# Practical 1

## Aim : Write a program for performing image enhancement

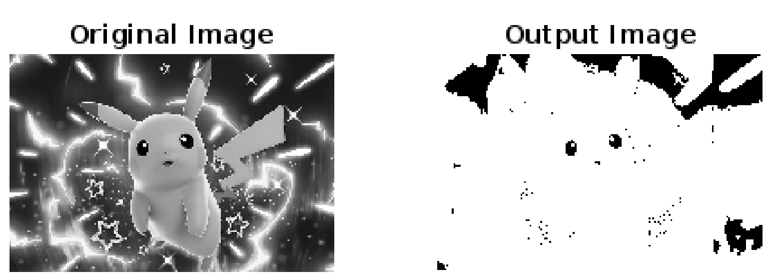
## 1. Thresholding

**Code:**

```
clc;
clear all;
close all;
img = imread('/MATLAB Drive/Pokemon.jpg');
imgg = rgb2gray(img);
imgSize = size(imgg);
row = imgSize(1,1);
col = imgSize(1,2);
m = input("ENTER THE THRESHOLDING POINT: ");
L = 256;
subplot(1,2,1);
imshow(imgg);
title('Original Image');
for i=1:row
    for j=1:col
        if(imgg(i,j)<=m)
            imgg(i,j)=0;
        else
            imgg(i,j)=L-1;
        end
    end
end
subplot(1,2,2);
imshow(imgg);
title('Output Image');
```

**Output:**

```
ENTER THE THRESHOLDING POINT:
34
```



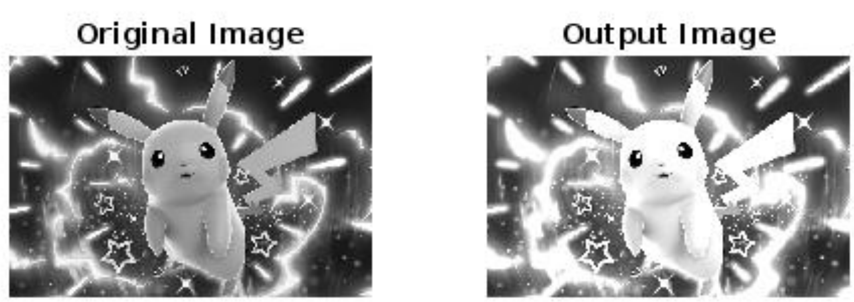
## 2. Contrast Adjustment

**Code:**

```
clc;
clear all;
close all;
img = imread('/MATLAB Drive/Pokemon.jpg');
imgg = rgb2gray(img);
row = size(imgg,1);
col = size(imgg,2);
L = 256;
imgcon = imgg;
r1 = input("Enter the first contrast Point: ");
r2 = input("Enter the Second contrast Point: ");
for i=1:row
    for j=1:col
        if(imgg(i,j)>=r1)
            imgcon(i,j)=imgg(i,j)*1.5;
        elseif(imgg(i,j) >= r1 && imgg(i,j)>=r2)
            imgcon(i,j)=imgg(i,j)*0.5;
        elseif(imgg(i,j)>=r2)
            imgcon(i,j)=imgg(i,j)*2;
        end
    end
end
subplot(1,2,1);
imshow(imgg);
title('Original Image');
subplot(1,2,2);
imshow(imgcon);
title('Output Image');
```

**Output:**

```
Enter the first contrast Point:
23
Enter the Second contrast Point:
45
```



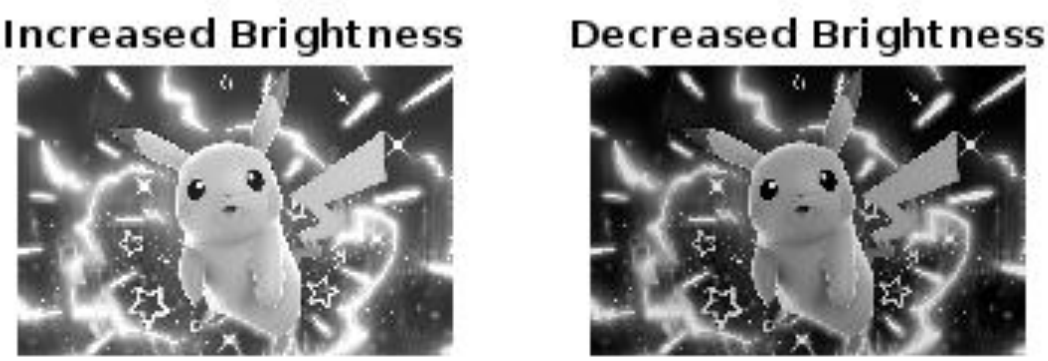
3. **Brightness Adjustment**

**Code:**

```
clc;
clear all;
close all;
img = imread("/MATLAB Drive/Pokemon.jpg")
imgg = rgb2gray(img);
bfact = input("Enter the brightness factor: ");
imgbi = double(imgg) + bfact;
imgbd = double(imgg) - bfact;
imshow(imgg);
title('Original Image');
subplot(1,2,1);
imshow(uint8(imgbi));
title('Increased Brightness');
subplot(1,2,2);
imshow(uint8(imgbd));
title('Decreased Brightness');
```

**Output:**

```
Enter the brightness factor:
24
```



## 4. Grayscale slicing with or without background

**Code:**

```
clc;
clear all;
close all;
img = imread("/MATLAB Drive/Pokemon.jpg");
imgg = rgb2gray(img);
row = size(imgg,1);
col = size(imgg,2);
L = 256;
imgcon = imgg;
choice = input("Enter  the choice for : with background (1) & without background (0): ")
r1 = input('Enter the first contrast point: ');
r2 = input('Enter the second contrast point: ');
for i=1:row
    for j=1:col
        if(imgg(i,j) < r1 || imgg(i,j) > r2)
            if choice == 1
                imgcon(i,j)=imgg(i,j);
            else
                imgcon(i,j)=0
            end
        else
            imgcon(i,j)=L-1;
        end
    end
end
subplot(1,2,1);
imshow(imgg);
title('Original Image');
subplot(1,2,2);
imshow(imgcon);
title('Output Image');
```

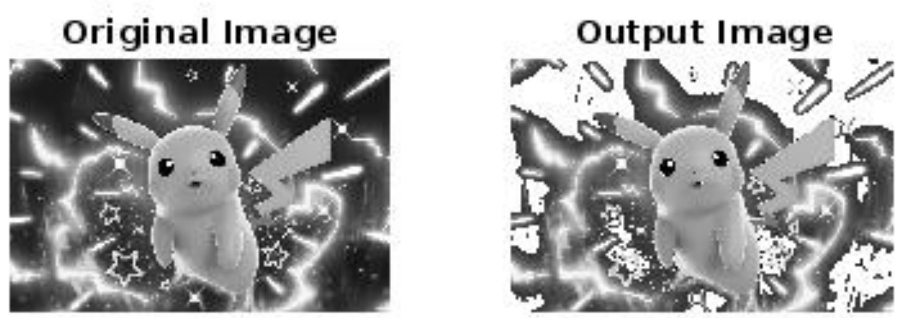
**Output:**

```
Enter  the choice for : with background (1) & without background (0):
1

choice =

     1

Enter the first contrast point:
25
Enter the second contrast point:
50
```



# Practical 2

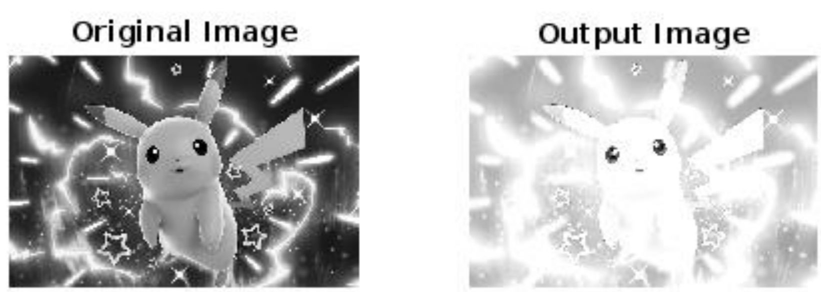
## Aim : Logarithmic Transformation

**Code:**

```
clc;
clear all;
close all;
img = imread('/MATLAB Drive/Pokemon.jpg');
imgg = rgb2gray(img);
row = size(imgg,1);
col = size(imgg,2);
constant = input("Enter the logarithmic factor: ");
imglog = constant * log(1+double(imgg));
subplot(1,2,1);
imshow(imgg);
title("Original Image");
subplot(1,2,2);
imshow(imglog);
title('Output Image');
```

**Output:**

```
Enter the logarithmic factor:
0.2
```

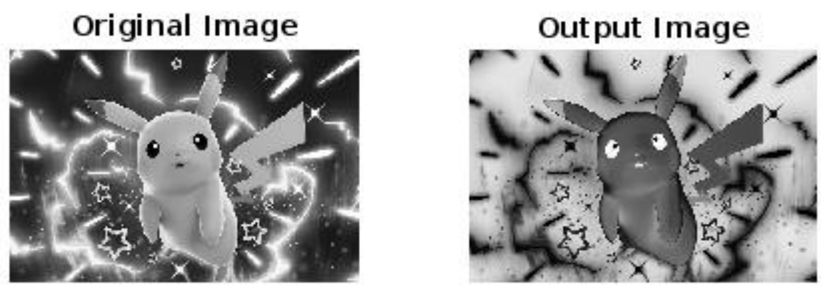


**b) Negative of the image:-**

**Code:**

```
clc;
clear all;
close all;
img = imread("/MATLAB Drive/Pokemon.jpg");
imgg = rgb2gray(img);
imgn = 255-double(imgg);
subplot(1,2,1);
imshow(imgg);
title('Original Image');
subplot(1,2,2);
imshow(uint8(imgn));
title("Output Image");
```

**Output:**



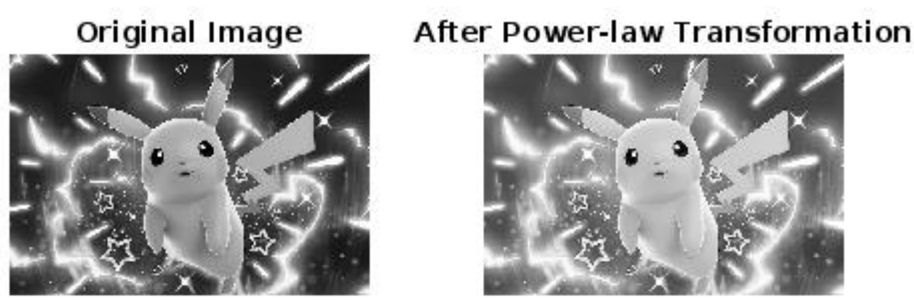
**c)Power Law Transformation**

**Code:**

```
clc;
clear all;
close all;
img = imread("/MATLAB Drive/Pokemon.jpg");
imgg = rgb2gray(img);
imgg = im2double(imgg);
[m,n] = size(imgg);
c = 1;
imgo = zeros(m,n);
y = input('Enter the value of Gamma: ');
for i=1:m
    for j = 1:n
        imgo(i,j) = c* (imgg(i,j)^y);
    end
end
subplot(1,2,1)
imshow(imgg);
title('Original Image');
subplot(1,2,2)
imshow(imgo);
title('After Power-law Transformation');
```

**Output:**

```
Enter the value of Gamma:
25
```



# Practical 3

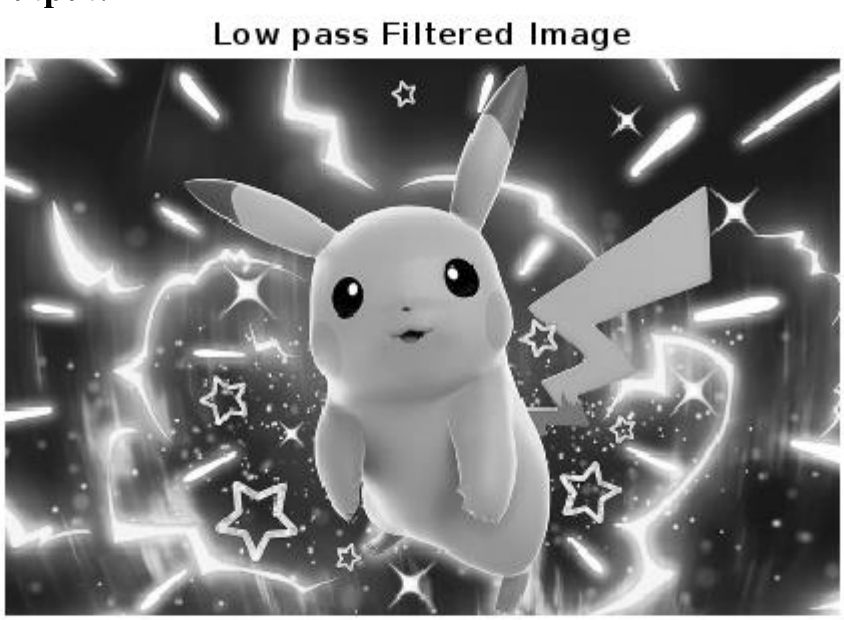
## Write a matlab code to demonstrate following filters

### a. Low Pass Filter

**Code:**

```
image = imread('/MATLAB Drive/images.jpg');
image = rgb2gray(image);
figure;
imshow(image);
title('Original Image');
% Create a Gaussian Low-Pass Filter
filterSize = 15;
sigma = 2;
h = fspecial('gaussian', filterSize, sigma);
filteredImage = imfilter(image, h, 'replicate');
figure;
imshow(filteredImage);
title('Low-Pass Filtered Image');
```

**Output:**

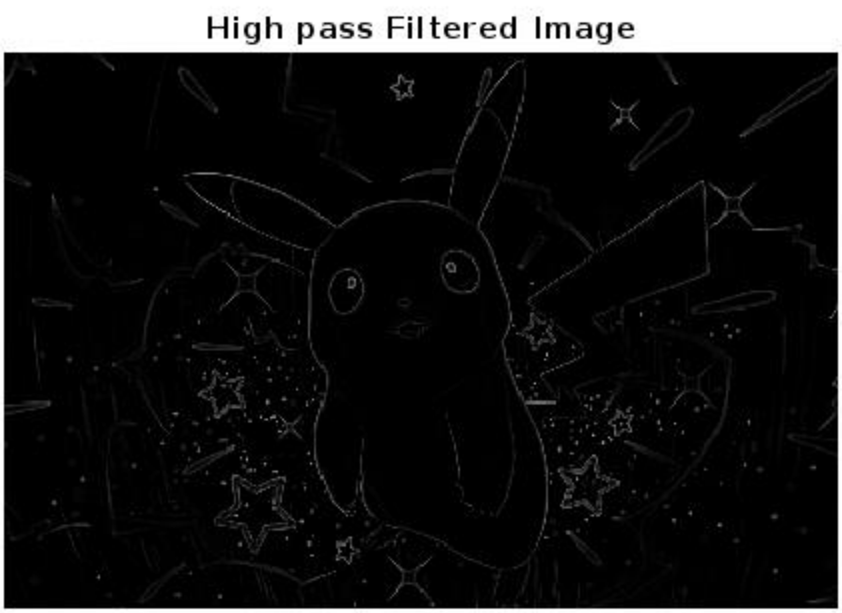


**b. High Pass FIlter**

**Code:**

```
image = imread('/MATLAB Drive/images.jpg');
image = rgb2gray(image);
figure;
imshow(image);
title('Original Image');
% Create a Gaussian Low-Pass Filter
filterSize = 15;
sigma = 2;
lowPassFilter = fspecial('gaussian', filterSize, sigma);
lowFrequencyImage = imfilter(image, lowPassFilter, 'replicate');
highFrequencyImage = image - lowFrequencyImage; m
figure;
imshow(highFrequencyImage);
title('High-Pass Filtered Image');
```

**Output:**



**c. Log (Laplacian of Gaussian)**

**Code:-**

```
image = imread("/MATLAB Drive/Pokemon.jpg");
image = rgb2gray(image);
figure;
imshow(image);
title("Original Image");
filterSize = 15;
sigma = 2;
logFilter = fspecial('log',filterSize,sigma);
filteredImage = imfilter(double(image),logFilter,'replicate');
filteredImage = mat2gray(filteredImage);
imshow(filteredImage);
title('Laplacian of Gaussian Filtered Image');
```

**Output:-**

Laplacian of Gaussian Filtered Image

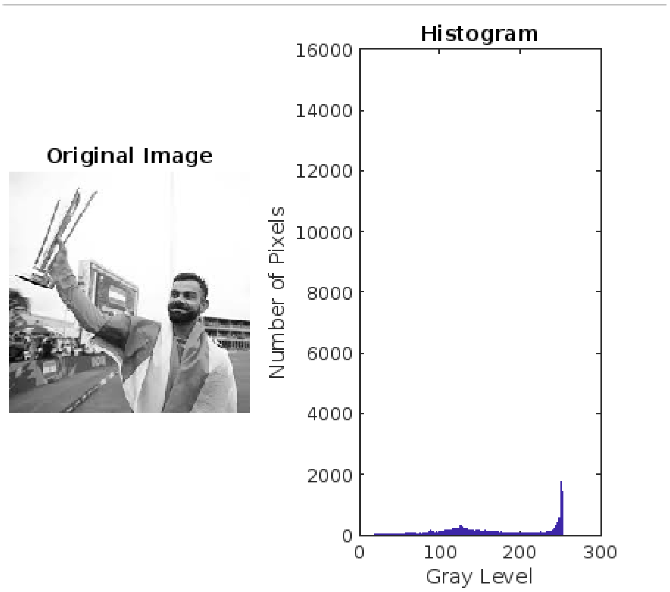
# Practical 4

## Aim: Write a matlab code to plot a histogram

**Code:**

```
clc;
clear all;
img = imread('/MATLAB Drive/images.jpg');
if size(img, 3) == 3
    img = rgb2gray(img);
end
imgo = zeros(1, 256);
[row, col] = size(img);
for i = 1:row
    for j = 1:col
        grayValue = img(i, j) + 1;
        imgo(grayValue) = imgo(grayValue) + 1;
    end
end
subplot(1, 2, 1);
imshow(img), title('Original Image');
subplot(1, 2, 2);
bar(0:255, imgo, 'histc'), title('Histogram');
xlabel('Gray Level');
ylabel('Number of Pixels');
```

**Output:**



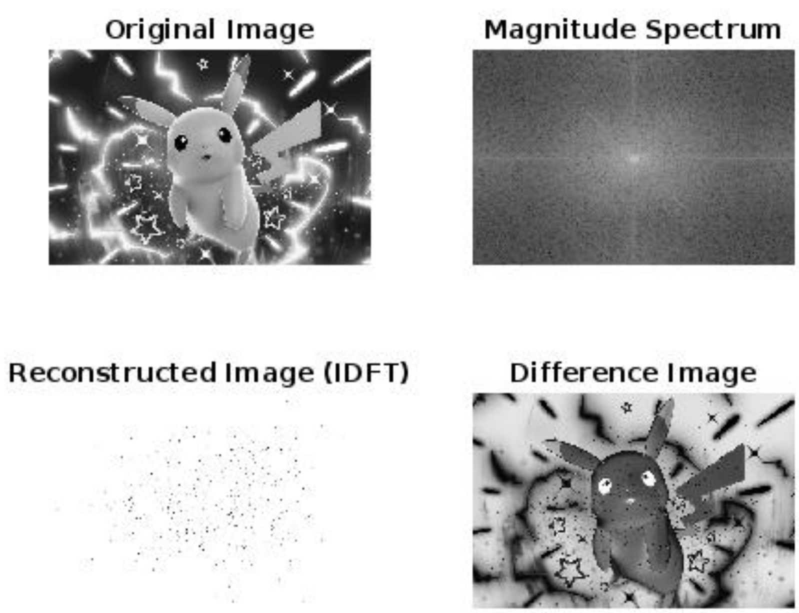
# Practical 5

## Aim : Write a matlab program to apply DFT on image and perform the inverse operation (DFT & IDFT).

**Code:**

```
image = imread('/MATLAB Drive/images (1).jpg');
if size(image, 3) == 3
    grayImage = rgb2gray(image);
else
    grayImage = image;
end
fftImage = fft2(double(grayImage));
fftShifted = fftshift(fftImage);
magnitudeSpectrum = log(1 + abs(fftShifted));
ifftImage = ifft2(fftImage);
reconstructedImage = uint8(abs(ifftImage));
figure;
subplot(2, 2, 1);
imshow(grayImage, []); title('Original Image');
subplot(2, 2, 2);
imshow(magnitudeSpectrum, []); title('Magnitude Spectrum');
subplot(2, 2, 3);
imshow(reconstructedImage, []); title('Reconstructed Image (IDFT)');
difference = imabsdiff(grayImage, reconstructedImage);
subplot(2, 2, 4);
imshow(difference, []);
title('Difference Image');
```

**Output:**



## Practical 6

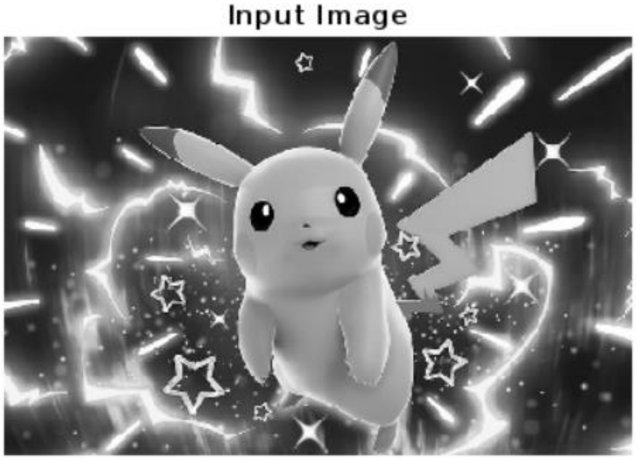
## Aim : Computation of mean, standard deviation, correlation, coefficient of given image.

**Code:-**

```
clc;
clear;
close all;
image = imread('/MATLAB Drive/images (1).jpg');
if size(image, 3) == 3
    image = rgb2gray(image);
end
image = double(image);
mean_value = mean(image(:));
std_dev = std(image(:));
corr_matrix = corrcoef(image);
disp(['Mean:', num2str(mean_value)]);
disp(['Standard Deviation:', num2str(std_dev)]);
disp('Corrleation Coefficient Matrix ');
disp(corr_matrix);
figure;
imshow(uint8(image));
title('Input Image');
```

**Output:-**

Input Image



```
Mean:75.079
Standard Deviation:47.2907
Corrleation Coefficient Matrix
  Columns 1 through 21

    1.0000    0.9997    0.9981    0.9940
    0.9997    1.0000    0.9991    0.9960
    0.9981    0.9991    1.0000    0.9986
    0.9940    0.9960    0.9986    1.0000
```

# Practical 7

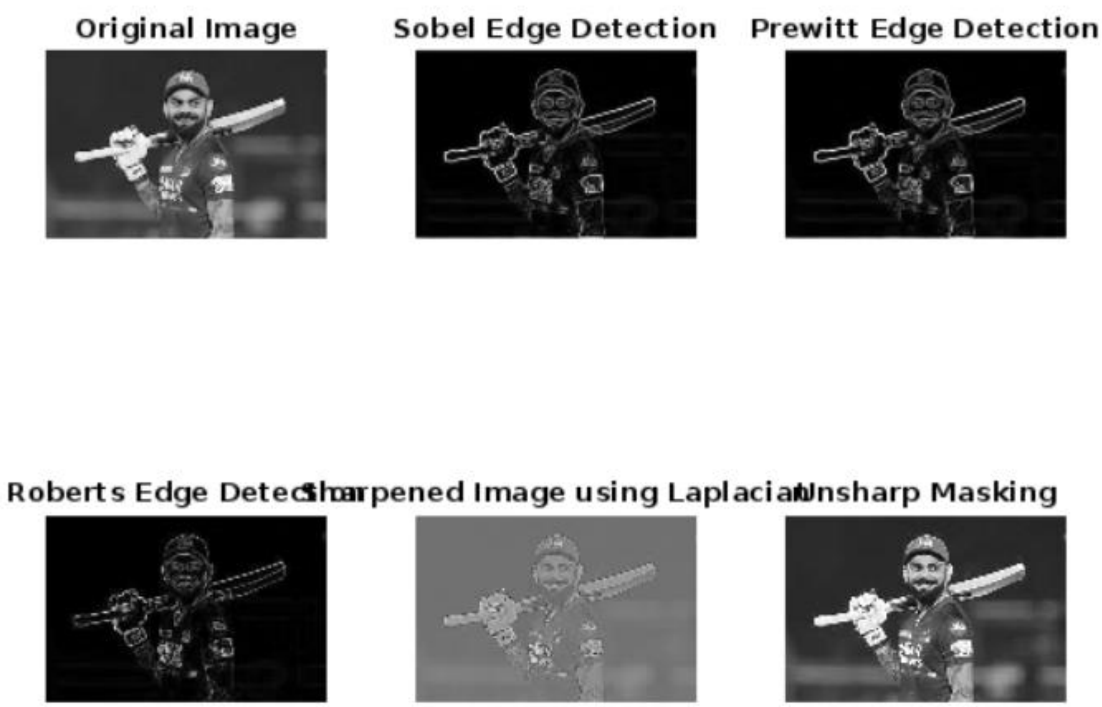
## Aim : Implementation of image sharpening filters and edge detection using gradient filters.

**Code:-**

```
clc;
close all;
clear;
img = imread('/MATLAB Drive/images (1).jpg');
imgg = rgb2gray(img);
figure;
subplot(2,3,1);
imshow(imgg);
title('Original Image');
sobel_x=fspecial('sobel');
sobel_y=sobel_x';
edge_sobel_x=imfilter(double(imgg ),sobel_x);
edge_sobel_y=imfilter(double(imgg ),sobel_y);
edge_sobel=sqrt(edge_sobel_x.^2+edge_sobel_y.^2);
subplot(2,3,2);
imshow(edge_sobel,[]);
title('Sobel Edge Detection');
prewitt_x=fspecial('prewitt');
prewitt_y=prewitt_x';
edge_prewitt_x=imfilter(double(imgg ),prewitt_x);
edge_prewitt_y=imfilter(double(imgg ),prewitt_y);
edge_prewitt=sqrt(edge_prewitt_x.^2+edge_prewitt_y.^2);
subplot(2,3,3);
imshow(edge_prewitt,[]);
title('Prewitt Edge Detection')
% Apply Roberts edge detection
roberts_x = [1 0; 0 -1];
roberts_y = [0 1; -1 0];
edge_roberts_x = imfilter(double(imgg ), roberts_x);
edge_roberts_y = imfilter(double(imgg ), roberts_y);
```

```
edge_roberts = sqrt(edge_roberts_x.^2 + edge_roberts_y.^2);
subplot(2,3,4);
imshow(edge_roberts, []);
title('Roberts Edge Detection');
% Image sharpening using Laplacian filter
laplacian_filter = fspecial('laplacian', 0.2);
sharpened_img = double(imgg) - imfilter(double(imgg), laplacian_filter);
subplot(2,3,5);
imshow(sharpened_img, []);
title('Sharpened Image using Laplacian');
% Image sharpening using unsharp masking
unsharp_mask = imgg  + 1.5 * (imgg  - imgaussfilt(imgg , 1));
subplot(2,3,6);
imshow(unsharp_mask, []);
title('Unsharp Masking');
```

**Output:-**



# Practical 8

## Aim: Implementation of image intensity slicing technique for image enhancement.

**Code:**

```
clc;
clear;
close all;
img = imread('/MATLAB Drive/Pokemon.jpg');
gray_img = mat2gray(img);
lower_thresh = 0.3;
upper_thresh = 0.7;
sliced_img = zeros(size(gray_img));
sliced_img((gray_img>=lower_thresh) & (gray_img <= upper_thresh)) = 1;
subplot(1,2,1);
imshow(gray_img);
title('Original Grayscale image');
subplot(1,2,2);
imshow(sliced_img);
title('Intensity sliced image');
```

**Output:**

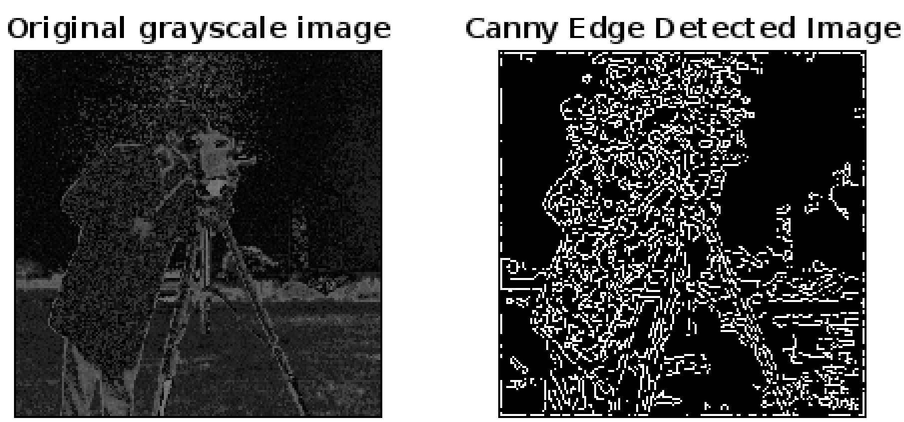
## Practical 9

## Aim: Implementation of canny edge detection algorithm.

**Code:**

```
clc;
clear;
close all;
img = imread('/MATLAB Drive/images (1).jpg');
gray_img = rgb2gray(img);
edges = edge(gray_img, 'Canny');
figure;
subplot(1, 2, 1);
imshow(gray_img);
title('Original Grayscale Image');
subplot(1, 2, 2);
imshow(edges);
title('Canny Edge Detected Image');
```

**Output:**



## Practical 10

## Aim: Write a matlab code for image compression by DCT and Huffman coding.

**Code:**

```
img = imread('/MATLAB Drive/images (1).jpg');
gray_img = rgb2gray(img);
block_size = 8;
[rows, cols] = size(gray_img);
dct_img = zeros(rows, cols);
for i = 1:block_size:rows
    for j = 1:block_size:cols
        block = double(gray_img(i:min(i+block_size-1, rows), j:min(j+block_size-1, cols)));
        dct_block = dct2(block);
        dct_img(i:min(i+block_size-1, rows), j:min(j+block_size-1, cols)) = dct_block;
    end
end
Q = 10;
quantized_dct = round(dct_img / Q);
rle_data = runLengthEncode(quantized_dct);
symbols = rle_data(:, 1); % RLE symbols
[unique_symbols, ~, idx] = unique(symbols);  % Get unique symbols and their indices
frequencies = histc(symbols, unique_symbols);  % Count the frequency of each unique symbol
probabilities = frequencies / sum(frequencies);  % Normalize to get probabilities
[~, sorted_idx] = sort(probabilities);  % Sort based on probabilities
unique_symbols = unique_symbols(sorted_idx);  % Reorder symbols
probabilities = probabilities(sorted_idx);
[dict, avg_len] = huffmancoding(unique_symbols, probabilities);
symbol_to_code = containers.Map(unique_symbols, dict);
encoded_data = huffmanencode(symbols, symbol_to_code);
save('compressed_data.mat', 'encoded_data', 'dict', 'Q');
load('compressed_data.mat');
decoded_symbols = huffmandecode(encoded_data, symbol_to_code);
disp('Decoded Symbols (first 10 elements):');
disp(decoded_symbols(1:min(10, end)));
rle_data_reconstructed = reconstructRLE(decoded_symbols);
disp('Reconstructed RLE (first 10 elements):');
disp(rle_data_reconstructed(1:min(10, end), :));
decoded_dct = runLengthDecode(rle_data_reconstructed, rows, cols) * Q;
```

```
decompressed_img = zeros(rows, cols);
for i = 1:block_size:rows
    for j = 1:block_size:cols
        block = decoded_dct(i:min(i+block_size-1, rows), j:min(j+block_size-1, cols));
        decompressed_img(i:min(i+block_size-1, rows), j:min(j+block_size-1, cols)) = idct2(block);
    end
end
subplot(1,2,1), imshow(gray_img), title('Original Image');
subplot(1,2,2), imshow(uint8(decompressed_img)), title('Decompressed Image');
function rle = runLengthEncode(img)
    rle = [];
    i = 1;
    while i <= numel(img)
        symbol = img(i);
        count = 1;
        while i + count <= numel(img) && img(i + count) == symbol
            count = count + 1;
        end
        rle = [rle; symbol, count];  % Store the symbol and its run length
        i = i + count;
    end
end
function decoded = runLengthDecode(rle, rows, cols)
    if size(rle, 2) ~= 2
        error('The RLE data must have two columns: symbol and run length.');
    end
    decoded = zeros(1, rows * cols);  % Create a vector for the decoded data
    idx = 1;
    for i = 1:size(rle, 1)
        value = rle(i, 1);  % The symbol
        count = rle(i, 2);  % The run length
        decoded(idx:idx + count - 1) = value;  % Assign the value for the run length
        idx = idx + count;  % Move to the next position
    end
    decoded = reshape(decoded, rows, cols);  % Reshape back into a 2D matrix
end
function rle = reconstructRLE(symbols)
    rle = [];
    i = 1;
    while i <= numel(symbols)
```

```
        symbol = symbols(i);
        count = 1;
        while i + count <= numel(symbols) && symbols(i + count) == symbol
            count = count + 1;
        end
        rle = [rle; symbol, count];  % Store the symbol and its run length
        i = i + count;
    end
end
function [dict, avg_len] = huffmancoding(symbols, probabilities)
    n = length(symbols);
    heap = cell(n, 1);
    for i = 1:n
        heap{i} = struct('symbol', symbols(i), 'probability', probabilities(i), 'left', [], 'right', []);
    end
    while length(heap) > 1
        % Sort heap based on probability using cellfun
        probabilities_heap = cellfun(@(x) x.probability, heap);
        [~, sorted_idx] = sort(probabilities_heap);  % Sort based on probabilities
        heap = heap(sorted_idx);  % Reorder heap
        left = heap{1};
        right = heap{2};
        heap(1:2) = [];
        new_node = struct('symbol', [], 'probability', left.probability + right.probability, 'left', left, 'right',
right);
        heap = [heap; new_node];
    end
    dict = cell(1, n);
    avg_len = 0;
    [dict, avg_len] = generateHuffmanCodes(heap{1}, '', dict, avg_len, symbols);
end
function [dict, avg_len] = generateHuffmanCodes(node, code, dict, avg_len, symbols)
    if isempty(node.left) && isempty(node.right)
        % Leaf node, assign code
        dict{find([symbols == node.symbol], 1)} = code;
        avg_len = avg_len + length(code);
    else
        % Internal node, traverse further
        dict = generateHuffmanCodes(node.left, [code '0'], dict, avg_len, symbols);
        dict = generateHuffmanCodes(node.right, [code '1'], dict, avg_len, symbols);
```

```
    end
end
function encoded_data = huffmanencode(symbols, symbol_to_code)
    encoded_data = [];
    for i = 1:length(symbols)
        encoded_data = [encoded_data, symbol_to_code(symbols(i))];
    end
end
function decoded_symbols = huffmandecode(encoded_data, symbol_to_code)
    % Reverse the map from symbols to Huffman codes
    code_to_symbol = containers.Map(values(symbol_to_code), keys(symbol_to_code));
    decoded_symbols = [];
    current_code = '';
    for i = 1:length(encoded_data)
        current_code = [current_code, encoded_data(i)];
        if isKey(code_to_symbol, current_code)
            decoded_symbols = [decoded_symbols, code_to_symbol(current_code)];
            current_code = '';
        end
    end
end
```

**Output:**

```
Decoded Symbols (first 10 elements):
    35     0    36     0    36     0    36     0    37     0

Reconstructed RLE (first 10 elements):
    35     1
     0     1
    36     1
     0     1
    36     1
     0     1
    36     1
     0     1
    37     1
     0     1
```