Experiment - 1

Student Name: Kush Bhasin
UID: 20BCS7677
Branch: CSE
Section/Group: 20BCS-1
Semester: 7th
Date of Performance: 02/08/2023
Subject Name: Computer Vision Lab
Subject Code: 20CSP- 422

Aim: Write a program to implement various feature extraction techniques for image classification

Objective: The objective of this experiment is to extract descriptive features from an image using different feature extraction techniques: Color Histogram, SIFT, ORB, and HOG.

Feature Extraction Techniques:

• SIFT:

i. Description: Scale-Invariant Feature Transform (SIFT) is a feature extraction technique that identifies key points in an image and computes descriptors based on the local image gradients, which are invariant to scale and rotation.

ii. Steps:

1. Import necessary libraries.
2. Load the image using OpenCV.
3. Define a function to extract SIFT features.
4. Create a SIFT object.
5. Detect key points and compute descriptors using sift.detectAndCompute.
6. Return the descriptors as features.

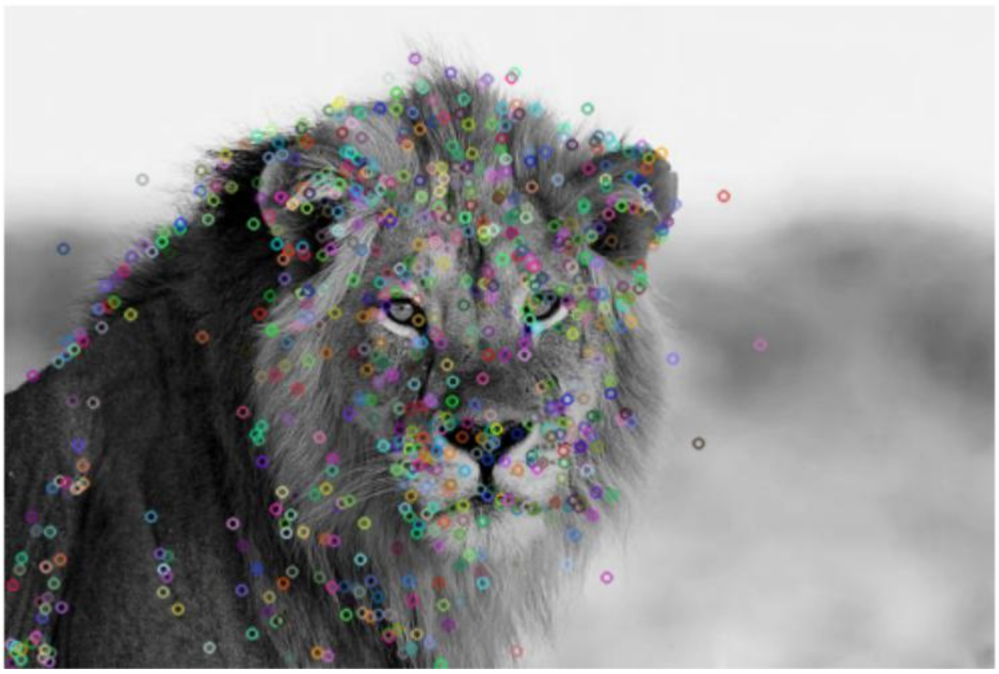
iii. Code:

```
import cv2
from matplotlib import pyplot as plt
import pandas as pd
import numpy as np
from skimage.io import imread,imshow
from skimage.transform import resize
from skimage import io, color
from skimage.feature import hog
# Load an image
gray_img = imread('/content/lion.jpg',as_gray=True)

# Convert to 8-bit grayscale
gray_img = np.uint8(gray_img * 255)
# Function to extract SIFT features
sift = cv2.SIFT_create()
```

```python
# Display or use the extracted features
keypoints, descriptors = sift.detectAndCompute(gray_img, None)
# Draw keypoints on the image
img_with_keypoints = cv2.drawKeypoints(gray_img, keypoints, None)
plt.axis('off')
plt.imshow(img_with_keypoints, cmap='gray')
plt.show()
```

Output;



ORB:

i. Description: Oriented FAST and Rotated BRIEF (ORB) is a feature extraction and matching technique commonly used in computer vision and image processing. It is designed to identify key points or interest points in an image and compute descriptors that represent the local image structure.

ii. Steps:

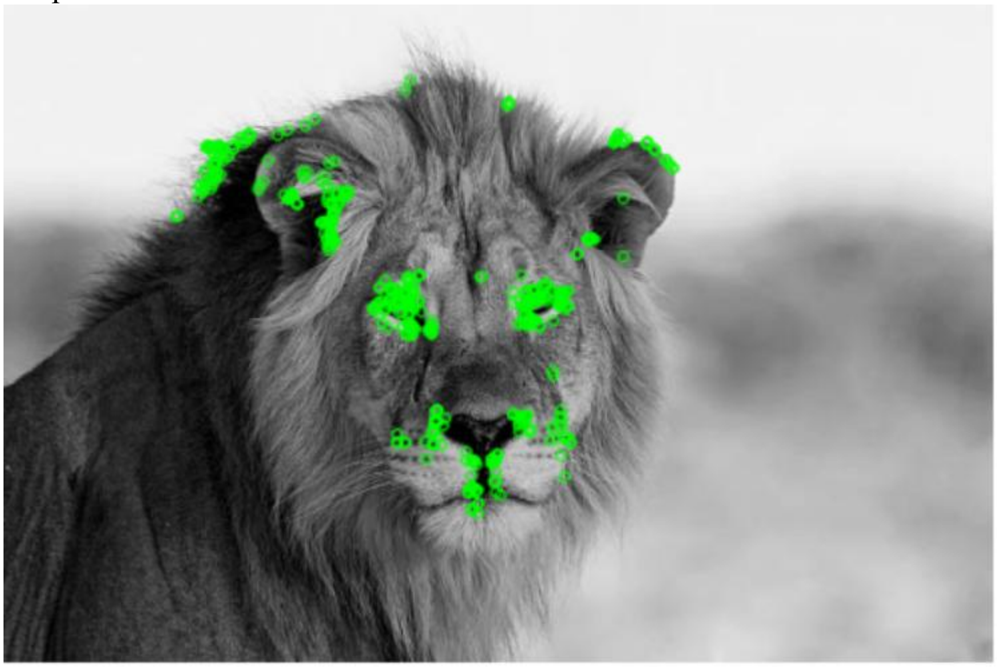
1. Import necessary libraries.
2. Load the image using OpenCV.
3. Define a function to extract ORB features.
4. Create a orb object.
5. Detect key points and compute descriptors using orb.detectAndCompute.
6. Return the descriptors as features.

Code ;

```python
# Initialize ORB detector
orb = cv2.ORB_create()
# Detect keypoints and compute descriptors
keypoints, descriptors = orb.detectAndCompute(gray_img, None)
```

```
# Draw keypoints on the image
img_with_keypoints = cv2.drawKeypoints(gray_img, keypoints, None,
color=(0, 255, 0), flags=0)
# Display the image with keypoints
plt.axis('off')
plt.imshow(img_with_keypoints, cmap='gray')
plt.show()
# Print the number of keypoints and descriptor shape
print("Number of keypoints:", len(keypoints))
print("Descriptor shape:", descriptors.shape)
```

Output:



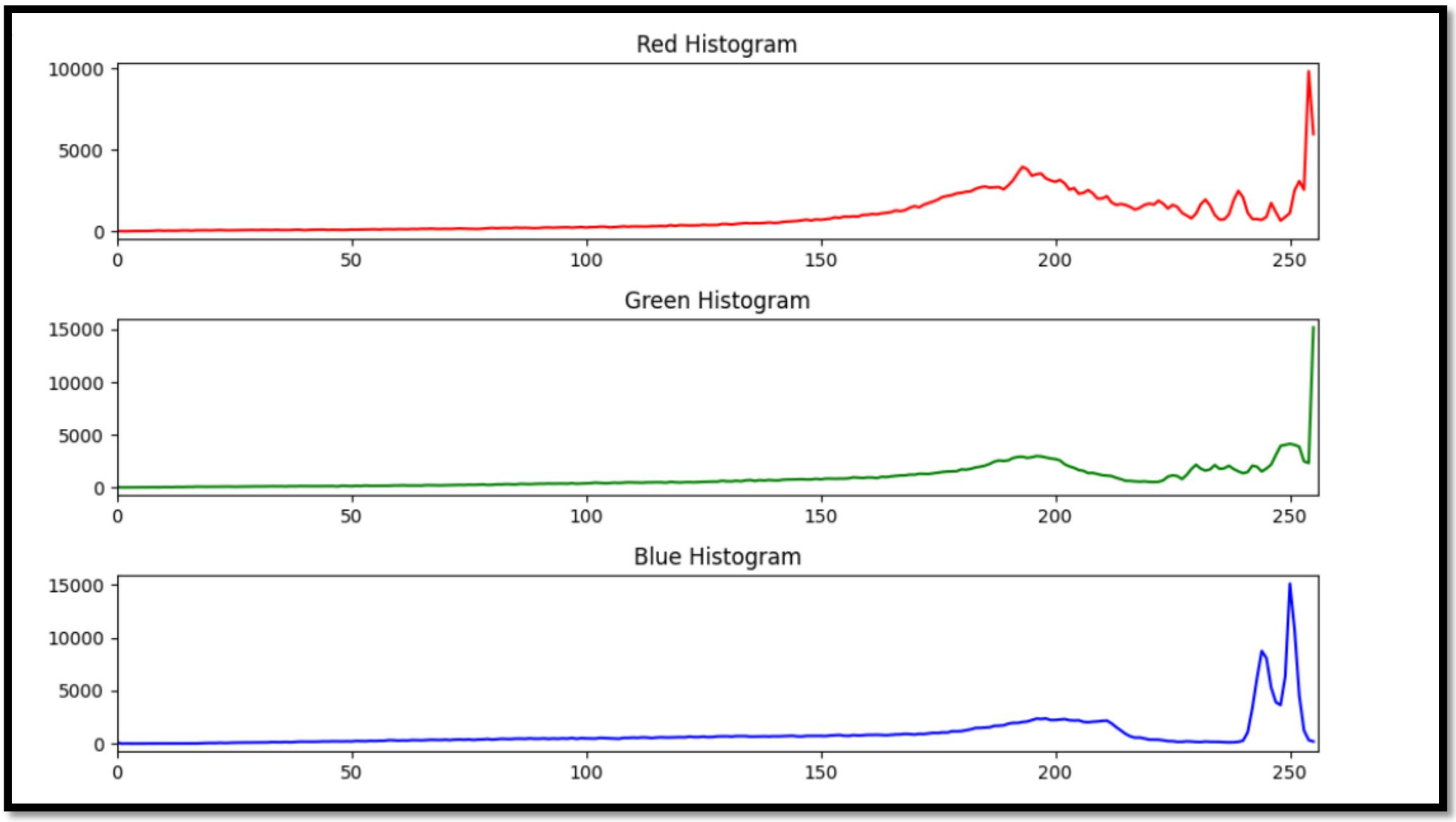
Color Histograms: -

Description: - A color histogram is a graphical representation of the distribution of colors in an image. It provides valuable information about the prevalence of different colors in the image, which can be useful for various image processing and computer vision tasks, including image analysis, object recognition, and image retrieval. Color histograms are commonly used in both color and grayscale images.

Code:

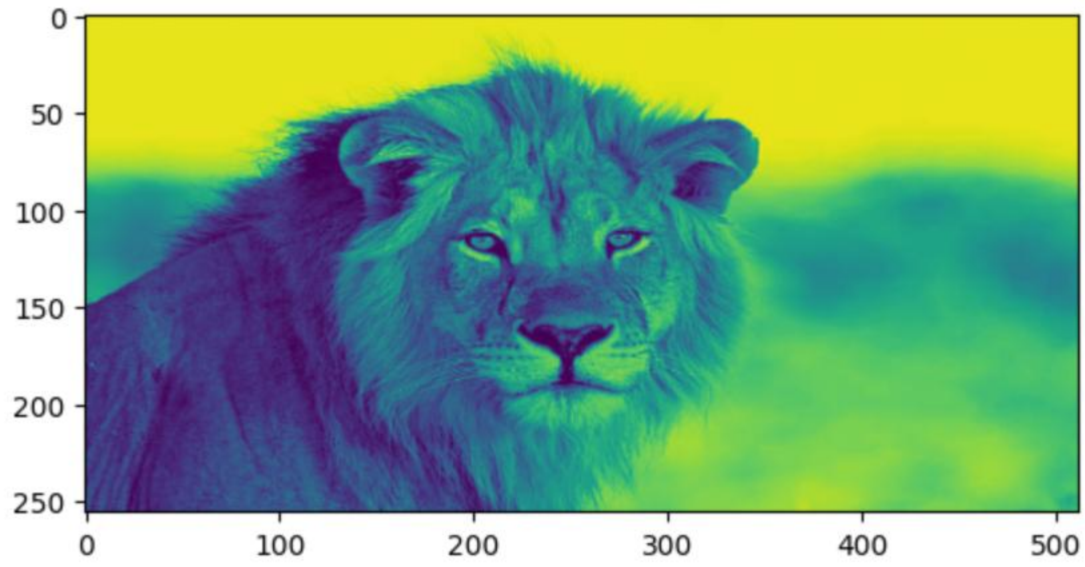
```
# Convert the image from BGR to RGB (OpenCV loads images as BGR by default)
cat_image = imread('/content/lion.jpg')
# Calculate the color histograms for each channel (R, G, B)
hist_red = cv2.calcHist([cat_image], [0], None, [256], [0, 256])
hist_green = cv2.calcHist([cat_image], [1], None, [256], [0, 256])
hist_blue = cv2.calcHist([cat_image], [2], None, [256], [0, 256])
# Plot the histograms
plt.figure(figsize=(10, 6))
plt.subplot(311)
plt.plot(hist_red, color='red')
plt.title('Red Histogram')
plt.xlim([0, 256])
plt.subplot(312)
plt.plot(hist_green, color='green')
plt.title('Green Histogram')
plt.xlim([0, 256])
plt.subplot(313)
plt.plot(hist_blue, color='blue')
plt.title('Blue Histogram')
plt.xlim([0, 256])
plt.tight_layout()
plt.show()
```

Output:



```
resized_img = resize(gray_img, (64*4, 128*4))
# plt.axis("off")
plt.imshow(resized_img)
print(resized_img.shape)
```

```
(256, 512)
```


0
50
100
150
200
250
0
100
200
300
400
500