Introduction

This Python script is designed for image feature extraction, utilizing various techniques such as color analysis, edge detection, texture analysis, Gabor filtering, and Histogram of Oriented Gradients (HOG). The script allows users to upload an image, performs several feature extraction tasks, and visualizes the results. It makes use of popular libraries such as OpenCV, NumPy, Matplotlib, and Scikit-Image.

Feature Extraction Functions

Color Features: Color features are essential in image processing and computer vision as they represent the color distribution within an image. A **color histogram** is a graphical representation of the frequency of each color in the image. Color features help in various tasks like object recognition, image segmentation, and matching. By analyzing how colors are distributed in an image, you can extract useful information about the content.

RGB Histogram: The RGB histogram represents the distribution of colors across the three-color channels: Red, Green, and Blue. It is widely used for color-based segmentation and matching.

Code:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from google.colab import files

# Upload image
uploaded = files.upload()

# Load uploaded image
image_path = list(uploaded.keys())[0]  # Get the filename of the uploaded image
image = cv2.imread(image_path)

# Ensure the image is loaded properly
if image is None:
    print("Error: Could not load image.")
else:
    # RGB Histogram
    colors = ('b', 'g', 'r')
    for i, color in enumerate(colors):
        hist = cv2.calcHist([image], [i], None, [256], [0, 256])
        plt.plot(hist, color=color)
    plt.title('RGB Histogram')
    plt.xlabel('Pixel Intensity')
    plt.ylabel('Frequency')
    plt.show()
```

HSV Histogram (Hue Channel): The HSV histogram represents the distribution of colors in terms of Hue, Saturation, and Value. The Hue channel is often used in color-based image analysis because it captures the actual color (e.g., red, green, blue), while saturation and value capture the intensity and brightness.

Code:

```
import cv2
```

```
import numpy as np
import matplotlib.pyplot as plt
from google.colab import files

# Upload image
uploaded = files.upload()

# Load uploaded image
image_path = list(uploaded.keys())[0]  # Get the filename of the uploaded image
image = cv2.imread(image_path)

# Ensure the image is loaded properly
if image is None:
    print("Error: Could not load image.")
else:
    # Convert image to HSV
    hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

    # HSV Histogram (Hue channel)
    hue_hist = cv2.calcHist([hsv], [0], None, [256], [0, 256])
    plt.plot(hue_hist, color='b')
    plt.title('Hue Histogram')
    plt.xlabel('Hue Value')
    plt.ylabel('Frequency')
    plt.show()
```

Texture Features: Texture features capture the patterns, structures, or the surface characteristics in an image. Techniques like Local Binary Patterns (LBP) are commonly used to extract textural information which is helpful in object classification, facial recognition, and texture analysis.

Local Binary Patterns (Uniform LBP): Local Binary Patterns (LBP) is a simple yet effective texture descriptor that encodes local image texture by comparing each pixel with its neighbors.

Code:

```
!pip install -q scikit-image

from skimage.feature import local_binary_pattern
import cv2
import matplotlib.pyplot as plt
from google.colab.patches import cv2_imshow  # For displaying images in Colab

# Upload image
from google.colab import files
uploaded = files.upload()

# Load uploaded image
image_path = list(uploaded.keys())[0]  # Get the filename of the uploaded image
image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

# Ensure the image is loaded properly
if image is None:
```

```
    print("Error: Could not load image.")
else:
    # Uniform LBP
    lbp_uniform = local_binary_pattern(image, P=8, R=1, method='uniform')

    # Plot the result
    plt.imshow(lbp_uniform, cmap='gray')
    plt.title('Uniform LBP')
    plt.show()
```

Local Binary Patterns (Non-uniform LBP): The Non-uniform LBP is an extension of the basic LBP, allowing for a larger number of distinct patterns and capturing more complex textures.

Code:

```
!pip install -q scikit-image

from skimage.feature import local_binary_pattern
import cv2
import matplotlib.pyplot as plt
from google.colab import files

# Upload image
uploaded = files.upload()

# Load the uploaded image in grayscale
image_path = list(uploaded.keys())[0]  # Get the filename of the uploaded image
image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

# Ensure the image is loaded properly
if image is None:
    print("Error: Could not load image.")
else:
    # Non-Uniform LBP
    lbp_non_uniform = local_binary_pattern(image, P=8, R=1, method='default')

    # Plot the result
    plt.imshow(lbp_non_uniform, cmap='gray')
    plt.title('Non-uniform LBP')
    plt.show()
```

Edge Features: Edge features highlight the boundaries within an image. They are important for object detection and segmentation. Canny edge detection is a popular algorithm for detecting edges in images.

Canny Edge Detection (Edge Magnitude): Edge magnitude refers to the strength of the edges detected using the Canny algorithm.

Code:

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

```
from google.colab import files

# Upload image
uploaded = files.upload()

# Load the uploaded image in grayscale
image_path = list(uploaded.keys())[0]  # Get the filename of the uploaded image
image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

# Ensure the image is loaded properly
if image is None:
    print("Error: Could not load image.")
else:
    # Canny edge detection
    edges = cv2.Canny(image, 100, 200)

    # Display the edge magnitude
    plt.imshow(edges, cmap='gray')
    plt.title('Edge Magnitude (Canny)')
    plt.show()
```

Canny Edge Detection (Edge Direction): Edge direction refers to the angle of the gradient at each edge pixel, showing the orientation of edges in the image.

Code:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from google.colab import files

# Upload image
uploaded = files.upload()

# Load the uploaded image in grayscale
image_path = list(uploaded.keys())[0]  # Get the filename of the uploaded image
image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

# Ensure the image is loaded properly
if image is None:
    print("Error: Could not load image.")
else:
    # Compute gradients using Sobel
    gradient_x = cv2.Sobel(image, cv2.CV_64F, 1, 0, ksize=3)  # Gradient in X direction
    gradient_y = cv2.Sobel(image, cv2.CV_64F, 0, 1, ksize=3)  # Gradient in Y direction

    # Compute edge direction
    direction = cv2.phase(gradient_x, gradient_y, angleInDegrees=True)

    # Display the edge direction
    plt.imshow(direction, cmap='gray')
    plt.title('Edge Direction')
```

```
    plt.colorbar(label='Angle (degrees)')
    plt.show()
```

Histogram of Oriented Gradients (HOG): HOG features capture edge and gradient information, making them useful for object detection, particularly in human detection. It works by computing the gradient of image pixels and encoding them into histograms.

Gradient Direction: The gradient direction helps in determining the direction of edges or features in the image.

Code:

```
!pip install -q scikit-image

from skimage.feature import hog
from skimage import exposure
import cv2
import matplotlib.pyplot as plt
from google.colab import files

# Upload image
uploaded = files.upload()

# Load the uploaded image in grayscale
image_path = list(uploaded.keys())[0]  # Get the filename of the uploaded image
image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

# Ensure the image is loaded properly
if image is None:
    print("Error: Could not load image.")
else:
    # Compute HOG features and visualize
    features, hog_image = hog(image, orientations=9, pixels_per_cell=(8, 8),
                              cells_per_block=(2, 2), visualize=True)

    # Rescale image for visualization
    hog_image_rescaled = exposure.rescale_intensity(hog_image, in_range=(0, 10))

    # Plot the HOG image
    plt.imshow(hog_image_rescaled, cmap='gray')
    plt.title('HOG Image')
    plt.show()
```

Block Normalization: Normalization of blocks within HOG helps reduce the impact of lighting variations.

Code:

```
!pip install -q scikit-image

from skimage.feature import hog
from skimage import exposure
import cv2
import matplotlib.pyplot as plt
```

```
from google.colab import files

# Upload image
uploaded = files.upload()

# Load the uploaded image in grayscale
image_path = list(uploaded.keys())[0]  # Get the filename of the uploaded image
image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

# Ensure the image is loaded properly
if image is None:
    print("Error: Could not load image.")
else:
    # Compute HOG features and normalize
    features, hog_image = hog(image, orientations=9, pixels_per_cell=(8, 8),
                              cells_per_block=(2, 2), visualize=True, block_norm='L2-Hys')

    # Rescale and visualize normalized HOG image
    hog_image_rescaled = exposure.rescale_intensity(hog_image, in_range=(0, 10))

    # Plot the result
    plt.imshow(hog_image_rescaled, cmap='gray')
    plt.title('Normalized HOG Image')
    plt.show()
```

SIFT (Scale-Invariant Feature Transform): SIFT is a robust feature extraction technique used to detect and describe local features in images. It is invariant to scaling, rotation, and partially invariant to affine transformations.

Keypoints (Scale Invariance): SIFT detects keypoints at multiple scales and orientations, making it robust to changes in size and rotation.

Code:

```
!pip install opencv-python-headless

import cv2
import matplotlib.pyplot as plt
from google.colab import files

# Upload image
uploaded = files.upload()

# Load the uploaded image
image_path = list(uploaded.keys())[0]  # Get the filename of the uploaded image
image = cv2.imread(image_path)

# Ensure the image is loaded properly
if image is None:
    print("Error: Could not load image.")
else:
    # Convert image to grayscale
```

```
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    # Detect SIFT features
    sift = cv2.SIFT_create()
    keypoints, descriptors = sift.detectAndCompute(gray, None)

    # Draw keypoints
    image_with_keypoints = cv2.drawKeypoints(image, keypoints, None,
flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)

    # Convert image from BGR to RGB for displaying with Matplotlib
    image_with_keypoints_rgb = cv2.cvtColor(image_with_keypoints, cv2.COLOR_BGR2RGB)

    # Display the image with keypoints
    plt.imshow(image_with_keypoints_rgb)
    plt.title('SIFT Keypoints')
    plt.axis('off')
    plt.show()
```

Conclusion

Feature extraction techniques are crucial for understanding and processing images in computer vision tasks. Depending on the application (e.g., object recognition, classification, or segmentation), different features like color histograms, textures, shapes, edges, and deep learning-based features are commonly used to capture important patterns in the images. The above examples demonstrate how to apply various feature extraction techniques in Python using libraries like OpenCV, scikit-image, and TensorFlow/Keras.