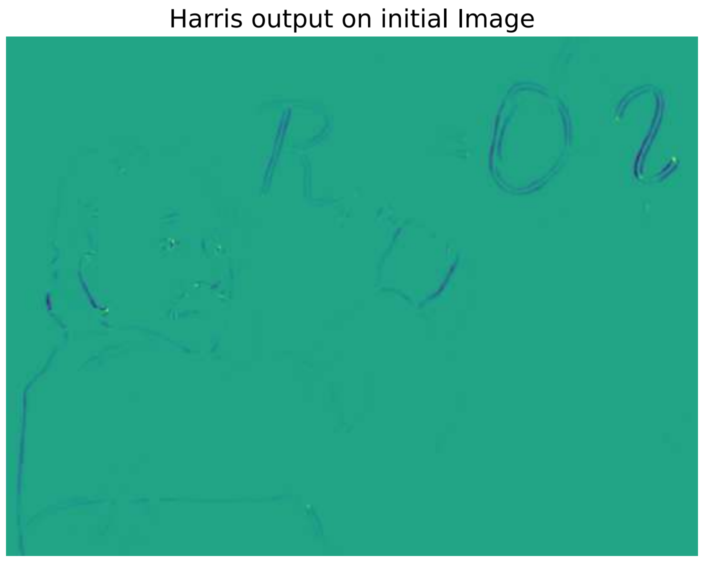

Figure 18: Harris Output

Finally, the original Image together with the circles representing the identified Blobs is as follows Fig-19:

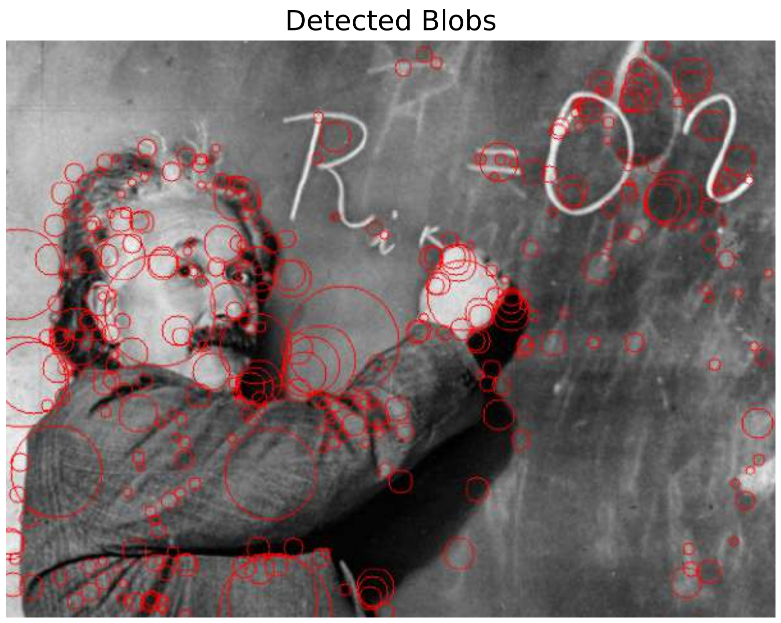

Figure 19: Einstein identified blobs

The above Strategy A blob detection method was run for parameters $n = 15$, $k = 1.25$, $\sigma_0 = 1.8$

2.2 Strategy B

The second strategy is to downscale the Image, instead of upscaling the filter and convolving with the same unchanged filter. For a given filter at each scale we do the same total number of convolutions with Strategy A, only at this time the number of operations within the convolution itself is constant and equal to $O(k^2)$, where k is the kernel width/height. In the case of Strategy A then intra-convolution operations become larger with the upscaling of the filter at every scale.

However, Strategy B incorporates some type of Interpolation to create an Image with the initial dimensions. This overhead is not sufficient to render the Strategy B slower.

The resulting detected blobs are the following:

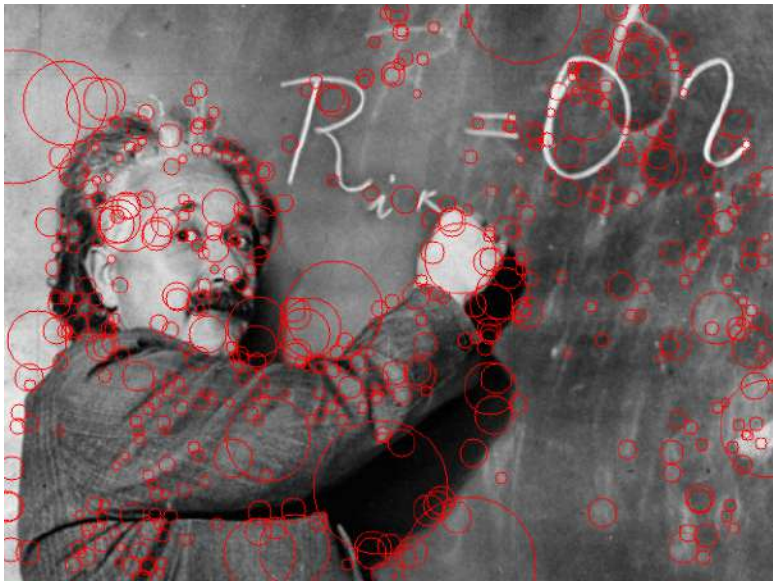

Figure 20: Einstein identified blobs, strategy B

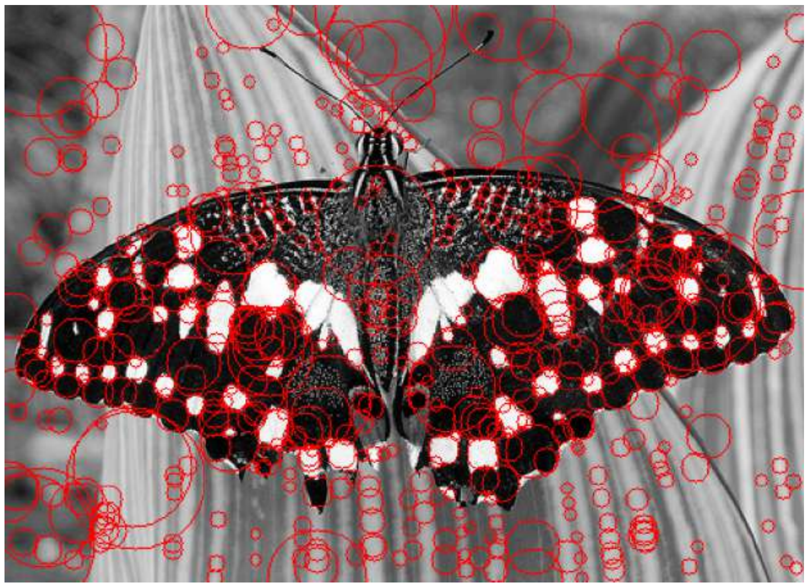

Figure 21: Butterfly identified blobs, strategy B

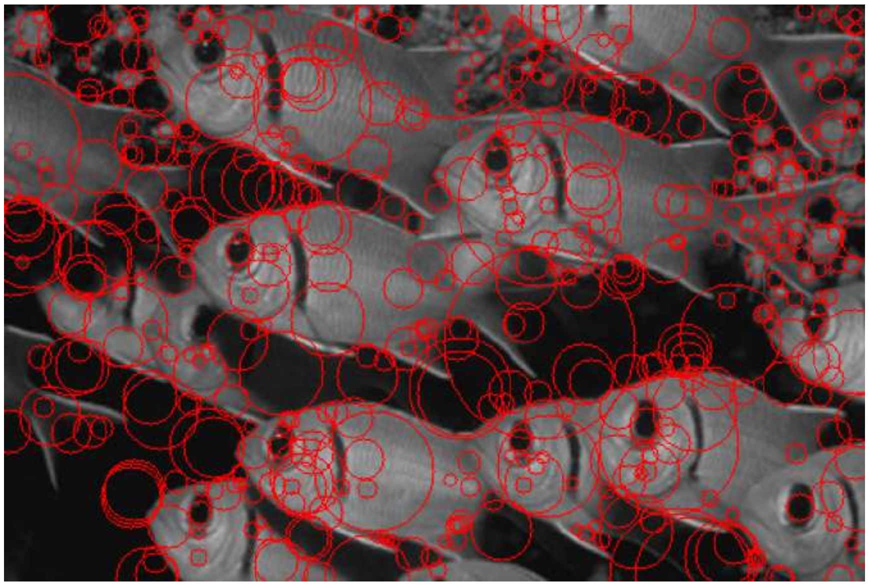

Figure 22: Fishes identified blobs, strategy B

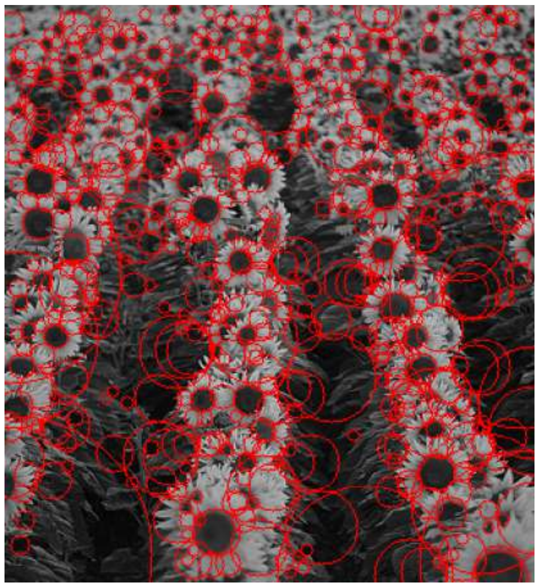

Figure 23: Sunflowers identified blobs, strategy B

Three great paintings and one Installation are also included for blob detection and edge detection.

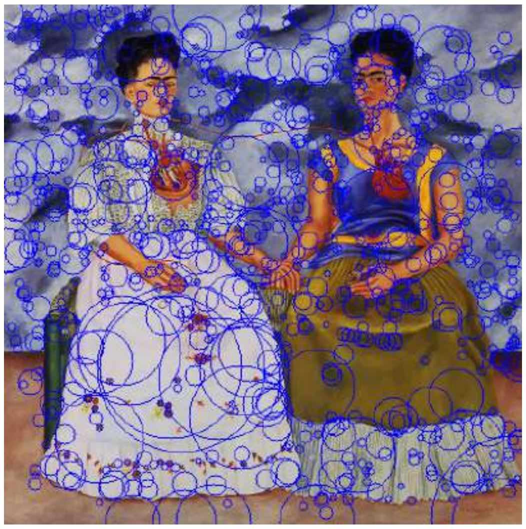

Figure 24: Frida Painting identified blobs, strategy B

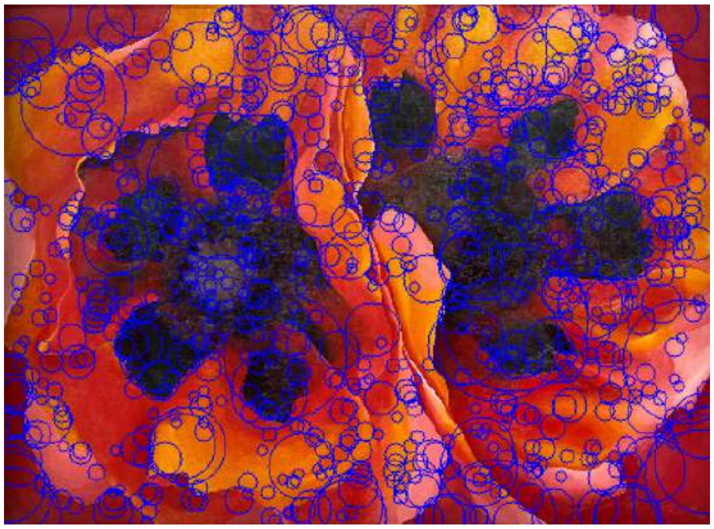

Figure 25: Georgia O' Keefe Painting identified blobs, strategy B

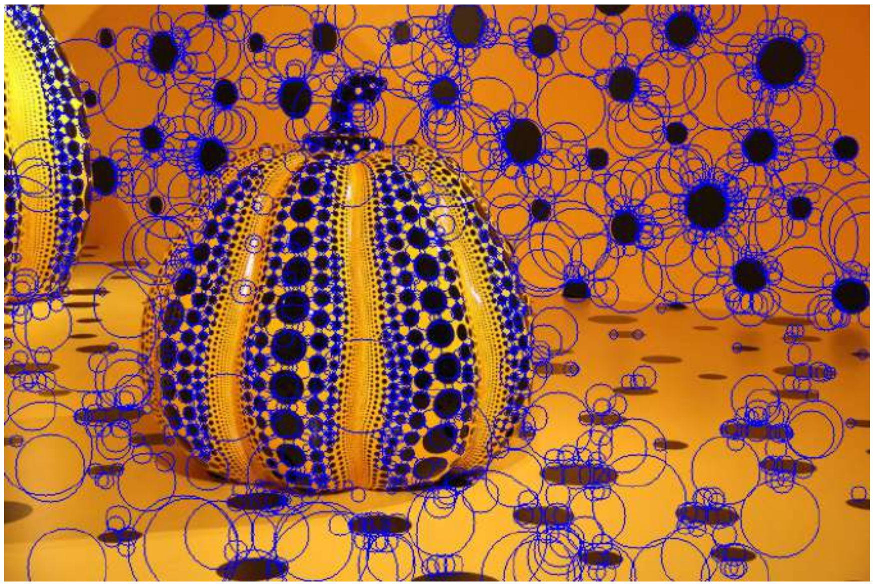

Figure 26: Kusama Installation identified blobs, strategy B

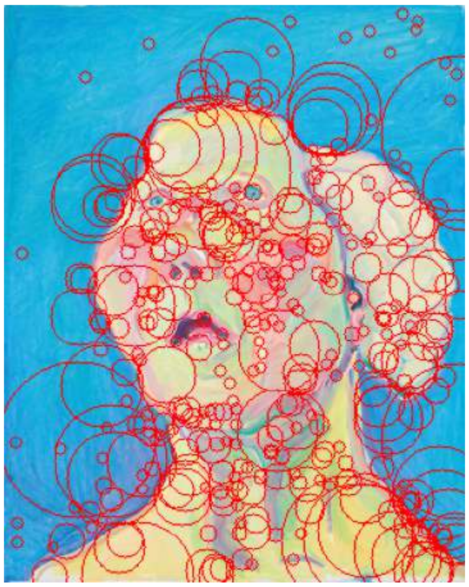

Figure 27: Maria Lasnig Painting identified blobs, strategy B

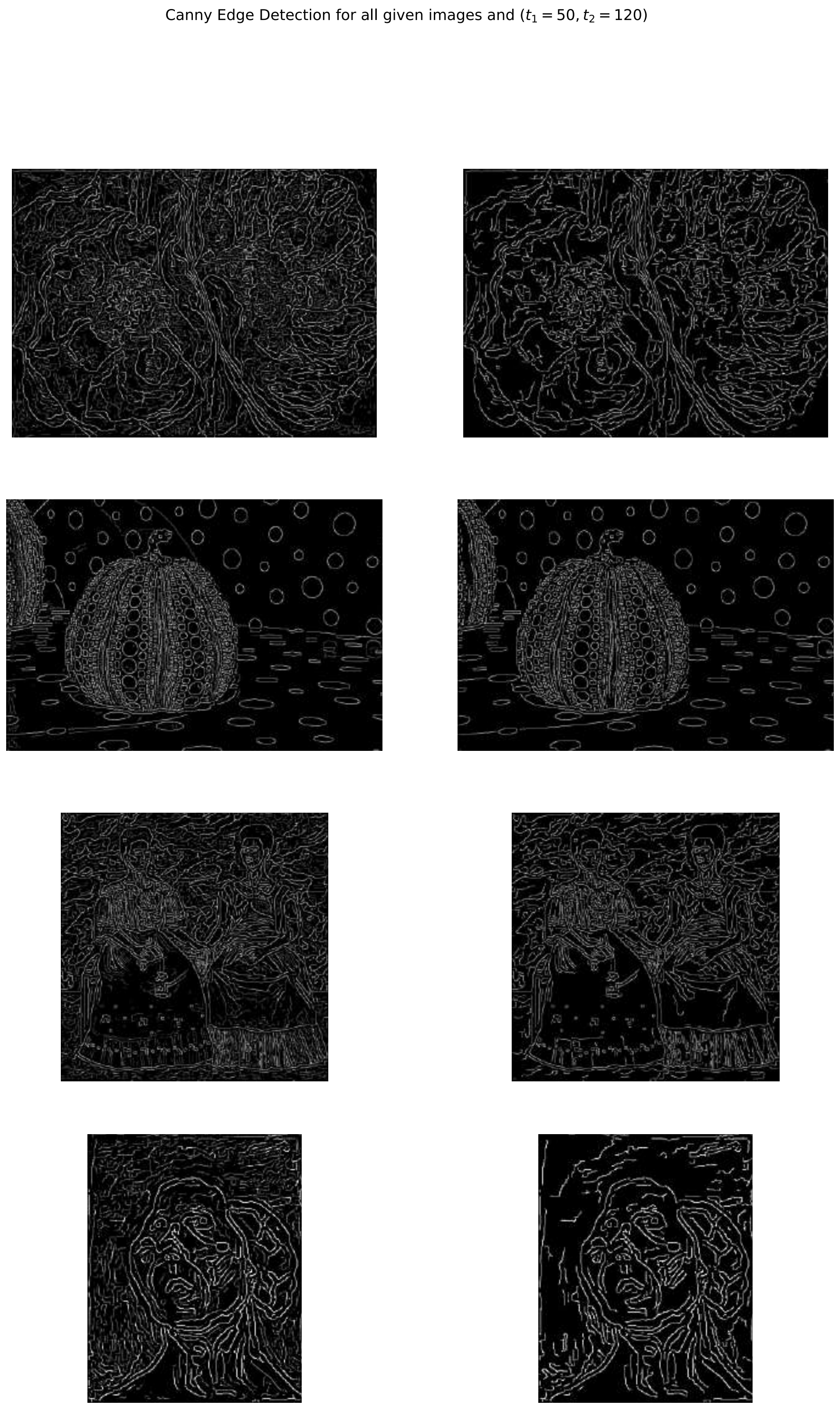
Canny Edge Detection for all given images and ($t_1 = 50, t_2 = 120$)

Figure 28: Canny Edge Detection for all paintings and installation $t_1 = 10, t_2 = 30$

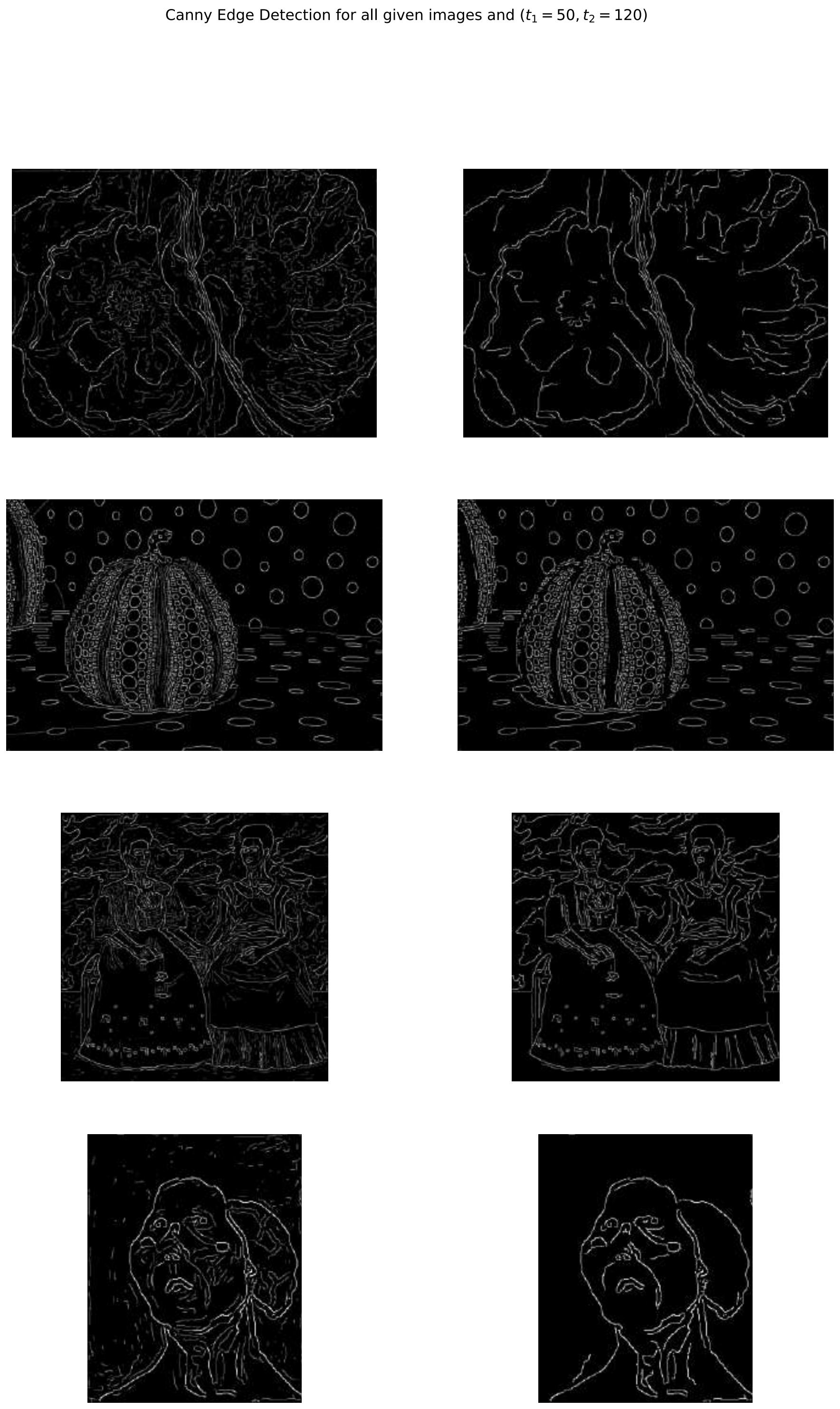
Canny Edge Detection for all given images and ($t_1 = 50, t_2 = 120$)

Figure 29: Canny Edge Detection for all paintings and installation $t_1 = 30, t_2 = 80$

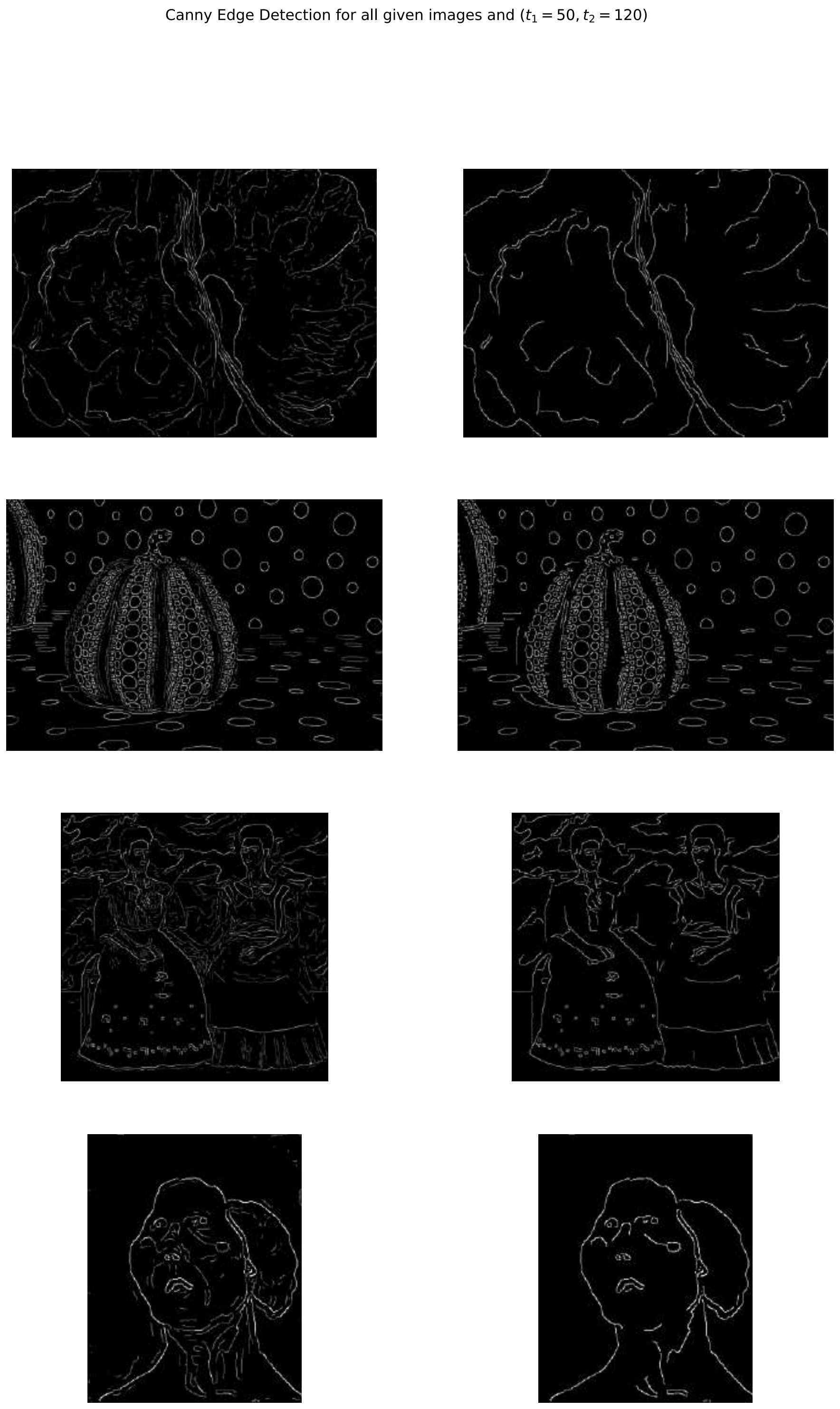
Canny Edge Detection for all given images and ($t_1 = 50, t_2 = 120$)

Figure 30: Canny Edge Detection for all paintings and installation $t_1 = 50, t_2 = 120$

Finally, the run-times for blob detection using both strategies on the Einstein Picture are calculated. For Strategy A we get 182 seconds and for Strategy B is 47 seconds.