# 15ECE212 - SIGNAL PROCESSING II

# BLUR DETECTION & EDGE DETECTION

## ABSTRACT:

### BLUR DETECTION BY FFT:

This paper analyzes how novices and experts can easily identify and eliminate images that are of low quality, or more precisely images riddled with blurs, and extricate themselves from the pain of having to manually go through each one of the photos captured through their phones or cameras. It is to be noted that there are add-on options available in a few Chinese Android Applications, but however, these are quite rare and not universally available. Thus, the problem of maintaining a junk-free gallery still persists for most users and urges the need for an easier method. We apply the concepts of Fast Fourier Transforms (FFT) and thresholding to achieve the desired results. We will understand why the FFT based blur detection method is a far easier and less time-consuming method when it comes to blur detection.

### EDGE DETECTION BY CONVOLUTION:

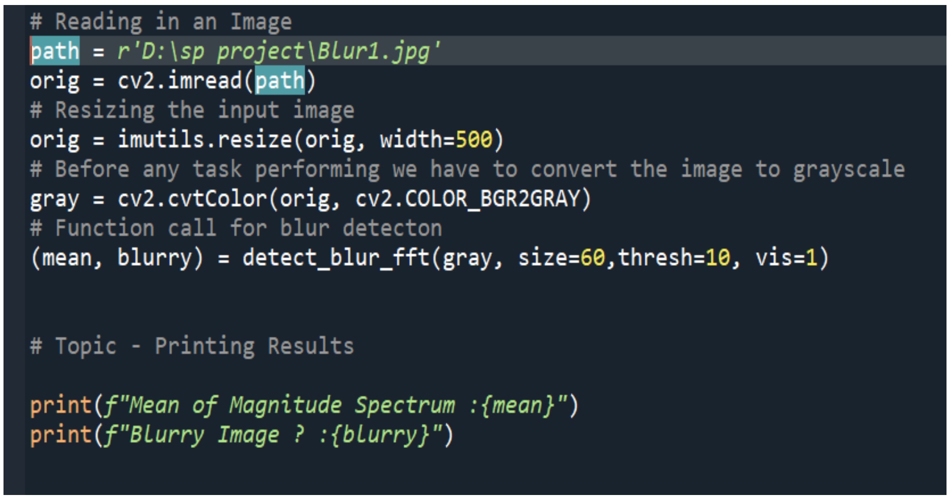
This paper carries out the analysis of pollen in flowers and uses the technique of edge detection to investigate the intricacies in pollen distribution in a given piece of land. Area which has better pollination leads to better yield. Thus, pollination is controlled artificially and balanced plantation is achieved. We make use of the concepts of Kernel Convolution and Filtering to bring about the desired results. The application of different filters/kernels in our study will positively help us to promote balanced plantation.

## METHODOLOGY:

### 1. BLUR DETECTION:

We will look into the code step by step:

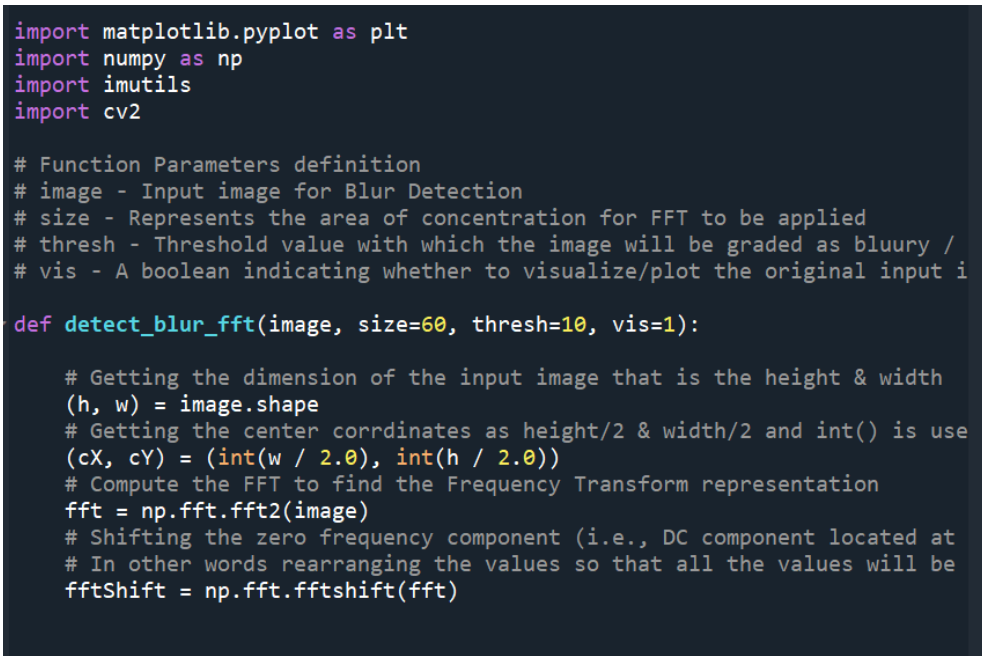
Step 1: Reading the input image



```
# Reading in an Image
path = r'D:\sp project\Blur1.jpg'
orig = cv2.imread(path)
# Resizing the input image
orig = imutils.resize(orig, width=500)
# Before any task performing we have to convert the image to grayscale
gray = cv2.cvtColor(orig, cv2.COLOR_BGR2GRAY)
# Function call for blur detecton
(mean, blurry) = detect_blur_fft(gray, size=60,thresh=10, vis=1)


# Topic - Printing Results

print(f"Mean of Magnitude Spectrum :{mean}")
print(f"Blurry Image ? :{blurry}")
```

In this step we are reading the input image and resizing it to a standard format width and converting the image into grayscale. We are converting because it is easier to work with grayscale rather than the color image. Then we are calling the function **detect_blur_fft()** for blur detection which will be explained in the coming steps and then we print the result.

Step 2: Defining the function



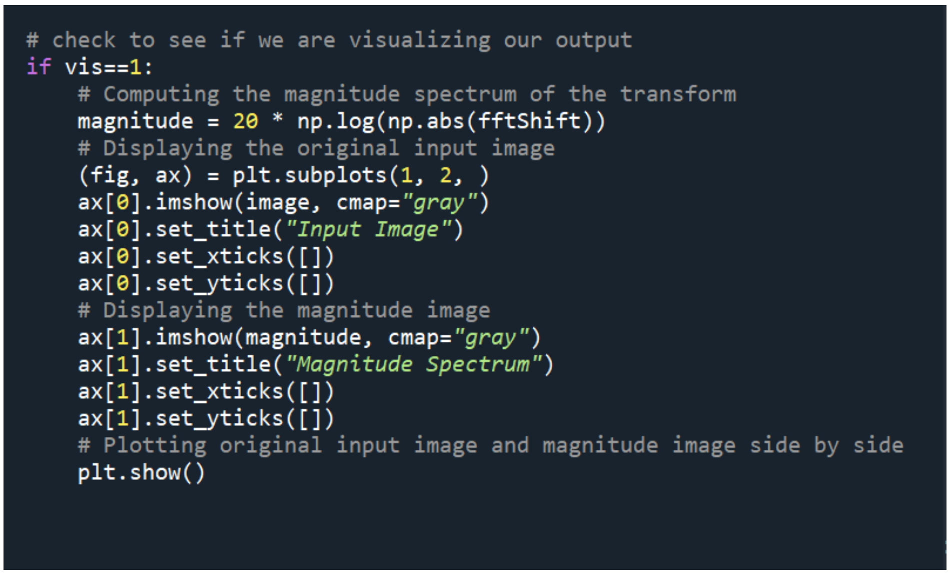
```
import matplotlib.pyplot as plt
import numpy as np
import imutils
import cv2

# Function Parameters definition
# image - Input image for Blur Detection
# size - Represents the area of concentration for FFT to be applied
# thresh - Threshold value with which the image will be graded as bluury /
# vis - A boolean indicating whether to visualize/plot the original input i

def detect_blur_fft(image, size=60, thresh=10, vis=1):

    # Getting the dimension of the input image that is the height & width
    (h, w) = image.shape
    # Getting the center corrdinates as height/2 & width/2 and int() is use
    (cX, cY) = (int(w / 2.0), int(h / 2.0))
    # Compute the FFT to find the Frequency Transform representation
    fft = np.fft.fft2(image)
    # Shifting the zero frequency component (i.e., DC component located at
    # In other words rearranging the values so that all the values will be
    fftShift = np.fft.fftshift(fft)
```

In this step we define the fuction and its parameters which does the following fuctions:

- **image**: Gray scale image is obtained in the last step is given as input
- **size**: The size of the radius around the center point of the image for which we will zero out the FFT shift
- **thresh**: A value with which the mean value of the FFT magnitude (more on that later) will be compared to for determining whether an image is considered blurry or not blurry
- **vis**: A Boolean indicating whether to visualize/plot the original input image and magnitude spectrum image using matplotlib

Coming to the function we get the dimensions of the image its height and width with which we compute the centre coordinates. Then we find FFT of the image and shift it using inbuilt NumPy functions so that the 0th frequency component is at the centre for ease of manipulation.

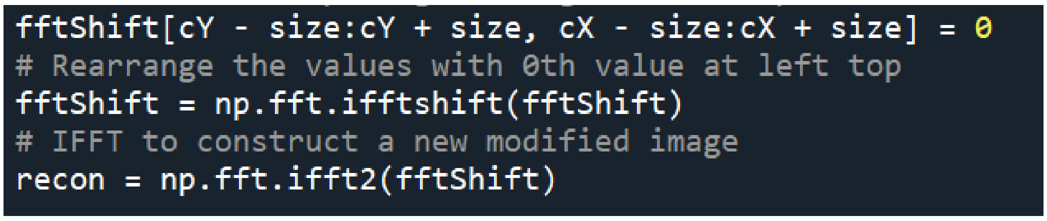
Step 3: Plotting



```python
# check to see if we are visualizing our output
if vis==1:
    # Computing the magnitude spectrum of the transform
    magnitude = 20 * np.log(np.abs(fftShift))
    # Displaying the original input image
    (fig, ax) = plt.subplots(1, 2, )
    ax[0].imshow(image, cmap="gray")
    ax[0].set_title("Input Image")
    ax[0].set_xticks([])
    ax[0].set_yticks([])
    # Displaying the magnitude image
    ax[1].imshow(magnitude, cmap="gray")
    ax[1].set_title("Magnitude Spectrum")
    ax[1].set_xticks([])
    ax[1].set_yticks([])
    # Plotting original input image and magnitude image side by side
    plt.show()
```

In this step we calculate the magnitude spectrum of the transform in terms of decibels (dB) and then we are plotting the input image and the magnitude spectrum image alongside each other.

Step 4: Image Reconstruction



```python
fftShift[cY - size:cY + size, cX - size:cX + size] = 0
# Rearrange the values with 0th value at left top
fftShift = np.fft.ifftshift(fftShift)
# IFFT to construct a new modified image
recon = np.fft.ifft2(fftShift)
```

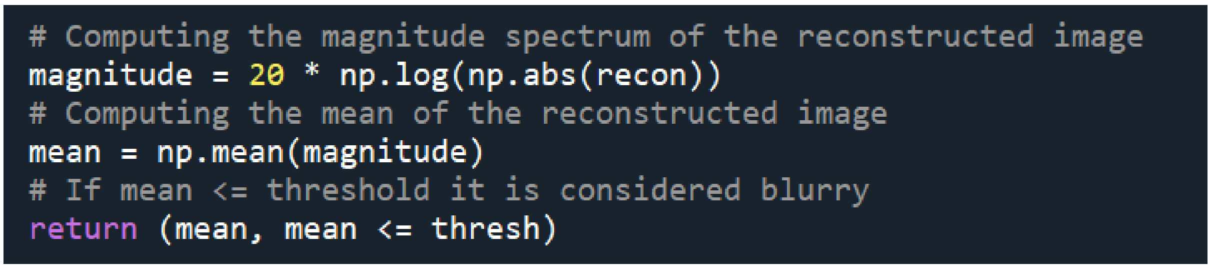
In this step, we are checking the blur by following methods,

- Removing low frequency components by setting them to 0 (High pass filter action)
- Shifting is done so that 0th frequency component appears at the top left as usual
- Image is reconstructed by using the IFFT numpy function

**NOTE:**

- Image smoothing is one in which edges & lines are not observed profusely. A blurry image is formed out of a low pass filter.
- Image sharpening is one in which edges & lines are enhanced. This occurs at high frequencies

Step 5: Blur Checking



```python
# Computing the magnitude spectrum of the reconstructed image
magnitude = 20 * np.log(np.abs(recon))
# Computing the mean of the reconstructed image
mean = np.mean(magnitude)
# If mean <= threshold it is considered blurry
return (mean, mean <= thresh)
```

In this step we are finding the mean of the magnitude and comparing it with the threshold value, if the mean is less than or equal to the threshold value, then it is considered to be a blur image else it is not a blur image.

The above can be understood as when:

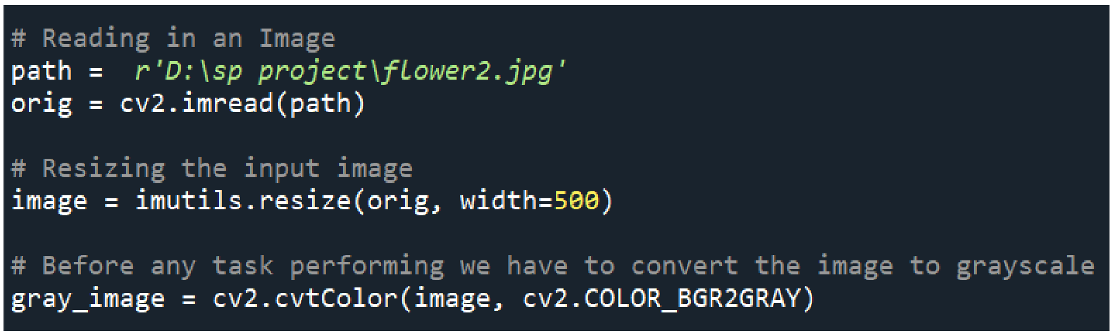
- Magnitude > threshold = It is a sharp image (Not Blurry)
- Magnitude < threshold = It is not a sharp image (Blurry)

**NOTE:** Here we are comparing the degree of sharpness to determine whether an image is blurry or not.

## 2. EDGE DETECTION:

As we did for Blur detection the same way we will look into the code of edge detection step by step:

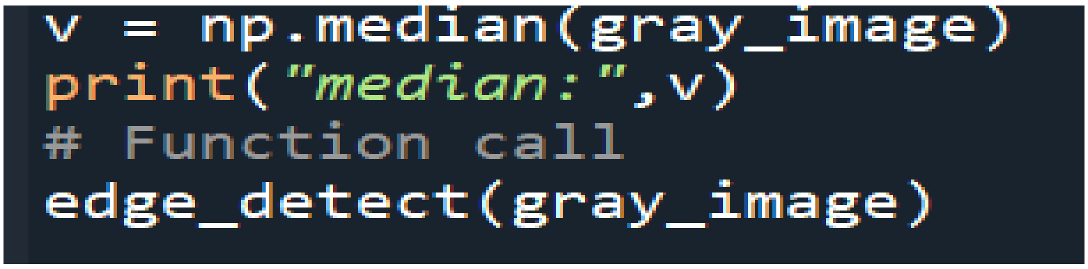
Step 1: Reading input image



```
# Reading in an Image
path =  r'D:\sp project\flower2.jpg'
orig = cv2.imread(path)

# Resizing the input image
image = imutils.resize(orig, width=500)

# Before any task performing we have to convert the image to grayscale
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
```

In this step we are reading the input image and resizing it to a standard format width and converting the image into grayscale. We are converting because it is easier to work with grayscale rather than the color image.

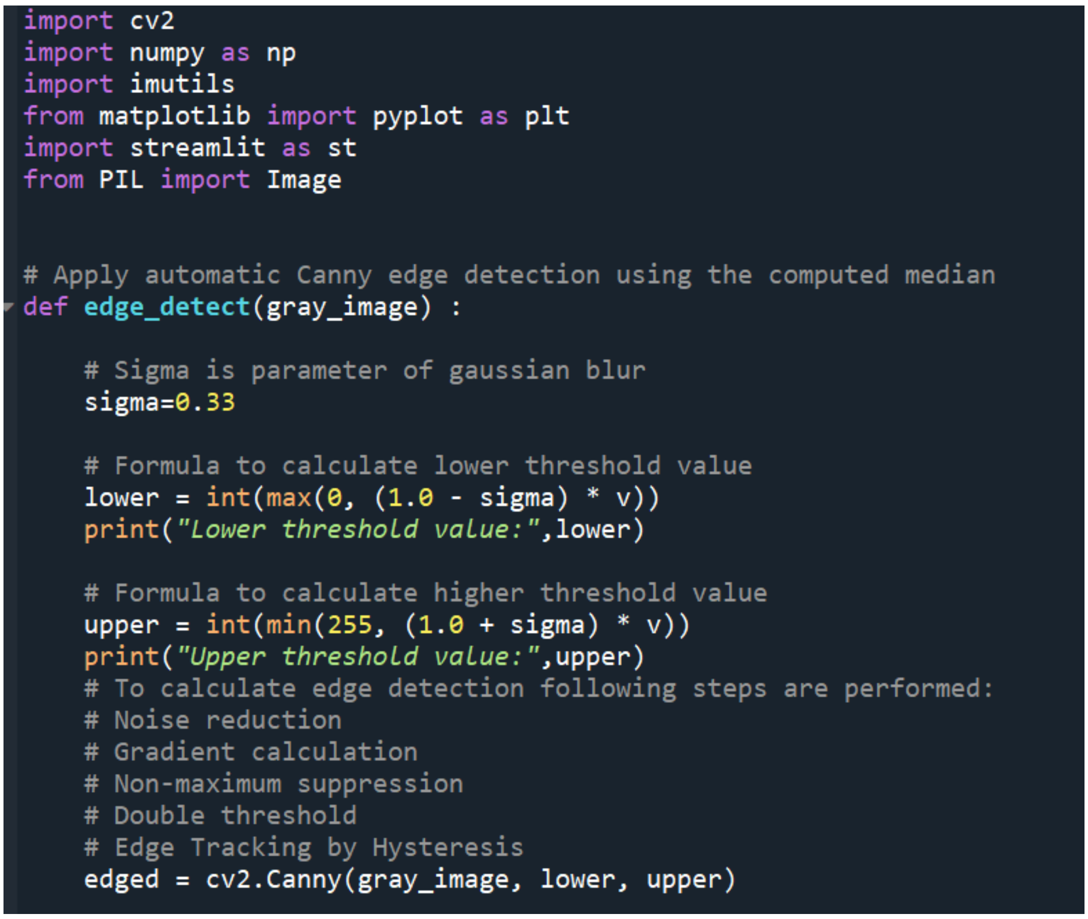
Step 2: Finding median



```
v = np.median(gray_image)
print("median:",v)
# Function call
edge_detect(gray_image)
```

In this step we find the median of the grayscale image so that using median we can calculate the lower and upper threshold values and we also call the edge detect function **edge_detect()** .

Step 3: Edge detection:



```python
import cv2
import numpy as np
import imutils
from matplotlib import pyplot as plt
import streamlit as st
from PIL import Image


# Apply automatic Canny edge detection using the computed median
def edge_detect(gray_image) :

    # Sigma is parameter of gaussian blur
    sigma=0.33

    # Formula to calculate lower threshold value
    lower = int(max(0, (1.0 - sigma) * v))
    print("Lower threshold value:",lower)

    # Formula to calculate higher threshold value
    upper = int(min(255, (1.0 + sigma) * v))
    print("Upper threshold value:",upper)
    # To calculate edge detection following steps are performed:
    # Noise reduction
    # Gradient calculation
    # Non-maximum suppression
    # Double threshold
    # Edge Tracking by Hysteresis
    edged = cv2.Canny(gray_image, lower, upper)
```

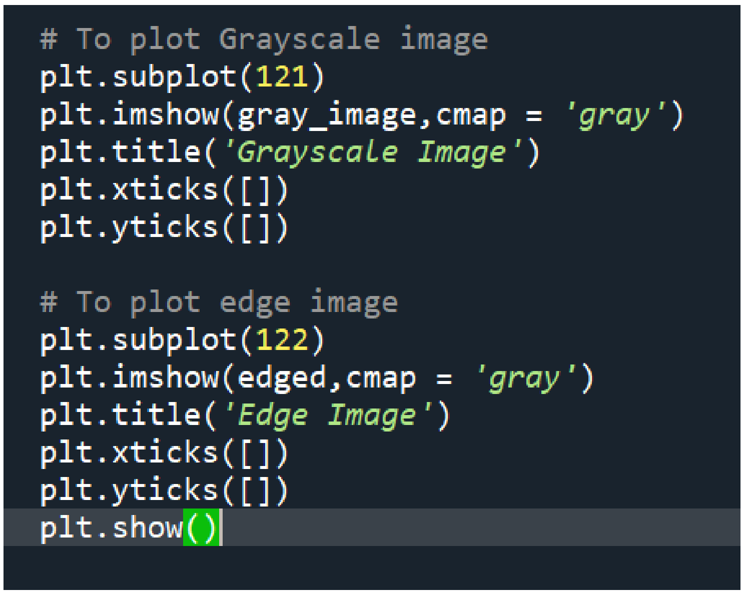
In this step we take sigma value which is a parameter of gaussian blur to be the standard value of 0.33, and with that we find the lower and upper threshold values.

To perform Edge detection we have used an inbuilt function **Canny()** which performs the following operations:

- **Noise reduction:** Edge detection results are highly sensitive to image noise. One way to get rid of the noise on the image, is by applying Gaussian blur to smooth it. To do so, image convolution technique is applied with a Gaussian Kernel (3x3, 5x5, 7x7 etc.…). The kernel size depends on the expected blurring effect. Basically, the smallest the kernel, the less visible is the blur.
- **Gaussian blur:** A type of low-pass filter, Gaussian blur smoothens uneven pixel values in an image by cutting out the extreme outliers.
- **Gradient calculation:** The Gradient calculation step detects the edge intensity and direction by calculating the gradient of the image using edge detection operators. Edges correspond to a change of pixels intensity. To detect it, the easiest way is to apply filters that highlight this intensity change in both directions: horizontal (x) and vertical (y). When the image is smoothed, the derivatives Ix and Iy w.r.t. x and y are calculated. It can be implemented by convolving I with Sobel kernels Kx and Ky, respectively.
- **Non-maximum suppression:** Pixels that would be highlighted, but seem too far from any edge, are removed. This is called non-maximum suppression, and the result is edge lines that are thinner than those produced by other methods. The principle is simple: the algorithm goes through all the points on the gradient intensity matrix and finds the pixels with the maximum value in the edge directions.

- **Double threshold:** A double threshold is applied to determine potential edges. Here extraneous pixels caused by noise or milder color variation than desired are eliminated. If a pixel's gradient value – based on the Sobel differential –is above the high threshold value, it is considered a strong candidate for an edge. If the gradient is below the low threshold value, it is turned off. If the gradient is in between, the pixel is considered a weak candidate for an edge pixel.
- **Edge Tracking by Hysteresis:** Based on the threshold results, the hysteresis consists of transforming weak pixels into strong ones, if and only if at least one of the pixels around the one being processed is a strong one.

Step 4: Plotting



```
# To plot Grayscale image
plt.subplot(121)
plt.imshow(gray_image,cmap = 'gray')
plt.title('Grayscale Image')
plt.xticks([])
plt.yticks([])

# To plot edge image
plt.subplot(122)
plt.imshow(edged,cmap = 'gray')
plt.title('Edge Image')
plt.xticks([])
plt.yticks([])
plt.show()
```

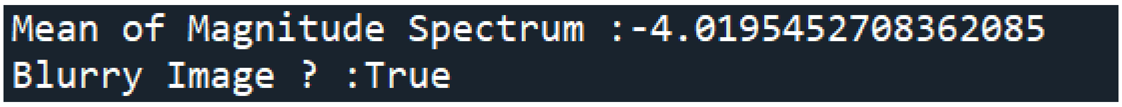
In this step graysacle image and edge image are plotted alongside each other.

**RESULTS AND DISCUSSIONS:**

1. **BLUR DETECTION:**

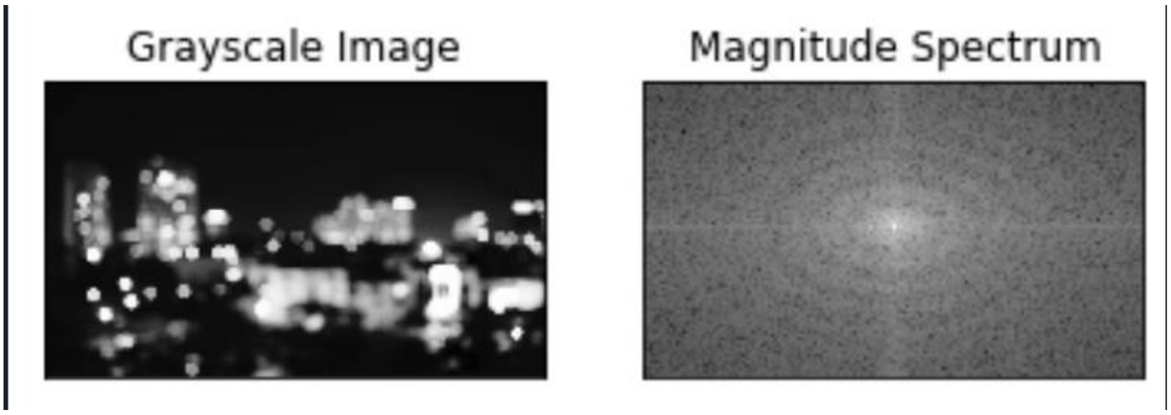
**For Blurry Image :**

**Output:**



```
Mean of Magnitude Spectrum :-4.0195452708362085
Blurry Image ? :True
```

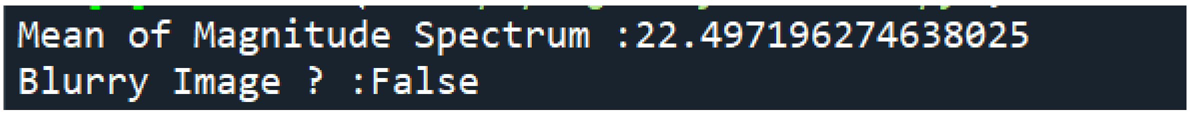
**Plot:**



From the output we can see the mean is around -4 which is very much less than the threshold value of 10. Therefore, the image is a blur image.

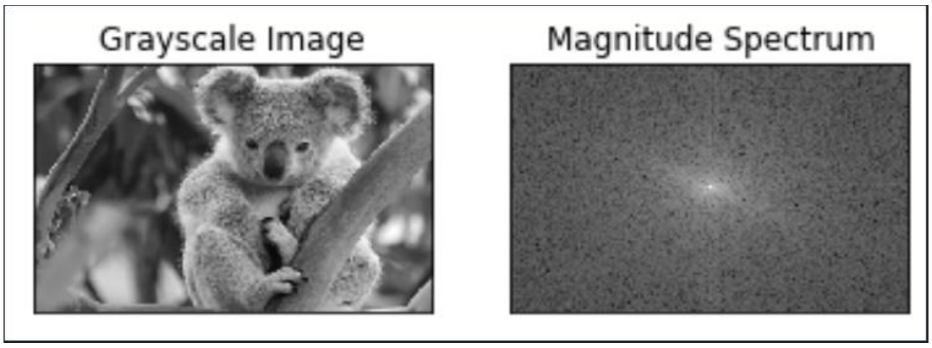
**For Not a Blurry Image :**

**Output:**



```
Mean of Magnitude Spectrum :22.497196274638025
Blurry Image ? :False
```

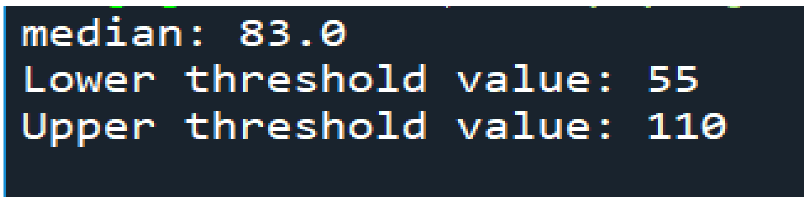
**Plot:**



From the output we can see the mean is around 22 which is very much greater than the threshold value of 10. Therefore, the image is a not a blur image.

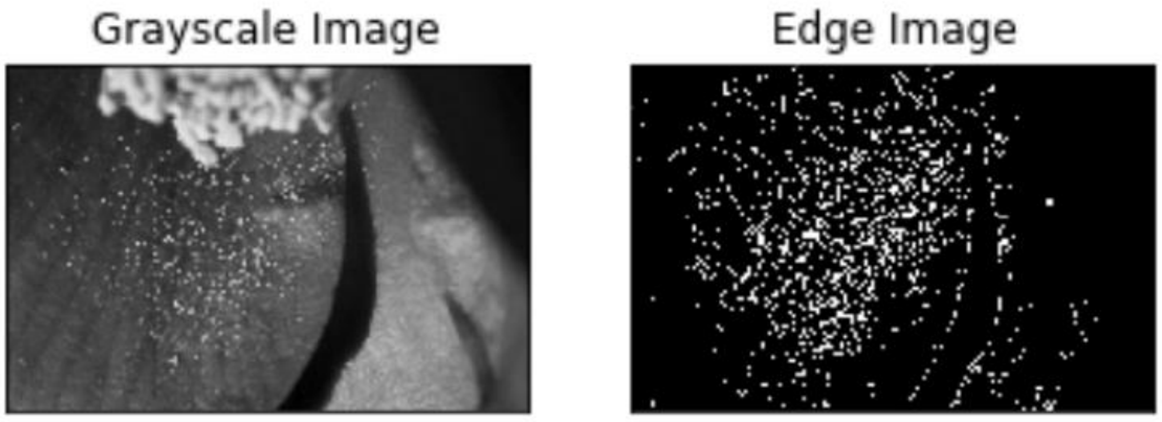
2. **EDGE DETECTION:**

**Output:**



```
median:  83.0
Lower threshold value:  55
Upper threshold value:  110
```

**Plot:**



Using edge detection we are able to find the spread of pollen grains. From this we are able to identify whether the area is suitable for plantation or not. As the spread of pollen increases, the possibility of the pollen used for the plantation decreases.

**FINAL NOTE :** We have also automated the above two detection processes by creating a Web application hosted on a local server. The Web application will be shown during the final evaluation phase.

**GROUP MEMBERS:**

| | |
|---|---|
| Chaitanya M Varier | (CB.EN.U4EIE18012) |
| Kaliswar Adhish R | (CB.EN.U4EIE18025) |
| S Sanjeev | (CB.EN.U4EIE18049) |
| Senthil Nathan M | (CB.EN.U4EIE18052) |
| Shivani S | (CB.EN.U4EIE18054) |