KVGCE Hackwise Hackathon: Problem 1 - Satellite Image Brightness Normalizer

Overview

Satellite imagery is critical for applications like environmental monitoring and urban planning, but variations in brightness across images can skew analysis. The Satellite Image Brightness Normalizer challenge requires teams to develop a program that normalizes the brightness of 10 grayscale PNG images (256x256 pixels) to match a global average intensity (e.g., 127.5 for the sample test case). The goal is to ensure each output image's average intensity is within ±1 of the global average, preserving image quality for downstream processing.

Objective: Read a ZIP file containing 10 grayscale PNGs, normalize their brightness, and output 10 normalized PNGs named normalized_image1.png to normalized_image10.png.

Approach

To solve this problem, follow these steps:

1. **Extract Input: Unzip satellite_images.zip to access 10 PNG files (image1.png to image10.png).**
2. **Load Images: Use an image processing library (e.g., PIL in Python, OpenCV, or ImageIO in Java) to read each PNG as a grayscale array.**
3. **Compute Global Average: Calculate the average pixel intensity across all 10 images combined.**
4. **Normalize Images: For each image:**
    - **Compute its current average intensity.**
    - **Apply a scaling factor to adjust pixel values so the image's average matches the global average.**
    - **Ensure pixel values stay within 0–255.**
5. **Save Output: Write normalized images as normalized_image1.png to normalized_image10.png in the working directory.**
6. **Validate: Check that each output image's average intensity is within ±1 of the global average.**

Suggested Tools:

- **Python: PIL (Pillow), NumPy.**
- **C++: OpenCV.**
- **Java: ImageIO, BufferedImage.**
- **JavaScript: Canvas API (Node.js with sharp).**

Datasets

- **Source: Datasets are available at https://github.com/arshad-muhammad/kvgce-hackwise.**
- **Access:**

1. **Clone the repository: git clone https://github.com/arshad-muhammad/kvgce-hackwise.git.**
2. **Navigate to test_cases/problem1/sample_input/ for the sample test case (satellite_images.zip).**
3. **The ZIP contains 10 grayscale PNGs (image1.png to image10.png, 256x256 pixels).**

- **Test Cases:**
  - **Sample Test Case: Provided in the repository with a global average intensity of 127.5.**
  - **Hidden Test Cases: Two additional test cases (hidden1, hidden2) with different global averages (e.g., 125.0, 130.0) will be used for evaluation but are not publicly available.**
  - **Each test case provides a satellite_images.zip with 10 PNGs.**

Pin-to-Pin Details

- **Input:**
  - **File: satellite_images.zip.**
  - **Contents: 10 grayscale PNGs (image1.png to image10.png), each 256x256 pixels, 8-bit depth (0–255).**
  - **Global average intensity is provided implicitly (compute from images) or specified (e.g., 127.5 for sample).**
- **Output:**
  - **10 grayscale PNGs named normalized_image1.png to normalized_image10.png.**
  - **Each output image must be 256x256 pixels, 8-bit grayscale.**
  - **Average intensity of each output image must be within ±1 of the global average.**
- **Constraints:**
  - **Processing time: <10 seconds per test case (8GB RAM, Intel i5).**
  - **Pixel values: 0–255 (clip if necessary).**
  - **No external libraries requiring internet access (offline environment).**
- **Scoring:**
  - **For each test case, score = (correct / 10) * 10, where correct is the number of PNGs with average intensity within ±1 of the global average.**
  - **Maximum 10 points per test case (sample, hidden1, hidden2), totaling 30 points.**

Exact Approach

Algorithm:

1. **Unzip Input:**

- **Use zipfile (Python), zlib (C++), or java.util.zip (Java) to extract satellite_images.zip.**

2. **Read Images:**
    - **Load each PNG as a 256x256 array of pixel values (0–255).**
    - **Example (Python with PIL):**
    - **from PIL import Image**
    - **import numpy as np**
    - **images = [np.array(Image.open(f"image{i}.png").convert("L")) for i in range(1, 11)]**
3. **Compute Global Average:**
    - **Flatten all images into a single array and compute the mean:**
    - **global_avg = np.mean([img.flatten() for img in images])**
4. **Normalize Each Image:**
    - **For each image:**
        - **Compute current average: current_avg = img.mean().**
        - **Calculate scaling factor: factor = global_avg / current_avg.**
        - **Scale pixels: normalized = img * factor.**
        - **Clip values to 0–255: normalized = np.clip(normalized, 0, 255).**
        - **Convert to 8-bit: normalized = normalized.astype(np.uint8).**
5. **Save Outputs:**
    - **Save each normalized array as a PNG:**
    - **for i, img in enumerate(normalized_images, 1):**
    - **Image.fromarray(img).save(f"normalized_image{i}.png")**
6. **Validate:**
    - **Compute each output image's average and ensure abs(avg - global_avg) <= 1.**

Error Handling:

- **Handle missing PNGs or invalid formats (return 0 score for test case).**
- **Ensure output filenames match exactly (normalized_image1.png, not image1_normalized.png).**

Submission Requirements

Teams must submit the following by the hackathon deadline:

1. **PowerPoint Presentation (PPT):**
    - **Format: .pptx, max 10 slides.**

- **Contents:**
  - **Title Slide: Team name, ID, problem number, KVGCE Hackwise logo.**
  - **Problem Understanding: Summarize the problem and its importance.**
  - **Solution Approach: Describe your algorithm and tools used.**
  - **Implementation: Highlight key code snippets and challenges faced.**
  - **Results: Show sample test case results (e.g., before/after images).**
  - **Team Contribution: List each member's role.**
  - **References: Cite libraries or tutorials used.**
- **Submission: Upload to the hackathon portal or email as specified.**

2. **Documentation:**
   - **Format: .pdf, 5–10 pages.**
   - **Contents:**
     - **Introduction: Problem overview and objectives.**
     - **Methodology: Detailed algorithm, including pseudocode or flowcharts.**
     - **Implementation Details:**
       - **Programming language and libraries used.**
       - **Code structure (e.g., main functions, file organization).**
       - **How to run the code (e.g., command-line instructions).**
     - **Dataset Handling: How you accessed and processed satellite_images.zip.**
     - **Output Format: Description of normalized_image1.png to normalized_image10.png.**
     - **Challenges and Solutions: Technical issues faced and how resolved.**
     - **Test Case Results: Performance on the sample test case (e.g., 10/10 PNGs correct).**
     - **Future Improvements: Potential optimizations or enhancements.**
     - **References: Cite datasets, libraries, or resources.**
   - **Submission: Upload with PPT or include in GitHub repository.**
3. **Code Submission:**
   - **Repository:**
     - **Create a public GitHub repository (e.g., github.com/your-team/kvgce-hackwise-problem1).**
     - **Upload all source code, including scripts and any helper files.**

- Include a README.md with:
    - Team name and ID.
    - Problem number and title.
    - Instructions to run the code (e.g., python main.py).
    - Dependencies (e.g., pip install pillow numpy).
    - Expected input (satellite_images.zip) and output (normalized_image1.png to normalized_image10.png).
- Collaboration:
    - Invite https://github.com/arshad-muhammad as a collaborator:

1. Go to repository Settings > Collaborators.
2. Add arshad-muhammad as a collaborator.
3. Ensure they have write access.

- Structure:
    - Example:
    - kvgce-hackwise-problem1/
    - ├── main.py
    - ├── requirements.txt
    - ├── README.md
    - ├── docs/
    - │ └── documentation.pdf
    - └── sample_output/
    - ├── normalized_image1.png
    - └── ...
- Submission: Share the repository URL in your documentation and PPT.

Evaluation Matrix

The following matrix applies to all five problems (per artifact ID: 71566a0e):

Criteria	Description	Points
Functionality	Correctness of output per test case (sample, hidden1, hidden2). Score = (correct / 10) * 10 per test case for Problem 1.	30 (10 per test case)
Code Quality	Readability, modularity, comments, and adherence to language best practices (evaluated by Judge 1).	20

Innovation	**Novel algorithms or optimizations beyond the basic solution (e.g., adaptive normalization).**	**20**
Presentation	**Clarity, completeness, and professionalism of PPT and documentation (evaluated by Judge 2).**	**20**
Teamwork	**Evidence of balanced contribution among team members, shown in PPT/documentation.**	**10**
Total		**100**

- **Functionality Testing:**
  - **The testing framework (artifact ID: 1 to 5) runs your code against three test cases.**
  - **Sample test case: Publicly available in the GitHub repository.**
  - **Hidden test cases: Used by judges, with different global averages (e.g., 125.0, 130.0).**
  - **Output is compared to expected PNGs using average intensity checks (±1 tolerance).**
- **Judging:**
  - **Judges manually review code quality, innovation, presentation, and teamwork.**
  - **Submit early to verify compatibility with the testing framework.**

Test Cases

- **Provided Test Case:**
  - **Sample Test Case: Available at test_cases/problem1/sample_input/satellite_images.zip.**
  - **Contains 10 grayscale PNGs (image1.png to image10.png).**
  - **Global average intensity: 127.5 (verify by computing mean across all pixels).**
  - **Expected output: 10 PNGs (normalized_image1.png to normalized_image10.png) with averages within 127.5 ± 1.**
- **Hidden Test Cases:**
  - **Two additional test cases (hidden1, hidden2) used for evaluation.**
  - **Same format: satellite_images.zip with 10 PNGs.**
  - **Different global averages (e.g., 125.0, 130.0).**
  - **Not publicly available; ensure your code generalizes to any global average.**

- **Format:**
  - **Input: ZIP file with 10 PNGs (256x256, 8-bit grayscale).**

    - **Output: 10 PNGs with exact filenames and same dimensions.**
- **Testing:**
    - **Use the sample test case to validate your code.**
    - **Ensure outputs match the expected format and intensity constraints.**

Notes

- **Environment: Offline, 8GB RAM, Intel i5. Pre-install libraries like PIL, NumPy, or OpenCV.**
- **Languages: Any (Python, C++, Java, etc.), but specify execution command in documentation (e.g., python main.py).**
- **Constraints: No internet access during testing; include all dependencies in your repository.**
- **Tips:**
    - **Test with the sample test case to ensure 10/10 PNGs are correct.**
    - **Optimize for speed (<10 seconds per test case).**
    - **Document challenges (e.g., handling edge cases) to score higher on innovation.**
- **Support: Post questions on the hackathon discussion forum or contact organizers.**

Acknowledgments

Thank you for participating in KVGCE Hackwise! We appreciate your effort in solving real-world problems through innovative coding. Good luck, and we look forward to your submissions!