Name = Hemangi Budhathoki

Student Id= 2357944

2.1 Exercise - 1:

Complete all the Task.

1. Read and display the image. • Read the image using the Pillow library and display it. • You can also use matplotlib to display the image.

```
!pip install pillow
```

```
Requirement already satisfied: pillow in /usr/local/lib/python3.11/dist-packages (11.1.(
```

```
from PIL import Image
```

```
picture=Image.open("/content/drive/MyDrive/AI and ML college 6 sem/lenna_image.png")
```

```
picture
```

```
print("Format",picture.format)
print("Mode:", picture.mode)
print("size",picture.size)
```

```
Format PNG
Mode: RGBA
size (366, 357)
```

```
picture=picture.convert("RGB")
print("Mode",picture.mode)
```

```
Mode RGB
```

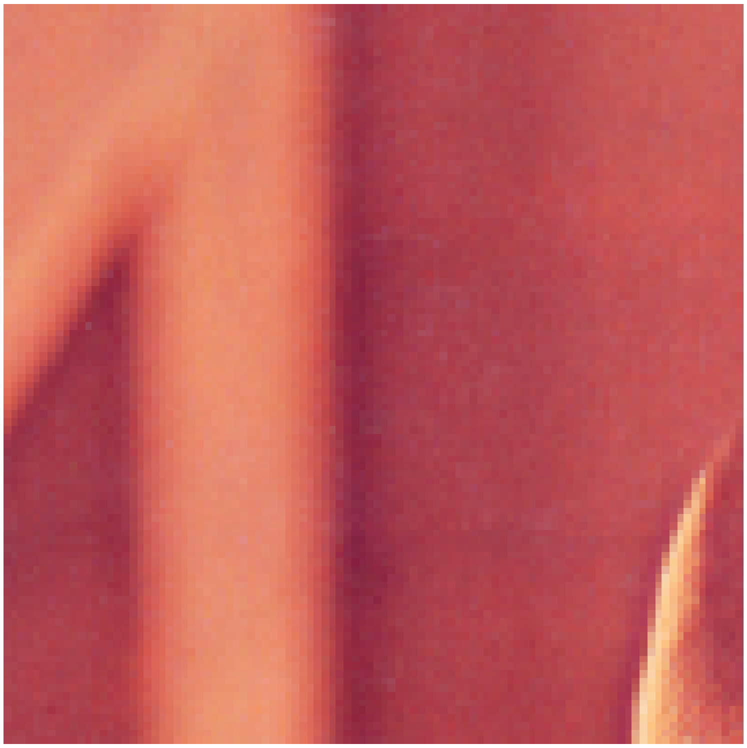
2. Display only the top left corner of 100x100 pixels.

• Extract the top-left corner of the image (100x100 pixels) and display it using NumPy and Array Indexing.

```
#convert image into numpy array
import numpy as np
import matplotlib.pyplot as plt
picture_array=np.array(picture)
```

```
top_left_cornerp=picture_array[:100,:100]

plt.imshow(top_left_cornerp)
plt.axis('off')
plt.show()
```



3. Show the three color channels (R, G, B).

• Separate the image into its three color channels (Red, Green, and Blue) and display them individually, labeling each channel as R, G, and B.{Using NumPy.}

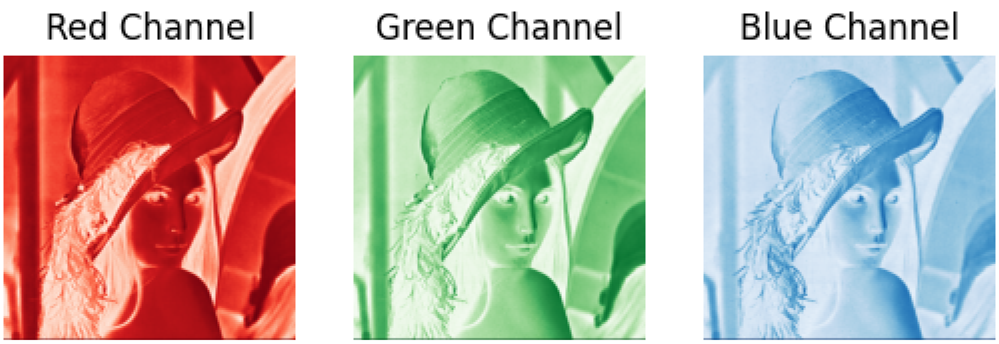
```
red_channel=picture_array[:,:,0]
green_channel=picture_array[:,:,1]
blue_channel=picture_array[:,:,2]

plt.subplot(1, 3, 1)
plt.imshow(red_channel, cmap='Reds')  # Use 'Reds' colormap for red channel
plt.title('Red Channel')
plt.axis('off')

# Green Channel
plt.subplot(1, 3, 2)
plt.imshow(green_channel, cmap='Greens')  # Use 'Greens' colormap for green channel
plt.title('Green Channel')
plt.axis('off')

# Blue Channel
plt.subplot(1, 3, 3)
plt.imshow(blue_channel, cmap='Blues')  # Use 'Blues' colormap for blue channel
plt.title('Blue Channel')
plt.axis('off')
```

```
plt.show()
```




```
# Step 2: Modify the top-left 100x100 region to have a value of 210
picture_array[:100, :100] = 210  # Set all channels (R, G, B) to 210 in the top-left 100x10(

# Step 3: Display the modified image
plt.imshow(picture_array)
plt.axis('off')  # Hide axis
plt.title('Modified Image (Top 100x100 Pixels = 210)')
plt.show()
```

2.2 Exercise - 2:

Complete all the Task.

1. Load and display a grayscale image. • Load a grayscale image using the Pillow library. • Display the grayscale image using matplotlib.

```
image_gray=Image.open("/content/drive/MyDrive/AI and ML college 6 sem/cameraman.png")
```

```
# Step 3: Display the grayscale image using matplotlib
plt.imshow(image_gray, cmap='gray')  # Use 'gray' colormap for grayscale images
plt.axis('off')
plt.title('Grayscale Image')
plt.show()
```

Grayscale Image

2. Extract and display the middle section of the image (150 pixels).

• Extract a 150 pixel section from the center of the image using NumPy array slicing. • Display this cropped image using matplotlib.

```
gray_image_array=np.array(image_gray)
# Step 3: Calculate the starting and ending indices for the center 150x150 region
height, width = gray_image_array.shape[:2]  # Get the height and width of the image
start_x = (width // 2) - 75  # Starting x-coordinate (center - 75)
```

```
start_y = (height // 2) - 75  # Starting y-coordinate (center - 75)
end_x = start_x + 150  # Ending x-coordinate
end_y = start_y + 150  # Ending y-coordinate

# Step 4: Extract the middle 150x150 section using NumPy array slicing
middle_section = gray_image_array[start_y:end_y, start_x:end_x]

# Step 5: Display the cropped image using matplotlib
plt.imshow(middle_section)
plt.axis('off')  # Hide axis
plt.title('Middle 150x150 Section')
plt.show()
```

Middle 150x150 Section

3. Apply a simple threshold to the image (e.g., set all pixel values below 100 to 0).

• Apply a threshold to the grayscale image: set all pixel values below 100 to 0, and all values above 100 to 255 (creating a binary image). • Display the resulting binary image.

```
import cv2

threshold_value = 100
binary_image = np.where(gray_image_array >= threshold_value, 255, 0)  # Set values >= 100 tc
```

```
# Step 4: Display the binary image
plt.imshow(binary_image, cmap='gray')  # Use 'gray' colormap for binary images
plt.axis('off')
plt.title('Binary Image (Threshold = 100)')
plt.show()
```

Binary Image (Threshold = 100)

4. Rotate the image 90 degrees clockwise and display the result.

• Rotate the image by 90 degrees clockwise using the Pillow rotate method or by manipulating the image array. • Display the rotated image using matplotlib.

```
rotate_image=image_gray.rotate(-90, expand=True)
plt.imshow(rotate_image)
plt.title("Rotated 90 Degrees Clockwise")
plt.axis("off")
plt.show()
```

Rotated 90 Degrees Clockwise

```
rgb_array = np.stack([gray_image_array] * 3, axis=-1)

# Step 4: Display the converted RGB image
plt.imshow(rgb_array)
plt.axis('off')  # Hide axis
plt.title('Converted RGB Image')
plt.show()
```

Converted RGB Image

3 Image Compression and Decompression using PCA.

In this exercise, build a PCA from scratch using explained variance method for image compression task. You are expected to compute the necessary matrices from the scratch. Dataset: Use image of your choice.

1. Load and Prepare Data: • Fetch an image of you choice.{If colour convert to grayscale} • Center the dataset - Standaridze the Data. • Calculate the covaraince matrix of the Standaridze data.

```
# "Step 1: Fetch an Image"
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Load an image (replace 'your_image.jpg' with the path to your image)
image = cv2.imread('/content/drive/MyDrive/AI and ML college 6 sem/lenna_image.png')

# Convert to grayscale if it's a color image
if len(image.shape) == 3:
    image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Display the grayscale image
plt.imshow(image, cmap='gray')
```

```
plt.title('Grayscale Image')
plt.axis('off')
plt.show()
```

Grayscale Image

```
# Flatten the image into a 1D array
flattened_image = image.flatten()

# Center the data by subtracting the mean
mean_value = np.mean(flattened_image)
centered_data = flattened_image - mean_value

# Reshape the centered data back to the original image shape (for visualization)
centered_image = centered_data.reshape(image.shape)

# Display the centered image
plt.imshow(centered_image, cmap='gray')
plt.title('Centered Image')
plt.axis('off')
plt.show()
```

Centered Image

```
# Step 3: Calculate the Covariance Matrix
# Reshape the centered data into a 2D array (each row is a sample, each column is a feature)
# For an image, each pixel is a feature, so we reshape it to (num_pixels, 1)
data_matrix = image.reshape(-1, 1)

# Calculate the covariance matrix
covariance_matrix = np.cov(data_matrix, rowvar=False)

print("Covariance Matrix:")
print(covariance_matrix)
```

```
Covariance Matrix:
2347.7310811378093
```

```
print(covariance_matrix.shape)
```

```
()
```

```
covariance_matrix = np.cov(image, rowvar=False)
```

```
covariance_matrix = np.array(covariance_matrix, ndmin=2)
```

```
# Compute eigenvalues and eigenvectors of the covariance matrix
eigenvalues, eigenvectors = np.linalg.eig(covariance_matrix)

# Eigenvalues and eigenvectors are returned in ascending order, so we sort them in descendir
sorted_indices = np.argsort(eigenvalues)[::-1]
eigenvalues = eigenvalues[sorted_indices]
eigenvectors = eigenvectors[:, sorted_indices]

print("Eigenvalues:")
print(eigenvalues)
print("\nEigenvectors:")
print(eigenvectors)
```

```
 [ 0.03775167+0.j          0.01998026+0.j         -0.02721533+0.j
  ...  0.04485696+0.06565299j  0.0516691 +0.j
  -0.01213738+0.j        ]
 [ 0.037324  +0.j          0.01674716+0.j         -0.0248318 +0.j
  ...  0.10002072-0.00614088j  0.08202038+0.j
   0.00514847+0.j        ]
 ...
 [-0.0731084 +0.j          0.1165575 +0.j          0.03536637+0.j
  ...  0.08217545+0.05212653j -0.05098893+0.j
  -0.02253369+0.j        ]
 [-0.07142531+0.j          0.11716339+0.j          0.03261266+0.j
  ... -0.07503788-0.06403603j  0.01302903+0.j
  -0.00429886+0.j        ]
 [-0.06948438+0.j          0.11869226+0.j          0.03031827+0.j
  ... -0.03480594+0.02009475j -0.04839693+0.j
   0.08014746+0.j        ]]
```

```
# Choose the top k eigenvectors (for example, k=1)
k = 1
top_k_eigenvectors = eigenvectors[:, :k]

print("Top k Eigenvectors:")
print(top_k_eigenvectors)
```

```
       [-0.09974658+0.j]
       [-0.09808794+0.j]
       [-0.0960504 +0.j]
       [-0.09474278+0.j]
       [-0.09295809+0.j]
       [-0.0914196 +0.j]
       [-0.09063382+0.j]
       [-0.08924722+0.j]
       [-0.08875622+0.j]
       [-0.08711876+0.j]
       [-0.08594058+0.j]
       [-0.08488834+0.j]
       [-0.08367183+0.j]
       [-0.08173681+0.j]
       [-0.08027779+0.j]
       [-0.07952248+0.j]
       [-0.0787635 +0.j]
       [-0.07810469+0.j]
       [-0.07743258+0.j]
       [-0.07788763+0.j]
       [-0.0772662 +0.j]
       [-0.07591752+0.j]
       [-0.07459012+0.j]
       [-0.07311844+0.j]
       [-0.0731084 +0.j]
       [-0.07142531+0.j]
       [-0.06948438+0.j]]
```

```
# Calculate the explained variance ratio
explained_variance_ratio = eigenvalues / np.sum(eigenvalues)

# Calculate the cumulative explained variance
cumulative_explained_variance = np.cumsum(explained_variance_ratio)

# Plot the cumulative explained variance
plt.plot(cumulative_explained_variance, marker='o')
plt.xlabel('Number of Principal Components')
plt.ylabel('Cumulative Explained Variance')
plt.title('Cumulative Explained Variance Plot')
plt.grid()
plt.show()
```

```
/usr/local/lib/python3.11/dist-packages/matplotlib/cbook.py:1709: ComplexWarning: Castir
  return math.isfinite(val)
/usr/local/lib/python3.11/dist-packages/matplotlib/cbook.py:1345: ComplexWarning: Castir
  return np.asarray(x, float)
```