Satellite Image Brightness Normalizer

Submitted by:
Team Name: TECH ORBIT

Team Members:

1) **PANCHAKARLA LEELA SAI RAM**
2) **KUSHI K.T**
3) **MANASWI KOCHI**
4) **CHENDANA P**

Problem Number: Problem 1
Problem Title: Satellite Image Brightness Normalizer

Institution:
KVG College of Engineering, HackWise Hackathon 2025

Submission Date:
26 April 2025

Contents

1. Introduction
2. Problem Overview
3. Methodology and Algorithm
4. Implementation Details
5. How to Run the Code
6. Dataset Handling
7. Output Format
8. Challenges and Solutions
9. Test Case Results
10. Future Improvements
11. Conclusion
12. References

Satellite Image Brightness Normalizer

1.Introduction

In the field of satellite image processing, consistent image brightness is crucial for accurate analysis, particularly in machine learning and remote sensing applications. Satellite images often come from various sources and are affected by changing lighting conditions, exposure settings, and sensor differences. This inconsistency in image brightness can degrade the performance of automated systems that rely on uniform input data.

2.Problem Overview

The goal of this project is to normalize the brightness levels of a set of grayscale satellite images by ensuring that their average pixel intensity matches a common global average. This global average is computed based on the collective pixel values of all input images. The normalization process ensures that each output image has a pixel intensity close to the shared average, maintaining fidelity while improving uniformity across the dataset.

3.Methodology

The problem is approached by designing a brightness normalization algorithm. The algorithm starts by reading all input images and converting them to grayscale arrays. It calculates the global average intensity by computing the mean value of all pixels across all images. Then, each individual image is normalized by adjusting its pixel values with a scaling factor derived from its current average and the computed global average.

The pseudocode for the algorithm is as follows:

FUNCTION normalize_all_images(input_directory, output_directory):

images ← load all images from input_directory

global_average ← compute mean of all pixel values across images

FOR each image in images: image_average ←

mean of the image pixels scaling_factor ←

global_average / image_average

normalized_image ← clip(image × scaling_factor, 0,

255) save normalized_image to output_directory

validate normalized images using global_average

This algorithm ensures that every image is adjusted using a proportional scale, such that it retains its structure and content but conforms to the desired brightness standard.

4.Implementation Details

The entire system is implemented in Python, leveraging well-established libraries such as Pillow for image processing, NumPy for numerical operations, and tkinter for building a graphical user interface (GUI). The codebase is organized in a modular fashion with separate functions for loading images, computing the global average, normalizing individual images, saving outputs, and validating the results.

The primary file is main.py, which serves as the entry point. A GUI is integrated into this file to allow users to select input and output directories without needing to use the command line. A progress bar visually indicates the normalization process, and users receive a detailed validation report after processing.

The GUI interface supports the following workflow:

1. Users select the folder containing the input images (sample_input).
2. They choose or create a folder where the output should be stored (sample_output).
3. On clicking "Normalize Images", the process begins and displays the computed global average and scaling factors.

This enhances usability and accessibility, especially for users unfamiliar with programming.

5.How to Run the Code

To execute the program, users must have Python installed on their machine. After installing the necessary dependencies listed in requirements.txt, users can run the application using the command:

python main.py

6.Dataset Handling

Unlike some initial setups that required extracting images from ZIP files, this solution directly uses the sample_input/ folder containing 10 grayscale images named image1.png through image10.png. These images are assumed to be of uniform format and resolution. Each image is loaded into memory using Pillow and converted to grayscale mode ("L") to simplify pixel manipulation.

All image arrays are flattened to calculate the global average across all images. After normalization, the processed images are saved into the sample_output/ folder as normalized_image1.png to normalized_image10.png.

7.Output Format

The output consists of 10 PNG images with adjusted brightness levels. Each output image matches the structure and resolution of its input but is normalized to have an average intensity close to the global average within a margin of ±1. This ensures brightness consistency while preserving the visual content.

Each image is saved in PNG format under a new directory, typically named sample_output/. Users can verify the output visually or through the printed validation report that shows the new average intensity of each image and whether it meets the threshold criteria.

8.Challenges and Solutions

One of the challenges in this project was achieving consistency across varying input image intensities without introducing noise or distortions. Direct pixel manipulation can often lead to overflow errors or banding artifacts. This was resolved by using NumPy's clip() function to constrain pixel values between 0 and 255, ensuring valid grayscale values.

Another challenge was making the tool accessible to non-programmers. This was addressed by integrating a simple yet effective graphical interface using tkinter, enabling users to navigate folders and perform normalization with minimal effort.

In early versions, the program assumed a fixed global average. This was later improved by computing the actual global average dynamically, ensuring the code works for any test case, including hidden test scenarios with unknown pixel distributions.

9.Test Case Results

The tool was tested using the provided sample input folder containing 10 grayscale images. The global average computed was approximately 130.20, and all normalized images had average values within 1 unit of this average, confirming the accuracy and reliability of the algorithm.

During hidden test evaluations, the tool is expected to generalize effectively, as the scaling factor is dynamically calculated based on the actual input image data, rather than any hardcoded values.

10.Future Improvements

Future versions of this tool can incorporate the following enhancements:

- Add support for batch processing of multiple folders.
- Enable visualization of original vs. normalized images within the GUI.
- Include exportable reports in CSV or PDF format showing scaling factors and validation metrics.
- Integrate histogram matching or contrast stretching for more sophisticated normalization.
- Extend support for RGB or multispectral satellite images with separate channel normalization.

11.Conclusion

The *Satellite Image Brightness Normalizer* effectively standardizes brightness levels across satellite images by adjusting each image's intensity to match a global average. This ensures consistency, which is essential for further image analysis tasks.

The tool performs well across all test cases, with outputs maintaining average intensities within ±1 of the global mean. The addition of a simple GUI enhances usability, making the tool accessible even to non-technical users.

Overall, the project meets its objectives and sets the stage for future improvements like batch processing, better visual feedback, and support for larger datasets.

References

- Pillow Library Documentation: https://pillow.readthedocs.io
- NumPy Library Documentation: https://numpy.org/
- Tkinter GUI Documentation: https://docs.python.org/3/library/tkinter.html
- OpenAI. (2025). ChatGPT (April 2025 version) [Large language model]. https://chat.openai.com/

GIT URL: https://github.com/Leelasairam03/tech-orbit-kvgcehackwise-problem1