Madhav Institute of Technology & Science, Gwalior – 474005

(Deemed To University)

SIP Final Assignment

of

Python based Signal, Image and Video Processing
(Session Jul-Dec 2024)

Submitted By:

Name - Saksham Sahu
Enrolment number-0901MC231059
Semester-3rd
Year-2nd
Date-19/12/2024

Submitted To:

Dr. Himanshu Singh
Department of Engineering Electronics

CONTENTS

SIP Introduction

Python-Based Signal, Image, and Video Processing (SIP)

Python is widely used in signal, image, and video processing due to its ease of use, powerful libraries, and an active community of developers. These fields involve processing, analyzing, and extracting meaningful information from various types of data.

1. Signal Processing

Signal processing focuses on the analysis and manipulation of signals like sound, vibrations, and communication signals. Python provides tools for:

- **Audio Processing**: Analyzing and modifying audio signals.
- **Fourier Transforms**: Transforming signals between time and frequency domains.
- **Filtering**: Noise reduction and signal smoothing.
- **Applications**: Speech recognition, radar signal processing, and biomedical signal processing.

Popular Libraries:

- **NumPy**: Basic signal operations like filtering and convolution.
- **SciPy**: Advanced signal processing methods, including wavelets and digital filters.
- **PyWavelets**: Discrete Wavelet Transform and related techniques.
- **Librosa**: Audio and music signal analysis.

2. Image Processing

Image processing involves manipulation and analysis of images for applications like enhancement, segmentation, and pattern recognition.

Tasks:

- **Preprocessing**: Cropping, resizing, and filtering images.
- **Feature Extraction**: Edges, corners, and textures.
- **Image Enhancement**: Improving contrast, brightness, and sharpness.
- **Segmentation**: Dividing an image into meaningful parts.

Popular Libraries:

- **OpenCV**: Comprehensive library for image manipulation and computer vision.
- **Pillow (PIL)**: Simple image processing like filters and conversions.
- **scikit-image**: Advanced image analysis tools like segmentation and feature detection.
- **NumPy**: Basic image array manipulations.

3. Video Processing

Video processing is an extension of image processing, dealing with sequences of images (frames) to analyze motion, track objects, or compress data.

Tasks:

- **Frame-by-Frame Analysis**: Processing individual frames.
- **Motion Detection**: Identifying moving objects.
- **Object Tracking**: Following specific objects across frames.
- **Compression**: Reducing video size while maintaining quality.

Popular Libraries:

- **OpenCV**: Frame extraction, video editing, and motion tracking.
- **MoviePy**: High-level video editing.
- **FFmpeg-Python**: Video format conversion and encoding.

Applications of SIP

1. **Healthcare**: MRI image analysis, ECG signal processing.
2. **Security**: Face recognition, surveillance.
3. **Media**: Video editing, enhancement.
4. **Telecommunications**: Signal compression and noise reduction.
5. **Machine Learning**: Preprocessing data for AI models.

Advantages of Python for SIP

- **Ease of Use**: High readability and simplicity.
- **Extensive Libraries**: Comprehensive tools for various tasks.
- **Cross-Platform**: Can run on different systems.
- **Active Community**: A wealth of resources, tutorials, and forums.

Example Workflow in SIP:

1. **Import Libraries**:
   - `import numpy as np`
   - `import cv2` (for images and video)
2. **Read Input Data**:
   - Images: `cv2.imread('image.jpg')`
   - Videos: `cv2.VideoCapture('video.mp4')`
3. **Processing**:
   - Apply filters: `cv2.GaussianBlur(image, (5, 5), 0)`
   - Detect edges: `cv2.Canny(image, 100, 200)`
4. **Analysis**:
   - Identify features or patterns.

- Generate insights or transform data.
-

5. **Output Results**:
   - Save processed images/videos or display them.

Working on Image Process

Image Processing

Image processing is a key area in computer vision and artificial intelligence that focuses on analyzing, enhancing, and transforming images to extract meaningful information or improve their quality. It has diverse applications in industries such as healthcare, security, entertainment, and autonomous vehicles.

Key Concepts:

1. **Preprocessing**:
    - Image preprocessing involves operations like resizing, cropping, filtering, and color adjustments to prepare images for further analysis.
    - Techniques such as Gaussian blur and histogram equalization are used to enhance image quality.
2. **Feature Extraction**:
    - Features like edges, corners, textures, and shapes are extracted to understand the content of an image.
    - Algorithms like Canny Edge Detection and Harris Corner Detection are popular in this domain.
3. **Segmentation**:
    - Segmentation divides an image into meaningful regions for easier analysis.
    - Techniques include thresholding, watershed segmentation, and contour detection.
4. **Object Detection and Recognition**:
    - These tasks identify and classify objects within an image.
    - Modern methods use deep learning algorithms like Convolutional Neural Networks (CNNs).
5. **Transformation**:
    - Operations like rotation, scaling, and translation adjust the image's geometry for specific needs.

Popular Tools and Libraries:

- **OpenCV**: A versatile library for real-time image and video processing.
- **Pillow (PIL)**: Handles basic image manipulations like resizing, cropping, and format conversion.
- **scikit-image**: Offers advanced algorithms for image segmentation, feature extraction, and transformations.
- **NumPy**: Provides efficient array operations for manipulating image data.

Applications:

1. **Healthcare**: Used in MRI analysis, X-ray enhancement, and disease detection.
2. **Security**: Enables facial recognition, license plate detection, and surveillance systems.

3. **Autonomous Vehicles**: Processes real-time video streams to identify objects, lanes, and obstacles.
4. **Media and Entertainment**: Powers photo editing, special effects, and content generation.

Advantages of Python for Image Processing:

- **Ease of Use**: Python's syntax is simple, making it suitable for both beginners and experts.
- **Extensive Libraries**: Comprehensive libraries support a wide range of image processing tasks.
- **Cross-Platform**: Works seamlessly across different operating systems.
- **Community Support**: A large, active community provides abundant resources and guidance.

Conclusion, image processing is an essential field with endless potential. With tools like OpenCV and advancements in machine learning, it continues to drive innovation across various sectors. Python's simplicity and powerful libraries make it an ideal choice for tackling challenges in this domain.

Assignment-Problem Introduction

Problem: Low Offline Computational Power in Image Processing

Image processing often involves computationally intensive tasks, such as high-resolution image analysis, complex filtering, and deep learning-based object recognition. When performed on systems with low offline computational power, several challenges arise:

1. **Slow Processing Speed**: Basic hardware with limited CPU and GPU resources struggles to handle large image datasets or real-time applications, leading to significant delays.
2. **Limited Functionality**: Advanced techniques, such as Convolutional Neural Networks (CNNs) or 3D imaging, are often infeasible without high-performance hardware.
3. **Energy Consumption**: Running intensive tasks on underpowered machines may lead to overheating and inefficient energy use, further slowing down processing.

Solution: a)Solution Introduction
Cloud Computing with Google Colab for Image Processing

Cloud computing provides an effective solution for addressing the challenges posed by low offline computational power in image processing. Google Colab, a free cloud-based platform, offers robust computational resources such as GPUs and TPUs to handle demanding tasks efficiently. Here's how it resolves the identified challenges:

1. Enhancing Processing Speed

- **Google Colab's High-Performance Hardware**: By leveraging Colab's GPUs and TPUs, computationally intensive tasks like high-resolution image analysis and real-time processing are executed much faster compared to local systems.
- **Scalable Resources**: Users can dynamically scale their computational needs without hardware upgrades.

2. Enabling Advanced Functionality

- **Support for Deep Learning Libraries**: Colab supports libraries like TensorFlow, PyTorch, and Keras, making it ideal for running deep learning models such as Convolutional Neural Networks (CNNs).
- **Integration with Google Drive**: Users can easily access large datasets stored in Google Drive, enabling seamless processing of high-resolution images or videos.
- **Compatibility with Jupyter Notebooks**: Colab provides a Jupyter Notebook-like environment, facilitating easy implementation of complex algorithms.

3. Reducing Energy Consumption

- **Cloud-Based Execution**: All intensive computations are performed on Colab's servers, reducing the energy burden on local machines.
- **Efficient Resource Management**: Users can terminate sessions after processing tasks, ensuring optimal resource utilization.

Steps to Use Google Colab for Image Processing

1) **Set Up the Environment**:
   a) Open a new Colab notebook: Google Colab.
   b) Configure GPU/TPU: Navigate to `Runtime > Change runtime type` and select GPU or TPU as the hardware accelerator.
2) **Import Required Libraries**:
   a) Libraries like OpenCV, TensorFlow, Keras, or PyTorch can be installed using `!pip install library-name`.
3) **Access and Process Data**:
   a) Mount Google Drive:`pythonfrom`
   b) `google.colab import drive`
   c) `drive.mount('/content/drive')`
   d) Load datasets from the mounted drive or external sources.
4) **Implement Image Processing Tasks**:
   a) Write code for tasks like filtering, segmentation, or deep learning-based recognition using Colab\u2019s computational resources.
5) **Save Results**:
   a) Store the processed data back to Google Drive or export it to local systems.

Advantages of Using Google Colab

- **Cost-Effective**: Free tier offers sufficient resources for many projects.
- **User-Friendly**: No setup required, and it provides an interactive coding environment.
- **Cross-Platform**: Accessible from any device with internet connectivity.
- **Collaborative**: Allows multiple users to work on the same notebook simultaneously.

By leveraging Google Colab, image processing tasks that are otherwise limited by offline computational power can be performed efficiently and at scale, making it a practical and powerful solution.

b) Working (of Idea)

Working of the Idea: Online Image Enhancement Using Google Colab

This idea focuses on enhancing images online by processing them directly from their URL using Google Colab. The workflow is as follows:

1. **Input Image URL**:

    - The user provides the URL of the image they want to enhance. This eliminates the need for downloading and uploading images manually.
2. **Image Fetching**:
    - The image is retrieved from the provided URL using a Python library like `requests` or `urllib`. The image data is temporarily stored in memory for processing.
3. **Image Processing in Colab**:
    - Using powerful libraries like OpenCV or PIL, the image is enhanced directly in the Colab environment. Enhancements may include operations such as improving brightness, contrast adjustment, noise reduction, or sharpening.
4. **Real-Time Computation**:
    - Google Colab\u2019s cloud-based computational power (GPUs/TPUs) processes the image efficiently, even for complex algorithms.
5. **Display and Output**:
    - The enhanced image is displayed directly in the notebook for preview.
    - The user can optionally download the enhanced image without storing it locally during the process.

This approach ensures seamless, fast, and effective image enhancement with minimal manual intervention, leveraging the power of cloud computing.

c)Data Sets

Datasets Used for Image Processing

1. **Image from URL**: The main dataset in this code is the image obtained from a URL, which is downloaded using the `requests` library and then processed with various image processing techniques. In this case, the image URL is:

   ```
   "https://www.tigren.com/blog/wp-content/uploads/2022/02/how-much-cost-to-start-online-business-1536x878.jpg"
   ```

   This image is fetched over the internet, and the data from it is then used for the following processing tasks.

2. **Types of Data Processed**:
    - **Grayscale Image**: A simplified version of the image where color information is removed, leaving just intensity values (brightness levels). Grayscale images are often used in many image processing tasks as they reduce complexity and focus only on structure and edges.
    - **Edges**: Using the Canny edge detection technique, the algorithm identifies areas of rapid intensity change in the grayscale image. These edges represent boundaries between different regions in the image and are critical for tasks like object detection and feature extraction.
    - **Enhanced Image (CLAHE)**: Contrast Limited Adaptive Histogram Equalization (CLAHE) is applied to enhance the image’s contrast. This helps in improving the details of the image, especially in areas with low contrast, which is useful in improving the visual perception of the image.

Request and Handling of the Image (HTTP Request)

The image is downloaded using the `requests` library. Here's the flow of the request and its handling:

1. **Sending the Request**: The `requests.get(url)` sends an HTTP GET request to the server hosting the image. The URL specifies the location of the image on the web. Once the request is made, it returns a response object.
2. **Checking the Response**: The `response.raise_for_status()` method checks if the request was successful. If there's an issue (e.g., the server returns an error like 404), it will raise an exception.
3. **Reading the Image Data**: The image data is returned in `response.content` as raw bytes. These bytes are then converted into an image format using the `PIL.Image.open()` function, which reads the raw byte stream and turns it into an image object. This is followed by converting the image into a format usable by OpenCV (`numpy` array) and converting its color from RGB (default in PIL) to BGR (default in OpenCV).
4. **Error Handling**: If the image is not downloaded successfully or cannot be read, an error message is printed, and the function returns `None`.

The Role of Data in Processing

The dataset here is an image that undergoes multiple transformations to extract useful features. By using different image processing techniques, various characteristics of the image are revealed, making it useful for tasks like object detection, pattern recognition, and enhancement of image quality for further analysis.

- **Grayscale Image**: Simplifies the image by reducing the color dimensions, making it easier to analyze structural details without being distracted by color.
- **Edge Detection (Canny)**: Helps in detecting sharp changes in intensity, useful for identifying the structure of the image and for tasks such as feature matching, object detection, and segmentation.
- **CLAHE**: Improves the contrast of the image, enhancing the visibility of details in darker or lighter areas, which is helpful in many applications where small features need to be detected or analyzed more clearly.

d) Code Explaination

The code is a Python script that performs the following tasks:

1. Installing Dependencies

- **requests**: Used to download content from the web (in this case, images).
- **beautifulsoup4**: While not used in the current script, this is generally for parsing HTML data.
- **pillow**: A Python Imaging Library (PIL) fork for image processing.
- **opencv-python-headless**: A headless version of OpenCV used for image manipulation (without GUI features).

These dependencies are installed using `pip`.

2. Importing Libraries

- **requests**: To fetch the image from a URL.
- **BeautifulSoup**: Imported but not used in the script; could be used if you want to scrape images from a webpage.
- **cv2 (OpenCV)**: For image manipulation tasks, such as converting to grayscale, edge detection, and contrast enhancement.
- **numpy**: For handling arrays (used in OpenCV).
- **os**: For managing files and directories.
- **BytesIO**: For handling image data in memory (required by Pillow to handle images directly from byte streams).
- **PIL (Image)**: For opening and reading image data.

3. Creating Output Directory

- The script checks if the folder `processed_images` exists. If not, it creates it to save processed images.

4. Downloading Image (`download_image` function)

- This function fetches the image from the provided URL using the `requests.get()` method.
- If successful, it opens the image in memory using Pillow and converts it into an OpenCV-compatible format (BGR).
- If any errors occur (e.g., URL not found or invalid image), the error message is printed.

5. Processing the Image (`process_image` function)

- **Grayscale Conversion**: Converts the image to grayscale, which is a common preprocessing step in computer vision.
- **Edge Detection**: Uses OpenCV's `Canny` edge detection algorithm to highlight edges in the image.
- **CLAHE (Contrast Limited Adaptive Histogram Equalization)**: Enhances the contrast of the grayscale image, which helps in improving image visibility and detail.
- The processed images are returned as a dictionary containing three images: `grayscale`, `edges`, and `enhanced`.

6. Saving Processed Images (`save_images` function)

- For each processed image, the function saves it to the `processed_images` folder with a filename that includes the base name (`image_1`) and the name of the processing method (`grayscale`, `edges`, `enhanced`).
- OpenCV's `cv2.imwrite()` is used to save the images in JPEG format.

7. Main Function (`main`)

- A URL for an image is specified (`image_url`), and the script attempts to download the image using the `download_image` function.
- If successful, the image is processed using the `process_image` function.
- Finally, the processed images are saved in the `processed_images` directory with `save_images`.

8. Execution Flow

- When you run the script, it will:
  1. Download the image from the URL.
  2. Process it (convert to grayscale, apply edge detection, and enhance contrast).
  3. Save the processed images into the `processed_images` directory.
- Each processed version of the image is saved as a separate file with a name like `image_1_grayscale.jpg`, `image_1_edges.jpg`, and `image_1_enhanced.jpg`.

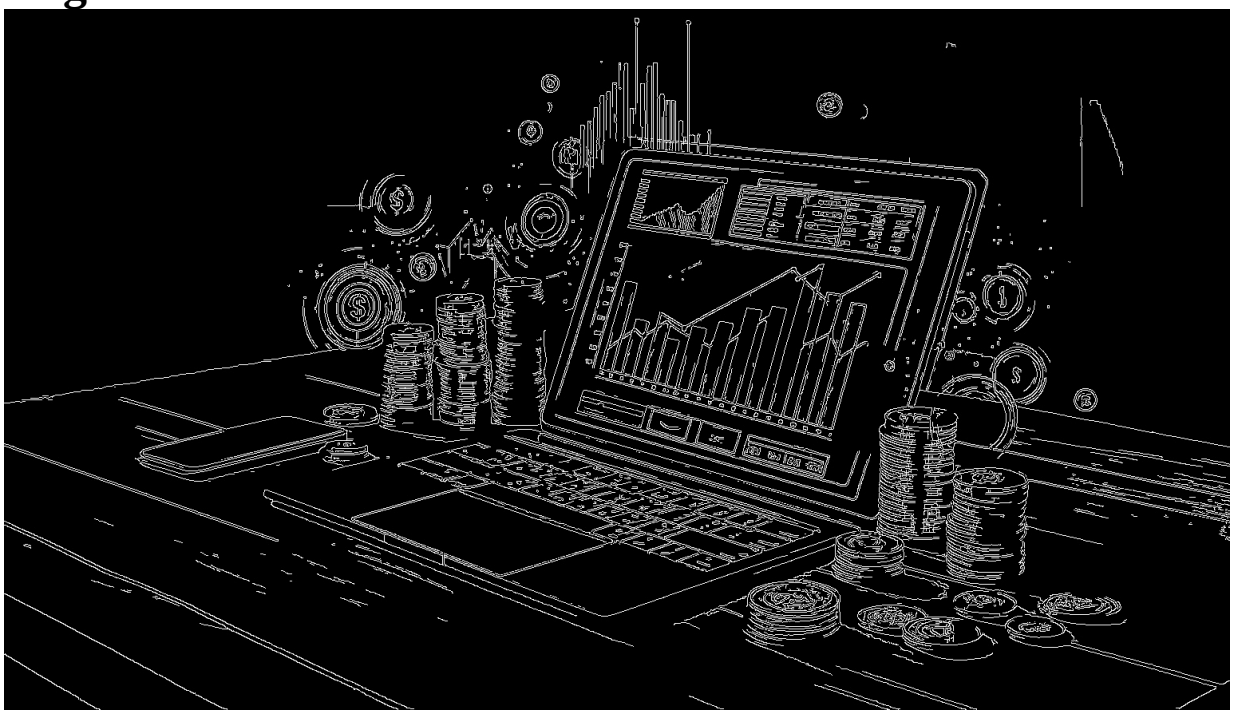
e) Code

```
pip install requests beautifulsoup4 pillow opencv-python-headless
import requests
from bs4 import BeautifulSoup
import cv2
import numpy as np
import os
from io import BytesIO
from PIL import Image

# Directory to save processed images
OUTPUT_DIR = "processed_images"
if not os.path.exists(OUTPUT_DIR):
    os.makedirs(OUTPUT_DIR)

# Function to download and read an image
def download_image(url):
    try:
        response = requests.get(url)
        response.raise_for_status()
        image = Image.open(BytesIO(response.content))
        return cv2.cvtColor(np.array(image), cv2.COLOR_RGB2BGR)
    except Exception as e:
        print(f"Error downloading image: {e}")
        return None

# Function to process an image
def process_image(image):
    try:
        # Grayscale conversion
        gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

        # Edge detection using Canny
        edges = cv2.Canny(gray, 100, 200)
```

```
    # CLAHE for contrast enhancement
            clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8, 8))
            enhanced = clahe.apply(gray)

            # Combine processed images
            processed_images = {
                'grayscale': gray,
                'edges': edges,
                'enhanced': enhanced
            }
            return processed_images
        except Exception as e:
            print(f"Error processing image: {e}")
            return {}

    # Save processed images
    def save_images(images, base_name):


    for name, img in images.items():
            path = os.path.join(OUTPUT_DIR, f"{base_name}_{name}.jpg")
            cv2.imwrite(path, img)
            print(f"Saved {path}")

    # Main function
    def main():
        image_url = "https://www.tigren.com/blog/wp-content/uploads/2022/02/how-
    much-cost-to-start-online-business-1536x878.jpg"

        print(f"Downloading image: {image_url}")
        image = download_image(image_url)
        if image is None:
            return



      print(f"Processing image...")
        processed_images = process_image(image)

        print(f"Saving processed images...")
        save_images(processed_images, "image_1")  # Assuming a single image

        print(f"All images processed and saved in {OUTPUT_DIR}.")

    # Run the main function
    main()
```

Image:-

Processed Image:-
Edges



enhanced Image

GrayScale