Introduction:

Image processing is a field that focuses on analyzing and transforming images to enhance their quality or extract meaningful information. It involves techniques like filtering, edge detection, segmentation, and feature extraction to make images suitable for applications such as medical diagnostics, security systems, and machine vision. By processing raw image data, it enables systems to interpret visual information and perform tasks like object detection, pattern recognition, and image restoration.

Feature Extraction:

Feature extraction is the process of identifying and isolating key attributes or patterns from data, such as images, to simplify analysis and improve accuracy in tasks like classification, recognition, and detection. In image processing, it involves techniques to capture meaningful details like edges, textures, colors, and shapes, reducing the data size while preserving critical information. This step is crucial in applications like facial recognition, object detection, and medical imaging, as it enables systems to focus on the most relevant aspects of the input.

Different Techniques of Feature Extraction:

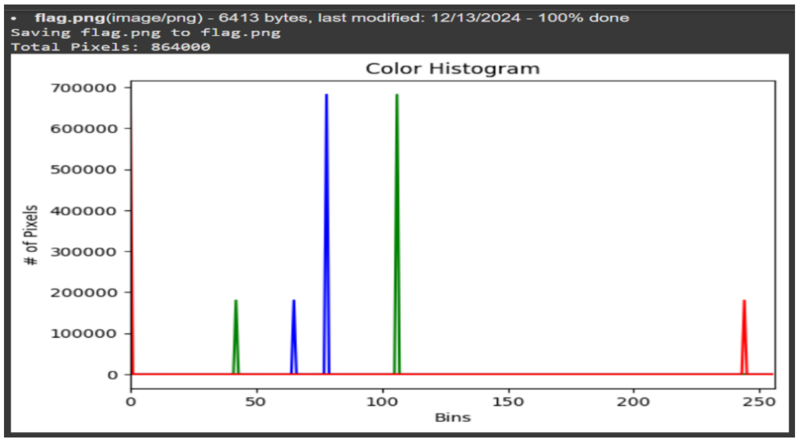
1. Color-Based Features:

- **Total Pixels and Color Histogram Visualization :** This code calculates the total number of pixels in an uploaded image and visualizes its color distribution through a histogram. It plots the intensity distribution for each color channel (Red, Green, Blue) to analyze the image's color composition.

CODE

```
import cv2
import matplotlib.pyplot as plt
from google.colab import files
# Step 1: Upload the image
uploaded = files.upload()
# Step 2: Load the image
image_path = list(uploaded.keys())[0]  # Get the name of the uploaded file
image = cv2.imread(image_path, cv2.IMREAD_COLOR)
# Step 3: Check if the image is loaded properly
if image is None:
raise FileNotFoundError("Image not found. Please upload a valid image file.")
# Step 4: Compute the total number of pixels
height, width, _ = image.shape  # Get image dimensions
total_pixels = height * width  # Calculate total pixels
print(f"Total Pixels: {total_pixels}")
# Step 5: Compute and plot the color histogram
colors = ('b', 'g', 'r')  # Channels for blue, green, and red
for i, color in enumerate(colors):
```

```
hist = cv2.calcHist([image], [i], None, [256], [0, 256])  # Histogram for each color
plt.plot(hist, color=color)  # Plot histogram
plt.xlim([0, 256])  # Set x-axis limits to the range of pixel intensity
# Step 6: Display the histogram
plt.title('Color Histogram')
plt.xlabel('Bins')
plt.ylabel('# of Pixels')
plt.show()
```



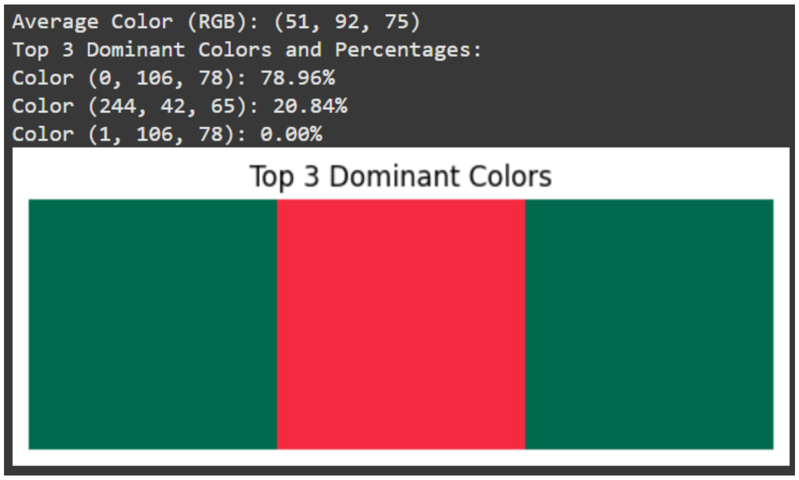
```
•  flag.png(image/png) - 6413 bytes, last modified: 12/13/2024 - 100% done
Saving flag.png to flag.png
Total Pixels: 864000
```

- **Average and Dominant Color Analysis of an Image:** This code calculates and displays the average color of an uploaded image, as well as the top 3 most dominant colors and their percentages. It also visualizes the dominant colors, providing insights into the image's color composition.

<u>CODE</u>

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from google.colab import files
from collections import Counter
# Step 1: Upload the image
uploaded = files.upload()  # Upload the image file
# Step 2: Load the image
image_path = list(uploaded.keys())[0]  # Get the name of the uploaded file
image = cv2.imread(image_path)  # Load the image using OpenCV
# Step 3: Check if the image is loaded properly
if image is None:
    raise FileNotFoundError("Image not found. Please upload a valid image file.")
# Step 4: Convert from BGR to RGB (fixing the channel order)
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
```

```
# Step 5: Calculate Average Color
average_color = image_rgb.mean(axis=(0, 1))  # Calculate mean of R, G, B channels
average_color_rgb = tuple(int(c) for c in average_color)  # Convert to integer tuple
print(f"Average Color (RGB): {average_color_rgb}")
# Step 6: Dominant Colors
pixels_reshaped = image_rgb.reshape(-1, 3)  # Reshape image to a list of pixels (RGB)
pixel_list = [tuple(pixel) for pixel in pixels_reshaped]  # Convert each pixel to tuple
color_counts = Counter(pixel_list)  # Count the occurrences of each color
top_colors = color_counts.most_common(3)  # Get top 3 most common colors
# Step 7: Display Dominant Colors and Percentages
print("Top 3 Dominant Colors and Percentages:")
total_pixels = len(pixel_list)
for color, count in top_colors:
    percentage = (count / total_pixels) * 100
    print(f"Color {color}: {percentage:.2f}%")
# Step 8: Visualizing the Top 3 Dominant Colors
dominant_colors = [color[0] for color in top_colors]  # Extract top 3 colors
plt.figure(figsize=(8, 2))
plt.imshow([dominant_colors])  # Display the top 3 colors as a row
plt.axis('off')
plt.title("Top 3 Dominant Colors")
plt.show()
```



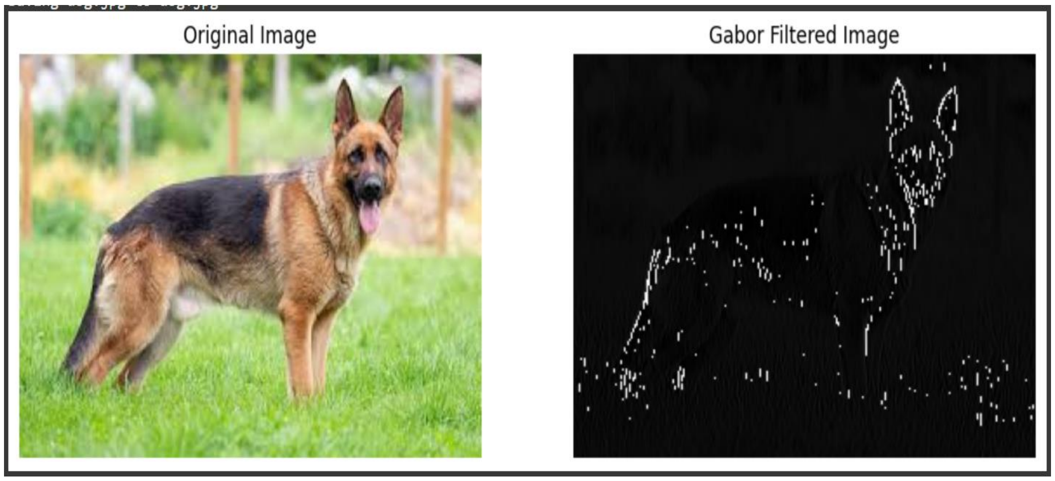
```
Average Color (RGB): (51, 92, 75)
Top 3 Dominant Colors and Percentages:
Color (0, 106, 78): 78.96%
Color (244, 42, 65): 20.84%
Color (1, 106, 78): 0.00%
```

2. Texture-Based Features:

- **Gabor Filters**: Gabor filters are linear filters used for texture and edge analysis. They capture spatial frequency content in a specific direction and scale, making them effective for feature extraction in texture-based applications**.**

CODE

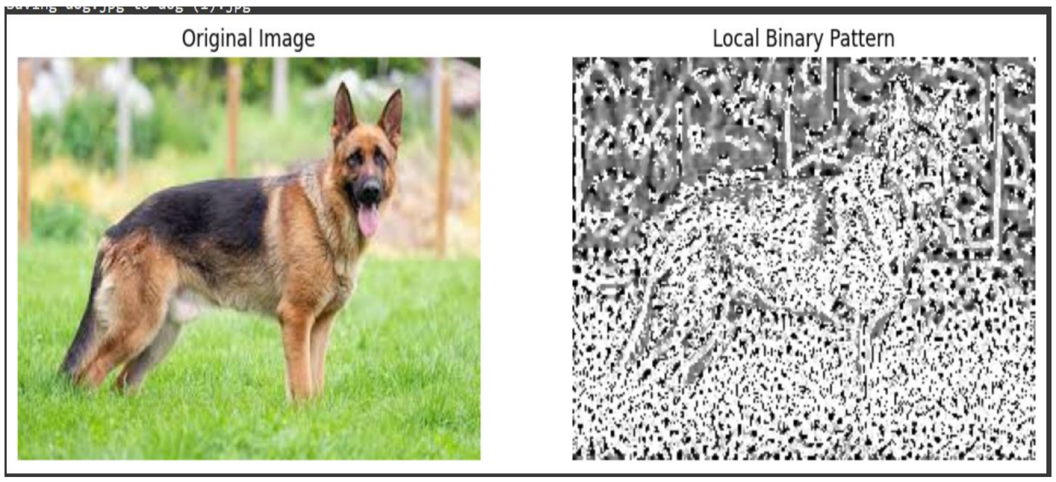
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from skimage.filters import gabor
from google.colab import files
# Step 1: Upload the image
uploaded = files.upload()  # Prompt to upload an image file
# Step 2: Load the image
image_path = list(uploaded.keys())[0]  # Get the name of the uploaded file
image = cv2.imread(image_path, cv2.IMREAD_COLOR)  # Load the image in color
# Step 3: Convert the image to grayscale
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
# Step 4: Apply Gabor filter
frequency = 0.6  # Frequency of the sinusoidal wave
gabor_filtered, _ = gabor(gray_image, frequency=frequency)
# Step 5: Visualize the original and Gabor-filtered images
plt.figure(figsize=(12, 6))
# Original Image
plt.subplot(1, 2, 1)
plt.title("Original Image")
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.axis('off')
# Gabor Filtered Image
plt.subplot(1, 2, 2)
plt.title("Gabor Filtered Image")
plt.imshow(gabor_filtered, cmap='gray')
plt.axis('off')
plt.show()
```



- **Local Binary Patterns (LBP):** Local Binary Patterns (LBP) is a texture descriptor that captures the local structure of an image by comparing each pixel with its neighbors. It's widely used for tasks like texture classification and face recognition.

CODE

```
import cv2
import numpy as np
from skimage.feature import local_binary_pattern
import matplotlib.pyplot as plt
from google.colab import files
# Step 1: Upload the image
uploaded = files.upload()  # Prompt to upload an image file
# Step 2: Load the image
image_path = list(uploaded.keys())[0]  # Get the name of the uploaded file
image = cv2.imread(image_path, cv2.IMREAD_COLOR)  # Load the image in color
# Step 3: Convert the image to grayscale
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
# Step 4: Define parameters for LBP
radius = 3  # Radius of the neighborhood
n_points = 8 * radius  # Number of neighbors
# Step 5: Compute LBP
lbp = local_binary_pattern(gray_image, n_points, radius, method='uniform')
# Step 6: Visualize the original and LBP images
plt.figure(figsize=(12, 6))
# Original Image
plt.subplot(1, 2, 1)
plt.title("Original Image")
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.axis('off')
# LBP Image
plt.subplot(1, 2, 2)
plt.title("Local Binary Pattern")
plt.imshow(lbp, cmap='gray')
plt.axis('off')
plt.show()
```



- **Image Dimension Analysis and Aspect Ratio Calculation:** This code calculates the width, height, and total dimensions of an uploaded image. It also determines the aspect ratio to analyze the image's proportions accurately.

  CODE:

  ```
  import cv2
  from google.colab import files
  # Step 1: Upload the image
  uploaded = files.upload()  # Prompt to upload an image file
  # Step 2: Load the image
  image_path = list(uploaded.keys())[0]  # Get the name of the uploaded file
  image = cv2.imread(image_path)
  # Step 3: Check if the image is loaded properly
  if image is None:
  raise FileNotFoundError("Image not found. Please upload a valid image file.")
  # Step 4: Get image dimensions
  height, width = image.shape[:2]  # Extract height and width
  # Step 5: Calculate aspect ratio
  aspect_ratio = width / height  # Width divided by height
  print(f"Image Dimensions: Width = {width}, Height = {height}")
  print(f"Aspect Ratio: {aspect_ratio:.2f} (Width:Height)")
  ```

- **Image Format Detection Using Python:** This code identifies and displays the format (e.g., JPEG, PNG) of an uploaded image using the Pillow library in Google Colab.

  CODE:

  ```
  from google.colab import files
  from PIL import Image
  # Step 1: Upload the image
  uploaded = files.upload()  # Prompt to upload an image file
  # Step 2: Load the image
  image_path = list(uploaded.keys())[0]  # Get the name of the uploaded file
  image = Image.open(image_path)  # Open the image using Pillow
  # Step 3: Get the image format
  image_format = image.format  # Retrieve the format of the image (e.g., JPEG, PNG)
  print(f"Image Format: {image_format}")
  ```

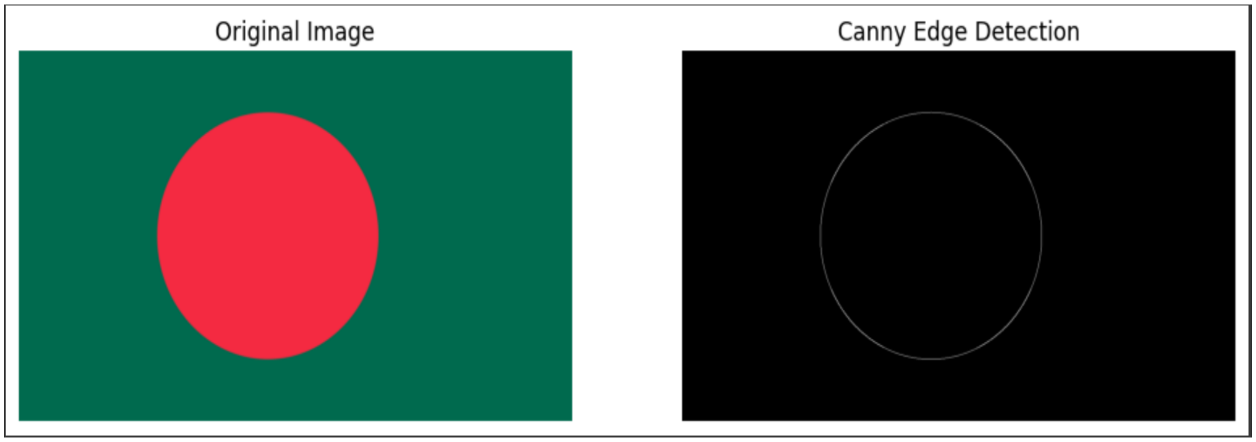
3. Shape-Based Features:

- **Canny Edge Detection Visualization:** This code applies the Canny edge detection algorithm to an uploaded image, displaying both the original image and the detected edges.

It helps visualize the boundaries and features in the image by highlighting areas with significant intensity changes.

CODE

```
import cv2
import matplotlib.pyplot as plt
from google.colab import files
# Step 1: Upload the image
uploaded = files.upload()  # Prompt to upload an image file
# Step 2: Load the image
image_path = list(uploaded.keys())[0]  # Get the name of the uploaded file
image = cv2.imread(image_path, cv2.IMREAD_COLOR)
# Step 3: Convert to grayscale
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
# Step 4: Apply Canny Edge Detection
canny_edges = cv2.Canny(gray_image, threshold1=100, threshold2=200)
# Step 5: Display the original image and Canny edges
plt.figure(figsize=(12, 6))
# Original Image
plt.subplot(1, 2, 1)
plt.title("Original Image")
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.axis("off")
# Canny Edges
plt.subplot(1, 2, 2)
plt.title("Canny Edge Detection")
plt.imshow(canny_edges, cmap="gray")
plt.axis("off")
plt.show()
```



4. Keypoint-Based Features:

- **SIFT (Scale-Invariant Feature Transform):** SIFT detects and describes keypoints in images, invariant to scale, rotation, and illumination, for reliable image matching.

  CODE

```
import cv2
import matplotlib.pyplot as plt
from google.colab import files
# Step 1: Upload the image
uploaded = files.upload()  # Prompt to upload an image file
# Step 2: Load the image
image_path = list(uploaded.keys())[0]  # Get the name of the uploaded file
image = cv2.imread(image_path, cv2.IMREAD_COLOR)  # Load the image in color
# Step 3: Check if the image is loaded properly
if image is None:
raise FileNotFoundError("Image not found. Please upload a valid image file.")
# Step 4: Convert the image to grayscale
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
# Step 5: Initialize the SIFT detector
sift = cv2.SIFT_create()
# Step 6: Detect keypoints and compute descriptors
keypoints, descriptors = sift.detectAndCompute(gray_image, None)
# Step 7: Draw keypoints on the image
sift_image = cv2.drawKeypoints(image, keypoints, None,
flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
# Step 8: Display the original image and SIFT features
plt.figure(figsize=(12, 6))
# Original Image
plt.subplot(1, 2, 1)
plt.title('Original Image')
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.axis('off')
# SIFT Features
plt.subplot(1, 2, 2)
plt.title('SIFT Features')
plt.imshow(cv2.cvtColor(sift_image, cv2.COLOR_BGR2RGB))
plt.axis('off')
plt.show()
# Step 9: Print the number of keypoints detected
print(f"Number of Keypoints Detected: {len(keypoints)}")
```

Conclusion :

This discussion covered various image processing techniques, from basic attribute extraction like dimensions and format to more advanced methods like edge detection and feature extraction. Each topic demonstrated the potential of computational methods to analyze and process images, emphasizing their role in diverse applications such as object recognition, content analysis, and quality assessment. By integrating these techniques, a comprehensive understanding of an image's structural, visual, and informational characteristics was achieved, showcasing the versatility and importance of image processing in modern technology.