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

import cv2
# read image
im = cv2.imread("elephant.jpg")
h,w = im.shape[:2]
print(h,w)
# save image
cv2.imwrite("result.png" ,im)
```

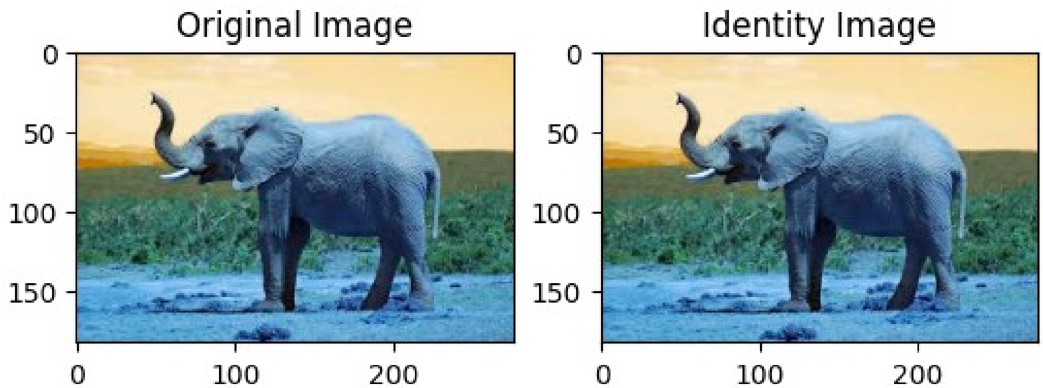
```
183 276
```

```
True
```

Ch1: Difference between original and identity image

```python
image = cv2.imread("elephant.jpg")
"""
Apply identity kernel
"""
# import imshow
kernel1 = np.array([[0, 0, 0],
                    [0, 1, 0],
                    [0, 0, 0]])
# filter2D() function can be used to apply kernel to an image.
# Where ddepth is the desired depth of final image. ddepth is -1 if...
# ... depth is same as original or source image.
identity = cv2.filter2D(src=image, ddepth=-1, kernel=kernel1)

# We should get the same image
plt.subplot(1, 2, 1)
plt.imshow(image)
plt.title('Original Image')

plt.subplot(1, 2, 2)
plt.imshow(identity)
plt.title('Identity Image')
cv2.imwrite('identity.jpg', identity)
```

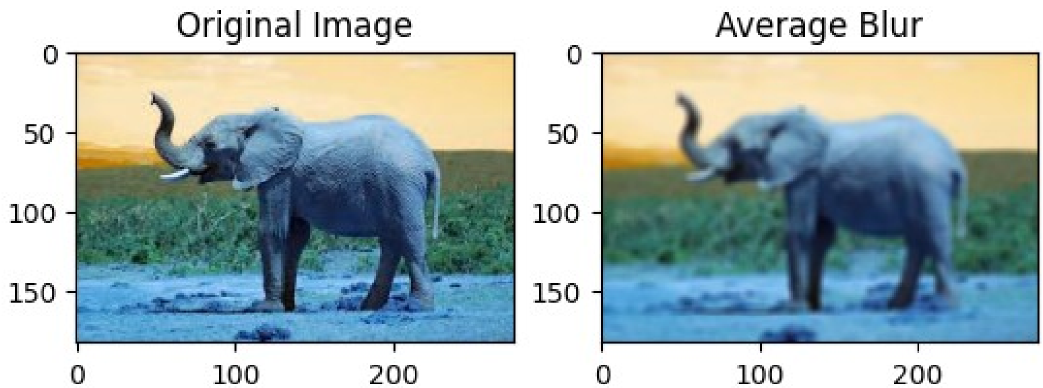
```
True
```



EXERCISE2:

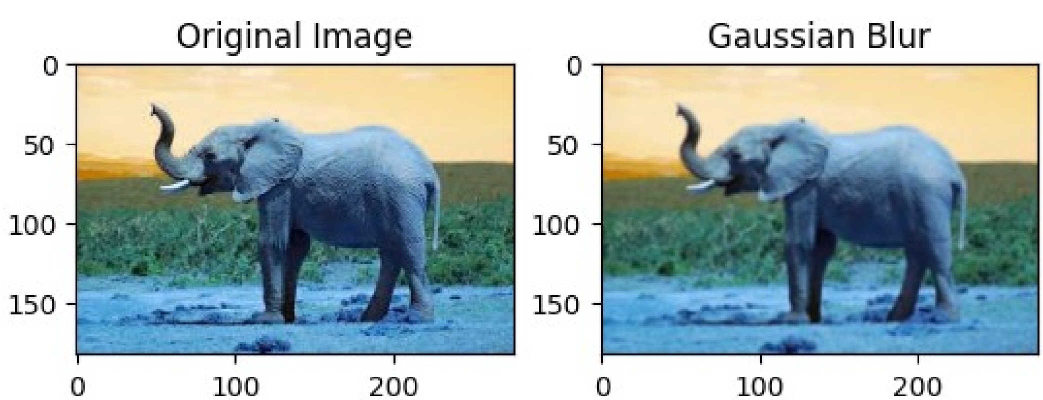
Explore image smoothing techniques to reduce noise and improve image quality. Averaging Gaussian Blur Median blur Bilateral blur

```
#AVERAGING
# Importing the modules
import cv2
import numpy as np

# Reading the image
image = cv2.imread('elephant.jpg')

# Applying the filter
averageBlur = cv2.blur(image, (5, 5))

# Showing the image
# cv2.imshow('Original', image)
# cv2.imshow('Average blur', averageBlur)

plt.subplot(1, 2, 1)
plt.imshow(image)
plt.title('Original Image')

plt.subplot(1, 2, 2)
plt.imshow(averageBlur)
plt.title('Average Blur')

# Show the plot
plt.show()
```



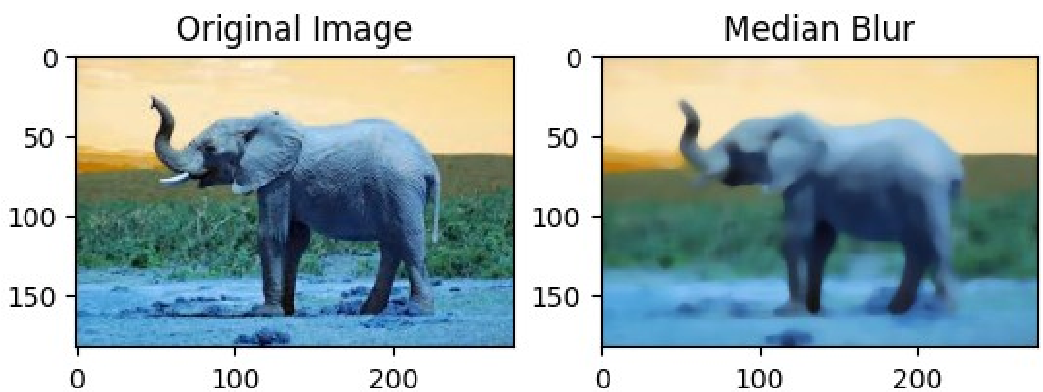

```python
#GAUSSIAN BLUR
# Importing the module
import cv2
import numpy as np

# Reading the image
image = cv2.imread('elephant.jpg')

# Applying the filter
gaussian = cv2.GaussianBlur(image, (3, 3), 0)

# Showing the image
plt.subplot(1, 2, 1)
plt.imshow(image)
plt.title('Original Image')

plt.subplot(1, 2, 2)
plt.imshow(gaussian)
plt.title('Gaussian Blur')
```

```
Text(0.5, 1.0, 'Gaussian Blur')
```



```
#MEDIAN BLUR
# Importing the modules
import cv2
import numpy as np

# Reading the image
image = cv2.imread('elephant.jpg')

# Applying the filter
medianBlur = cv2.medianBlur(image, 9)

# Showing the image
plt.subplot(1, 2, 1)
plt.imshow(image)
plt.title('Original Image')

plt.subplot(1, 2, 2)
plt.imshow(medianBlur)
plt.title('Median Blur')
```

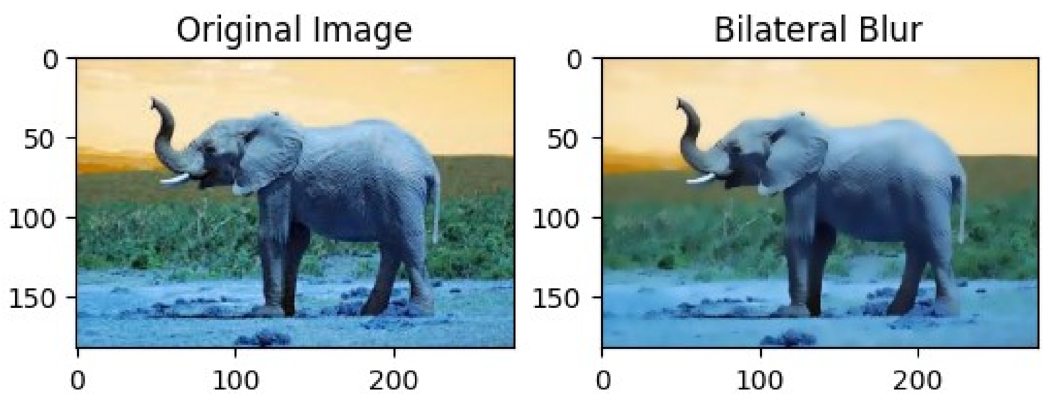
```
Text(0.5, 1.0, 'Median Blur')
```




```
#BILATERAL BLUR
# Importing the modules
import cv2
import numpy as np

# Reading the image
image = cv2.imread('elephant.jpg')

# Applying the filter
bilateral = cv2.bilateralFilter(image, 9, 75, 75)

# Showing the image
plt.subplot(1, 2, 1)
plt.imshow(image)
plt.title('Original Image')
```

```
plt.subplot(1, 2, 2)
plt.imshow(bilateral)
plt.title('Bilateral Blur')
```

```
Text(0.5, 1.0, 'Bilateral Blur')
```

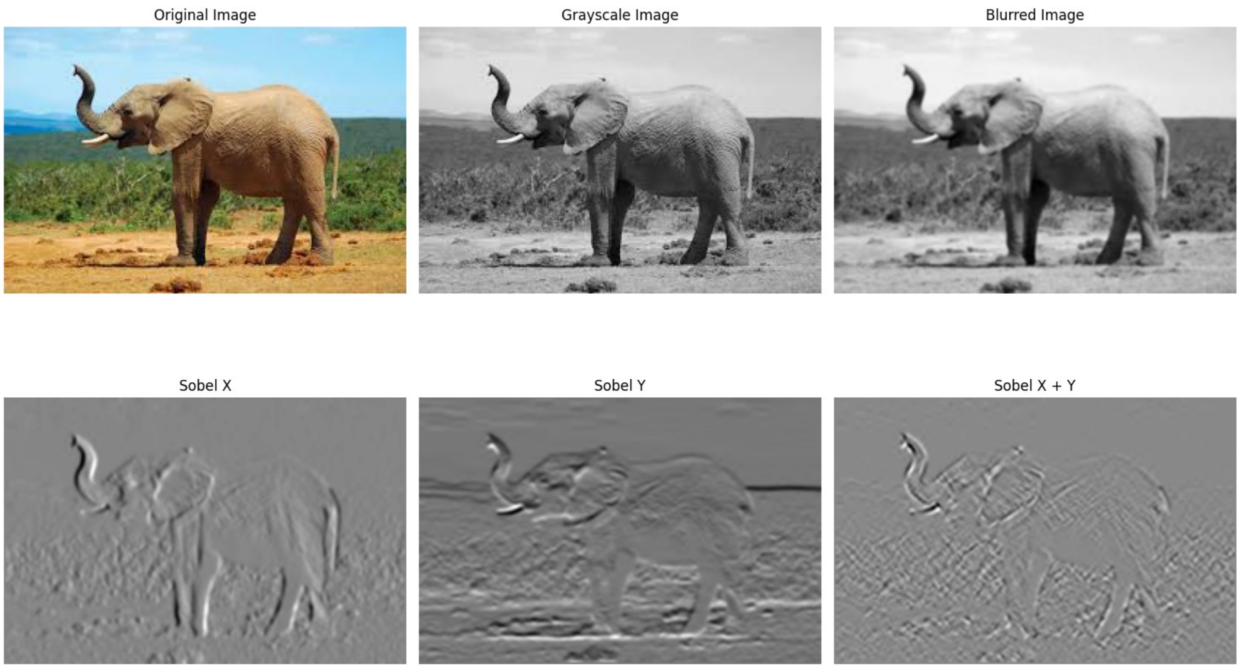


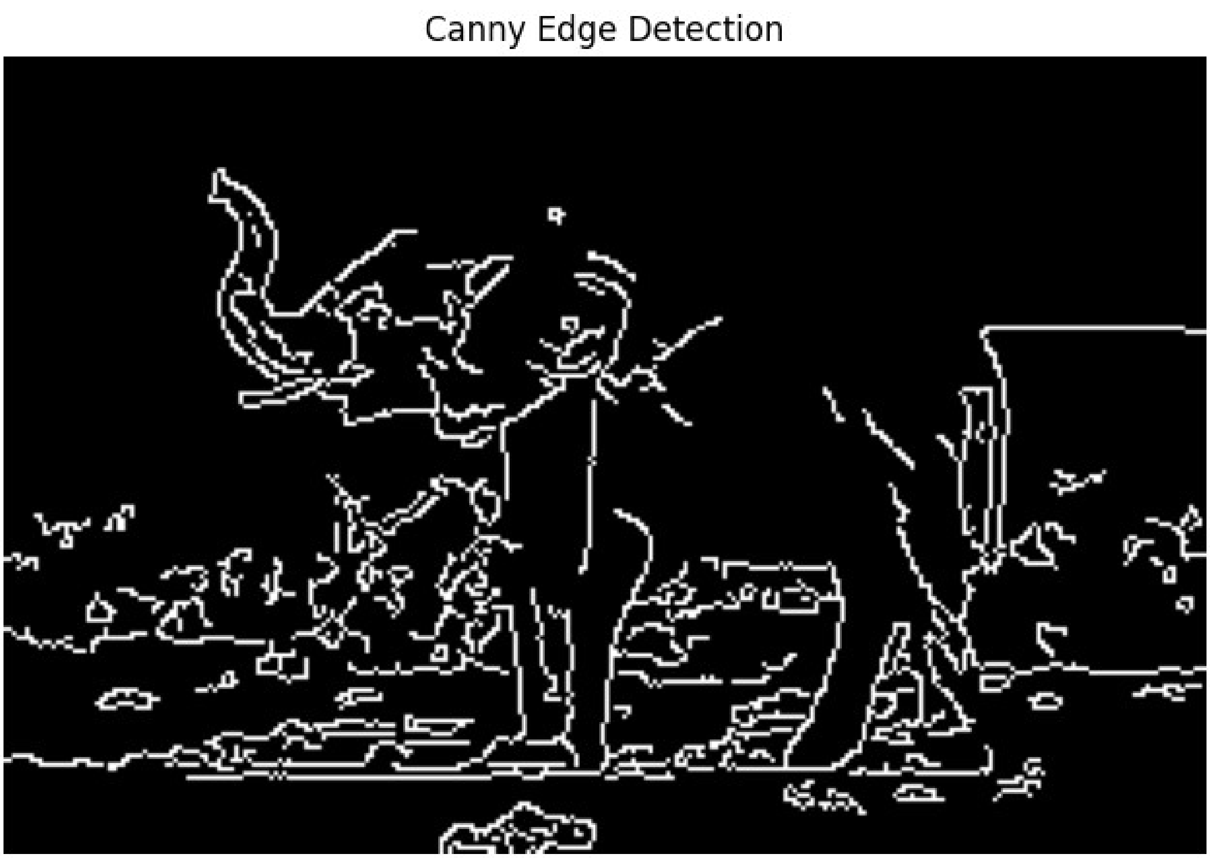
EXERCISE3:

Detect edges in an image using different edge detection algorithms.

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

# Read the original image
img = cv2.imread('elephant.jpg')

# Convert from BGR to RGB for displaying using matplotlib
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

# Convert to grayscale
img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Blur the image for better edge detection
img_blur = cv2.GaussianBlur(img_gray, (3, 3), 0)

# Sobel Edge Detection
sobelx = cv2.Sobel(src=img_blur, ddepth=cv2.CV_64F, dx=1, dy=0,
ksize=5)  # Sobel Edge Detection on X axis
sobely = cv2.Sobel(src=img_blur, ddepth=cv2.CV_64F, dx=0, dy=1,
ksize=5)  # Sobel Edge Detection on Y axis
sobelxy = cv2.Sobel(src=img_blur, ddepth=cv2.CV_64F, dx=1, dy=1,
ksize=5)  # Combined X and Y Sobel Edge Detection

# Canny Edge Detection
```

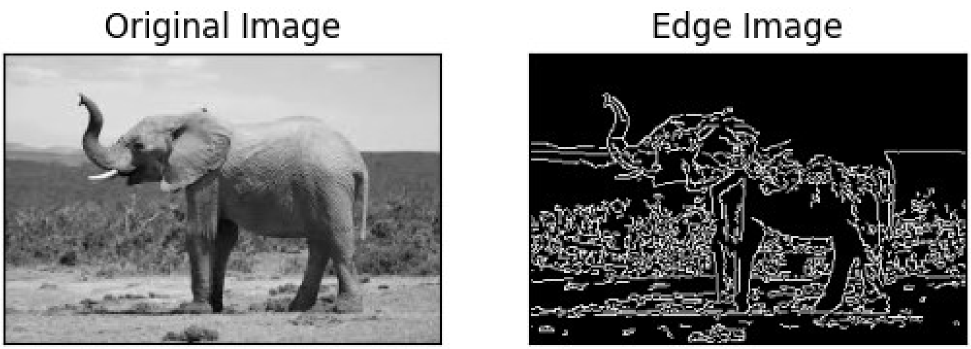
```python
edges = cv2.Canny(image=img_blur, threshold1=100, threshold2=200)

# Plot the images using matplotlib
fig, axs = plt.subplots(2, 3, figsize=(15, 10))

# Display the original image
axs[0, 0].imshow(img_rgb)
axs[0, 0].set_title('Original Image')
axs[0, 0].axis('off')

# Display the grayscale image
axs[0, 1].imshow(img_gray, cmap='gray')
axs[0, 1].set_title('Grayscale Image')
axs[0, 1].axis('off')

# Display the blurred image
axs[0, 2].imshow(img_blur, cmap='gray')
axs[0, 2].set_title('Blurred Image')
axs[0, 2].axis('off')

# Display Sobel X
axs[1, 0].imshow(sobelx, cmap='gray')
axs[1, 0].set_title('Sobel X')
axs[1, 0].axis('off')

# Display Sobel Y
axs[1, 1].imshow(sobely, cmap='gray')
axs[1, 1].set_title('Sobel Y')
axs[1, 1].axis('off')

# Display Sobel X and Y combined
axs[1, 2].imshow(sobelxy, cmap='gray')
axs[1, 2].set_title('Sobel X + Y')
axs[1, 2].axis('off')

# Adjust layout and show the plot
plt.tight_layout()
plt.show()

# Plot the Canny Edge Detection result separately
plt.figure(figsize=(8, 8))
plt.imshow(edges, cmap='gray')
plt.title('Canny Edge Detection')
plt.axis('off')
plt.show()
```






```python
#CANNY
import numpy as np
```

```
import cv2 as cv
from matplotlib import pyplot as plt

img = cv.imread('elephant.jpg', cv.IMREAD_GRAYSCALE)
assert img is not None, "file could not be read, check with
os.path.exists()"
edges = cv.Canny(img,100,200)

plt.subplot(121),plt.imshow(img,cmap = 'gray')
plt.title('Original Image'), plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(edges,cmap = 'gray')
plt.title('Edge Image'), plt.xticks([]), plt.yticks([])

plt.show()
```



Write a small application to find the Canny edge detection whose threshold values can be varied using two trackbars.

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
from ipywidgets import interact, IntSlider

# Load the image in grayscale
image = cv2.imread('elephant.jpg', 0)

# Function to update the Canny Edge Detection based on threshold
values
def update_canny(min_val, max_val):
    # Apply Canny edge detection
    edges = cv2.Canny(image, min_val, max_val)

    # Display the result
    plt.figure(figsize=(6,6))
    plt.imshow(edges, cmap='gray')
    plt.title(f'Canny Edge Detection (Min: {min_val}, Max:
{max_val})')
    plt.axis('off')
```

```
    plt.show()

# Create sliders for minimum and maximum threshold values
interact(update_canny,
         min_val=IntSlider(min=0, max=255, step=1, value=100,
description='Min Threshold'),
         max_val=IntSlider(min=0, max=255, step=1, value=200,
description='Max Threshold'));
```

{"model_id":"a00a82ea0ff14555a711db16b76320ab","version_major":2,"version_minor":0}

EXERCISE: Develop a program that demonstrates how K means clustering algorithm can be used to group pixels into meaningful segments based on color similarity.

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
from ipywidgets import interact, IntSlider

# Load the image
image = cv2.imread('elephant.jpg')  # replace 'sha3.jpeg' with
'elephant.jpg'

# Convert the image from BGR to RGB (OpenCV loads images in BGR by
default)
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# Reshape the image into a 2D array of pixels, where each pixel has 3
color values (RGB)
pixels = image_rgb.reshape((-1, 3))

# Convert to float32, required for K-means
pixels = np.float32(pixels)

# Define criteria for the K-means algorithm (type of termination
criteria, max iterations, epsilon)
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 100,
0.2)

# Function to apply K-means clustering and display the segmented image
def apply_kmeans(k):
    if k < 2:
        k = 2  # Minimum number of clusters

    # Apply K-means clustering
    _, labels, centers = cv2.kmeans(pixels, k, None, criteria, 10,
cv2.KMEANS_RANDOM_CENTERS)
```

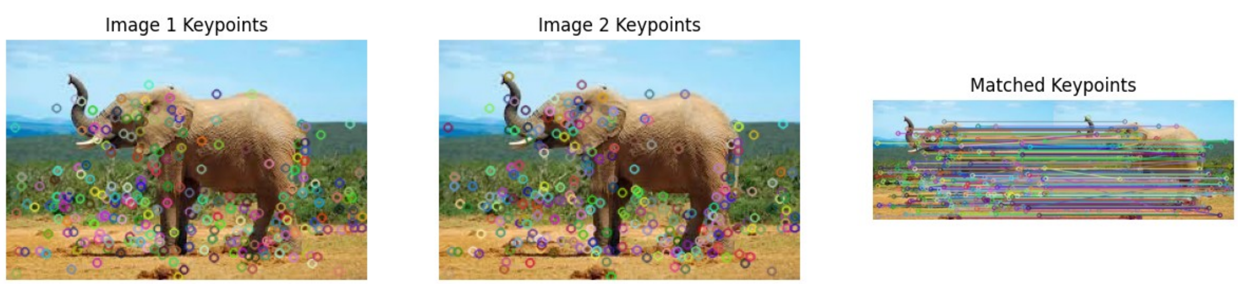
```
    # Convert the centers to uint8 (since they are colors)
    centers = np.uint8(centers)

    # Map each pixel to the color of its corresponding cluster center
    segmented_image = centers[labels.flatten()]

    # Reshape the image to its original dimensions
    segmented_image = segmented_image.reshape(image_rgb.shape)

    # Display the segmented image using matplotlib
    plt.figure(figsize=(10, 6))
    plt.subplot(1, 2, 1)
    plt.imshow(image_rgb)
    plt.title('Original Image')
    plt.axis('off')

    plt.subplot(1, 2, 2)
    plt.imshow(segmented_image)
    plt.title(f'Segmented Image with {k} Clusters')
    plt.axis('off')

    plt.show()

# Create an interactive slider to adjust the number of clusters (k)
interact(apply_kmeans, k=IntSlider(min=2, max=10, step=1, value=4,
description='Clusters'));
```


```
{"model_id":"bb2d39c20a784bf190063cc3b0831e49","version_major":2,"vers
ion_minor":0}
```


Implement SIFT algorithm to detect and match keypoints between two images.

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

# Load images
img1 = cv2.imread('elephant.jpg')
img2 = cv2.imread('identity.jpg')

# Convert to grayscale
gray1 = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
gray2 = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)

# Create SIFT object
sift = cv2.SIFT_create()

# Detect keypoints and extract descriptors
```

```
kp1, des1 = sift.detectAndCompute(gray1, None)
kp2, des2 = sift.detectAndCompute(gray2, None)

# Draw keypoints
img1_kp = cv2.drawKeypoints(img1, kp1, None)
img2_kp = cv2.drawKeypoints(img2, kp2, None)

# Create matcher
matcher = cv2.BFMatcher()
matches = matcher.match(des1, des2)

# Draw matches
match_img = cv2.drawMatches(img1, kp1, img2, kp2, matches, None)

# Display images using matplotlib
plt.figure(figsize=(15, 10))

plt.subplot(1, 3, 1)
plt.imshow(cv2.cvtColor(img1_kp, cv2.COLOR_BGR2RGB))
plt.title('Image 1 Keypoints')
plt.axis('off')

plt.subplot(1, 3, 2)
plt.imshow(cv2.cvtColor(img2_kp, cv2.COLOR_BGR2RGB))
plt.title('Image 2 Keypoints')
plt.axis('off')

plt.subplot(1, 3, 3)
plt.imshow(cv2.cvtColor(match_img, cv2.COLOR_BGR2RGB))
plt.title('Matched Keypoints')
plt.axis('off')

plt.show()
```



EXERCISE: Implement image resizing, clustering (K-Means), and recognition (SVM) to classify images into predefined categories.