ASSIGNMENT-1 BIOLOGICAL VISION

Image Edge Detection and Orientation Assignment

Name: R.Amshu Naik
Roll No: B20CS046

Part-1 Objective : Image Preprocessing

Steps of implementation:

a. *Loading of images:*

- Used gdown, Cv2, Matplotlib library to load and download the images from the link in “image_link_up” list.
- Here gdown library is used to download the images so as to to display the images and CV2 is used to load images with colors on it.

b. *Converting each image to grayscale:*

- To convert images to grayscale while preserving local details and enhancing contrast, using Adaptive Histogram Equalization (AHE), Contrast Limited Adaptive Histogram Equalization (CLAHE), and gamma correction
- Conversion of image to grayscale image is implemented using OpenCV library.

```
    gray_img = cv2.cvtColor(color_image, cv2.COLOR_BGR2GRAY)
```

- Here we are applying gamma correction to improve contrast of the grayscale image provided using the OpenCV library.

```
# code for CLAHE (Contrast Limited Adaptive Histogram
Equalization)
    clahe = cv2.createCLAHE(clipLimit=2.5, tileGridSize=(7,
7))
    clache_gray_img = clahe.apply(gray_img)

    #  code for gamma correction
    gamma = 1  # kept gamma value 1
    gamma_img = np.power(clache_gray_img / 255.0, gamma) *
255.0
    gamma_img = np.uint8(gamma_img)
```

Output - Displaying the images:

➢ ***image1***

```
Downloading...
From: https://drive.google.com/uc?id=10JqpfJd9mOXs2xst7UiKNpjHRP-jr46n
To: /content/10JqpfJd9mOXs2xst7UiKNpjHRP-jr46n.jpg
100%|████████| 83.2k/83.2k [00:00<00:00, 41.4MB/s]
```

➢ ***image 2***

```
Downloading...
From: https://drive.google.com/uc?id=1EN3FBoYe9kB6C8xWB7rln1faNT5bR29F
To: /content/1EN3FBoYe9kB6C8xWB7rln1faNT5bR29F.jpg
100%|████████| 39.4k/39.4k [00:00<00:00, 45.7MB/s]
```

➢ **image 3**

```
Downloading...
From: https://drive.google.com/uc?id=1rssiLKqIrLVJGvkpfQ09_F5IFWbud_dB
To: /content/1rssiLKqIrLVJGvkpfQ09_F5IFWbud_dB.jpg
100%|████████| 39.8k/39.8k [00:00<00:00, 50.8MB/s]
```

➢ image 4

```
Downloading...
From: https://drive.google.com/uc?id=1bMez--4ck-A4NgkmGufL1Uk0j6QG5oXC
To: /content/1bMez--4ck-A4NgkmGufL1Uk0j6QG5oXC.jpg
100%|██████████| 128k/128k [00:00<00:00, 23.9MB/s]
```

Loaded color Image

Grayscale Image

Gamma corrected Grayscale Image

➢ image 5

```
Downloading...
From: https://drive.google.com/uc?id=1BjagYvMHKCEOhD3D3zf2Lwp1iBJpZOn5
To: /content/1BjagYvMHKCEOhD3D3zf2Lwp1iBJpZOn5.jpg
100%|██████████| 71.4k/71.4k [00:00<00:00, 65.1MB/s]
```

Loaded color Image

Grayscale Image

Gamma corrected Grayscale Image

➢ image 6

```
Downloading...
From: https://drive.google.com/uc?id=1Q1VYb5s7laGKymNw3ccl5gm-Dd7Rh76D
To: /content/1Q1VYb5s7laGKymNw3ccl5gm-Dd7Rh76D.jpg
100%|██████████| 164k/164k [00:00<00:00, 50.3MB/s]
```

Loaded color Image

Grayscale Image

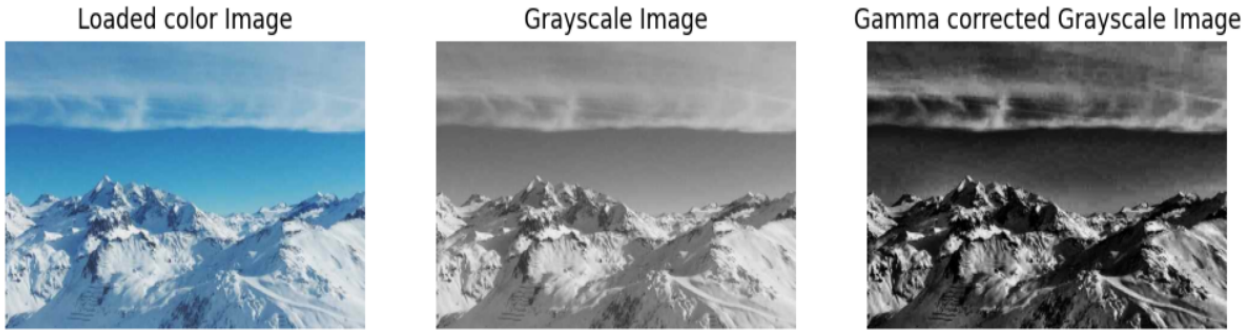
Gamma corrected Grayscale Image

➢ image 7

```
Downloading...
From: https://drive.google.com/uc?id=1vIk3SIB6GhQ7viPPgDkCiMXvHOUp-BzU
To: /content/1vIk3SIB6GhQ7viPPgDkCiMXvHOUp-BzU.jpg
100%|██████████| 84.7k/84.7k [00:00<00:00, 67.6MB/s]
```

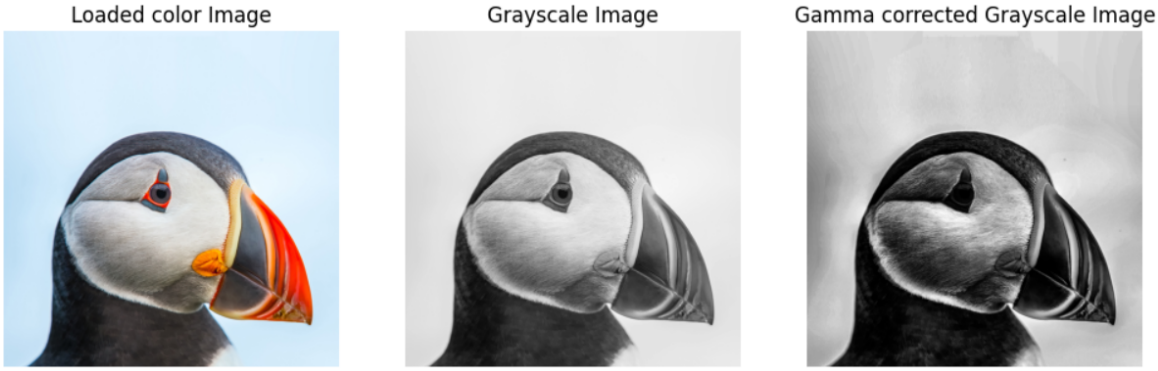
➢ image 8

```
Downloading...
From: https://drive.google.com/uc?id=1fQDy1-bXb1rVXq5-wI4VfQDslk2EVDRY
To: /content/1fQDy1-bXb1rVXq5-wI4VfQDslk2EVDRY.jpg
100%|██████████| 23.0k/23.0k [00:00<00:00, 34.8MB/s]
```



➢ image 9

```
From: https://drive.google.com/uc?id=1AKa9VntJ3lRvc96CLoPecxgzs_2dblNw
To: /content/1AKa9VntJ3lRvc96CLoPecxgzs_2dblNw.jpg
100%|██████████| 391k/391k [00:00<00:00, 66.1MB/s]
```



➢ **image 10**

```
Downloading...
From: https://drive.google.com/uc?id=1v-FXXKvxl8jBc1P01vRNa-xPbBh5XXuz
To: /content/1v-FXXKvxl8jBc1P01vRNa-xPbBh5XXuz.jpg
100%|██████████| 97.8k/97.8k [00:00<00:00, 49.3MB/s]
```

Challenges faced:

- ➔ To ensure that images have a minimum resolution of 400x400 pixels.
- ➔ Ensure that color images are displayed properly, for that need to use correct library is used
- ➔ Need to ensure that the images were not corrupted.
- ➔ Most importantly the images need to be shared properly, so that it can be accessible by the colab.
- ➔ While converting the image to grayscale ensure that the correct value of gamma is taken for better contrast.

Result:

- ➔ Keep gamma correction value low, for better image quality.
- ➔ The gamma corrected grayscale image is better that simple grayscale image in terms of contrast, brightness and highlight preservation of the image.

Part-2 Objective : Complex Orientation Filters

Steps of implementation:

- I selected first image link "https://drive.google.com/file/d/10JqpfJd9mOXs2xst7UiKNpjHRP-jr46n/view?usp=drive_link" and get the gamma_corrected grayscale image.
- Then apply a Gabor filter on the corrected grayscale image to analyze the texture patterns of the image at different scales and orientations and to detect edges which will be used in further parts.

```
# created a Gabor filter kernel using cv2
        gabor_kernel = cv2.getGaborKernel(
            (11, 11),                    # Kernel size
            4.0,                         # Standard deviation
            np.deg2rad(orientation),     # Orientation in radians
```

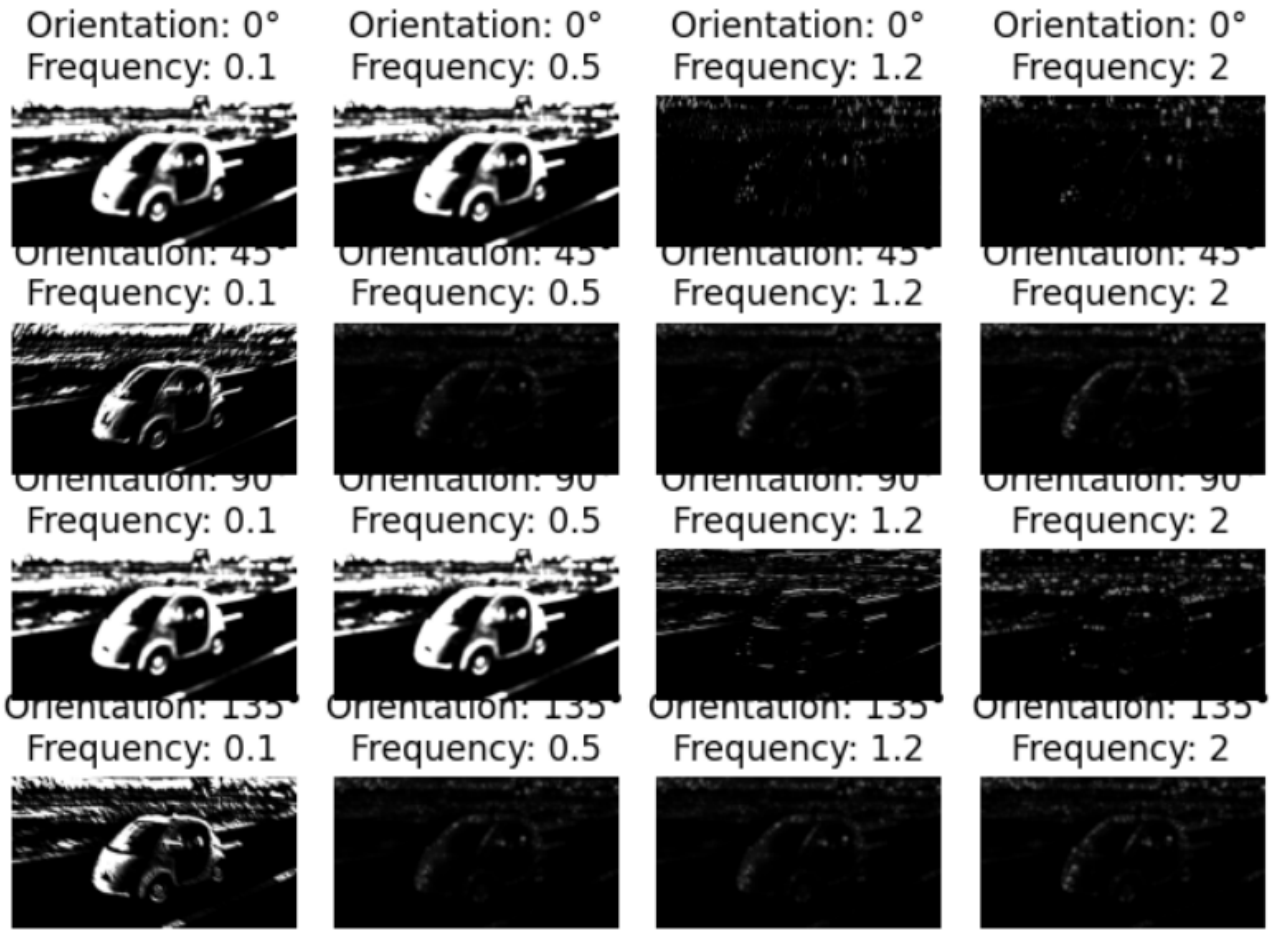
```
            frequency,                    # Frequency
            1.0,                          # Aspect ratio
            0,                            # Phase offset
            ktype=cv2.CV_32F
        )
        # Applying  the Gabor filter on the image( corrected
graysclawe image )
        filtered_image = cv2.filter2D(image, cv2.CV_8UC3,
gabor_kernel)
```

- Get the filtered images at different orientation and frequencies and append it in filtered images which contain the image feature at that particular orientation and frequency.

```
# Define Gabor filter parameters
angle = [0, 45, 90, 135]  # Orientations in degrees
freq = [0.2, 0.5, 1.2, 2]  # Frequencies
```

```
filtered_images.append((orientation, frequency, filtered_image))
```

- Display of the images/ **output** after applying filter.



Challenges:

- ➔ One of the challenges was choosing the correct frequency and orientation for image analysis.
- ➔ Visualizing the filtered images in an informative way was a bit challenging.
- ➔ Choosing the correct parameter for Gaborkernel was challenging.
- ➔ Getting clear and understandable image was challenging

Result:

- ➔ filtered images will highlight the features in the original image that are aligned with the orientation and frequency of the Gabor filter.
- ➔ From the output/filtered image displayed it is observed that Lower frequencies capture larger structures, than higher frequencies .
- ➔ Gabor filters are highly sensitive to orientation and correct orientation need to be chosen.,

Part-3 Objective: Winner-Takes-All and Normalization :

Steps of implementation:

- The filtered image analysis which we got using Gobler filter at different frequency and orientation will now be used to get dominant orientation for each pixel location for the WTA algorithm.
- After applying the algorithm and perform normalization on the output image.
- Normalizing the output image will enhance feature visibility and preserve contextual relationships.
- Normalization is done using the 'cv2.normalize' function.

```
normalized_output = cv2.normalize(wta_output, None, normalize_min,
normalize_max, cv2.NORM_MINMAX)
```

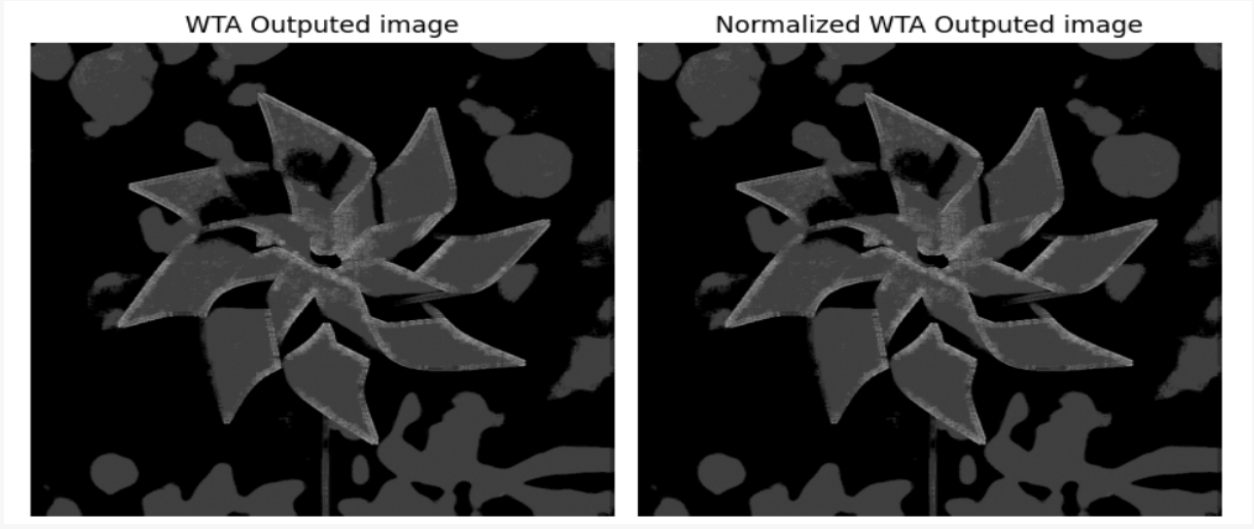
Main code of WTA implementation :- .

```
 filtered_orientation = np.arctan2(np.imag(filtered_image),
np.real(filtered_image))
          wta_mask = (np.abs(filtered_image) > wta_threshold)  #
Apply threshold
          # Ensure both arrays have the same dimensions for the
assignment
```

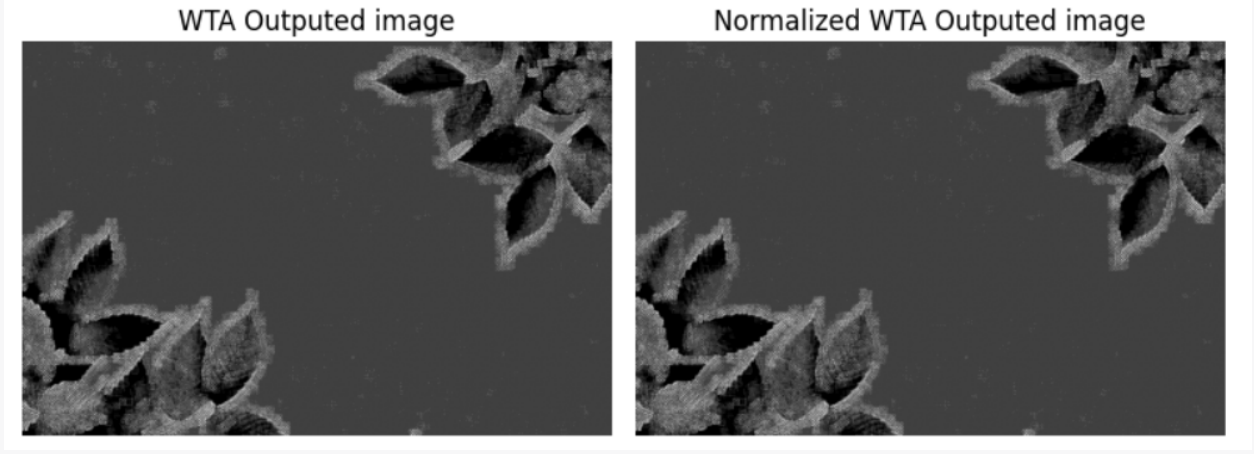
```
            filtered_orientation = np.where(wta_mask,
filtered_orientation, 0.0)

            filtered_orientation = np.array(filtered_orientation,
dtype='uint8')
            filtered_orientation = cv2.resize(filtered_orientation,
(wta_output.shape[1], wta_output.shape[0]),
interpolation=cv2.INTER_LINEAR)
            # Accumulate orientations using element-wise addition
            wta_output += filtered_orientation
```

Output : image after applying WTA and normalizing

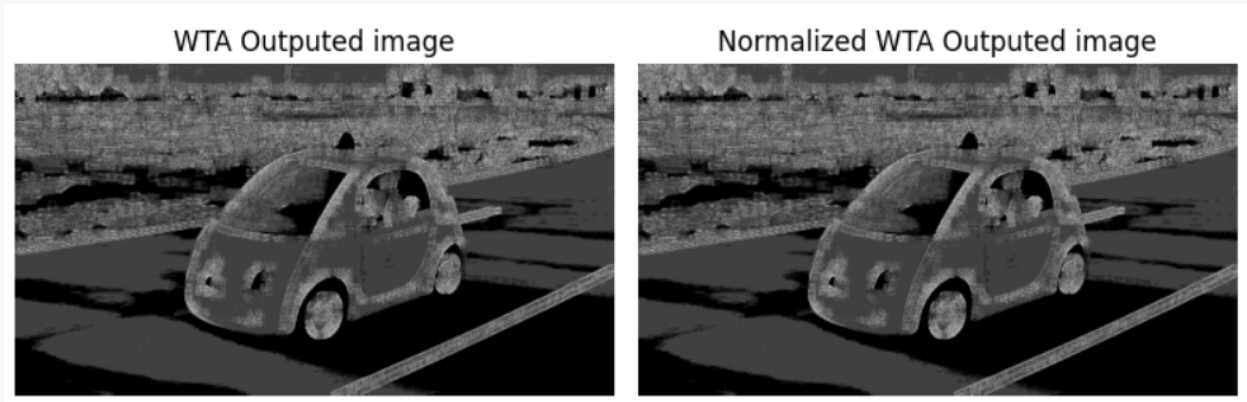
- **Image 1**



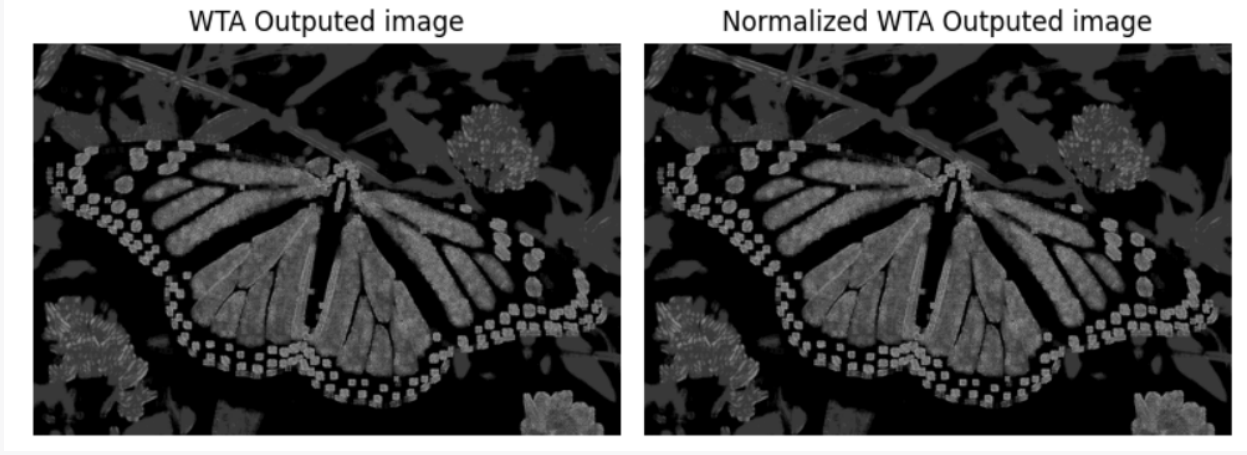
- **Image 2**



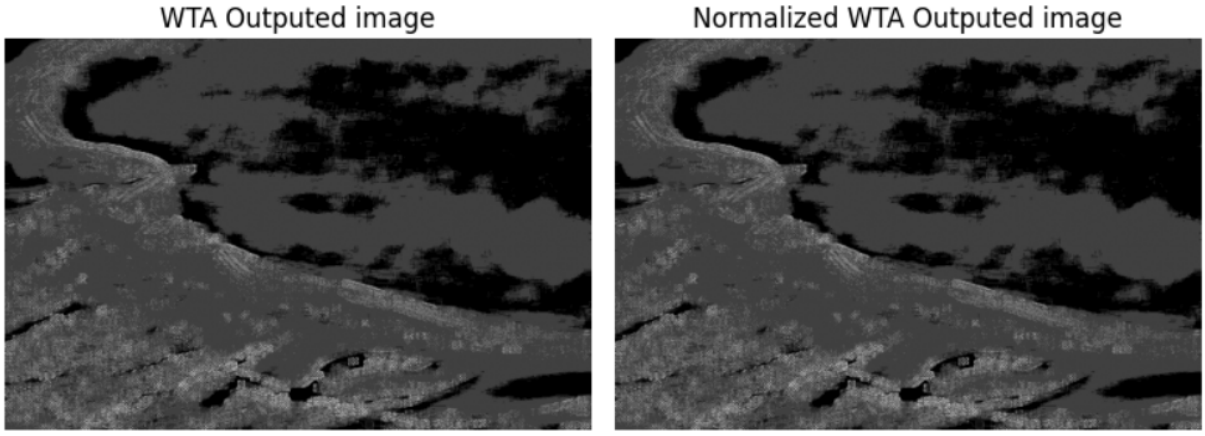
➢ **Image 3**



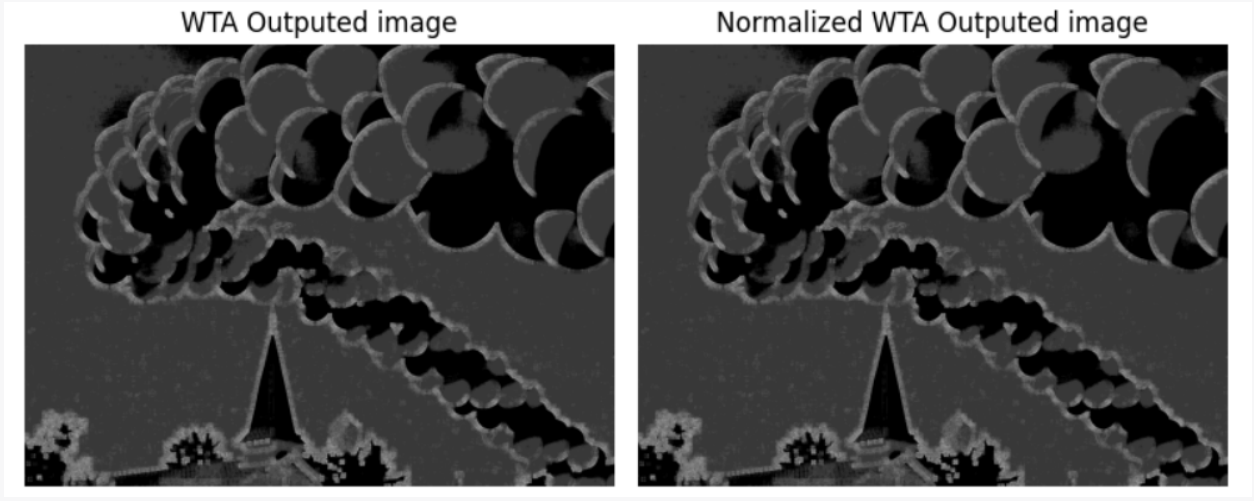
➢ **Image 4**



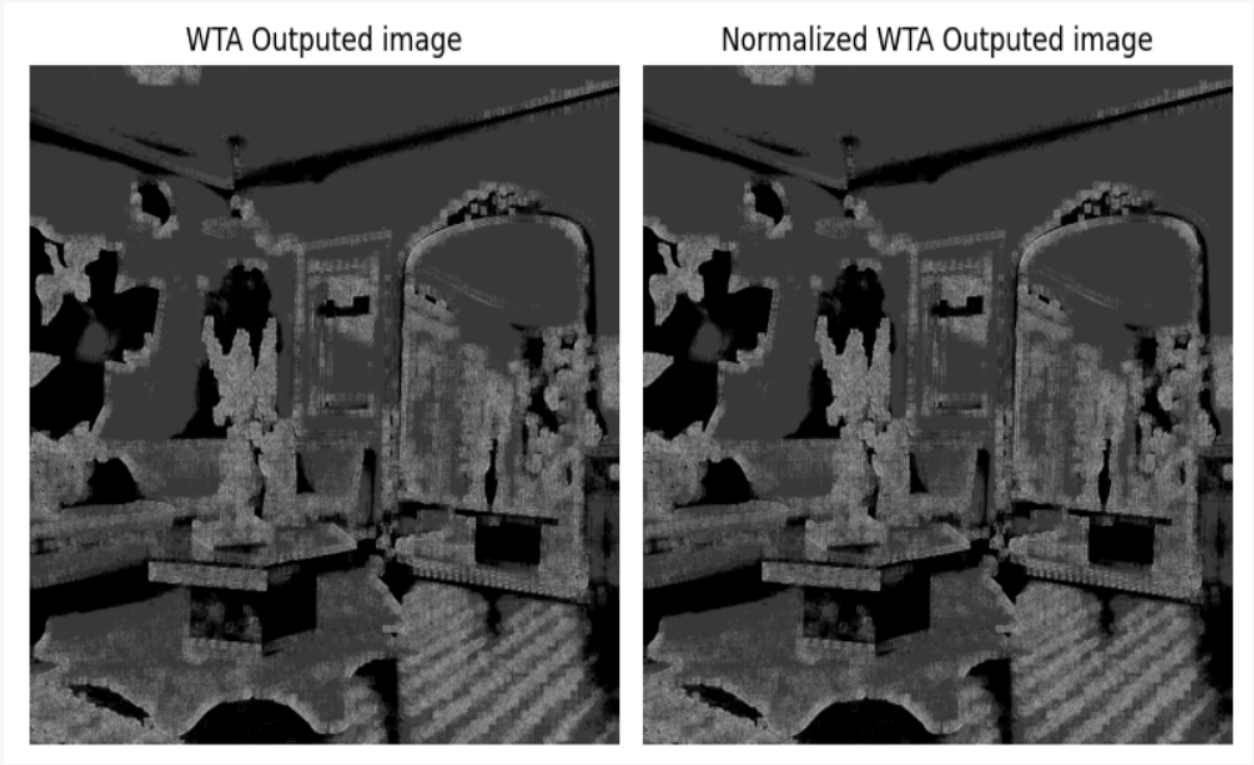
➢ **Image 5**



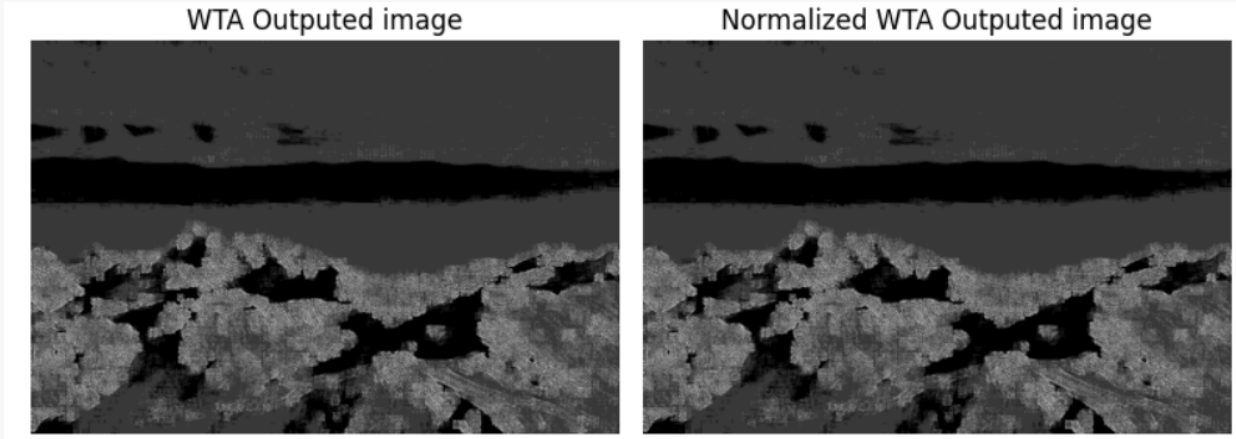
➢ **Image 6**



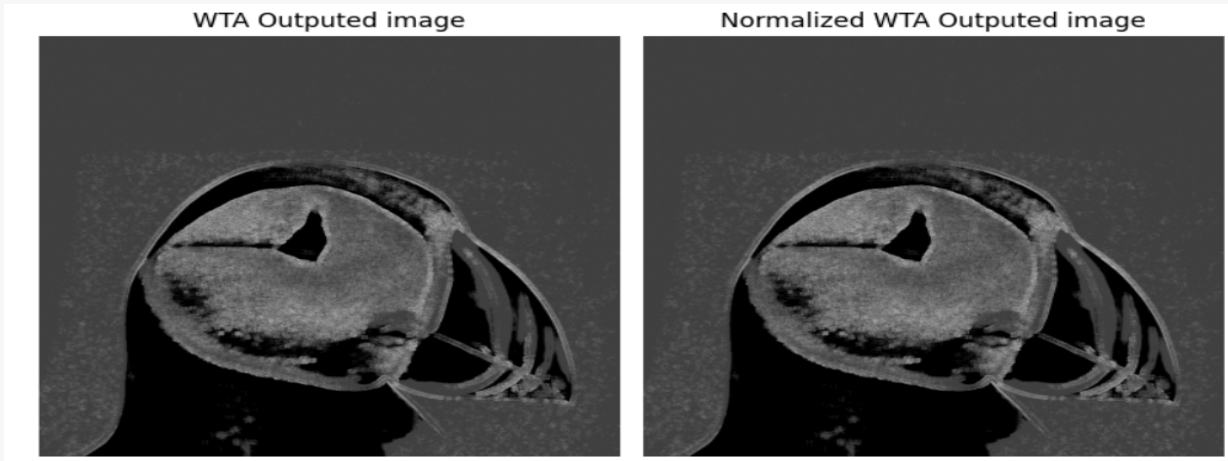
➢ **Image 7**



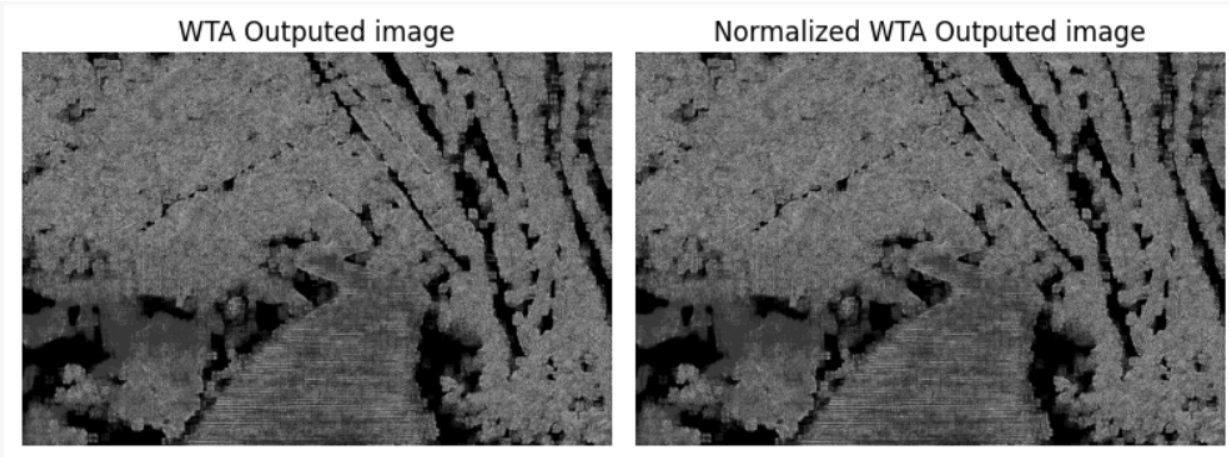
➢ **Image 8**



➢ **Image 9**



➢ **Image 10**



Challenges:

- ➔ The main challenge faced in this c=part was resizing of the filtered and wta_mask image to the same shape.
- ➔ To choose the correct wta_threshold for the algorithm so as to enhance certain features.
- ➔ Choosing correct normalization technique out of all is also important

Result:

- ➔ The WTA algorithm has enhanced the visibility of important features in the filtered images.
- ➔ Normalizing the WTA output images enhanced the feature visibility while preserving with contextual relationships.
- ➔ The final images displayed provide for informative knowledge that that produced after applying Gabor filter.

Part 4: Comparative Analysis

Steps of implementation:

- Implemented all the above 3 parts again on the image which are loading of images, conversion to grayscale image, Complex orientation filtering using Gobler filter using OpenCV library.
- Then apply the WTA algorithm on the filtered images to get dominant orientation for each pixel .
- Apply noise on the image (gaussian noise)

```
    noisy_image = cv2.add(image, np.random.normal(0, 25,
image.shape).astype(np.uint8))
```

- Then performed edge detection on original images and the noisy images and did comparison using SSIM and edge F-1 score.
- Here SSIM is used to measure how well the structures (textures, edges, patterns) in the two images match. ***Higher SSIM values indicate higher similarity between the images.***

```
    image = cv2.resize(image, (normalized_output.shape[1],
normalized_output.shape[0]))
    noisy_image= cv2.resize(noisy_image,
(normalized_noisy_output.shape[1],
normalized_noisy_output.shape[0]))
```

```
    ssim_original = ssim(image, normalized_output, win_size=3)
    ssim_noisy = ssim(noisy_image, normalized_noisy_output,
win_size=3)
```

- Whereas edge F-1 score is used to know how accurately the edges detected in an image match the ground truth edges. ***Higher F1-scores indicate better edge detection performance.***

```
    edge_f1_original = calculate_edge_f1_score(image,
normalized_output)
    edge_f1_noisy = calculate_edge_f1_score(image,
normalized_noisy_output)
```

Output of Edge Detection implementation:

```
/usr/local/lib/python3.10/dist-packages/skimage/_shared/utils.py:348:
UserWarning: Inputs have mismatched dtype.  Setting data_range based on
im1.dtype.
  return func(*args, **kwargs)
/usr/local/lib/python3.10/dist-packages/skimage/_shared/utils.py:348:
UserWarning: Inputs have mismatched dtype.  Setting data_range based on
im1.dtype.
  return func(*args, **kwargs)
Image no : 1
Shape of gt_edges: (408, 612)
Shape of normalized_output: (408, 612)
Shape of normalized_noisy_output: (408, 612)
SSIM (Original): 0.0027227493200805792
SSIM (Noisy): 0.008013693973213479
Edge F1-score (Original): 0.16689496626919276
Edge F1-score (Noisy): 0.16689496626919276


Image no : 2
Shape of gt_edges: (408, 612)
Shape of normalized_output: (408, 612)
Shape of normalized_noisy_output: (408, 612)
SSIM (Original): 0.062075394192038905
SSIM (Noisy): 0.016628599604471505
Edge F1-score (Original): 0.8570597970521515
Edge F1-score (Noisy): 0.8570597970521515
```

```
/usr/local/lib/python3.10/dist-packages/skimage/_shared/utils.py:348:
UserWarning: Inputs have mismatched dtype.  Setting data_range based on
im1.dtype.
  return func(*args, **kwargs)
/usr/local/lib/python3.10/dist-packages/skimage/_shared/utils.py:348:
UserWarning: Inputs have mismatched dtype.  Setting data_range based on
im1.dtype.
  return func(*args, **kwargs)
Image no : 3
Shape of gt_edges: (408, 612)
Shape of normalized_output: (408, 612)
Shape of normalized_noisy_output: (408, 612)
SSIM (Original): 0.004271073725768984
SSIM (Noisy): 0.011396622879568
Edge F1-score (Original): 0.34559006293518574
Edge F1-score (Noisy): 0.34559006293518574


Image no : 4
Shape of gt_edges: (408, 612)
Shape of normalized_output: (408, 612)
Shape of normalized_noisy_output: (408, 612)
SSIM (Original): 0.0029447557933550516
SSIM (Noisy): 0.009903832391006393
Edge F1-score (Original): 0.1827508373379933
Edge F1-score (Noisy): 0.1827508373379933


/usr/local/lib/python3.10/dist-packages/skimage/_shared/utils.py:348:
UserWarning: Inputs have mismatched dtype.  Setting data_range based on
im1.dtype.
  return func(*args, **kwargs)
Image no : 5
Shape of gt_edges: (408, 612)
Shape of normalized_output: (408, 612)
Shape of normalized_noisy_output: (408, 612)
SSIM (Original): 0.0013501104233867047
SSIM (Noisy): 0.01084825133177175
Edge F1-score (Original): 0.11973345688638769
Edge F1-score (Noisy): 0.11973345688638769


Image no : 6
Shape of gt_edges: (408, 612)
Shape of normalized_output: (408, 612)
Shape of normalized_noisy_output: (408, 612)
SSIM (Original): 0.0031566566787996353
```

```
SSIM (Noisy): 0.009361511155829024
Edge F1-score (Original): 0.3269035464938948
Edge F1-score (Noisy): 0.3269035464938948


/usr/local/lib/python3.10/dist-packages/skimage/_shared/utils.py:348:
UserWarning: Inputs have mismatched dtype.  Setting data_range based on
im1.dtype.
  return func(*args, **kwargs)
/usr/local/lib/python3.10/dist-packages/skimage/_shared/utils.py:348:
UserWarning: Inputs have mismatched dtype.  Setting data_range based on
im1.dtype.
  return func(*args, **kwargs)
Image no : 7
Shape of gt_edges: (408, 612)
Shape of normalized_output: (408, 612)
Shape of normalized_noisy_output: (408, 612)
SSIM (Original): 0.008893396557794917
SSIM (Noisy): 0.01051616805541004
Edge F1-score (Original): 0.4737699755722483
Edge F1-score (Noisy): 0.4737699755722483


Image no : 8
Shape of gt_edges: (408, 612)
Shape of normalized_output: (408, 612)
Shape of normalized_noisy_output: (408, 612)
SSIM (Original): 0.004014842764991037
SSIM (Noisy): 0.009957292139418721
Edge F1-score (Original): 0.25564655895342986
Edge F1-score (Noisy): 0.25564655895342986


/usr/local/lib/python3.10/dist-packages/skimage/_shared/utils.py:348:
UserWarning: Inputs have mismatched dtype.  Setting data_range based on
im1.dtype.
  return func(*args, **kwargs)
/usr/local/lib/python3.10/dist-packages/skimage/_shared/utils.py:348:
UserWarning: Inputs have mismatched dtype.  Setting data_range based on
im1.dtype.
  return func(*args, **kwargs)
Image no : 9
Shape of gt_edges: (408, 612)
Shape of normalized_output: (408, 612)
Shape of normalized_noisy_output: (408, 612)
SSIM (Original): 0.032576824456913386
SSIM (Noisy): 0.010550356809239467
```

```
Edge F1-score (Original): 0.8000480595938964
Edge F1-score (Noisy): 0.8000480595938964
```

```
Image no : 10
Shape of gt_edges: (408, 612)
Shape of normalized_output: (408, 612)
Shape of normalized_noisy_output: (408, 612)
SSIM (Original): -0.0024838147153631853
SSIM (Noisy): 0.0019074097449963442
Edge F1-score (Original): 0.35272026453524824
Edge F1-score (Noisy): 0.35272026453524824
```

```
/usr/local/lib/python3.10/dist-packages/skimage/_shared/utils.py:348:
UserWarning: Inputs have mismatched dtype.  Setting data_range based on
im1.dtype.
  return func(*args, **kwargs)
```

- ***Pros and cons of Gabor-based edge detection*** in challenging scenarios.
    - Pros:
        - It is effective in analyzing the texture patterns of the image at different scales and orientations .
        - And even in detecting and highlighting the edges.
        - It is robust to certain types of noise due to its complexity.
    - Cons:
        - Gabor based edge detection might not perform well at certain scales
        - Computationally more costly.
        - Its effectiveness and output depend on orientation and frequency scale chosen
        - Gabor-based edge detection might not perform well when edge occur at different scale.
        - It can be computationally more costly sometimes
        - Its effectives and output depend on the orientation and frequency scale chosen for the images.

Challenges:

- ➔ The main challenge faced in this part was resizing of the filtered and wta_mask image to the same shape.
- ➔ Automating the entire pipeline for multiple images can be complex and time taking.
- ➔ Applying noise to images after complete accessing of the data was very difficult.

Result:

- ➔ Higher SSIM values indicate higher similarity between the images.
- ➔ Higher F1-scores indicate better edge detection performance.
- ➔ According to the data, image 9 of the bird has a high Edge F1 score which indicates how well the detected image matches the ground truth edge.
- ➔ According to the data image 5 has high SSIM value out of all images which indicate that the processed image retains the structural details of the original image.

Part-5 Objective : Visualization

Steps of implementation:

- After all the analysis and edge and noise detection methods implement Overlaying edge-detected on the original image at different transparency level.

```
 # Print or store the results and metrics for analysis
    def overlay_edges_on_image(image, edges, transparency):
        # Explicitly specify the data type for the output image (use
np.uint8 for 8-bit images)
        overlay_egde_img = cv2.addWeighted(image.astype(np.uint8), 1
- transparency, edges.astype(np.uint8), transparency, 0)
        return overlay_egde_img

    # Visualization parameters
    transparency_levels = [0.1, 0.3, 0.5, 0.8]  # Adjust
transparency levels as needed
```

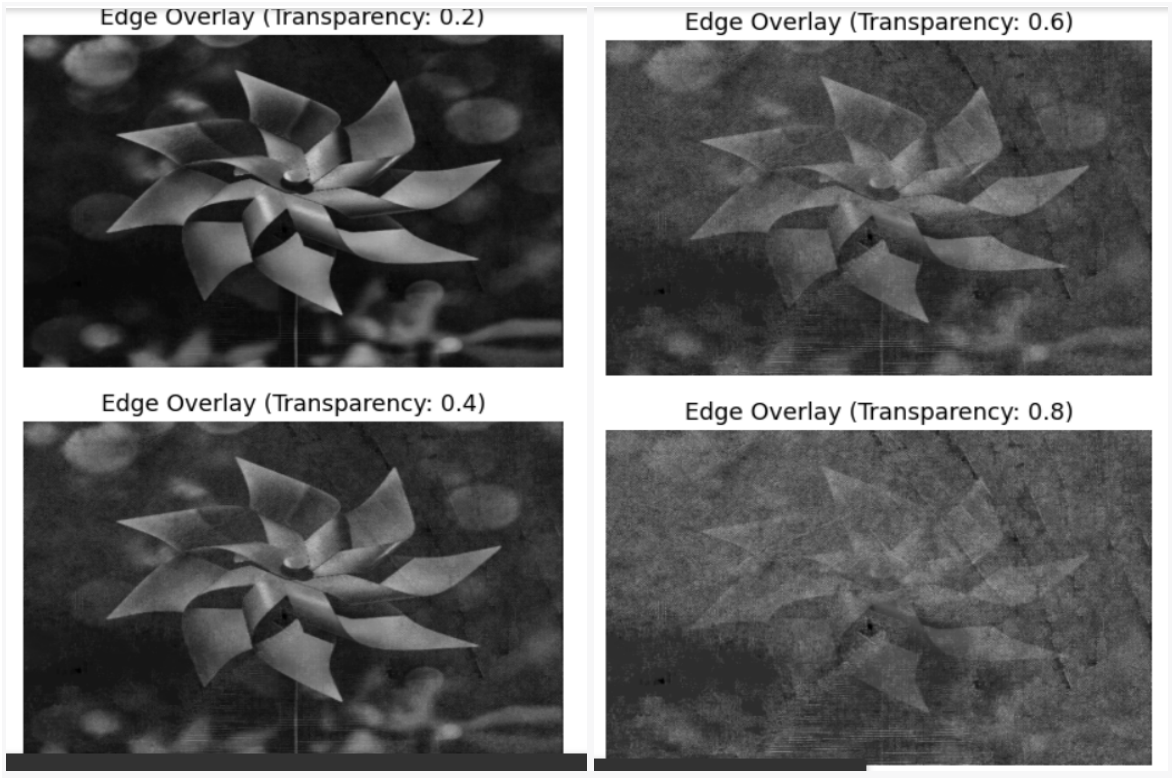
- To generate Gradient magnitude and orientation use the OpenCV library in python on the image.
- Here we made use of cv2.Sobel function to perform edge detection by computing the gradient of an image which basically is used to represent rate of change of pixel values in both the horizontal (x) and vertical (y) directions.

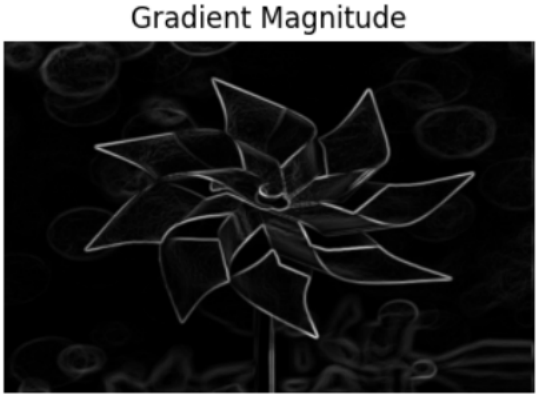
```
# Generate gradient magnitude and orientation maps on the original
image.
    x_dir = cv2.Sobel(image, cv2.CV_64F, 1, 0, ksize=5)  # Compute
x-gradient
```

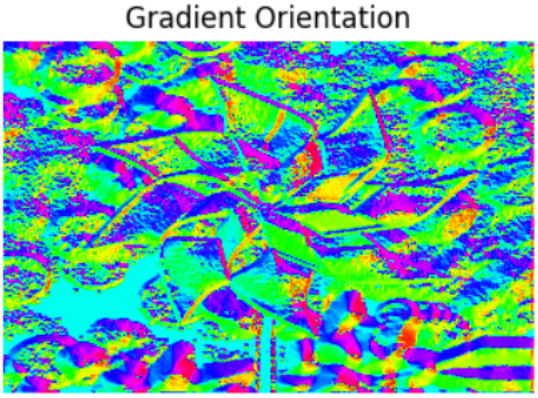
```
    y_dir = cv2.Sobel(image, cv2.CV_64F, 0, 1, ksize=5)  # Compute
y-gradient
    grad_magnitude = np.sqrt(x_dir**2 + y_dir**2)  # Compute
gradient magnitude
    grad_orientation = np.arctan2(y_dir, x_dir)  # Compute gradient
orientation
```

Output: *Images on different transparency level and their gradient magnitude and orientation*

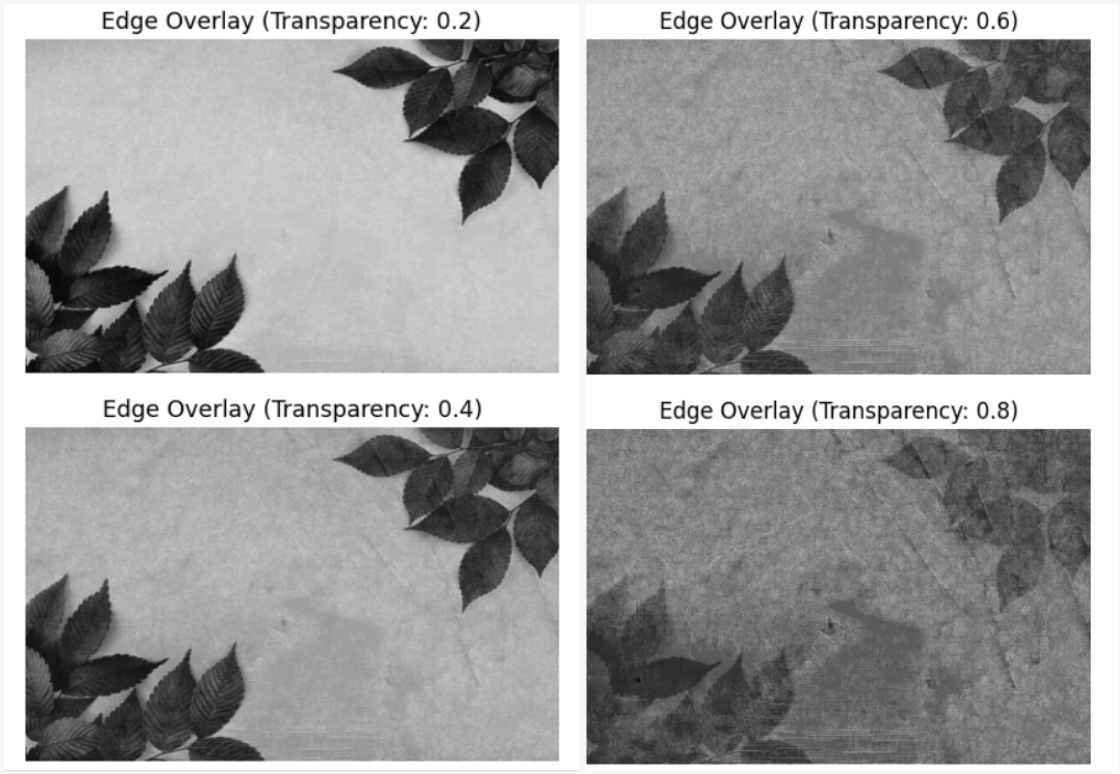
➢ **Image 1 :**

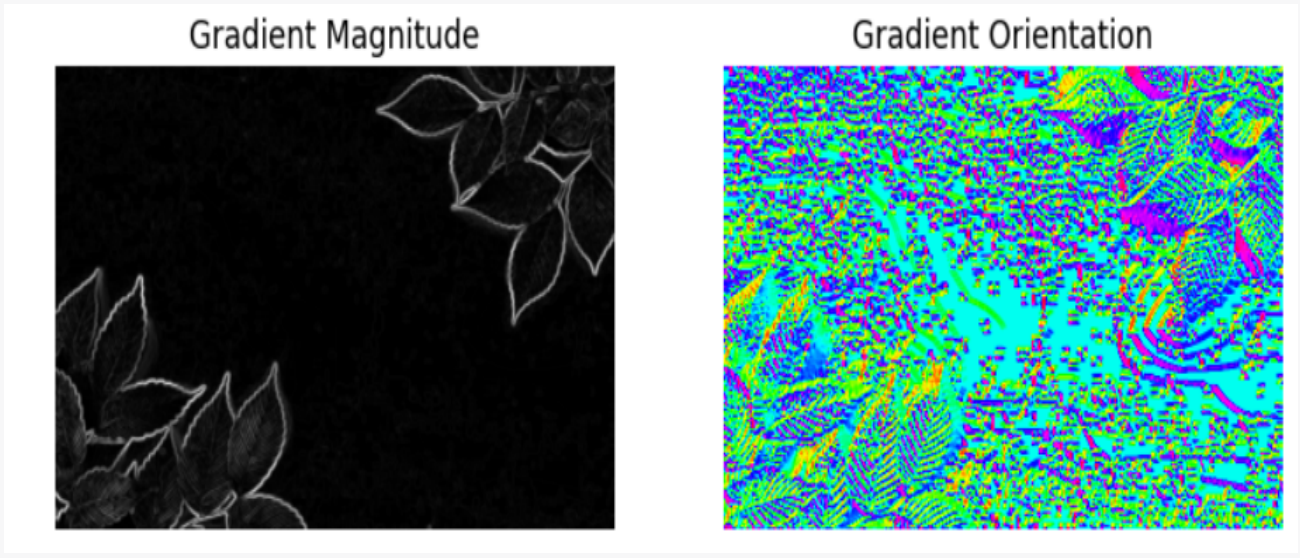




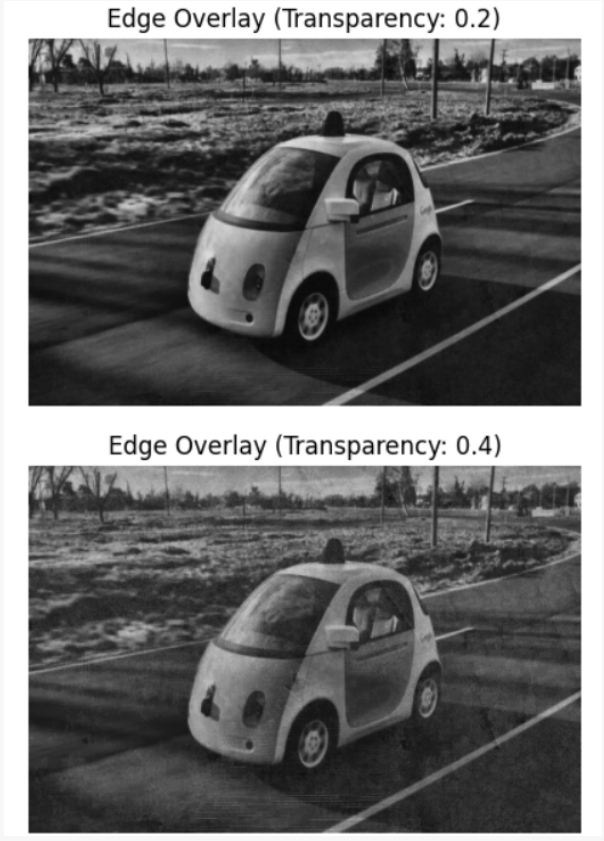


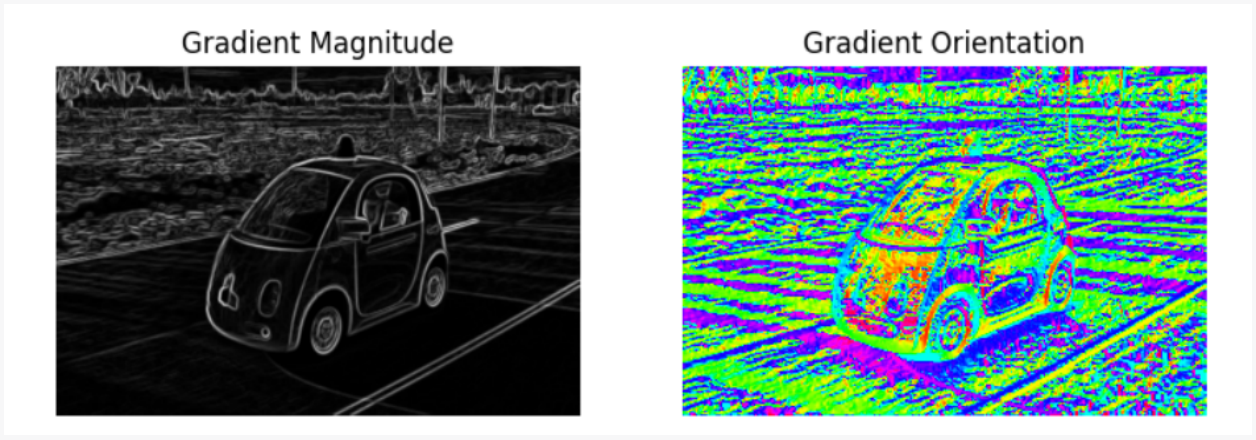
➢ **Image 2:**


Edge Overlay (Transparency: 0.2)
Edge Overlay (Transparency: 0.6)
Edge Overlay (Transparency: 0.4)
Edge Overlay (Transparency: 0.8)

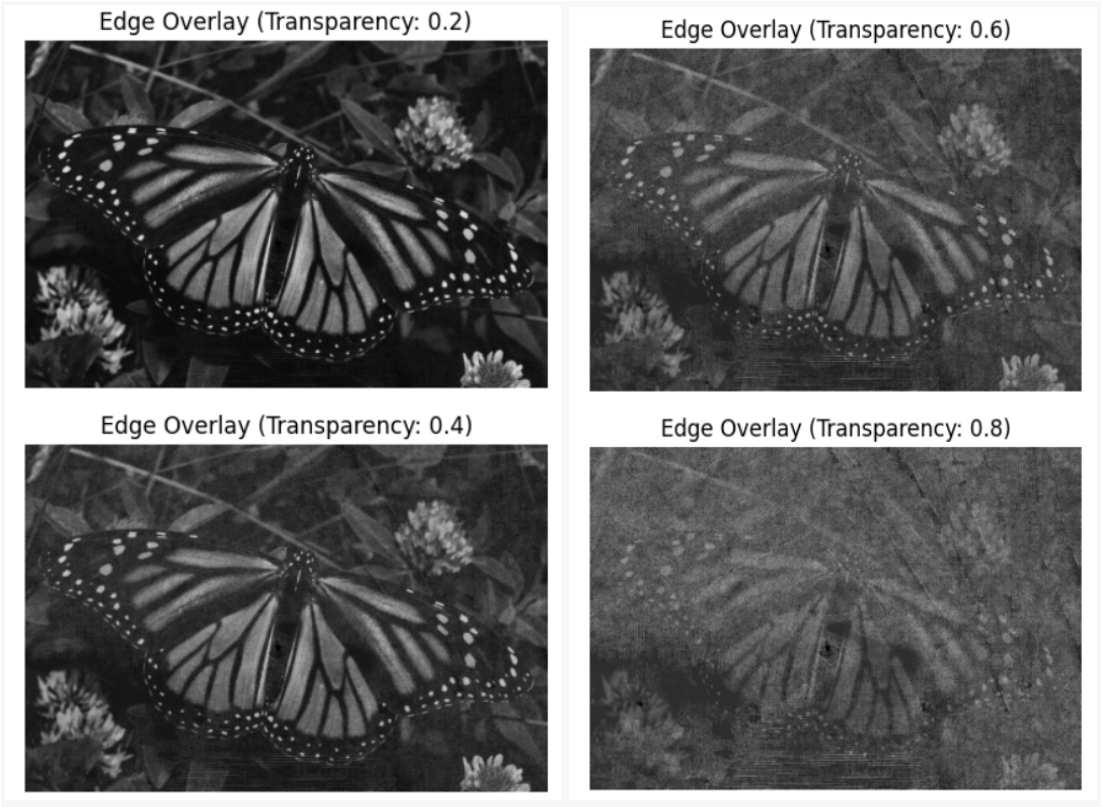

Gradient Magnitude
Gradient Orientation

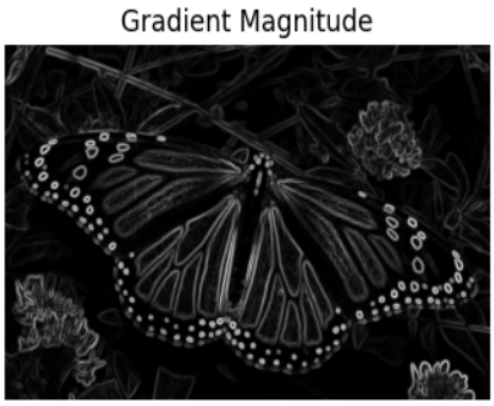
➢ **Image 3:**

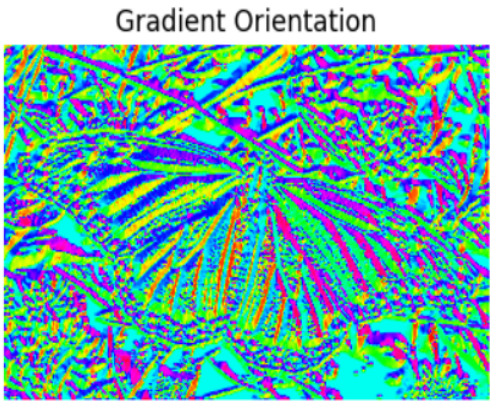

Edge Overlay (Transparency: 0.2)
Edge Overlay (Transparency: 0.6)
Edge Overlay (Transparency: 0.4)
Edge Overlay (Transparency: 0.8)


Gradient Magnitude
Gradient Orientation

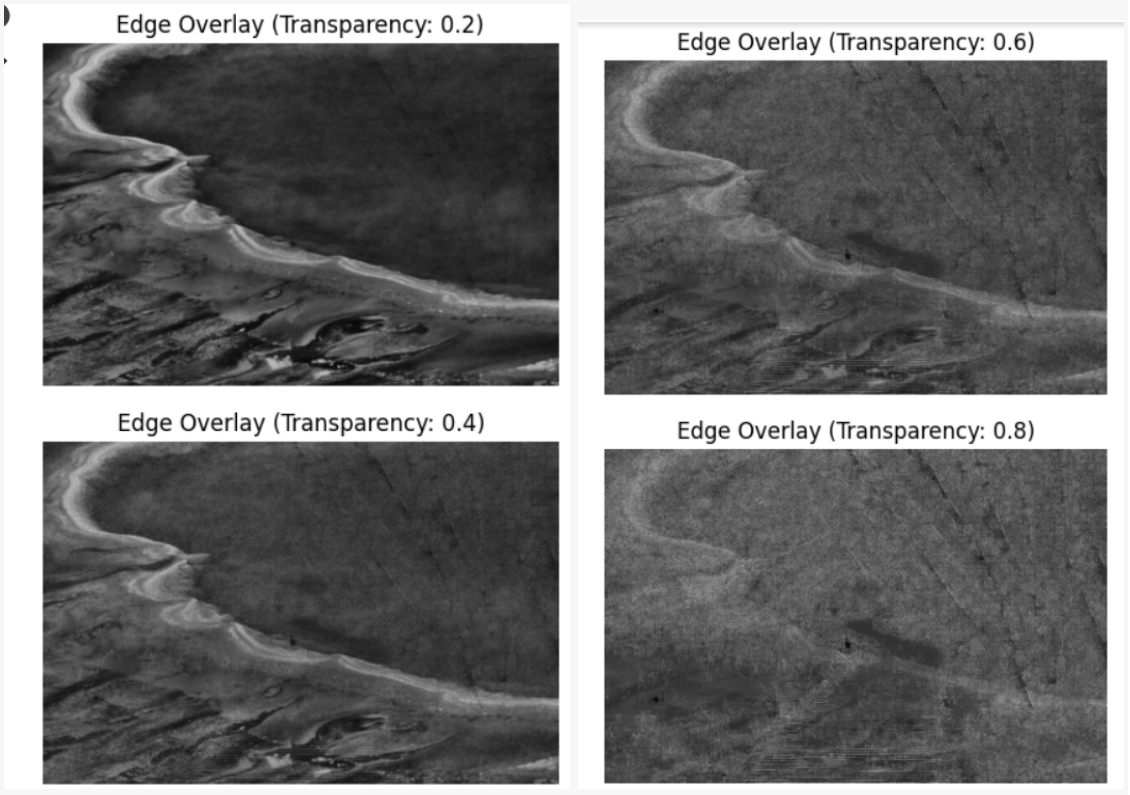
➢ Image 4

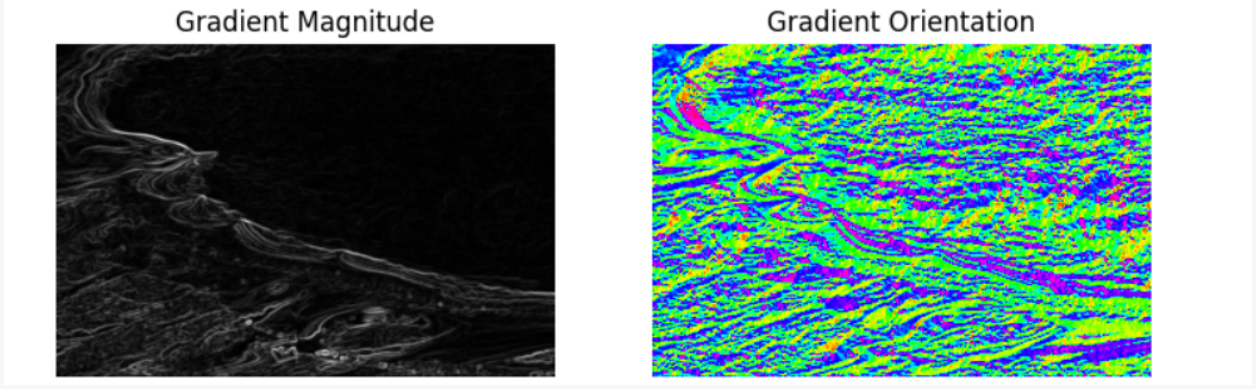

Edge Overlay (Transparency: 0.2)
Edge Overlay (Transparency: 0.6)
Edge Overlay (Transparency: 0.4)
Edge Overlay (Transparency: 0.8)


Gradient Magnitude

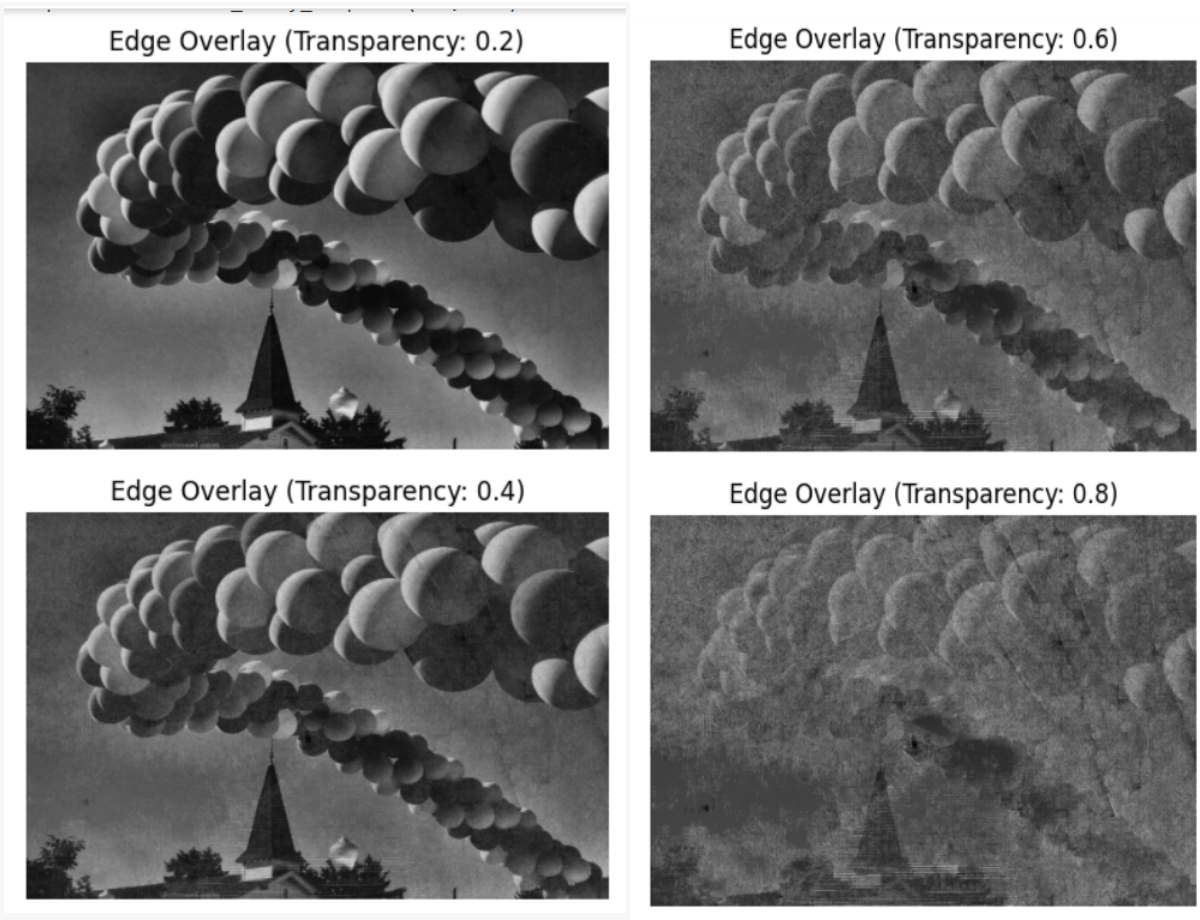

Gradient Orientation

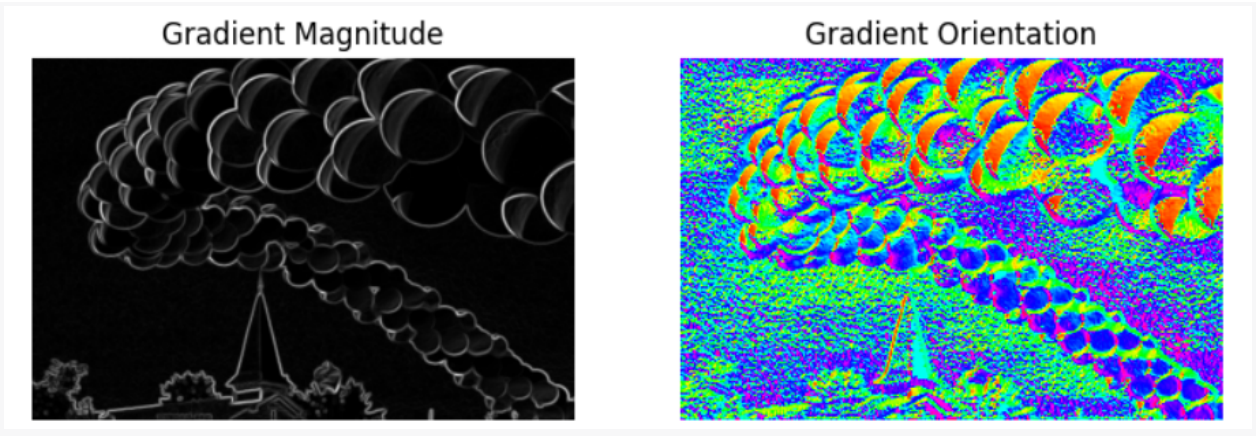
➢ **Image 5:**


Edge Overlay (Transparency: 0.2)
Edge Overlay (Transparency: 0.6)
Edge Overlay (Transparency: 0.4)
Edge Overlay (Transparency: 0.8)

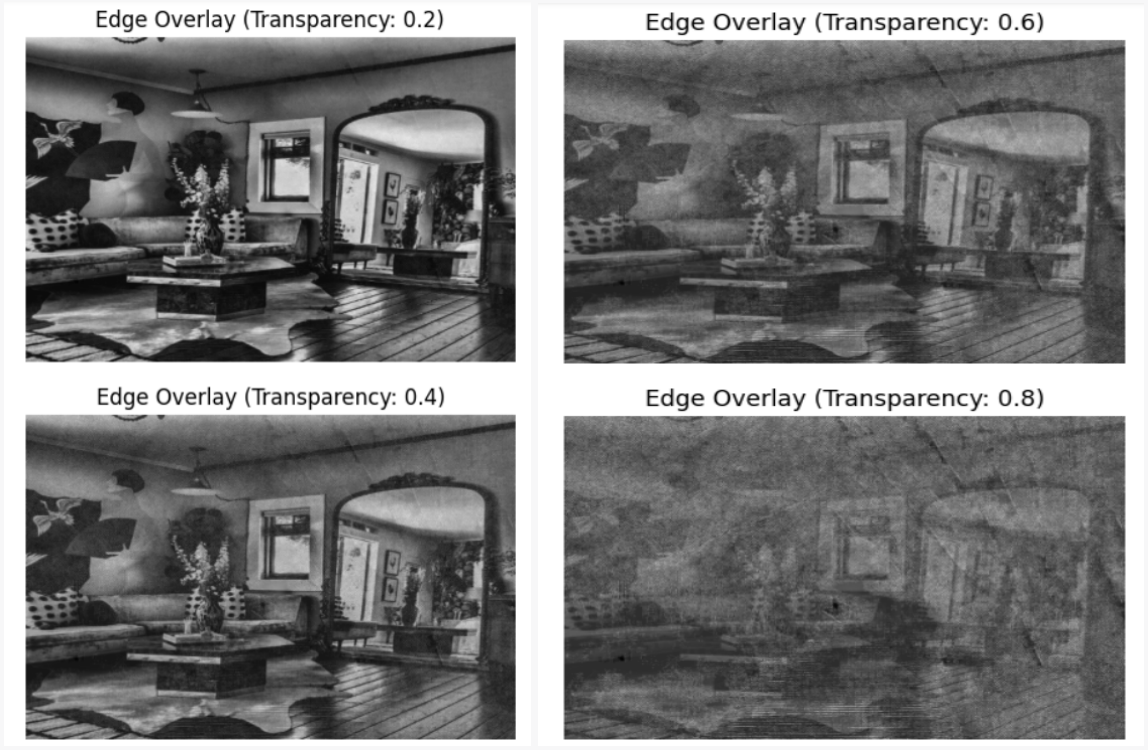

Gradient Magnitude
Gradient Orientation

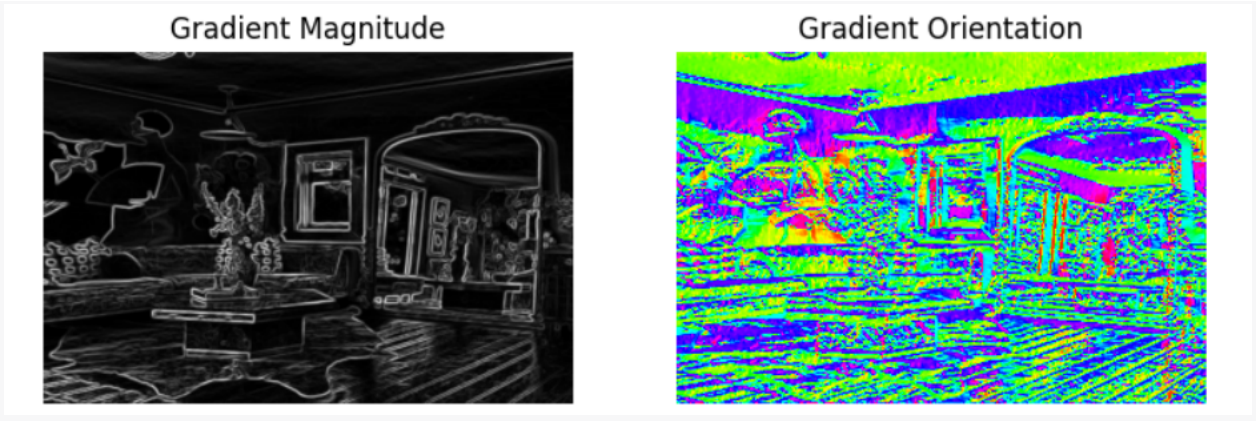
➢ **Image 6:**


Edge Overlay (Transparency: 0.2)
Edge Overlay (Transparency: 0.6)
Edge Overlay (Transparency: 0.4)
Edge Overlay (Transparency: 0.8)

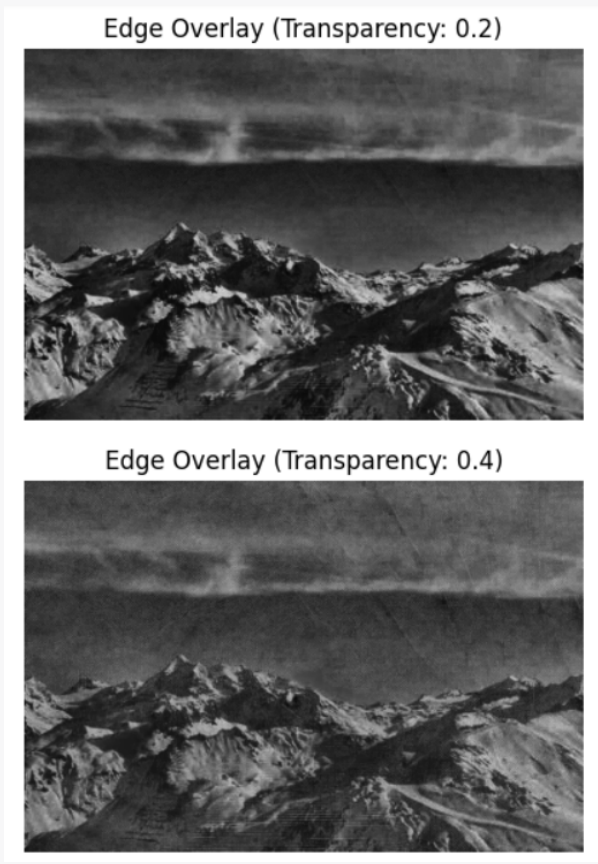

Gradient Magnitude
Gradient Orientation

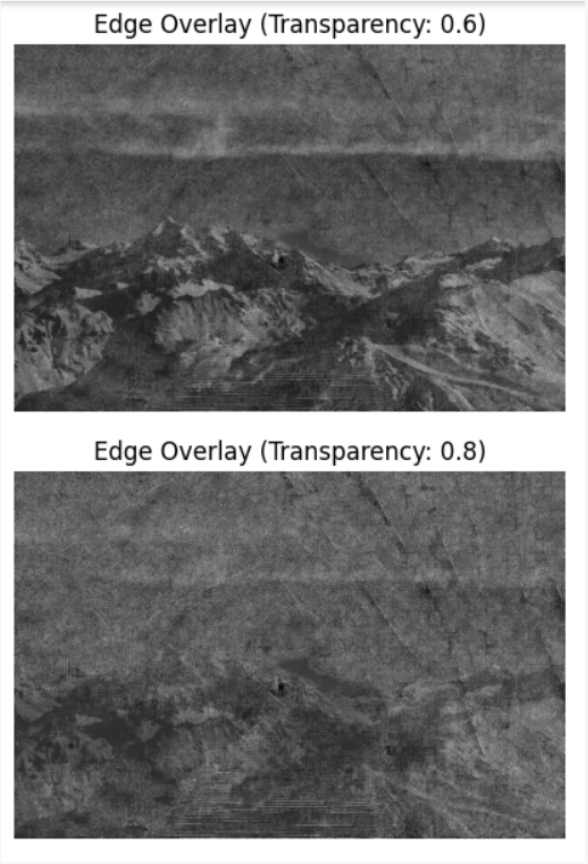
➢ Image 7:

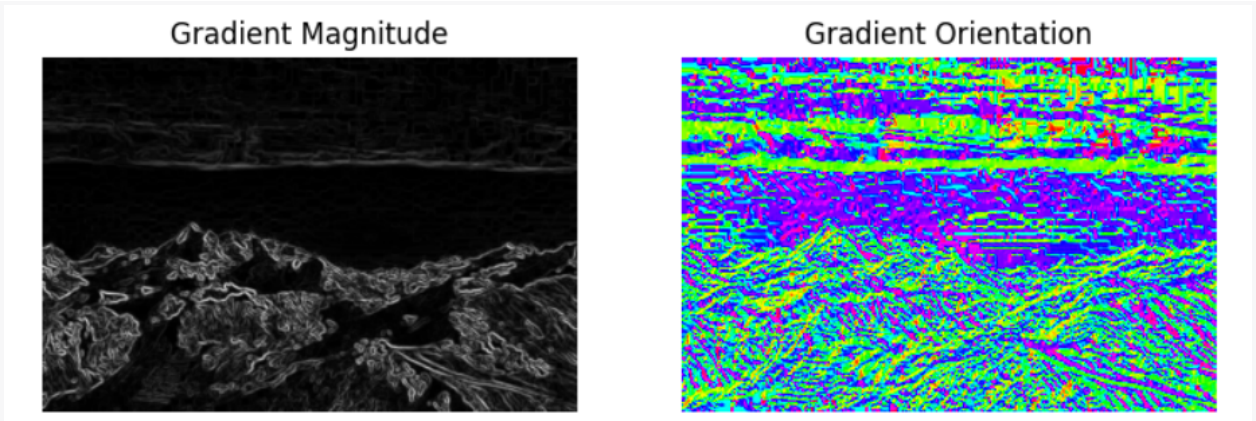

Edge Overlay (Transparency: 0.2)
Edge Overlay (Transparency: 0.6)
Edge Overlay (Transparency: 0.4)
Edge Overlay (Transparency: 0.8)


Gradient Magnitude
Gradient Orientation

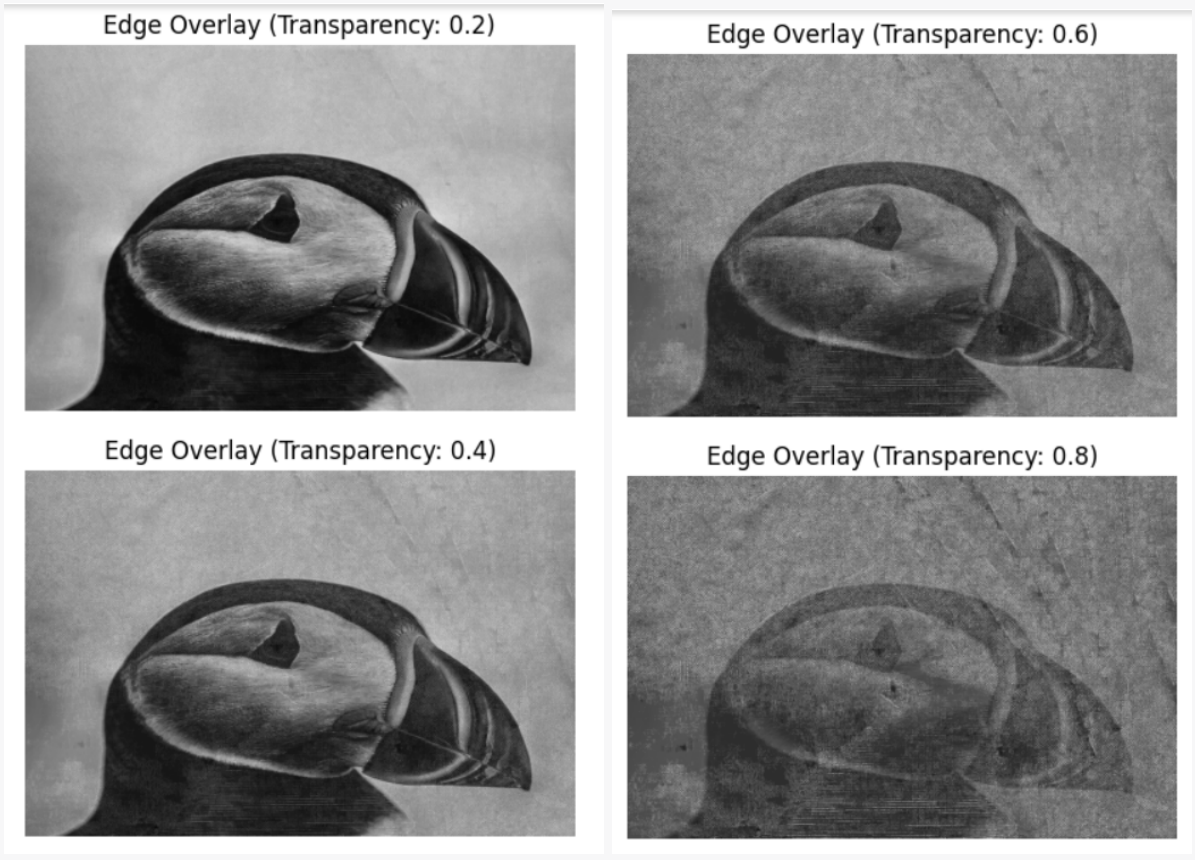
➢ Image 8

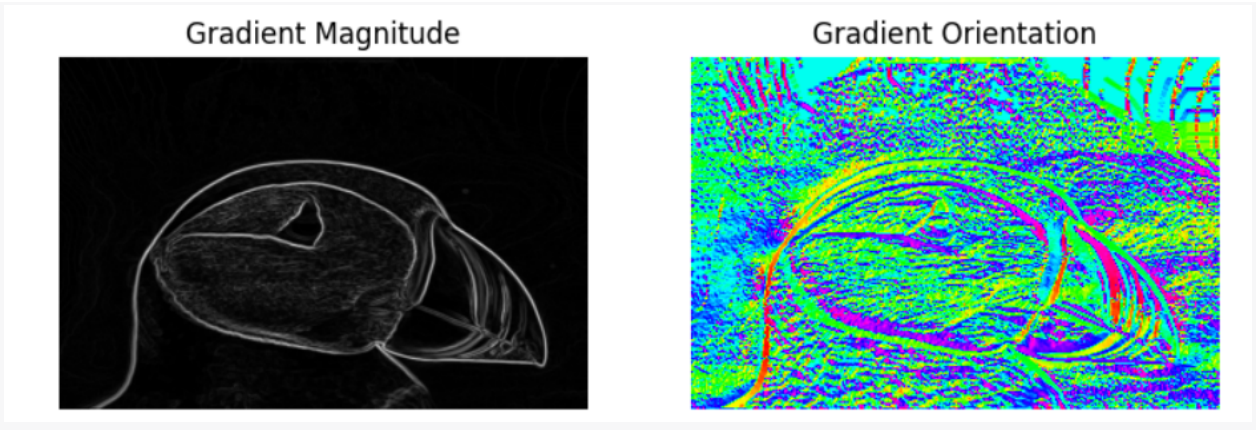

Edge Overlay (Transparency: 0.2)
Edge Overlay (Transparency: 0.4)


Edge Overlay (Transparency: 0.6)
Edge Overlay (Transparency: 0.8)

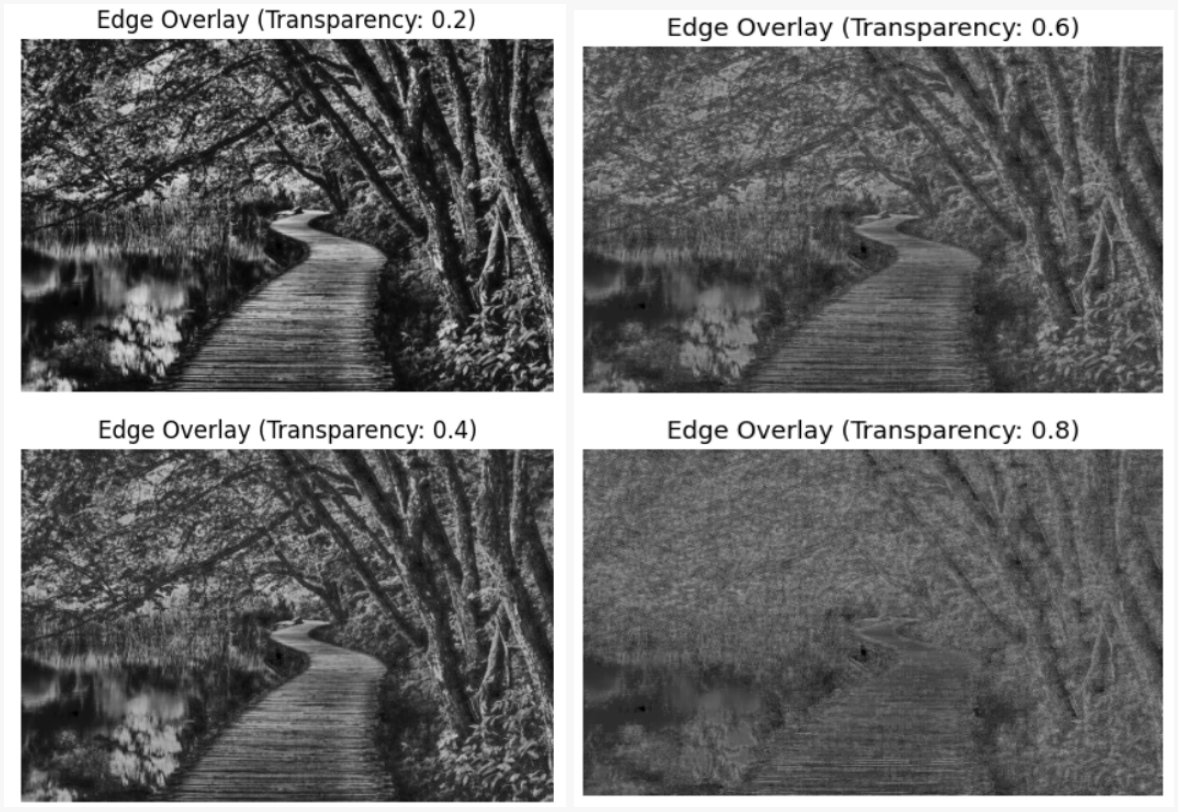

Gradient Magnitude
Gradient Orientation

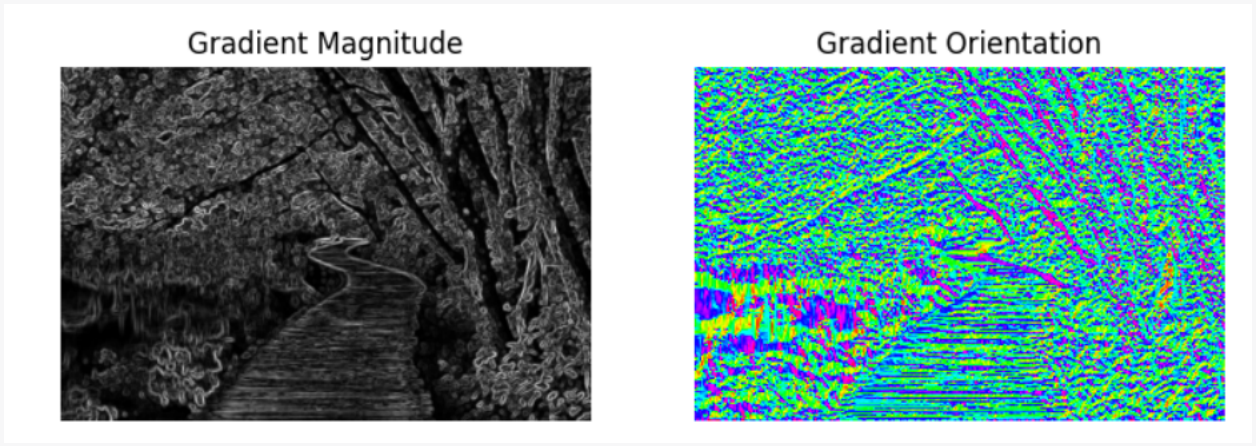
➢ **Image 9:**


Edge Overlay (Transparency: 0.2)
Edge Overlay (Transparency: 0.6)
Edge Overlay (Transparency: 0.4)
Edge Overlay (Transparency: 0.8)


Gradient Magnitude
Gradient Orientation

➢ **Image 10:**





Challenges:

- ➔ Choosing the correct scale ,color and thickness to overlay the images.
- ➔ Choosing correct scale for visualizing and computing the gradient magnitude
- ➔ Choosing the correct library for computing gradient magnitude and orientation.
- ➔ Here noise can impact the final result.
- ➔ choosing correct color scales and representation is necessary.

Result:

- ➔ Overlaying technique allows visual comparison between detected edges and original features at different transparency levels.
- ➔ Using gradient magnitude we compare the x magnitude and y magnitude to identify the region correctly.
- ➔ orientation map helps visualize the directions of edges and their orientations for correct and clear display of the images.

Conclusions from the assignment :

- ➔ The above process of image analysis pipeline is an iterative process and with each part improvement and more data was created and analyzed.
- ➔ During the start , Importing and downloading images is a necessary part of the above assignment.
- ➔ For this keep the gamma correction value low, for better image quality.
- ➔ The gamma corrected grayscale image is better that simple grayscale image in terms of contrast, brightness and highlight preservation of the image.
- ➔ Implementing complex orientation filters, such as Gabor wavelets, is used to extract features, including edges and textures, from images with better contrast.
- ➔ Then the WTA algorithm was used to enhance the visibility of important features in the filtered images.
- ➔ It is important to include noise in the image properly, without disturbing the working of code.
- ➔ Visualization techniques, such as overlaying detected edges, gradient magnitude and orientation maps, are tools for interpreting and assessing the quality of edge detection results with better quality.
- ➔ SSIM and Edge F1-score help in evaluating the accuracy of edge detection and structural symmetry. Along with this the image with higher SSIM values indicate higher similarity between the images and the image with higher F1-scores indicate better edge detection performance.