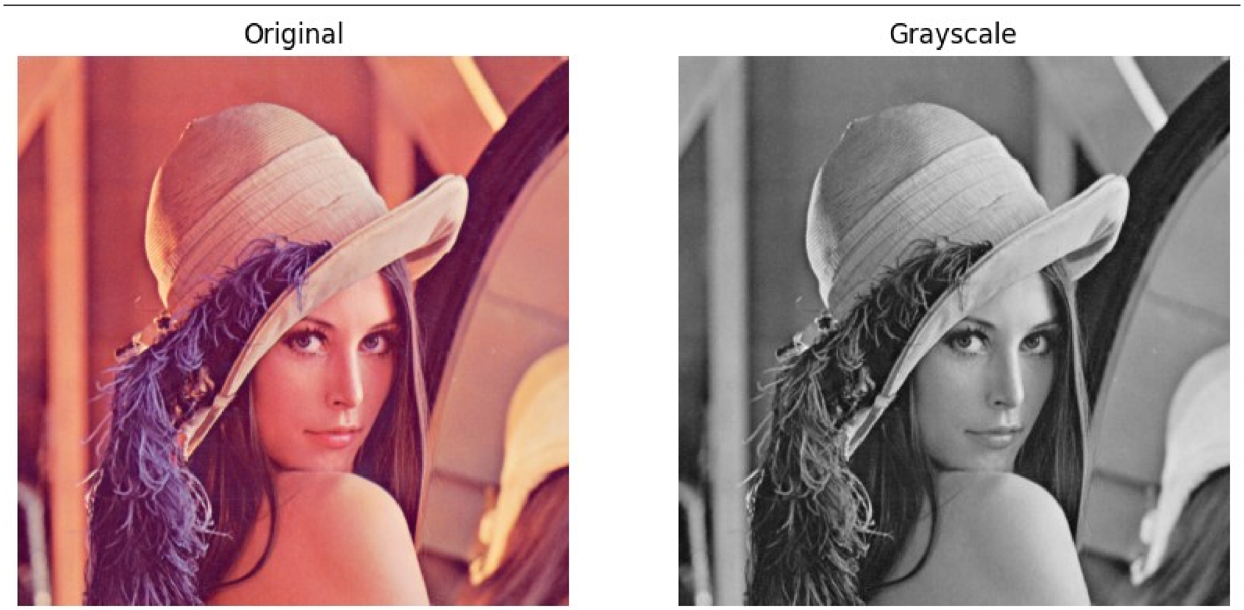
CYCLE – 1

1. Read an image and convert it into gray scale image without using builtin function for the function

Code:

```
rom PIL import Image
import matplotlib.pyplot as plt

def convert_to_grayscale(image_path):
img = Image.open(image_path)
width, height = img.size
grayscale_img = Image.new('L', (width, height))
for y in range(height):
for x in range(width):
pixel = img.getpixel((x, y))
gray_value = int(0.299 * pixel[0] + 0.587 * pixel[1] + 0.114 * pixel[2])
grayscale_img.putpixel((x, y), gray_value)
grayscale_img.save("grayscale_image.jpg")

# Display both images side by side
fig, axes = plt.subplots(1, 2, figsize=(10, 5))
# Plot original image
original_img = plt.imread(image_path)
axes[0].imshow(original_img)
axes[0].set_title('Original')
axes[0].axis('off') # Hide axes
# Plot grayscale image
grayscale_array = plt.imread("grayscale_image.jpg")
axes[1].imshow(grayscale_array, cmap='gray')
axes[1].set_title('Grayscale')
axes[1].axis('off') # Hide axes
plt.show()

image_path = "test1.png"
convert_to_grayscale(image_path)
```

Output:



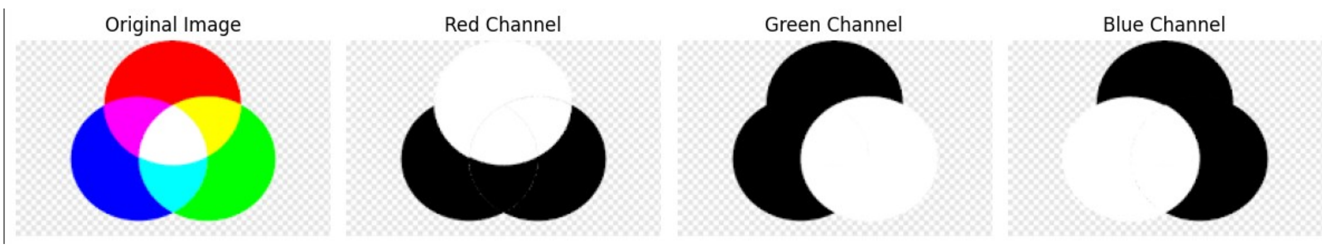
2. Read an image and display the RGB channel images separately.

Code:

```
from PIL import Image
import matplotlib.pyplot as plt

def display_rgb_channels(image_path):
    img = Image.open(image_path)

    # Convert the image to RGB mode if it's not already in RGB
    if img.mode != 'RGB':
        img = img.convert('RGB')

    # Split the image into RGB channels
    r, g, b = img.split()

    # Display original image
    plt.figure(figsize=(12, 4))

    plt.subplot(1, 4, 1)
    plt.title('Original Image')
    plt.imshow(img)
    plt.axis('off')

    # Display each channel separately
    plt.subplot(1, 4, 2)
    plt.title('Red Channel')
```

```
    plt.imshow(r, cmap='gray')
    plt.axis('off')

    plt.subplot(1, 4, 3)
    plt.title('Green Channel')
    plt.imshow(g, cmap='gray')
    plt.axis('off')

    plt.subplot(1, 4, 4)
    plt.title('Blue Channel')
    plt.imshow(b, cmap='gray')
    plt.axis('off')

    plt.tight_layout()
    plt.show()

# Provide the path to the image file
image_path = "rgb.png"  # Replace with your image file path
display_rgb_channels(image_path)
```

Output:



3. Read an image and convert it into binary image using threshold.

Code:

```
from PIL import Image
import matplotlib.pyplot as plt

def convert_to_binary(image_path, threshold_value):

img = Image.open(image_path)
img = img.convert('L')

binary_img = img.point(lambda p: 0 if p < threshold_value else 255, '1')
plt.figure(figsize=(8, 4))
plt.subplot(1, 2, 1)
plt.title('Original Image')
plt.imshow(img, cmap='gray')
```

```
plt.axis('off')
plt.subplot(1, 2, 2)
plt.title('Binary Image (Threshold={})'.format(threshold_value))
plt.imshow(binary_img, cmap='gray')
plt.axis('off')
plt.tight_layout()
plt.show()


image_path = "test1.png"
threshold = 129

convert_to_binary(image_path, threshold)
```

Output:

4. Display the histogram of the gray scale image.

Code:

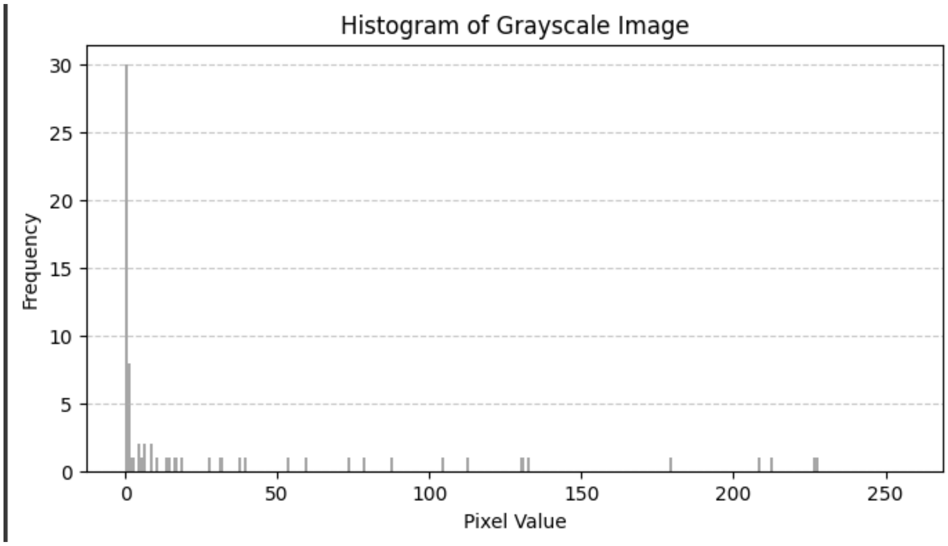
```
from PIL import Image
import matplotlib.pyplot as plt

def display_histogram(image_path):
# Open the image
img = Image.open(image_path)
# Convert the image to grayscale if it's not already in grayscale
img = img.convert('L')
```

# Calculate the histogram
histogram = img.histogram()
# Display the histogram
plt.figure(figsize=(8, 4))
plt.title('Histogram of Grayscale Image')
plt.xlabel('Pixel Value')
plt.ylabel('Frequency')
plt.hist(histogram, bins=256, range=(0, 256), color='gray', alpha=0.7)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()

# Provide the path to the image file
image_path = "grayscale_image.jpg" # Replace with your image file path

display_histogram(image_path)

Output:



5. Apply histogram equalization on an image and display the resultant image.

Code:

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

path = "test1.png"
img = cv.imread(path)

# Convert image to grayscale
gray_img = cv.cvtColor(img, cv.COLOR_BGR2GRAY)

```
# Perform histogram equalization
equ = cv.equalizeHist(gray_img)

# Plot original and equalized images
plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)
plt.imshow(gray_img, cmap='gray')
plt.title('Original Image')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(equ, cmap='gray')
plt.title('Equalized Image')
plt.axis('off')

plt.show()
```

Output:

6. Display the edge map of an image with any edge detection algorithm

Code:

```
import cv2
import matplotlib.pyplot as plt

# Read the image
image_path = "test1.png" # Replace with your image file path
img = cv2.imread(image_path, 0) # Read image in grayscale

# Apply Canny edge detection
```

```
edges = cv2.Canny(img, 100, 200) # Adjust threshold values for best results

# Plot the original and edge-detected images
plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)
plt.imshow(img, cmap='gray')
plt.title('Original Image')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(edges, cmap='gray')
plt.title('Canny Edge Detection')
plt.axis('off')

plt.show()
```

Output:

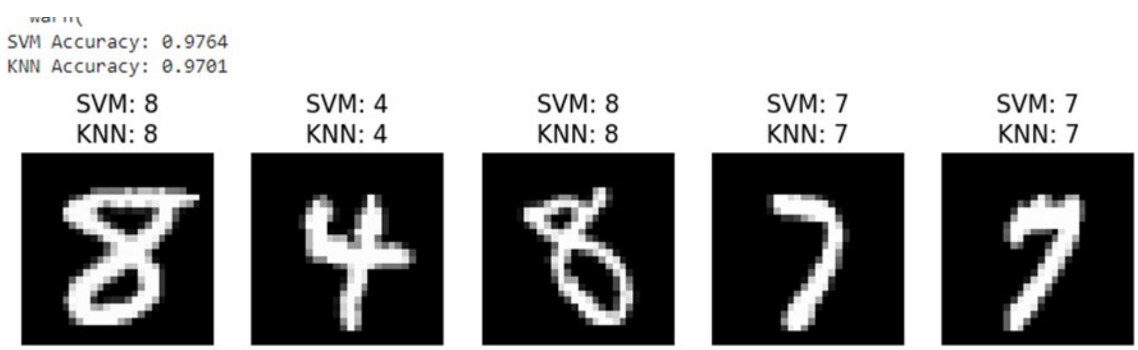
7. Download any OCR dataset and perform the classification with SVM and KNN. Compare the obtained result

Code:

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn import svm, neighbors, metrics
from sklearn.datasets import fetch_openml
import matplotlib.pyplot as plt

mnist = fetch_openml('mnist_784', version=1)
X = np.array(mnist.data.astype('int'))
```

```
y = np.array(mnist.target.astype('int'))
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

svm_classifier = svm.SVC()
svm_classifier.fit(X_train, y_train)
svm_predictions = svm_classifier.predict(X_test)

knn_classifier = neighbors.KNeighborsClassifier(n_neighbors=5)
knn_classifier.fit(X_train, y_train)
knn_predictions = knn_classifier.predict(X_test)

svm_accuracy = metrics.accuracy_score(y_test, svm_predictions)
knn_accuracy = metrics.accuracy_score(y_test, knn_predictions)

print(f"SVM Accuracy: {svm_accuracy:.4f}")
print(f"KNN Accuracy: {knn_accuracy:.4f}")

fig, axes = plt.subplots(1, 5, figsize=(10, 2))

for i, ax in enumerate(axes):
    ax.imshow(X_test[i].reshape(28, 28), cmap='gray')
    ax.set_title(f'SVM: {svm_predictions[i]}\nKNN: {knn_predictions[i]}')
    ax.axis('off')

plt.show()
```

Output:



8. Implement any two segmentation algorithms and compare the efficiency with ground truth

Code:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.cluster import MeanShift, estimate_bandwidth
from sklearn.datasets import make_blobs
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import normalized_mutual_info_score

# Create a synthetic ground truth image
np.random.seed(42)
ground_truth, _ = make_blobs(n_samples=300, centers=3, random_state=42, cluster_std=2.0)
ground_truth = StandardScaler().fit_transform(ground_truth)

# Generate image with ground truth labels
ground_truth_labels = np.argmax(ground_truth, axis=1)
ground_truth_labels = ground_truth_labels.reshape((10, 30))

# Plot ground truth
plt.figure(figsize=(12, 4))
plt.subplot(1, 3, 1)
plt.imshow(ground_truth_labels, cmap='viridis')
plt.title('Ground Truth')
plt.axis('off')

# Generate image data for clustering
data, _ = make_blobs(n_samples=300, centers=3, random_state=42, cluster_std=2.0)

# Apply K-Means clustering
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans_labels = kmeans.fit_predict(data)
kmeans_labels = kmeans_labels.reshape((10, 30))

# Plot K-Means clustering result
plt.subplot(1, 3, 2)
plt.imshow(kmeans_labels, cmap='viridis')
plt.title('K-Means Clustering')
plt.axis('off')

# Apply Mean Shift clustering
bandwidth = estimate_bandwidth(data, quantile=0.2, n_samples=300)
meanshift = MeanShift(bandwidth=bandwidth, bin_seeding=True)
meanshift_labels = meanshift.fit_predict(data)
meanshift_labels = meanshift_labels.reshape((10, 30))
```

```
# Plot Mean Shift clustering result
plt.subplot(1, 3, 3)
plt.imshow(meanshift_labels, cmap='viridis')
plt.title('Mean Shift Clustering')
plt.axis('off')

plt.tight_layout()
plt.show()

# Compare clustering results with ground truth
nmi_kmeans = normalized_mutual_info_score(ground_truth_labels.flatten(), kmeans_labels.flatten())
nmi_meanshift = normalized_mutual_info_score(ground_truth_labels.flatten(),
meanshift_labels.flatten())

print(f"Normalized Mutual Information (NMI) - K-Means: {nmi_kmeans:.4f}")
print(f"Normalized Mutual Information (NMI) - Mean Shift: {nmi_meanshift:.4f}")
```

Output:


```
Normalized Mutual Information (NMI) - K-Means: 0.5325
Normalized Mutual Information (NMI) - Mean Shift: 0.5229
```

9. Input an image and perform the following morphological operations

i) Dilation
ii) Erosion
iii) Opening
iv) Closing

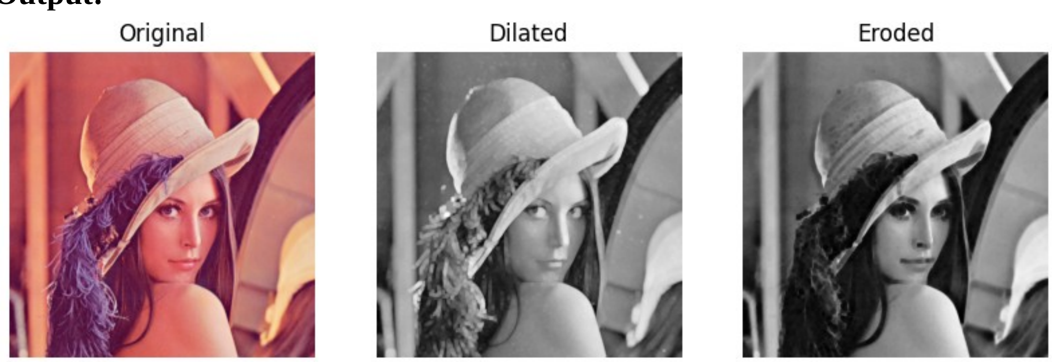
Display the results.

Code:

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

# Read the image
image = cv2.imread('test1.png')  # Replace with your image file

# Convert the image to grayscale
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Define the kernel for morphological operations
```

```
kernel = np.ones((5, 5), np.uint8)  # You can adjust the size of the kernel as needed

# Perform morphological operations
dilated_image = cv2.dilate(gray_image, kernel, iterations=1)
eroded_image = cv2.erode(gray_image, kernel, iterations=1)
opened_image = cv2.morphologyEx(gray_image, cv2.MORPH_OPEN, kernel)
closed_image = cv2.morphologyEx(gray_image, cv2.MORPH_CLOSE, kernel)

# Display the original and processed images
plt.figure(figsize=(10, 10))

plt.subplot(2, 3, 1)
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.title('Original')
plt.axis('off')

plt.subplot(2, 3, 2)
plt.imshow(dilated_image, cmap='gray')
plt.title('Dilated')
plt.axis('off')

plt.subplot(2, 3, 3)
plt.imshow(eroded_image, cmap='gray')
plt.title('Eroded')
plt.axis('off')

plt.subplot(2, 3, 4)
plt.imshow(opened_image, cmap='gray')
plt.title('Opened')
plt.axis('off')

plt.subplot(2, 3, 5)
plt.imshow(closed_image, cmap='gray')
plt.title('Closed')
plt.axis('off')

plt.show()
```

Output:



Opened

Closed

10. Implement any image restoration algorithm

Code:

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

def median_filter(data, filter_size):
    temp = []
    indexer = filter_size // 2
    data_final = np.zeros_like(data)
    for i in range(len(data)):
        for j in range(len(data[0])):
            for z in range(filter_size):
                if i + z - indexer < 0 or i + z - indexer > len(data) - 1:
                    for c in range(filter_size):
                        temp.append(0)
                else:
                    if j + z - indexer < 0 or j + indexer > len(data[0]) - 1:
                        temp.append(0)
                    else:
                        for k in range(filter_size):
                            temp.append(data[i + z - indexer][j + k - indexer])

            temp.sort()
            data_final[i][j] = temp[len(temp) // 2]
            temp = []
    return data_final

img = cv2.imread("test2.png", cv2.IMREAD_GRAYSCALE)
removed_noise = median_filter(img, 3)

# Plot original and restored images
plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)
plt.imshow(img, cmap='gray')
plt.title('Original Image')
```

```
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(removed_noise, cmap='gray')
plt.title('Restored Image')
plt.axis('off')

plt.show()
```

Output: