PRACTICAL - 1

Aim: Implement code to
i. read an image, observe image as function along with its attributes,create a sample image and store it for grayscale images
ii. read an image, observe image as function along with its attributes, split image in different color image planes, merge splitted image plane and observe effect on merged image by changing red, green and blue image plane values
iii. convert color image to grayscale and HSV image and observe HSV values
iv. create images of different shapes
v. resize image with different types of interpolation

Program:

i.

```
% Read an image and observe its attributes
imagePath = 'saulgoodman.png';
originalImage = imread(imagePath);
disp(['Original Image Size: ', num2str(size(originalImage))]);
disp(['Image Data Type: ', class(originalImage)]);


% Create a sample grayscale image
sampleImage = randi([0, 255], [300, 300], 'uint8');
imwrite(sampleImage, 'sample_grayscale_image.jpg');
```

Output:

```
Original Image Size: 393  439    3
Image Data Type: uint8
```

Program:

ii.

```
% Read a color image and observe its attributes
colorImagePath = 'saulgoodman.png';
colorImage = imread(colorImagePath);
disp(['Color Image Size: ', num2str(size(colorImage))]);
disp(['Color Image Data Type: ', class(colorImage)]);

% Split image into different color planes (R, G, B)
r = colorImage(:, :, 1);
g = colorImage(:, :, 2);
b = colorImage(:, :, 3);

% Merge the color planes
mergedImage = cat(3, r, g, b);

% Observe the effect by changing red, green, and blue image plane values
% For example, change red plane values:
r = r * 1.5;
% Merge the modified planes
modifiedImage = cat(3, r, g, b);
```

Output:

```
Color Image Size: 393  439    3
Color Image Data Type: uint8
```

Program:

iii.

% Convert color image to grayscale

grayImage = rgb2gray(colorImage);

% Convert color image to HSV

hsvImage = rgb2hsv(colorImage);

% Observe HSV values

disp(['Hue Values (min, max): ', num2str([min(hsvImage(:, :, 1)), max(hsvImage(:, :, 1))])]);

disp(['Saturation Values (min, max): ', num2str([min(hsvImage(:, :, 2)), max(hsvImage(:, :, 2))])]);

disp(['Value (Brightness) Values (min, max): ', num2str([min(hsvImage(:, :, 3)), max(hsvImage(:, :, 3))])]);

Output:

```
Hue Values (min, max): 0          0          0          0          0
Saturation Values (min, max): 0          0          0          0          0
Value (Brightness) Values (min, max): 0.054902    0.054902    0.054902    0.05490
```

Program:

iv.

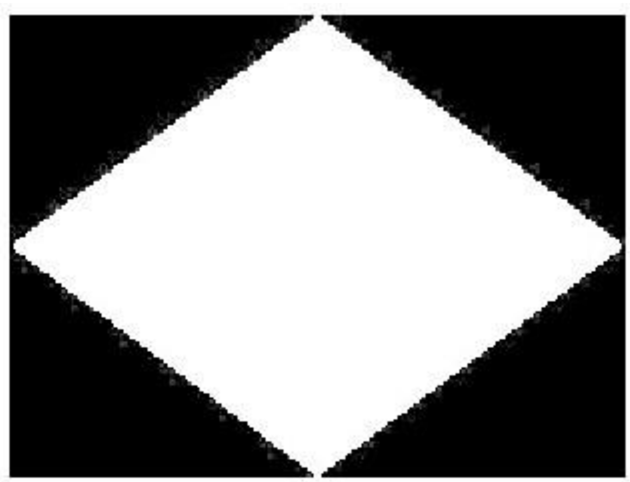
% Create images of different shapes

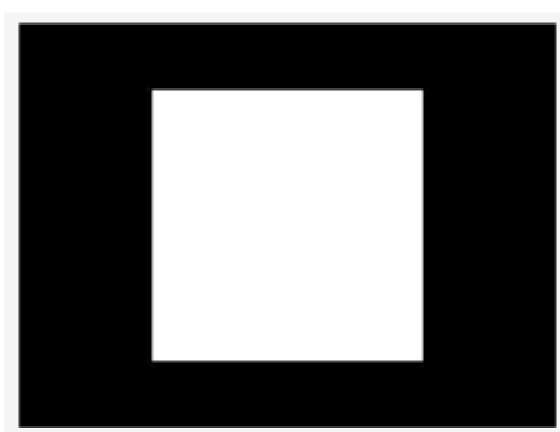
rectangleImage = zeros(300, 400);

rectangleImage(50:250, 100:300) = 255;

```
diamondImage = zeros(300, 400);
[x, y] = meshgrid(1:400, 1:300);
diamondMask = abs((x - 200) / 200) + abs((y - 150) / 150) <= 1;
diamondImage(diamondMask) = 255;
% Display or save the created images
imwrite(rectangleImage, 'rectangle_image.jpg');
imwrite(diamondImage, 'diamond_image.jpg');
```

Output:





Program:

v.

```
% Read an image to resize
imagePathToResize = 'saulgoodman.png';
imageToResize = imread(imagePathToResize);

% Resize image with different types of interpolation
resizedNearest = imresize(imageToResize, 0.5, 'nearest');
resizedBilinear = imresize(imageToResize, 0.5, 'bilinear');
resizedBicubic = imresize(imageToResize, 0.5, 'bicubic');

% Save resized images
imwrite(resizedNearest, 'resized_nearest.jpg');
imwrite(resizedBilinear, 'resized_bilinear.jpg');
imwrite(resizedBicubic, 'resized_bicubic.jpg');
```

Output:

Name	Size	Class	Value
resized_bicubic	197×220×3	uint8	197×220×3 uint8

Preview:

Name	Size	Class	Value
resized_nearest	197×220×3	uint8	197×220×3 uint8

Preview:

Name	Size	Class	Value
resized_bilinear	197×220×3	uint8	197×220×3 uint8

Preview:

Conclusion: In this practical, I performed image attribute observation, color channel manipulation, color space conversion, shape generation, and image resizing with different interpolation methods using OpenCV.

PRACTICAL - 2

Aim: Implement code to
i. obtain negative of given images
ii. perform contrast stretching on given images
iii. perform thresholding on given images
iv. perform Otsu's thresholding on given images
v. perform log transformation on given images
vi. perform gamma correction on given images

Program:

i.

```
% Read the original image
originalImage = imread('pexels-brett-sayles-6424244.jpg');
% Obtain negative of the image
negativeImage = 255 - originalImage;
% Display or save the negative image
imshow(negativeImage);
imwrite(negativeImage, 'negative_image.jpg');
```

Output:

Program:

ii.

```
% Read the original image
originalImage = imread('pexels-brett-sayles-6424244.jpg');

% Convert the original image to double data type
originalImage = double(originalImage);

% Perform contrast stretching
minIntensity = min(originalImage(:));
maxIntensity = max(originalImage(:));
stretchedImage = uint8((originalImage - minIntensity) * (255 / (maxIntensity - minIntensity)));

% Display or save the contrast-stretched image
imshow(stretchedImage);
imwrite(stretchedImage, 'contrast_stretched_image.jpg');
```

Output:

Program:

iii.

```
% Read the grayscale image
grayImage = imread('grayscale_image.jpg');

% Set a threshold value
threshold = 128;

% Perform thresholding
thresholdedImage = grayImage > threshold;

% Display or save the thresholded image
imshow(thresholdedImage);
imwrite(thresholdedImage, 'thresholded_image.jpg');
```

Output:

Program:

iv.

```
% Read the grayscale image
grayImage = imread('grayscale_image.jpg');

% Perform Otsu's thresholding
threshold = graythresh(grayImage);
otsuThresholdedImage = imbinarize(grayImage, threshold);

% Display or save the Otsu's thresholded image
imshow(otsuThresholdedImage);
imwrite(otsuThresholdedImage, 'otsu_thresholded_image.jpg');
```

Output:

Program:

v.

```
% Read the grayscale image
grayImage = imread('grayscale_image.jpg');

% Perform log transformation
c = 1; % Scaling factor
logTransformedImage = c * log(1 + double(grayImage));

% Scale the log-transformed image to [0, 255]
logTransformedImage = uint8((logTransformedImage /
max(logTransformedImage(:))) * 255);

% Display or save the log-transformed image
imshow(logTransformedImage);
imwrite(logTransformedImage, 'log_transformed_image.jpg');
```

Output:

Program:

vi.

```
% Read the grayscale image
grayImage = imread('grayscale_image.jpg');
% Set gamma value (adjust as needed)
gamma = 1.5;
% Perform gamma correction
gammaCorrectedImage = imadjust(grayImage, [], [], gamma);
% Display or save the gamma-corrected image
imshow(gammaCorrectedImage);
imwrite(gammaCorrectedImage, 'gamma_corrected_image.jpg');
```

Output:

Conclusion: In this practical, I implemented various image manipulation techniques, including obtaining negative images, enhancing contrast through stretching, segmenting objects using thresholding, applying automatic thresholding with Otsu's method, adjusting contrast with logarithmic transformation, and fine-tuning brightness and contrast using gamma correction.

PRACTICAL - 3

Aim: Implement code to
i. apply intensity slicing on given images
ii. apply bit plane slicing on given images and observe information on different bit planes
iii. calculate histograms on different contrast images.
iv. apply normal histogram equalization and CLAHE histogram equalization on given images.
v. apply histogram matching on given images

Program:

i.

```
% Read the original grayscale image
originalImage = imread('pexels-brett-sayles-6424244.jpg');

% Define lower and upper thresholds for intensity slicing
lowerThreshold = 100;
upperThreshold = 200;

% Perform intensity slicing
outputImage = originalImage;
outputImage(originalImage >= lowerThreshold & originalImage <=
upperThreshold) = 255;

% Display or save the intensity-sliced image
imshow(outputImage);
imwrite(outputImage, 'intensity_sliced_image.jpg');
```

Output:

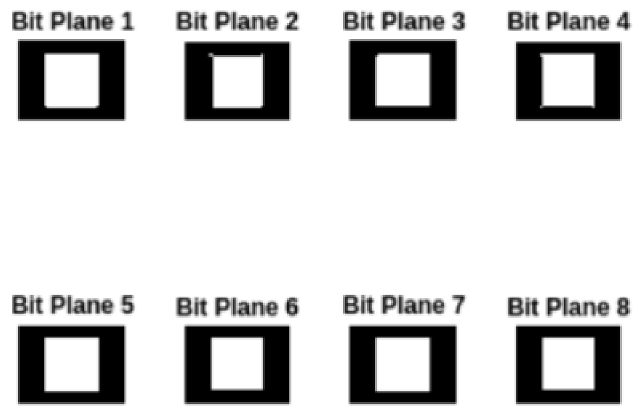
Program:

ii.

```
% Read the original grayscale image
originalImage = imread('rectangle_image.jpg');
% Convert the image to binary representation
binaryImage = imbinarize(originalImage);
% Perform bit plane slicing
bitPlanes = zeros(size(binaryImage, 1), size(binaryImage, 2), 8);
for i = 1:8
    bitPlanes(:, :, i) = bitget(originalImage, i);
end

% Display or save bit planes
for i = 1:8
    subplot(2, 4, i);
    imshow(bitPlanes(:, :, i) * 255);
```

```
    title(['Bit Plane ', num2str(i)]);
end
```

Output:



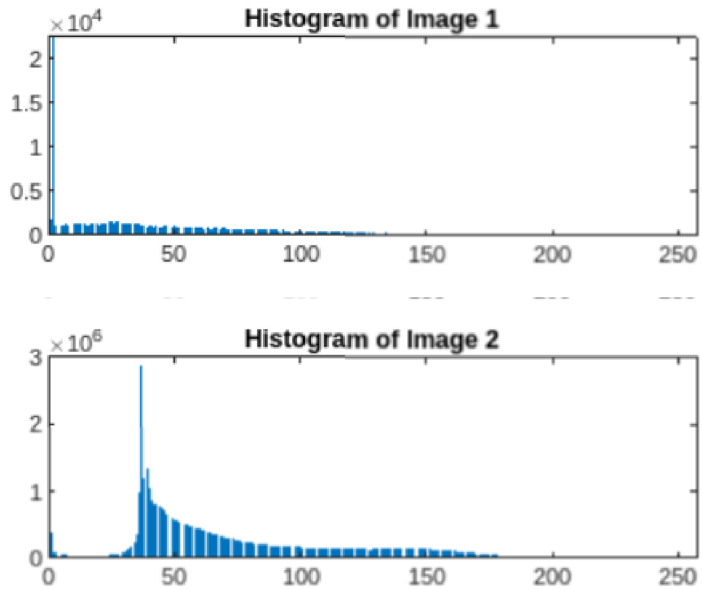
Program:

iii.

```
% Read different contrast images
image1 = imread('download (1).jpeg');
image2 = imread('pexels-brett-sayles-6424244.jpg');
% Calculate histograms
histogramImage1 = imhist(image1);
histogramImage2 = imhist(image2);
% Display histograms
figure;
subplot(2, 1, 1);
bar(histogramImage1);
title('Histogram of Image 1');
```

```
subplot(2, 1, 2);
bar(histogramImage2);
title('Histogram of Image 2');
```

Output:



Program:

iv.

```
% Read the original image
originalImage = imread('jesse.png');

% Convert the image to grayscale if it's a color image
if size(originalImage, 3) == 3
    originalImage = rgb2gray(originalImage);
end
```

```
% Perform normal histogram equalization
histeqImage = histeq(originalImage);

% Perform CLAHE (Contrast Limited Adaptive Histogram Equalization)
claheImage = adapthisteq(originalImage, 'ClipLimit', 0.02, 'Distribution',
'rayleigh');

% Display or save the processed images
imshowpair(histeqImage, claheImage, 'montage');
imwrite(histeqImage, 'histeq_image.jpg');
imwrite(claheImage, 'clahe_image.jpg');
```

Output:

Program:

v.

```
% Read the reference image and the image to be matched
referenceImage = imread('download (1).jpeg');
imageToMatch = imread('jesse.png');
```

```
% Perform histogram matching
matchedImage = imhistmatch(imageToMatch, referenceImage);

% Display or save the matched image
imshowpair(referenceImage, matchedImage, 'montage');
imwrite(matchedImage, 'matched_image.jpg');
```

Output:

Conclusion: In this practical, I learnt image enhancement techniques, such as intensity slicing, bit plane slicing, histogram analysis, histogram equalization (standard and CLAHE), and histogram matching.

PRACTICAL - 4

Aim: Implement code to
i. perform cross-correlation and convolution on images in spatial domain
ii. apply smoothing spatial filters of different kernel sizes on images
iii. apply sharpening spatial filters of different kernel sizes on images
iv. apply non-linear spatial filters of different kernel sizes on images
v. analyze noise removal with different smoothing spatial filters and non-linear filters
vi. perform unsharp masking and high-boost filtering on different images

Program:

i.

```
% Read the original grayscale image and kernel
originalImage = imread('pexels-brett-sayles-6424244.jpg');
grayImage = rgb2gray(originalImage);
kernel = fspecial('gaussian', [3 3], 1); % Example Gaussian kernel

% Perform cross-correlation
crossCorrelationResult = xcorr2(double(grayImage), kernel);

% Perform convolution
convolutionResult = conv2(double(grayImage), kernel, 'same');

% Display or save the results
imshow(crossCorrelationResult, []);
imwrite(uint8(convolutionResult), 'convolution_result.jpg');
```

Output:

Program:

ii.

```
% Apply smoothing spatial filters with different kernel sizes
smoothedImages = cell(1, 3);
kernelSizes = [3, 5, 7];
for i = 1:length(kernelSizes)
    kernel = fspecial('average', kernelSizes(i));
    smoothedImages{i} = conv2(double(grayImage), kernel, 'same');
end

% Display or save the smoothed images
for i = 1:length(kernelSizes)
    imshow(uint8(smoothedImages{i}), []);
    imwrite(uint8(smoothedImages{i}), ['smoothed_image_kernel_',
num2str(kernelSizes(i)), '.jpg']);
end
```

Output:

Program:

iii.

```
% Apply sharpening spatial filters with different kernel sizes
sharpenedImages = cell(1, 3);
kernelSizes = [3, 5, 7];
for i = 1:length(kernelSizes)
    % Create the kernel for sharpening
    kernel = -fspecial('average', kernelSizes(i)) + 2 * fspecial('gaussian',
kernelSizes(i), 1);
    sharpenedImages{i} = conv2(double(grayImage), kernel, 'same');
end
% Display or save the sharpened images
for i = 1:length(kernelSizes)
    imshow(uint8(sharpenedImages{i}), []);
    imwrite(uint8(sharpenedImages{i}), ['sharpened_image_kernel_',
num2str(kernelSizes(i)), '.jpg']);
end
```

Output:

Program:

iv.

```
% Apply non-linear spatial filters with different kernel sizes
filteredImages = cell(1, 3);
for i = 1:length(kernelSizes)
    filteredImages{i} = medfilt2(grayImage, [kernelSizes(i), kernelSizes(i)]);
end

% Display or save the filtered images
for i = 1:length(kernelSizes)
    imshow(uint8(filteredImages{i}), []);
    imwrite(uint8(filteredImages{i}), ['filtered_image_kernel_',
num2str(kernelSizes(i)), '.jpg']);
end
```

Output:

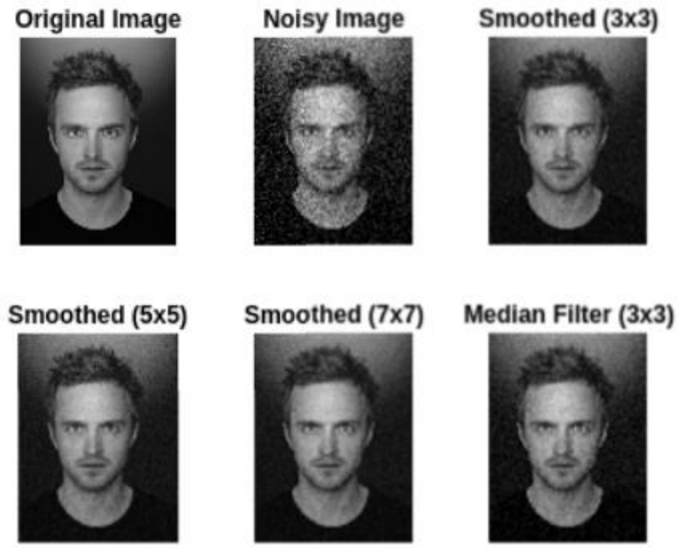
Program:

v.

```
% Read the original image
originalImage = imread('jesse.png');

% Convert the original image to grayscale
grayOriginalImage = rgb2gray(originalImage);

% Add noise to the grayscale image
noisyImage = imnoise(grayOriginalImage, 'gaussian', 0, 0.01); % Adding Gaussian
noise

% Apply smoothing spatial filters and non-linear filters to noisy image
smoothedImages = cell(1, length(kernelSizes));
filteredImages = cell(1, length(kernelSizes));

for i = 1:length(kernelSizes)
```

```
    % Apply smoothing spatial filter (Gaussian blur)
    gaussianKernel = fspecial('gaussian', kernelSizes(i), 1);
    smoothedImages{i} = imfilter(noisyImage, gaussianKernel, 'replicate');

    % Apply non-linear filter (median filter)
    filteredImages{i} = medfilt2(noisyImage, [kernelSizes(i), kernelSizes(i)]);
end

% Display the results
figure;
subplot(2, 3, 1), imshow(grayOriginalImage), title('Original Image');
subplot(2, 3, 2), imshow(noisyImage), title('Noisy Image');
subplot(2, 3, 3), imshow(smoothedImages{1}), title('Smoothed (3x3)');
subplot(2, 3, 4), imshow(smoothedImages{2}), title('Smoothed (5x5)');
subplot(2, 3, 5), imshow(smoothedImages{3}), title('Smoothed (7x7)');
subplot(2, 3, 6), imshow(filteredImages{1}), title('Median Filter (3x3)');

% Save the processed images
imwrite(smoothedImages{1}, 'smoothed_3x3.jpg');
imwrite(smoothedImages{2}, 'smoothed_5x5.jpg');
imwrite(smoothedImages{3}, 'smoothed_7x7.jpg');
imwrite(filteredImages{1}, 'median_filter_3x3.jpg');
```

Output:



Program:

vi.

```
% Read the original grayscale image
originalImage = imread('jesse.png');

% Apply unsharp masking for sharpening
smoothedImage = imgaussfilt(originalImage, 2); % Example Gaussian filter
unsharpMask = double(originalImage) - double(smoothedImage);
unsharpMaskedImage = double(originalImage) + 1.5 * unsharpMask;

% Apply high-boost filtering for sharpening
highBoostMaskedImage = double(originalImage) + 2 * unsharpMask;

% Convert the results back to uint8 for display
unsharpMaskedImage = uint8(unsharpMaskedImage);
```

```
highBoostMaskedImage = uint8(highBoostMaskedImage);

% Display or save the sharpened images
imshow(unsharpMaskedImage);
imwrite(unsharpMaskedImage, 'unsharp_masked_image.jpg');
imshow(highBoostMaskedImage);
imwrite(highBoostMaskedImage, 'high_boost_image.jpg');
```

Output:

Conclusion: In this practical, I leant about spatial domain operations, including cross-correlation, convolution, smoothing, sharpening, non-linear filtering, noise removal, unsharp masking, and high-boost filtering, demonstrating a wide range of image processing techniques.

PRACTICAL - 5

Aim: Implement code to
i. convert images from space domain to frequency domain and observe their spectrum
ii. observe aliasing in down-sampled images and apply anti-aliasing filter to reduce effect of aliasing
iii. apply frequency domain low-pass filters of different types and cut-off frequencies on images and observe their effects
iv. apply frequency domain high-pass filters of different types and cut-off frequencies on images and observe their effects
v. add periodic noise on images in frequency domain, apply notch filters to remove noise and restore original image

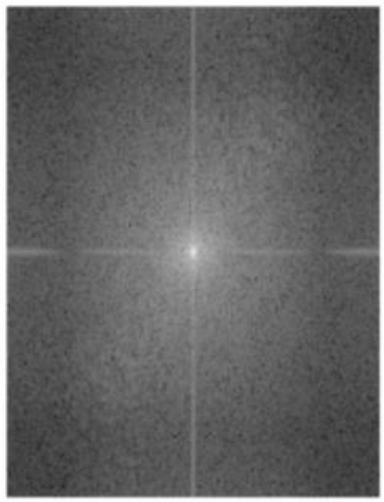
Program:

i.

```
% Read the original image
originalImage = imread('jesse.png');


% Convert image to grayscale if it's a color image
if size(originalImage, 3) == 3
    originalImage = rgb2gray(originalImage);
end


% Perform Fourier Transform to convert to frequency domain
frequencyDomainImage = fftshift(fft2(originalImage));


% Calculate magnitude spectrum for visualization
magnitudeSpectrum = abs(frequencyDomainImage);
```

```
% Display the magnitude spectrum
imshow(log(1 + magnitudeSpectrum), []);
```

Output:



Program:

ii.

```
% Downsample the original image (introducing aliasing)
downsampledImage = imresize(originalImage, 0.5);

% Apply anti-aliasing filter (Gaussian filter in this case)
filteredImage = imgaussfilt(downsampledImage, 1);

% Display the down-sampled and filtered images
figure;
subplot(1, 2, 1), imshow(downsampledImage), title('Down-sampled Image');
subplot(1, 2, 2), imshow(filteredImage), title('Filtered Image');
```

Output:

Program:

iii.

```
% Apply frequency domain low-pass filters (e.g., Gaussian filter)
cutoffFrequency = 50;
lowpassFilteredImage = frequencyDomainImage;
lowpassFilteredImage(abs(lowpassFilteredImage) > cutoffFrequency) = 0;
% Inverse Fourier Transform to obtain filtered image
filteredImage = abs(ifft2(ifftshift(lowpassFilteredImage)));
% Display the filtered image
imshow(filteredImage, []);
```

Output:

Program:

iv.

```
% Apply frequency domain high-pass filters (e.g., Ideal high-pass filter)
cutoffFrequency = 50;
highpassFilteredImage = frequencyDomainImage;
highpassFilteredImage(abs(highpassFilteredImage) < cutoffFrequency) = 0;

% Inverse Fourier Transform to obtain filtered image
filteredImage = abs(ifft2(ifftshift(highpassFilteredImage)));

% Display the filtered image
imshow(filteredImage, []);
```

Output:

Program:

v.

```
% Add periodic noise in frequency domain
noiseAmplitude = 20;
noiseFrequencyX = 50;
noiseFrequencyY = 30;
noisyFrequencyDomainImage = frequencyDomainImage;
noisyFrequencyDomainImage(noiseFrequencyY, noiseFrequencyX) = ...
    noisyFrequencyDomainImage(noiseFrequencyY, noiseFrequencyX) + noiseAmplitude;


% Apply notch filters to remove noise
notchFilterRadius = 10;
notchFilter = ones(size(noisyFrequencyDomainImage));
notchFilter(noiseFrequencyY-notchFilterRadius:noiseFrequencyY+notchFilterRadius, ...
    noiseFrequencyX-notchFilterRadius:noiseFrequencyX+notchFilterRadius) = 0;
denoisedFrequencyDomainImage = noisyFrequencyDomainImage .* notchFilter;


% Inverse Fourier Transform to obtain denoised image
denoisedImage = abs(ifft2(ifftshift(denoisedFrequencyDomainImage)));
% Display the noisy and denoised images
figure;
subplot(1, 2, 1), imshow(abs(ifft2(ifftshift(noisyFrequencyDomainImage))), []), title('Noisy Image');
subplot(1, 2, 2), imshow(denoisedImage, []), title('Denoised Image');
```

Output:

Conclusion: In this practical, I leant about aliasing and removing noise from images.

PRACTICAL - 6

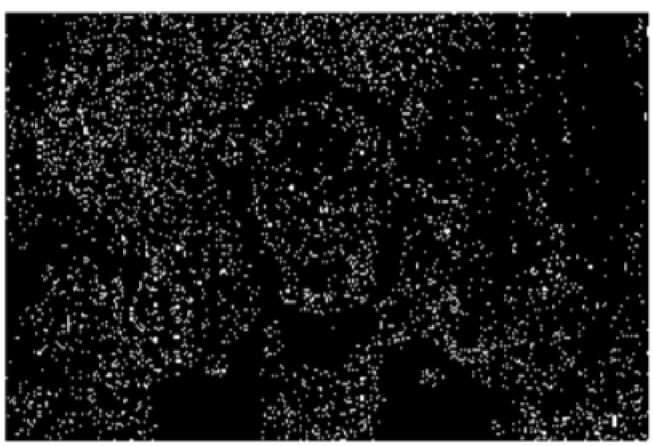
Aim: Implement code to detect
i. edges in different images using laplacian operator
ii. edges in different images using sobel operator
iii. edges in different images using prewitt operator
iv. edges in different images using canny operator
v. lines in different images using Hough Transform

Program:

i.

```
% Apply Laplacian operator for edge detection
laplacianEdges = edge(grayImage, 'log');

% Display or save the edges detected using Laplacian operator
imshow(laplacianEdges);
imwrite(laplacianEdges, 'laplacian_edges.jpg');
```

Output:

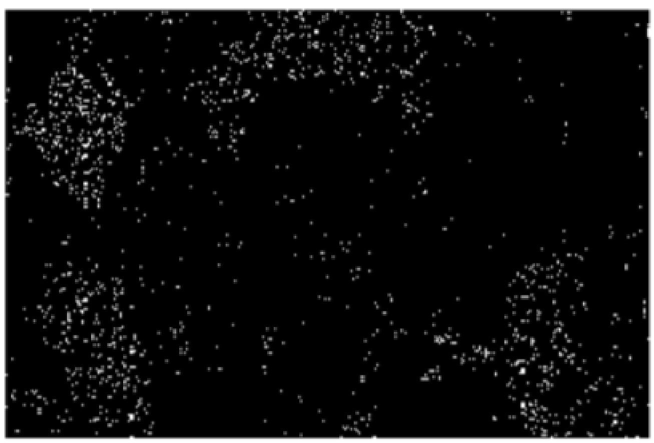


Program:

ii.

```
% Apply Sobel operator for edge detection (horizontal and vertical edges)
sobelEdges = edge(grayImage, 'sobel');

% Display or save the edges detected using Sobel operator
imshow(sobelEdges);
imwrite(sobelEdges, 'sobel_edges.jpg');
```

Output:

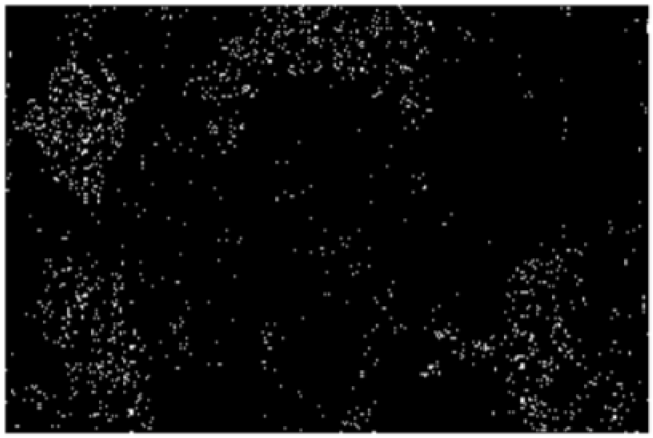


Program:

iii.

```
% Apply Prewitt operator for edge detection (horizontal and vertical edges)
prewittEdges = edge(grayImage, 'prewitt');
% Display or save the edges detected using Prewitt operator
imshow(prewittEdges);
imwrite(prewittEdges, 'prewitt_edges.jpg');
```

Output:



Program:

iv.

```
% Apply Canny operator for edge detection
cannyEdges = edge(grayImage, 'canny');

% Display or save the edges detected using Canny operator
imshow(cannyEdges);
imwrite(cannyEdges, 'canny_edges.jpg');
```

Output:

Program:

v.

```
% Apply edge detection (e.g., Canny) for better Hough transform results
edges = edge(grayImage, 'canny');

% Perform Hough Transform for line detection
[H,theta,rho] = hough(edges);
peaks = houghpeaks(H, 10); % Specify the number of peaks to detect

% Find lines in the image
lines = houghlines(edges, theta, rho, peaks);

% Draw lines on the original color image
figure, imshow(originalImage), hold on
for k = 1:length(lines)
   xy = [lines(k).point1; lines(k).point2];
   plot(xy(:,1), xy(:,2), 'LineWidth', 2, 'Color', 'r');
end
hold off;

% Display or save the image with detected lines
imwrite(originalImage, 'lines_detected_image.jpg');
```

Output:

Conclusion: In this practical, I implemented various edge detection operators, including Laplacian, Sobel, Prewitt, and Canny, as well as the Hough Transform for line detection.

PRACTICAL - 7

Aim: Implement code to detect features using
i. Harris corner detector
ii. Shi-Tomasi corner detector
iii. Scale Invariant Feature Transform (SIFT)
iv. Speeded up Robust Feature (SURF)
v. Oriented FAST and Rotated BRIEF (ORB)

Program:

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

image_path = "/content/pexels-brett-sayles-6424244 (1).jpg"
image = cv2.imread(image_path)
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Detect corners using Harris corner detector
corner_image = cv2.cornerHarris(gray_image, blockSize=2, ksize=3, k=0.04)

# Threshold the corner response to identify strong corners
threshold = 0.01 * corner_image.max()
corner_image[corner_image < threshold] = 0

# Dilate the corners to make them more visible
corner_image_dilated = cv2.dilate(corner_image, None)
```

```
# Mark detected corners on the original image
image_with_corners = image.copy()
image_with_corners[corner_image_dilated > 0.01 * corner_image_dilated.max()]
= [0, 0, 255]  # Red color

# Display the original image with detected corners
plt.figure(figsize=(12, 6))
plt.subplot(121), plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB)),
plt.title('Original Image')
plt.subplot(122), plt.imshow(cv2.cvtColor(image_with_corners,
cv2.COLOR_BGR2RGB)), plt.title('Image with Detected Corners')
plt.show()



# Detect corners using Shi-Tomasi corner detector
corners = cv2.goodFeaturesToTrack(gray_image, maxCorners=100,
qualityLevel=0.01, minDistance=10)

# Convert corners to integer coordinates
corners = np.int0(corners)

# Draw detected corners on the original image
image_with_corners = image.copy()
for corner in corners:
    x, y = corner.ravel()
    cv2.circle(image_with_corners, (x, y), 3, 255, -1)  # Draw a circle at each corner
```

```
# Display the original image with detected corners
plt.figure(figsize=(12, 6))
plt.subplot(121), plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB)), plt.title('Original Image')
plt.subplot(122), plt.imshow(cv2.cvtColor(image_with_corners, cv2.COLOR_BGR2RGB)), plt.title('Image with Detected Corners')
plt.show()

# Create an SIFT object
sift = cv2.SIFT_create()

# Detect keypoints and compute descriptors
keypoints, descriptors = sift.detectAndCompute(gray_image, None)

# Draw detected keypoints on the original image
image_with_keypoints = cv2.drawKeypoints(image, keypoints, None)

# Display the original image with detected keypoints
plt.figure(figsize=(12, 6))
plt.subplot(121), plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB)), plt.title('Original Image')
plt.subplot(122), plt.imshow(cv2.cvtColor(image_with_keypoints, cv2.COLOR_BGR2RGB)), plt.title('Image with Detected Keypoints')
plt.show()

# Create a SURF object
surf = cv2.SURF_create()
```

```
# Detect keypoints and compute descriptors
keypoints, descriptors = surf.detectAndCompute(gray_image, None)

# Draw detected keypoints on the original image
image_with_keypoints = cv2.drawKeypoints(image, keypoints, None, (0, 255, 0),
4)

# Display the original image with detected keypoints
plt.figure(figsize=(12, 6))
plt.subplot(121), plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB)),
plt.title('Original Image')
plt.subplot(122), plt.imshow(cv2.cvtColor(image_with_keypoints,
cv2.COLOR_BGR2RGB)), plt.title('Image with Detected Keypoints')
plt.show()

# Create an ORB object
orb = cv2.ORB_create()

# Detect keypoints and compute descriptors
keypoints, descriptors = orb.detectAndCompute(gray_image, None)

# Draw detected keypoints on the original image
image_with_keypoints = cv2.drawKeypoints(image, keypoints, None, (0, 255, 0),
4)

# Display the original image with detected keypoints
plt.figure(figsize=(12, 6))
```

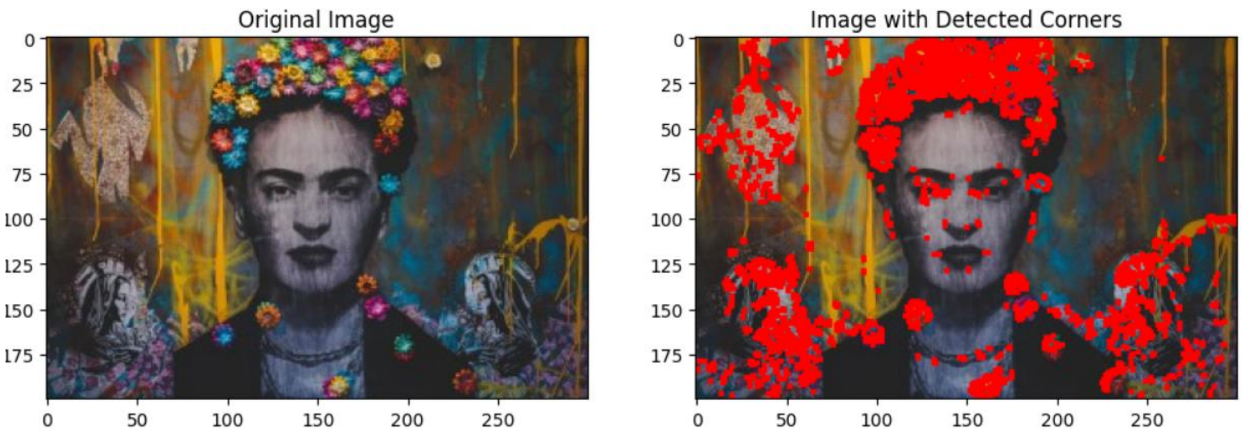
plt.subplot(121), plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB)), plt.title('Original Image')

plt.subplot(122), plt.imshow(cv2.cvtColor(image_with_keypoints, cv2.COLOR_BGR2RGB)), plt.title('Image with Detected Keypoints')

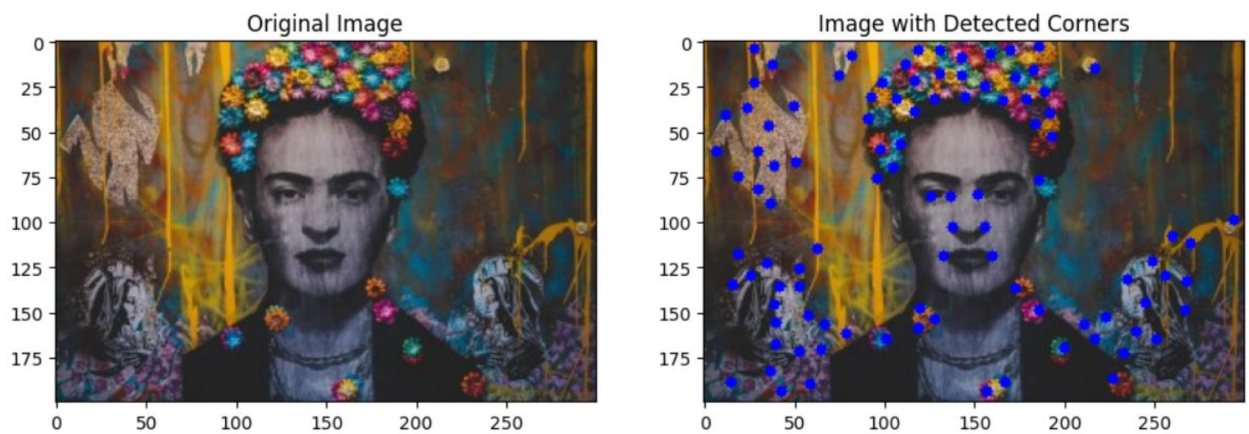
plt.show()

Output:

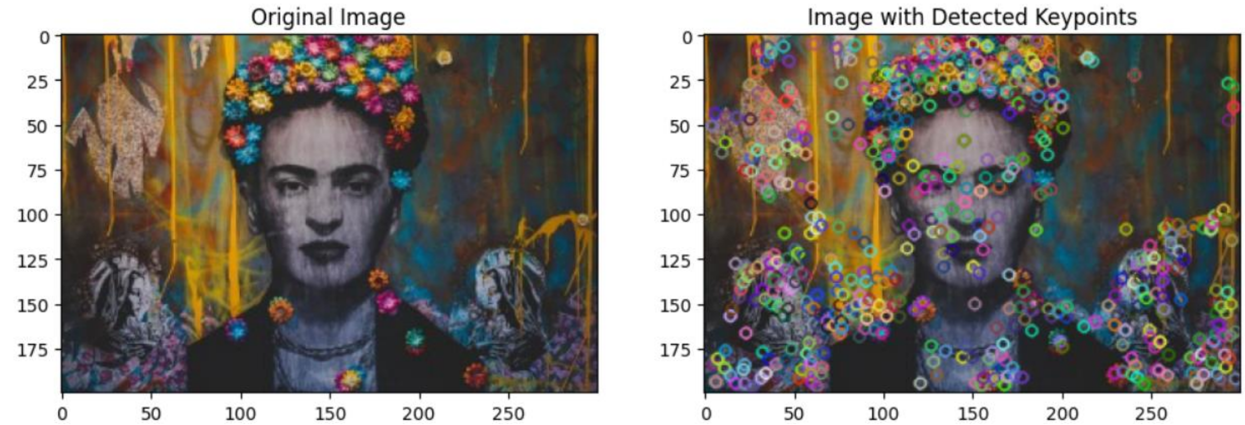
i



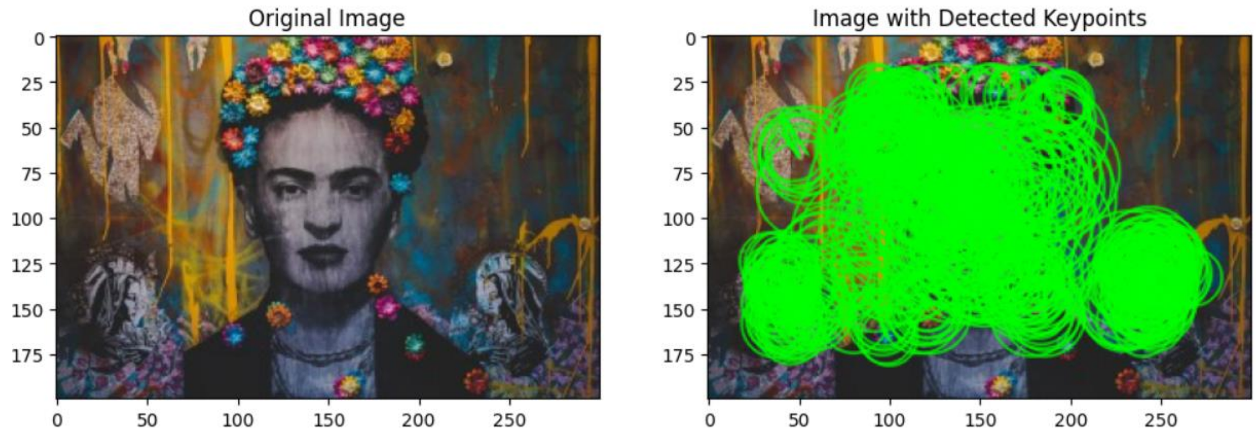
ii



iii



v



Conclusion: In this practical, I learnt various feature detection methods, including Harris and Shi-Tomasi corner detectors, SIFT, SURF, and ORB, for feature-based image analysis.

PRACTICAL - 8

Aim: Implement a code to segment an image
i. of mutually touching coins using distance transform along with watershed algorithms.
ii. using the K-means algorithm.
iii. using the Grabcut algorithm.

Program:

```
import cv2

import numpy as np

import matplotlib.pyplot as plt


image_path = "/content/pexels-brett-sayles-6424244 (1).jpg"

image = cv2.imread(image_path)

gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)


# Threshold the grayscale image to create a binary mask of the coins

_, binary_mask = cv2.threshold(gray_image, 0, 255, cv2.THRESH_BINARY_INV
+ cv2.THRESH_OTSU)


# Perform morphological operations to remove noise and separate touching coins

kernel = np.ones((3, 3), np.uint8)

opening = cv2.morphologyEx(binary_mask, cv2.MORPH_OPEN, kernel,
iterations=2)


# Calculate the distance transform

dist_transform = cv2.distanceTransform(opening, cv2.DIST_L2, 5)
```

```
_, sure_foreground = cv2.threshold(dist_transform, 0.7 * dist_transform.max(),
255, 0)

# Find sure background
sure_background = cv2.dilate(opening, kernel, iterations=3)

# Subtract sure background from sure foreground to get unknown region
sure_foreground = np.uint8(sure_foreground)
unknown = cv2.subtract(sure_background, sure_foreground)

# Label markers for watershed
_, markers = cv2.connectedComponents(sure_foreground)

# Add 1 to all labels to ensure that sure background is not 0 (unlabeled)
markers = markers + 1
markers[unknown == 255] = 0

# Apply the watershed algorithm
cv2.watershed(image, markers)
image[markers == -1] = [0, 0, 255]  # Mark segmented regions in red

# Display the original image with segmented coins
plt.figure(figsize=(12, 6))
plt.subplot(121), plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB)),
plt.title('Original Image')
```

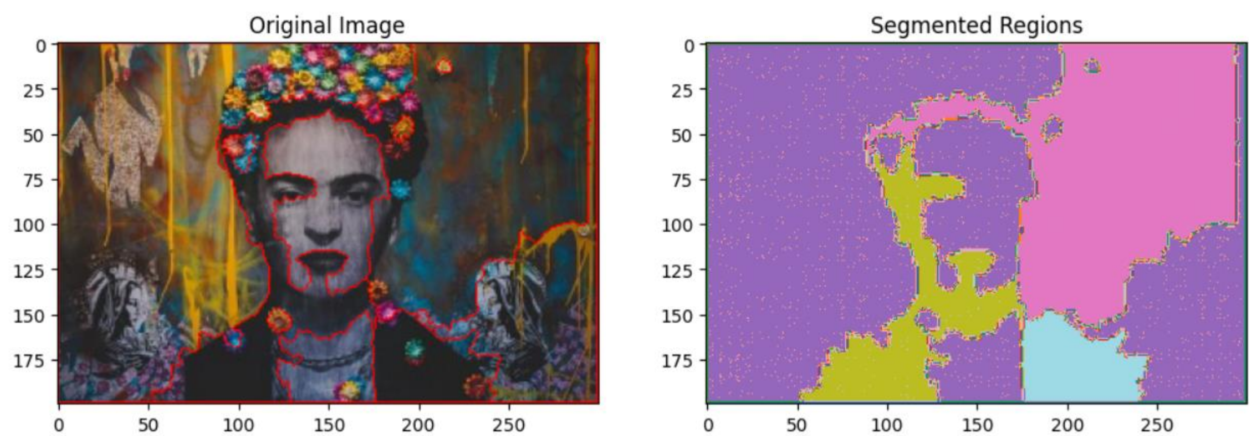
```
plt.subplot(122), plt.imshow(markers, cmap='tab20'), plt.title('Segmented Regions')

plt.show()



image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)  # Convert to RGB


# Reshape the image to a 2D array of pixels

pixels = image.reshape(-1, 3)


# Define the number of clusters (K)

K = 3

# Apply K-means clustering

criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 100, 0.2)

_, labels, centers = cv2.kmeans(np.float32(pixels), K, None, criteria, 10, cv2.KMEANS_RANDOM_CENTERS)


# Convert the labels to 8-bit for visualization

labels = labels.reshape(image.shape[0], image.shape[1]).astype(np.uint8)


# Create a mask for each segment

segmented_images = []

for i in range(K):

    mask = np.where(labels == i, 255, 0).astype(np.uint8)

    segmented_images.append(mask)
```

```
# Display the original image and segmented images
plt.figure(figsize=(12, 6))
plt.subplot(131), plt.imshow(image), plt.title('Original Image')
plt.subplot(132), plt.imshow(segmented_images[0], cmap='gray'),
plt.title('Segment 1')
plt.subplot(133), plt.imshow(segmented_images[1], cmap='gray'),
plt.title('Segment 2')
plt.show()

# Create a mask and initialize it with zeros
mask = np.zeros(image.shape[:2], np.uint8)

# Define a rectangular region of interest (ROI) for initialization
rect = (50, 50, image.shape[1] - 50, image.shape[0] - 50)

# Initialize the GrabCut algorithm with the image, mask, and ROI
bgdModel = np.zeros((1, 65), np.float64)
fgdModel = np.zeros((1, 65), np.float64)
cv2.grabCut(image, mask, rect, bgdModel, fgdModel, 5,
cv2.GC_INIT_WITH_RECT)

# Modify the mask to create a binary mask for the foreground
mask2 = np.where((mask == 2) | (mask == 0), 0, 1).astype('uint8')

# Multiply the original image with the binary mask to extract the segmented object
```

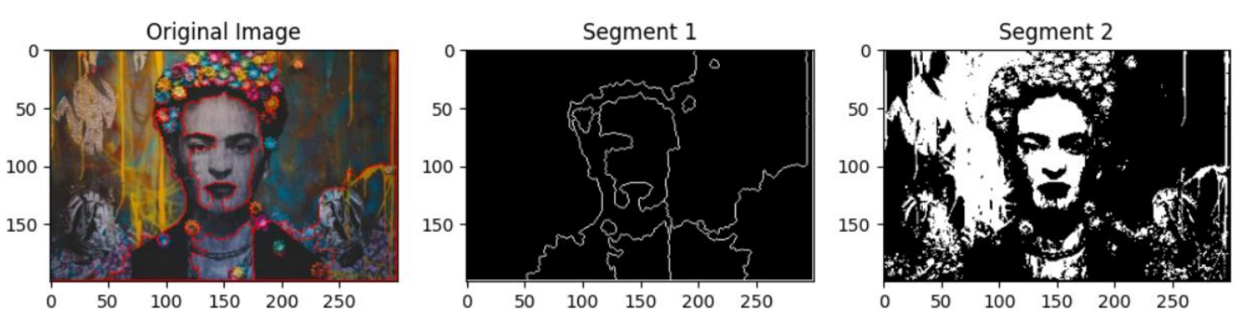
```
segmented_image = image * mask2[:, :, np.newaxis]


# Display the original image and the segmented result
plt.figure(figsize=(12, 6))
plt.subplot(121), plt.imshow(image), plt.title('Original Image')
plt.subplot(122), plt.imshow(segmented_image), plt.title('Segmented Image')
plt.show()
```

Output:

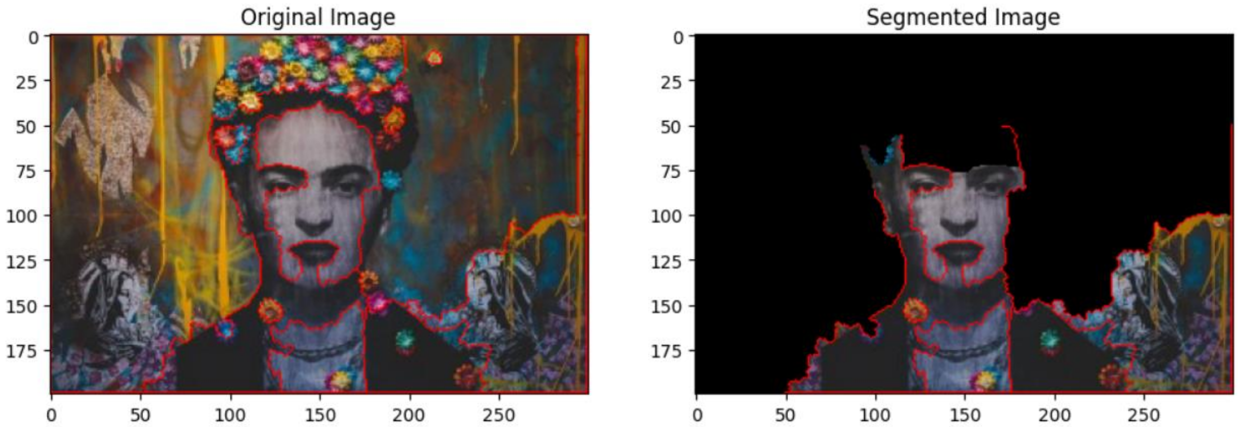
i



ii



iii



Conclusion: In this practical, I segmented images of mutually touching coins by employing distance transform with watershed, K-means, and Grabcut algorithms

PRACTICAL - 9

Aim: i. Implement face detection and eye detection using HAAR cascade classifiers.
ii. Implement face detection using Viola Jones method and Adaboost training algorithm.
iii. Implement car detection and pedestrian detection using HAAR cascade classifiers.

Program:

```
import cv2

from google.colab.patches import cv2_imshow  # Import cv2_imshow for Colab

import matplotlib.pyplot as plt


# Load pre-trained HAAR cascade classifiers for face and eye detection

face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 
'haarcascade_frontalface_default.xml')

eye_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 
'haarcascade_eye.xml')


# Load an image

image_path = "/content/saulgoodman.png"

image = cv2.imread(image_path)

gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)


# Detect faces in the image

faces = face_cascade.detectMultiScale(gray_image, scaleFactor=1.3, 
minNeighbors=5, minSize=(30, 30))
```

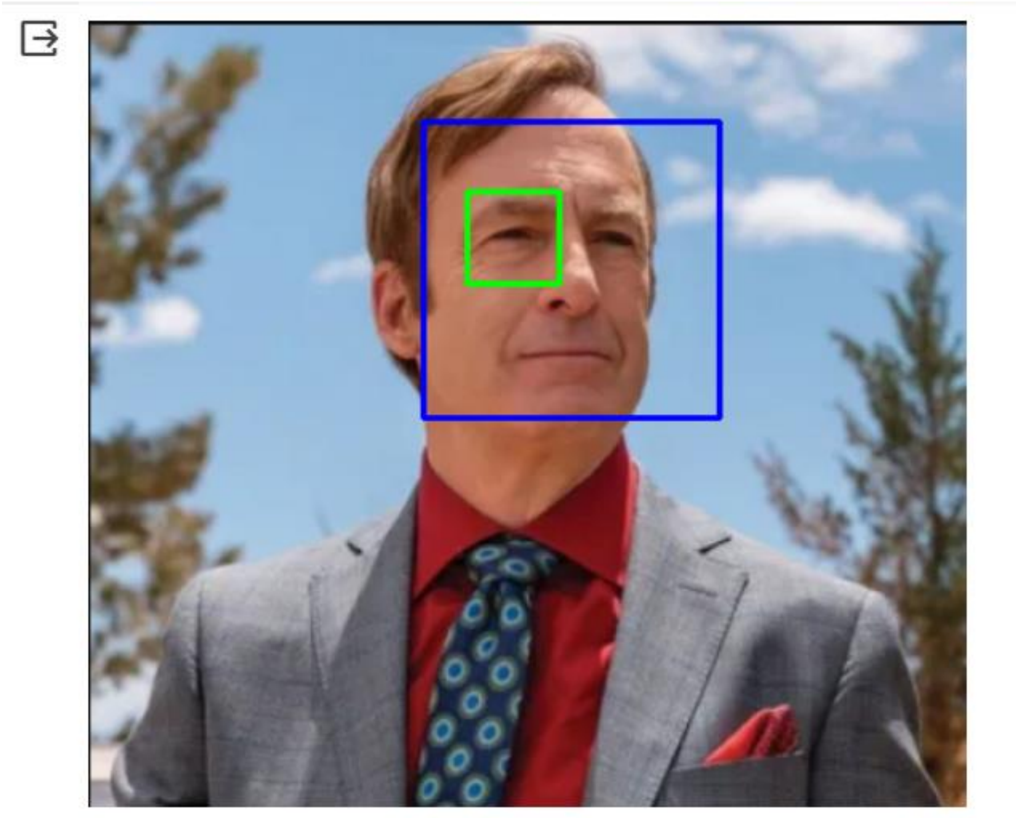
```
# Iterate over detected faces and draw rectangles around them
for (x, y, w, h) in faces:
    cv2.rectangle(image, (x, y), (x + w, y + h), (255, 0, 0), 2)
    roi_gray = gray_image[y:y + h, x:x + w]
    roi_color = image[y:y + h, x:x + w]

    # Detect eyes within each face region
    eyes = eye_cascade.detectMultiScale(roi_gray)
    for (ex, ey, ew, eh) in eyes:
        cv2.rectangle(roi_color, (ex, ey), (ex + ew, ey + eh), (0, 255, 0), 2)

# Display the image with detected faces and eyes using cv2_imshow
cv2_imshow(image)



# Detect faces in the image
faces = face_cascade.detectMultiScale(image, scaleFactor=1.3, minNeighbors=5,
minSize=(30, 30))

# Draw rectangles around detected faces
for (x, y, w, h) in faces:
    cv2.rectangle(image, (x, y), (x + w, y + h), (255, 0, 0), 2)

# Display the image with detected faces
cv2_imshow(image)
cv2.waitKey(0)
```

```
cv2.destroyAllWindows()

# Specify the paths to the Haar cascade XML files
car_cascade_path = '/content/haarcascade_car.xml'
pedestrian_cascade_path = '/content/haarcascade_fullbody.xml'

# Load the Haar cascade classifiers for car and pedestrian detection
car_cascade = cv2.CascadeClassifier(car_cascade_path)
pedestrian_cascade = cv2.CascadeClassifier(pedestrian_cascade_path)

# Load an image for detection
image_path = "/content/anthony-rosset-YLaLy6wlDiY-unsplash.jpg"
image = cv2.imread(image_path)
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Detect cars in the image
cars = car_cascade.detectMultiScale(gray_image, scaleFactor=1.1, 
minNeighbors=5, minSize=(20, 20))

# Detect pedestrians in the image
pedestrians = pedestrian_cascade.detectMultiScale(gray_image, scaleFactor=1.1, 
minNeighbors=5, minSize=(20, 20))

# Brighter colors for drawing rectangles
car_color = (0, 0, 255)  # Bright red (BGR color format)
pedestrian_color = (0, 255, 0)  # Bright green (BGR color format)
```

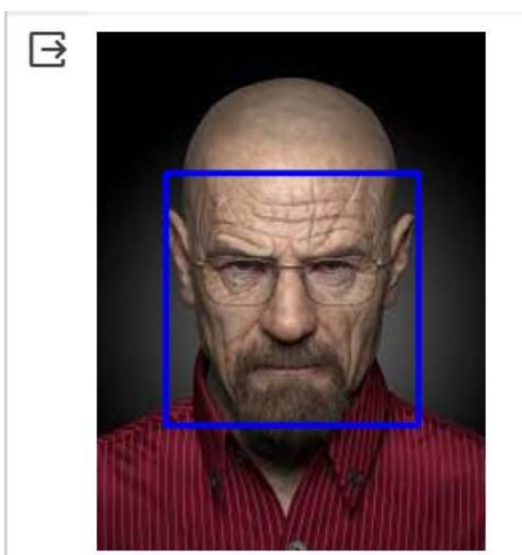
```
# Line thickness for drawing rectangles
line_thickness = 8  # Adjust as needed for thicker borders

# Loop through the detected cars and draw rectangles with thicker borders
for (x, y, w, h) in cars:
    cv2.rectangle(image, (x, y), (x + w, y + h), car_color, line_thickness)

# Loop through the detected pedestrians and draw rectangles with thicker borders
for (x, y, w, h) in pedestrians:
    cv2.rectangle(image, (x, y), (x + w, y + h), pedestrian_color, line_thickness)

# Convert the image from BGR to RGB for displaying with matplotlib
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# Display the image with detected cars and pedestrians
plt.imshow(image_rgb)
plt.axis('off')  # Turn off axis labels
plt.show()
```

Output:

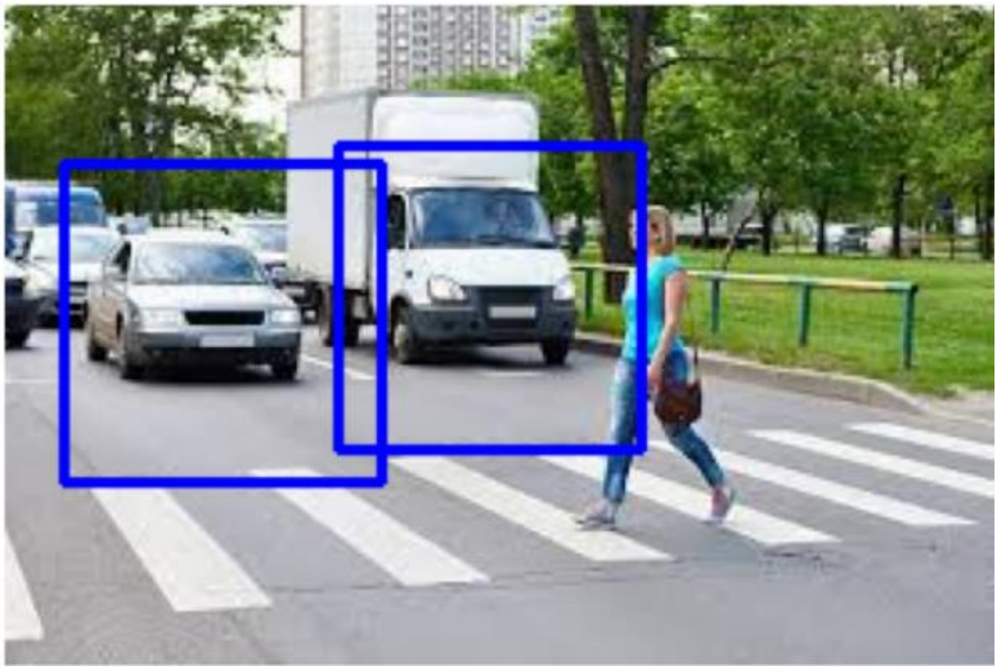
i



ii



iii



Conclusion: In this practical, I applied HAAR cascade classifiers for face, eye, car, and pedestrian detection, demonstrating robust object detection techniques in different contexts.

PRACTICAL - 10

Aim: i. Implement code to perform feature extraction on given images of faces using Histogram of Gradients.
ii. Implement code to apply Principal Component Analysis on extracted features in objective i.
iii. Implement code to recognize faces using the SVM classifier.

Program:

```
import matplotlib.pyplot as plt

from skimage import io, feature

from skimage.color import rgb2gray

from sklearn.decomposition import PCA

import numpy as np



# Load an image of a face (replace with your image path)

image_path = "/content/download (1).jpeg"

image = io.imread(image_path)


# Convert the image to grayscale

gray_image = rgb2gray(image)


# Compute HOG features

hog_features, hog_image = feature.hog(gray_image, visualize=True)


# Display the original image
```

```
plt.figure(figsize=(8, 4))
plt.subplot(121)
plt.imshow(image, cmap='gray')
plt.title('Original Image')
plt.axis('off')

# Display the HOG image
plt.subplot(122)
plt.imshow(hog_image, cmap='gray')
plt.title('HOG Features')
plt.axis('off')

plt.tight_layout()
plt.show()

# Print the HOG feature vector (histogram)
print("HOG Feature Vector (Histogram):")
print(hog_features)



# Extracted HOG features (replace with your own features)
hog_features = np.array([0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0])

# Create a PCA instance without specifying the number of components
pca = PCA()
```

```
# Fit the PCA model to the HOG features
pca.fit(hog_features.reshape(-1, 1))  # Reshape to a single feature per sample

# Transform the features into the PCA space
hog_features_pca = pca.transform(hog_features.reshape(-1, 1))

# Explained variance ratio of all components
explained_variance_ratio = pca.explained_variance_ratio_

# Print the transformed features and explained variance
print("Transformed HOG Features (PCA):")
print(hog_features_pca)
print("Explained Variance Ratio:")
print(explained_variance_ratio)
```

iii)

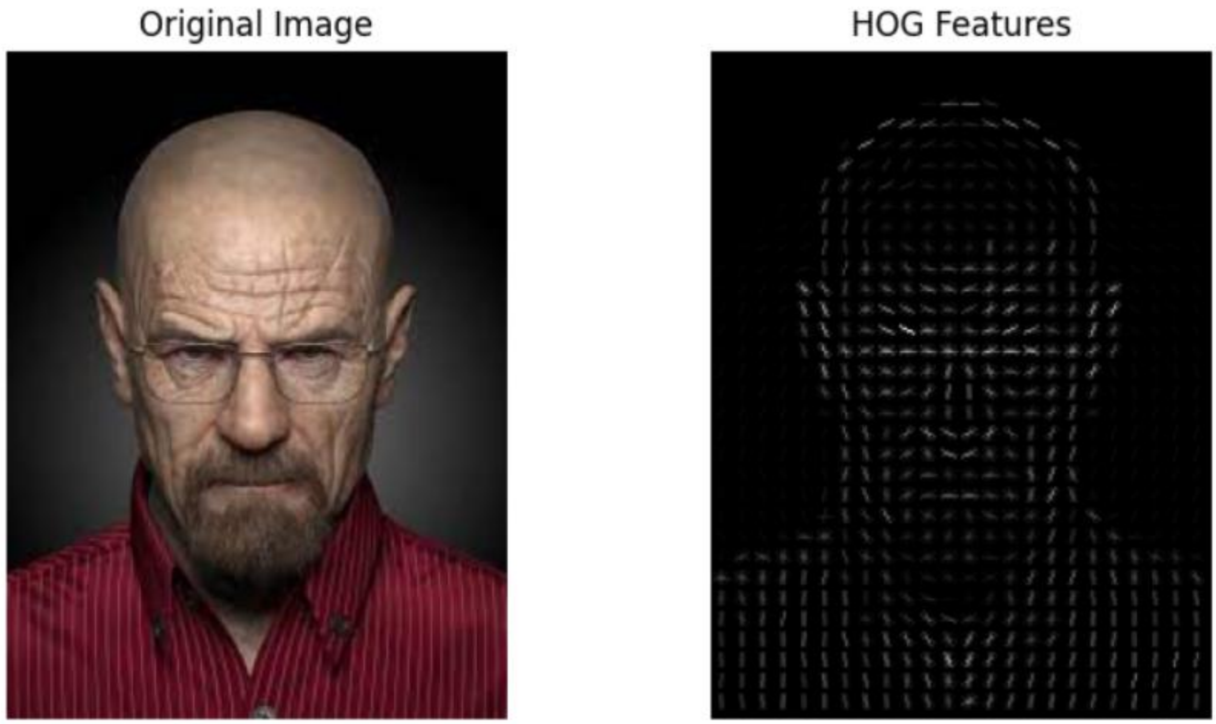
```
from sklearn.datasets import fetch_lfw_people
faces = fetch_lfw_people(min_faces_per_person=60)

faces.DESCR

import matplotlib.pyplot as plt

fig, splts = plt.subplots(2, 4)
for i, splts in enumerate(splts.flat):
```

```
    splts.imshow(faces.images[i], cmap='magma')
    splts.set(xticks=[], yticks=[],
            xlabel=faces.target_names[faces.target[i]])

from sklearn.model_selection import train_test_split

X = faces.data
y = faces.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4,
random_state=42)

from sklearn.svm import SVC
from sklearn.decomposition import PCA as RandomizedPCA
from sklearn.pipeline import make_pipeline

# For dimensionality reduction
pca = RandomizedPCA(n_components=150, whiten=True, random_state=42)
svc = SVC(kernel='rbf', class_weight='balanced')
model = make_pipeline(pca, svc)

model.fit(X_train, y_train)

from sklearn.metrics import accuracy_score

predictions = model.predict(X_test)
```

```
accuracy_score(predictions, y_test)

from colorama import Fore

incorrect = 0

length = len(predictions)

print("Actual\t\t\t\tPredicted\n")

for i in range(len(predictions)):
    if predictions[i] != y_test[i]: # if predictions and actual values are not equal
        prediction_name = faces.target_names[predictions[predictions[i]]] # Getting the predicted name
        actual_name = faces.target_names[y_test[y_test[i]]] # Getting the actual name
        incorrect+=1
        print("{}\t\t\t{}".format(Fore.GREEN + actual_name, Fore.RED+prediction_name))

print("{} are classified as correct and {} are classified as incorrect!".format(length-incorrect, incorrect))
```

Output:

i

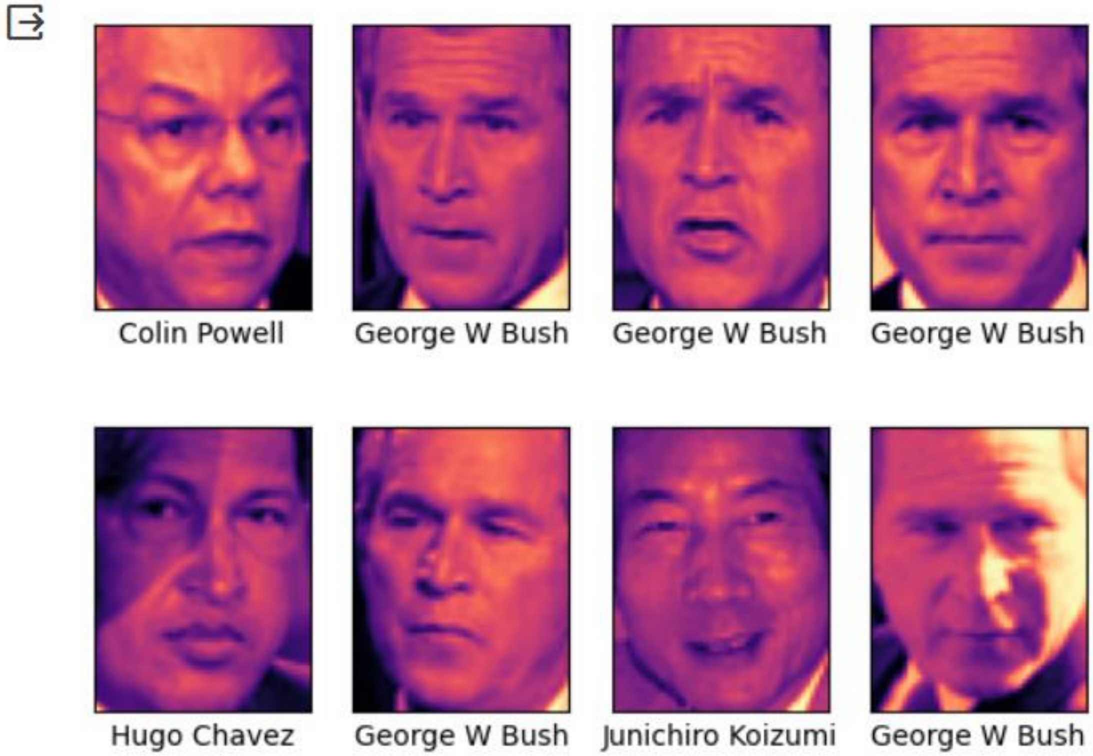



```
HOG Feature Vector (Histogram):
[0.         0.         0.         ... 0.01053878 0.00620127 0.24241371]
```

ii

```
Transformed HOG Features (PCA):
[[ 0.45]
 [ 0.35]
 [ 0.25]
 [ 0.15]
 [ 0.05]
 [-0.05]
 [-0.15]
 [-0.25]
 [-0.35]
 [-0.45]]
Explained Variance Ratio:
[1.]
```

iii


Colin Powell
George W Bush
George W Bush
George W Bush
Hugo Chavez
George W Bush
Junichiro Koizumi
George W Bush

```
Actual                          Predicted

Junichiro Koizumi                       Junichiro Koizumi
George W Bush                   George W Bush
Junichiro Koizumi                       George W Bush
Colin Powell                    Junichiro Koizumi
George W Bush                   Junichiro Koizumi
Colin Powell                    Junichiro Koizumi
Colin Powell                    George W Bush
George W Bush                   Junichiro Koizumi
George W Bush                   George W Bush
George W Bush                   Junichiro Koizumi
Junichiro Koizumi                       George W Bush
Colin Powell                    Junichiro Koizumi
Junichiro Koizumi                       George W Bush
Colin Powell                    George W Bush
Junichiro Koizumi                       George W Bush
George W Bush                   George W Bush
George W Bush                   Junichiro Koizumi
Junichiro Koizumi                       George W Bush
George W Bush                   George W Bush
Junichiro Koizumi                       George W Bush
Junichiro Koizumi                       George W Bush
George W Bush                   George W Bush

George W Bush                   Junichiro Koizumi
George W Bush                   George W Bush
George W Bush                   Junichiro Koizumi
George W Bush                   George W Bush
George W Bush                   Junichiro Koizumi
Colin Powell                    Junichiro Koizumi
Junichiro Koizumi                       George W Bush
Colin Powell                    George W Bush
Junichiro Koizumi                       Junichiro Koizumi
Gerhard Schroeder                       Junichiro Koizumi
George W Bush                   Junichiro Koizumi
Colin Powell                    Junichiro Koizumi
436 are classified as correct and 104 are classified as incorrect!
```

Conclusion: In this practical, I performed facial feature extraction with Histogram of Gradients, applied Principal Component Analysis (PCA) on the features, and achieved facial recognition using SVM classifiers.

PRACTICAL - 11

Aim:
i. Implement code to extract facial landmarks on given images.
ii. Implement code to merge faces (face swaps) using extracted facial landmark features on given images.
iii. Implement code to merge faces (face swaps) using extracted facial landmark features on live video.

Program:

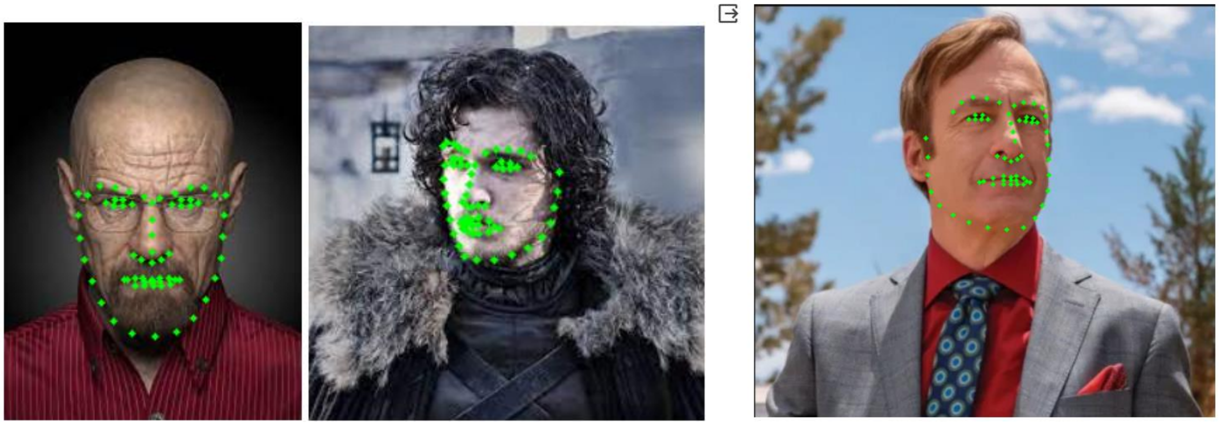
```
import cv2

import dlib

import numpy as np

from google.colab.patches import cv2_imshow


# Load the face detector from dlib (HOG-based)

face_detector = dlib.get_frontal_face_detector()

# Load the facial landmarks predictor from dlib

landmark_predictor =
dlib.shape_predictor("/content/drive/MyDrive/shape_predictor_68_face_landmarks
.dat")


# Load an image

image_path = "/content/saulgoodman.png"

image = cv2.imread(image_path)

# Convert the image to grayscale (required by dlib)

gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
```

```
# Detect faces in the image
faces = face_detector(gray_image)
# Loop over the detected faces
for face in faces:
    # Detect facial landmarks
    landmarks = landmark_predictor(gray_image, face)

    # Loop over the facial landmarks and draw them on the image
    for i in range(68):
        x, y = landmarks.part(i).x, landmarks.part(i).y
        cv2.circle(image, (x, y), 2, (0, 255, 0), -1)  # Draw a green circle at each
landmark point

# Display the image with facial landmarks
cv2_imshow(image)
# Load the source and target images
source_image = cv2.imread("/content/jesse.png")
target_image = cv2.imread("/content/download (1).jpeg")

# Detect faces in both images
source_faces = face_detector(source_image)
target_faces = face_detector(target_image)
# Ensure one face is detected in each image
if len(source_faces) != 1 or len(target_faces) != 1:
    print("Error: Exactly one face must be present in each image.")
else:
```

```
# Get facial landmarks for both faces
source_landmarks = landmark_predictor(source_image, source_faces[0])
target_landmarks = landmark_predictor(target_image, target_faces[0])

# Convert landmarks to NumPy arrays
source_landmarks = np.array([[p.x, p.y] for p in source_landmarks.parts()])
target_landmarks = np.array([[p.x, p.y] for p in target_landmarks.parts()])

# Compute the affine transformation matrix
transformation_matrix, _ = cv2.estimateAffinePartial2D(source_landmarks,
target_landmarks)

# Warp the source face to match the target face
warped_face = cv2.warpAffine(source_image, transformation_matrix,
(target_image.shape[1], target_image.shape[0]))

# Blend the warped face onto the target face
alpha = 0.7  # Adjust this value for blending intensity
beta = 1.0 - alpha
blended_face = cv2.addWeighted(target_image, alpha, warped_face, beta, 0)

# Save or display the resulting image
cv2.imwrite("output_face_swap.jpg", blended_face)
cv2_imshow( blended_face)
```

Output:

i



ii

Conclusion: In this practical, I extracted facial landmarks, performed face swaps using landmark features on static images, and extended the capability to live video.

PRACTICAL - 12

Aim: Implement Deep Learning concepts (using DIGITS/TensorFlow/Pytorch)
i. Image Classification
ii. Image Segmentation
iii.Object Detection
iv. Transfer Learning
v. Face Recognition
vi. Emotion Recognition

Program:

Output:

i

```
from google.colab import drive
drive.mount('/content/drive')
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.preprocessing.image import ImageDataGenerator
# Define your dataset directory on Google Colab
dataset_dir = '/content/drive/MyDrive/practical12/traindata/'
# Define hyperparameters
batch_size = 32
epochs = 10
input_shape = (224, 224, 3)  # Adjust the input shape according to your images
num_classes = 2  # Change this to the number of classes in your dataset
# Data augmentation and preprocessing
train_datagen = ImageDataGenerator(
```

```
    rescale=1.0/255.0,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    horizontal_flip=True,
    shear_range=0.2,
    zoom_range=0.2
)
# Create a generator for training data
train_generator = train_datagen.flow_from_directory(
    dataset_dir,
    target_size=input_shape[:2],
    batch_size=batch_size,
    class_mode='categorical',  # Use 'binary' for binary classification
    shuffle=True
)
# Build a convolutional neural network (CNN) model
model = keras.Sequential([
    keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=input_shape),
    keras.layers.MaxPooling2D((2, 2)),
    keras.layers.Conv2D(64, (3, 3), activation='relu'),
    keras.layers.MaxPooling2D((2, 2)),
    keras.layers.Conv2D(128, (3, 3), activation='relu'),
    keras.layers.MaxPooling2D((2, 2)),
    keras.layers.Flatten(),
```

```
    keras.layers.Dense(128, activation='relu'),
    keras.layers.Dense(num_classes, activation='softmax')
])
# Compile the model
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
# Train the model
history = model.fit(
    train_generator,
    steps_per_epoch=len(train_generator),
    epochs=epochs
)
# Save the trained model
model.save('image_classification_model.h5')
# Optionally, save training history for analysis
import pickle
with open('training_history.pkl', 'wb') as file:
    pickle.dump(history.history, file)
import os
import numpy as np
from tensorflow.keras.preprocessing import image
# Load the trained model
model = keras.models.load_model('image_classification_model.h5')
# Define a function to predict an individual image
```

```
def predict_image(image_path):
    img = image.load_img(image_path, target_size=(224, 224))
    img = image.img_to_array(img)
    img = np.expand_dims(img, axis=0)
    img = img / 255.0
    predictions = model.predict(img)
    class_index = np.argmax(predictions)
    return class_index
# Define the directory containing test images on Google Colab
test_dir = '/content/drive/MyDrive/practical12/traindata/testimage/'
# Loop through test images and make predictions
for filename in os.listdir(test_dir):
    if filename.endswith('.jpg'):
        image_path = os.path.join(test_dir, filename)
        class_index = predict_image(image_path)
        print(f"Image: {filename}, Predicted Class Index: {class_index}")

from google.colab import drive
drive.mount('/content/drive')
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.preprocessing.image import ImageDataGenerator
# Define your dataset directory on Google Colab
dataset_dir = '/content/drive/MyDrive/practical12/traindata/'
# Define hyperparameters
```

```
batch_size = 32
epochs = 10
input_shape = (224, 224, 3)  # Adjust the input shape according to your images
num_classes = 2  # Change this to the number of classes in your dataset
# Data augmentation and preprocessing
train_datagen = ImageDataGenerator(
    rescale=1.0/255.0,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    horizontal_flip=True,
    shear_range=0.2,
    zoom_range=0.2
)
# Create a generator for training data
train_generator = train_datagen.flow_from_directory(
    dataset_dir,
    target_size=input_shape[:2],
    batch_size=batch_size,
    class_mode='categorical',  # Use 'binary' for binary classification
    shuffle=True
)
# Build a convolutional neural network (CNN) model
model = keras.Sequential([
    keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=input_shape),
```

```
    keras.layers.MaxPooling2D((2, 2)),
    keras.layers.Conv2D(64, (3, 3), activation='relu'),
    keras.layers.MaxPooling2D((2, 2)),
    keras.layers.Conv2D(128, (3, 3), activation='relu'),
    keras.layers.MaxPooling2D((2, 2)),
    keras.layers.Flatten(),
    keras.layers.Dense(128, activation='relu'),
    keras.layers.Dense(num_classes, activation='softmax')
])
# Compile the model
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
# Train the model
history = model.fit(
    train_generator,
    steps_per_epoch=len(train_generator),
    epochs=epochs
)
# Save the trained model
model.save('image_classification_model.h5')
# Optionally, save training history for analysis
import pickle
with open('training_history.pkl', 'wb') as file:
    pickle.dump(history.history, file)
```

```
import os
import numpy as np
from tensorflow.keras.preprocessing import image
# Load the trained model
model = keras.models.load_model('image_classification_model.h5')
# Define a function to predict an individual image
def predict_image(image_path):
    img = image.load_img(image_path, target_size=(224, 224))
    img = image.img_to_array(img)
    img = np.expand_dims(img, axis=0)
    img = img / 255.0
    predictions = model.predict(img)
    class_index = np.argmax(predictions)
    return class_index

# Define the directory containing test images on Google Colab
test_dir = '/content/drive/MyDrive/practical12/traindata/testimage/'

# Loop through test images and make predictions
for filename in os.listdir(test_dir):
    if filename.endswith('.jpg'):
        image_path = os.path.join(test_dir, filename)
        class_index = predict_image(image_path)
        print(f"Image: {filename}, Predicted Class Index: {class_index}")
```

ii.

```
# Setting the dataset path
import pathlib
data_dir = pathlib.Path('/content/gdrive/MyDrive/Segmentation/dataset1')
image_count = len(list(data_dir.glob('*/*.png')))
print(image_count)
dir_data = "/content/gdrive/MyDrive/Segmentation/dataset1/"
dir_seg = dir_data + "/annotations_prepped_train/"
dir_img = dir_data + "/images_prepped_train/"

import cv2, os
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
## seaborn has white grid by default so I will get rid of this.
sns.set_style("whitegrid", {'axes.grid' : False})
ldseg = np.array(os.listdir(dir_seg))
## pick the first image file
fnm = ldseg[0]
print(fnm)
## read in the original image and segmentation labels
seg = cv2.imread(dir_seg + fnm ) # (360, 480, 3)
img_is = cv2.imread(dir_img + fnm )
print("seg.shape={}, img_is.shape={}".format(seg.shape,img_is.shape))
```

```
## Check the number of labels
mi, ma = np.min(seg), np.max(seg)
n_classes = ma - mi + 1
print("minimum seg = {}, maximum seg = {}, Total number of segmentation
classes = {}".format(mi,ma, n_classes))
fig = plt.figure(figsize=(5,5))
ax = fig.add_subplot(1,1,1)
ax.imshow(img_is)
ax.set_title("original image")
plt.show()
fig = plt.figure(figsize=(15,10))
for k in range(mi,ma+1):
    ax = fig.add_subplot(3, int(n_classes/3)+1, k+1)
    ax.imshow((seg == k)*1.0)
    ax.set_title("label = {}".format(k))
plt.show()
import random
def give_color_to_seg_img(seg,n_classes):
    '''
    seg : (input_width,input_height,3)
    '''
    if len(seg.shape)==3:
        seg = seg[:,:,0]
    seg_img = np.zeros( (seg.shape[0],seg.shape[1],3) ).astype('float')
    colors = sns.color_palette("hls", n_classes)
    for c in range(n_classes):
```

```
        segc = (seg == c)
        seg_img[:,:,0] += (segc*( colors[c][0] ))
        seg_img[:,:,1] += (segc*( colors[c][1] ))
        seg_img[:,:,2] += (segc*( colors[c][2] ))
    return(seg_img)
input_height , input_width = 224 , 224
output_height , output_width = 224 , 224
ldseg = np.array(os.listdir(dir_seg))
for fnm in ldseg[np.random.choice(len(ldseg),3,replace=False)]:
    fnm = fnm.split(".")[0]
    seg = cv2.imread(dir_seg + fnm + ".png") # (360, 480, 3)
    img_is = cv2.imread(dir_img + fnm + ".png")
    seg_img = give_color_to_seg_img(seg,n_classes)
    fig = plt.figure(figsize=(20,40))
    ax = fig.add_subplot(1,4,1)
    ax.imshow(seg_img)
    ax = fig.add_subplot(1,4,2)
    ax.imshow(img_is/255.0)
    ax.set_title("original image {}".format(img_is.shape[:2]))
    ax = fig.add_subplot(1,4,3)
    ax.imshow(cv2.resize(seg_img,(input_height , input_width)))
    ax = fig.add_subplot(1,4,4)
    ax.imshow(cv2.resize(img_is,(output_height , output_width))/255.0)
    ax.set_title("resized to {}".format((output_height , output_width)))
    plt.show()
```

iv.

```
import tensorflow as tf
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.layers import GlobalAveragePooling2D, Dense
from tensorflow.keras.models import Model

# Load the CIFAR-10 dataset
(X_train, y_train), (X_test, y_test) = cifar10.load_data()
# Normalize pixel values to between 0 and 1
X_train, X_test = X_train / 255.0, X_test / 255.0
# Load the pre-trained ResNet50 model (excluding the top classification layers)
base_model = ResNet50(weights='imagenet', include_top=False)
# Add custom classification layers on top of the pre-trained model
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(1024, activation='relu')(x)
predictions = Dense(10, activation='softmax')(x)  # Adjust the number of classes
# Create the transfer learning model
model = Model(inputs=base_model.input, outputs=predictions)
# Freeze the layers of the pre-trained model (optional)
for layer in base_model.layers:
    layer.trainable = False
# Compile the model
model.compile(optimizer='adam',
```

```
            loss='sparse_categorical_crossentropy',  # Adjust the loss function
            metrics=['accuracy'])
# Train the model
history = model.fit(X_train, y_train,
                 epochs=10,  # Adjust the number of epochs
                 validation_data=(X_test, y_test))
# Evaluate the model
test_loss, test_accuracy = model.evaluate(X_test, y_test)
print(f'Test accuracy: {test_accuracy:.4f}')
```

v.

```
#import OpenCV module
import cv2
import os
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
#function to detect face
def detect_face (img):
#convert the test image to gray image
gray = cv2.cvtColor (img, cv2.COLOR_BGR2GRAY)
#load OpenCV face detector
face_cas = cv2.CascadeClassifier ('-File name.xml-')
faces = face_cas.detectMultiScale (gray, scaleFactor=1.3, minNeighbors=4);
```

```
#if no faces are detected then return image
if (len (faces) == 0):
return None, None
#extract the face
faces [0]=(x, y, w, h)
#return only the face part
return gray[y: y+w, x: x+h], faces [0]
#this function will read all persons' training images, detect face #from each image
#and will return two lists of exactly same size, one list
def prepare_training_data(data_folder_path):

#------STEP-1--------
#get the directories (one directory for each subject) in data folder
dirs = os.listdir(data_folder_path)
faces = []
labels = []
for dir_name in dirs:
#our subject directories start with letter 's' so
#ignore any non-relevant directories if any
if not dir_name.startswith("s"):
continue;

#------STEP-2--------
#extract label number of subject from dir_name
#format of dir name = slabel
```

```
#, so removing letter 's' from dir_name will give us label
label = int(dir_name.replace("s", ""))
#build path of directory containin images for current subject subject
#sample subject_dir_path = "training-data/s1"
subject_dir_path = data_folder_path + "/" + dir_name
#get the images names that are inside the given subject directory
subject_images_names = os.listdir(subject_dir_path)

#------STEP-3--------
#go through each image name, read image,
#detect face and add face to list of faces
for image_name in subject_images_names:
#ignore system files like .DS_Store
if image_name.startswith("."):
continue;
#build image path
#sample image path = training-data/s1/1.pgm
image_path = subject_dir_path + "/" + image_name
#read image
image = cv2.imread(image_path)
#display an image window to show the image
cv2.imshow("Training on image...", image)
cv2.waitKey(100)
#detect face
face, rect = detect_face(image)
```

```
#------STEP-4--------
#we will ignore faces that are not detected
if face is not None:
#add face to list of faces
faces.append(face)
#add label for this face
labels.append(label)
cv2.destroyAllWindows()
cv2.waitKey(1)
cv2.destroyAllWindows()
return faces, labels
#let's first prepare our training data
#data will be in two lists of same size
#one list will contain all the faces
#and other list will contain respective labels for each face
print("Preparing data...")
faces, labels = prepare_training_data("training-data")
print("Data prepared")
#print total faces and labels
print("Total faces: ", len(faces))
print("Total labels: ", len(labels))
#create our LBPH face recognizer
face_recognizer = cv2.face.createLBPHFaceRecognizer()
#train our face recognizer of our training faces
face_recognizer.train(faces, np.array(labels))
```

```
#function to draw rectangle on image
#according to given (x, y) coordinates and
#given width and heigh
def draw_rectangle(img, rect):
(x, y, w, h) = rect
cv2.rectangle(img, (x, y), (x+w, y+h), (0, 255, 0), 2)
#function to draw text on give image starting from
#passed (x, y) coordinates.
def draw_text(img, text, x, y):
cv2.putText(img, text, (x, y), cv2.FONT_HERSHEY_PLAIN, 1.5, (0, 255, 0), 2)
#this function recognizes the person in image passed
#and draws a rectangle around detected face with name of the subject
def predict(test_img):
#make a copy of the image as we don't want to chang original image
img = test_img.copy()
#detect face from the image
face, rect = detect_face(img)
#predict the image using our face recognizer
label= face_recognizer.predict(face)
#get name of respective label returned by face recognizer
label_text = subjects[label]
#draw a rectangle around face detected
draw_rectangle(img, rect)
#draw name of predicted person
draw_text(img, label_text, rect[0], rect[1]-5)
```

```
return img
#load test images
test_img1 = cv2.imread("test-data/test1.jpg")
test_img2 = cv2.imread("test-data/test2.jpg")
#perform a prediction
predicted_img1 = predict(test_img1)
predicted_img2 = predict(test_img2)
print("Prediction complete")

#create a figure of 2 plots (one for each test image)
f, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 5))

#display test image1 result
ax1.imshow(cv2.cvtColor(predicted_img1, cv2.COLOR_BGR2RGB))
#display test image2 result
ax2.imshow(cv2.cvtColor(predicted_img2, cv2.COLOR_BGR2RGB))
#display both images
cv2.imshow("Tom cruise test", predicted_img1)
cv2.imshow("Shahrukh Khan test", predicted_img2)
cv2.waitKey(0)
cv2.destroyAllWindows()
cv2.waitKey(1)
cv2.destroyAllWindows()
```

vi.

```
# read image
img = cv2.imread('img1.jpg')
# call imshow() using plt object
plt.imshow(img[:, :, : : -1])
# display that image
plt.show()
# storing the result
result = DeepFace.analyze(img,
actions = ['emotion'])
# print result
print(result)
# import the required modules
import cv2
import matplotlib.pyplot as plt
from deepface import DeepFace
# read image
img = cv2.imread('img.jpg')
# call imshow() using plt object
plt.imshow(img[:,:,::-1])
# display that image
plt.show()
# storing the result
result = DeepFace.analyze(img,actions=['emotion'])
print(result)
```

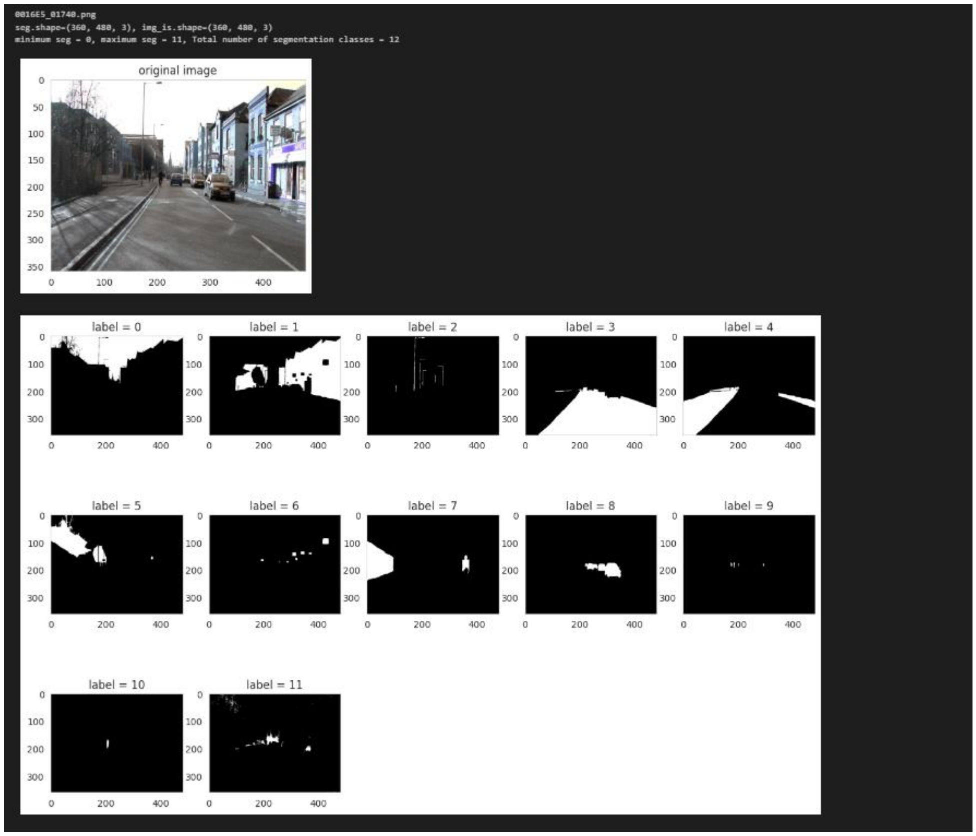
Output:

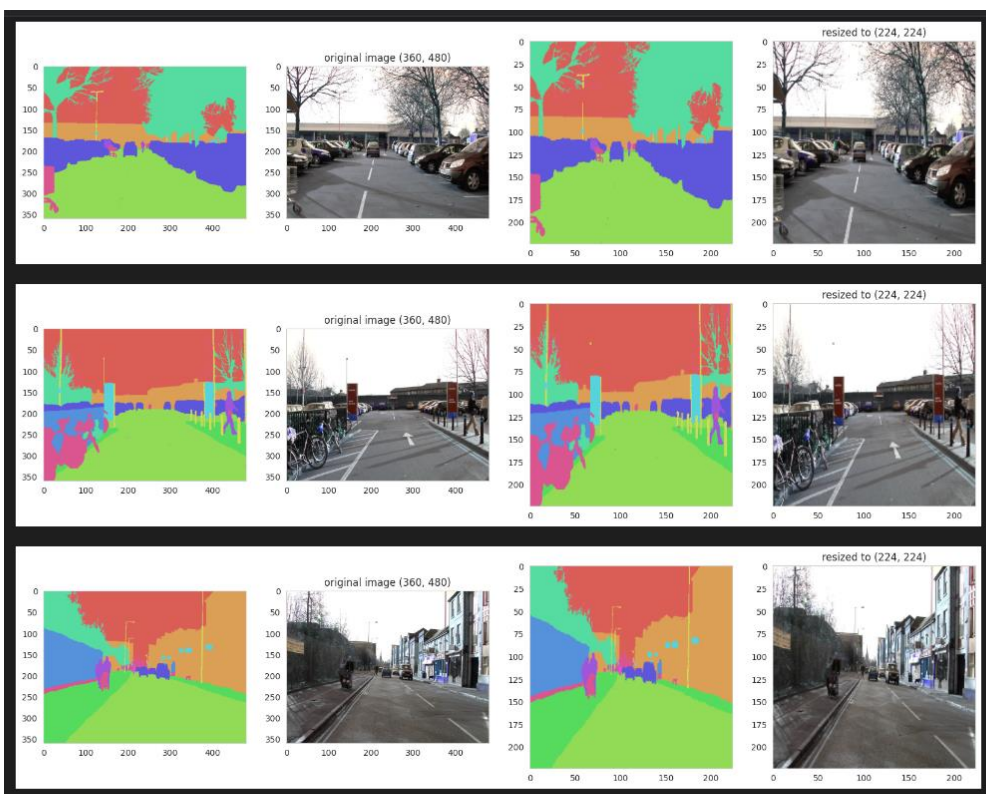
i.

```
Found 165 images belonging to 1 classes.
Epoch 1/10
6/6 [==============================] - 51s 8s/step - loss: 1.8926 - accuracy: 0.1879
Epoch 2/10
6/6 [==============================] - 3s 490ms/step - loss: 20.8844 - accuracy: 0.6121
Epoch 3/10
6/6 [==============================] - 3s 394ms/step - loss: 132.8162 - accuracy: 0.5818
Epoch 4/10
6/6 [==============================] - 3s 393ms/step - loss: 502.1111 - accuracy: 0.4182
Epoch 5/10
6/6 [==============================] - 2s 381ms/step - loss: 765.7403 - accuracy: 0.5818
Epoch 6/10
6/6 [==============================] - 3s 434ms/step - loss: 3444.7781 - accuracy: 0.4182
Epoch 7/10
6/6 [==============================] - 4s 575ms/step - loss: 6166.1294 - accuracy: 0.4182
Epoch 8/10
6/6 [==============================] - 3s 394ms/step - loss: 7368.7881 - accuracy: 0.6121
Epoch 9/10
6/6 [==============================] - 3s 398ms/step - loss: 14062.2969 - accuracy: 0.4182
Epoch 10/10
6/6 [==============================] - 3s 404ms/step - loss: 26948.4355 - accuracy: 0.7758
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3000: UserWarning: You are saving your mo
  saving_api.save_model(
```

```
1/1 [==============================] - 0s 70ms/step
Image: SPB_25-Mar_10-35-10_01.jpg, Predicted Class Index: 1
1/1 [==============================] - 0s 20ms/step
Image: Robo_25-Mar_12-33-22_01.jpg, Predicted Class Index: 1
```

ii.


0016E5_01740.png
seg.shape=(360, 480, 3), img_is.shape=(360, 480, 3)
minimum seg = 0, maximum seg = 11, Total number of segmentation classes = 12
original image
label = 0
label = 1
label = 2
label = 3
label = 4
label = 5
label = 6
label = 7
label = 8
label = 9
label = 10
label = 11

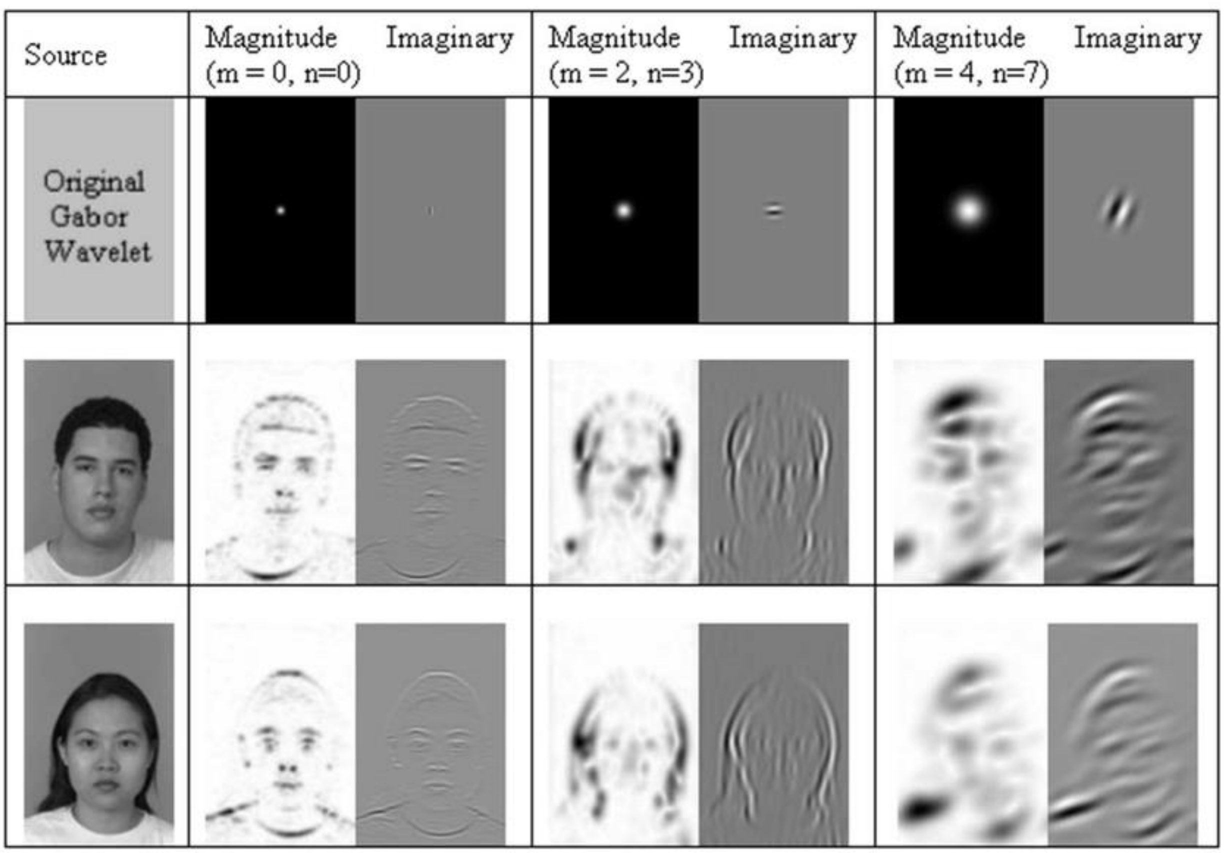

original image (360, 480)
resized to (224, 224)
original image (360, 480)
resized to (224, 224)
original image (360, 480)
resized to (224, 224)

iv.

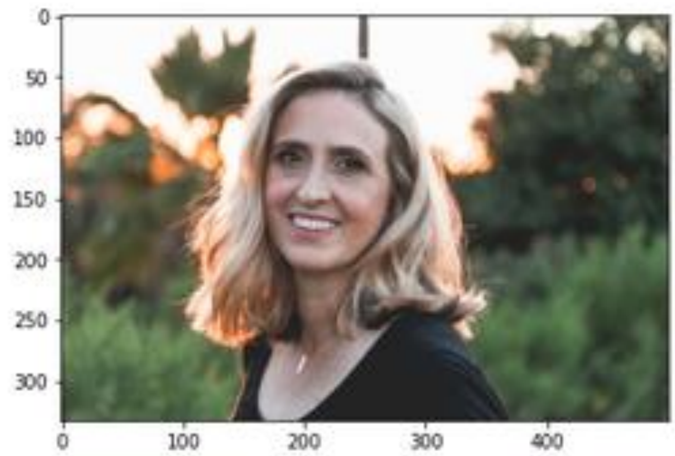
```
Epoch 1/10
1563/1563 [==============================] - 37s 16ms/step - loss: 2.0382 - accuracy: 0.2593 - val_loss: 1.9041 - val_accuracy: 0.2964
Epoch 2/10
1563/1563 [==============================] - 21s 13ms/step - loss: 1.8747 - accuracy: 0.3164 - val_loss: 1.8232 - val_accuracy: 0.3379
Epoch 3/10
1563/1563 [==============================] - 20s 13ms/step - loss: 1.8194 - accuracy: 0.3401 - val_loss: 1.7811 - val_accuracy: 0.3503
Epoch 4/10
1563/1563 [==============================] - 23s 15ms/step - loss: 1.7895 - accuracy: 0.3505 - val_loss: 1.7763 - val_accuracy: 0.3530
Epoch 5/10
1563/1563 [==============================] - 23s 15ms/step - loss: 1.7662 - accuracy: 0.3611 - val_loss: 1.7181 - val_accuracy: 0.3830
Epoch 6/10
1563/1563 [==============================] - 23s 15ms/step - loss: 1.7506 - accuracy: 0.3663 - val_loss: 1.7250 - val_accuracy: 0.3832
Epoch 7/10
1563/1563 [==============================] - 21s 13ms/step - loss: 1.7350 - accuracy: 0.3759 - val_loss: 1.6867 - val_accuracy: 0.3968
Epoch 8/10
1563/1563 [==============================] - 21s 13ms/step - loss: 1.7189 - accuracy: 0.3812 - val_loss: 1.7239 - val_accuracy: 0.3777
Epoch 9/10
1563/1563 [==============================] - 21s 13ms/step - loss: 1.7069 - accuracy: 0.3866 - val_loss: 1.7374 - val_accuracy: 0.3729
Epoch 10/10
1563/1563 [==============================] - 21s 13ms/step - loss: 1.7019 - accuracy: 0.3882 - val_loss: 1.7405 - val_accuracy: 0.3734
```

```
313/313 [==============================] - 3s 11ms/step - loss: 1.7405 - accuracy: 0.3734
Test accuracy: 0.3734
```

v.



vi.




```
facial_expression_model_weights.h5 will be downloaded...
Downloading...
From: https://github.com/serengil/deepface_models/releases/download/v1.0/facial_expression_model_weights.h5
To: /root/.deepface/weights/facial_expression_model_weights.h5
100%|██████████| 5.98M/5.98M [00:00<00:00, 23.5MB/s]
{'emotion': {'angry': 7.147069602808642e-07, 'disgust': 1.7738306483383592e-12, 'fear': 7.72365460477431e-07, 'happy': 96.74765467643738, '
```


```
facial_expression_model_weights.h5 will be downloaded...
Downloading...
From: https://github.com/serengil/deepface_models/releases/download/v1.0/facial_expression_model_weights.h5
To: /root/.deepface/weights/facial_expression_model_weights.h5
100%|██████████| 5.98M/5.98M [00:00<00:00, 23.5MB/s]
{'emotion': {'angry': 7.147069602808642e-07, 'disgust': 1.7738306483383592e-12, 'fear': 7.72365460477431e-07, 'happy': 96.74765467643738, '
```


Conclusion: In this practical, I implemented deep learning concepts using DIGITS, TensorFlow, and PyTorch covered a wide spectrum of applications, including image classification, segmentation, object detection, transfer learning, face recognition, and emotion recognition