# TEAM AMRIT

presents…


## KVGCE HACKWISE

NATIONAL LEVEL 24 HOURS HACAKTHON

25th – 26th APRIL 2025

**THEME: AI HACKTHON PUSHING BOUNDARIES IN SPACE CHALLENGES**

## Problem 1 - Satellite Image Brightness Normalizer

**Team members:**

1. Ganesh Mali
2. Ramchandra Potadar
3. Sandip Sargar

# Table of Content

# 1.Introduction

## Problem Overview:

Satellite imagery analysis often suffers from brightness inconsistencies across images, which can affect critical applications such as environmental monitoring and urban development. This project addresses the normalization of brightness across satellite images, enhancing consistency and reliability in downstream analysis

Satellite image datasets captured from different sensors or under varying lighting conditions result in brightness bias, making comparison across scenes difficult. The normalization of intensity improves interoperability in machine learning models for classification or segmentation.

## Objective:

To develop a program that:

- Accepts a ZIP file containing 10 grayscale PNG images (256x256 pixels), 8-bit depth (0-255.
- Computes a Global Average Pixel Intensity value across all 10 images.
- Verify each image's Average intensity is within the ±1 of Global Average.
- Saves outputs as normalized_image1.png to normalized_image10.png.

**Additional Goal:** Ensure the program performs efficiently in constrained offline environments, supporting real-world scenarios such as edge computing in space systems.

# 2. Methodology

## Algorithm Steps:

**Step 1:** Extract ZIP Archive – use zipfile module to extract all images.

**Step 2:** Read Images - Load each image as a grayscale 256x256 NumPy array

**Step 3:** Compute Global Average: Combine all pixel values across all 10 images and compute the mean.

**Step 4:** Normalize Each Image:

- Compute individual image average.
- Derive scaling factor = global average / current average.
- Scale pixel values and clip to [0, 255].
- Convert to 8-bit unsigned integers.

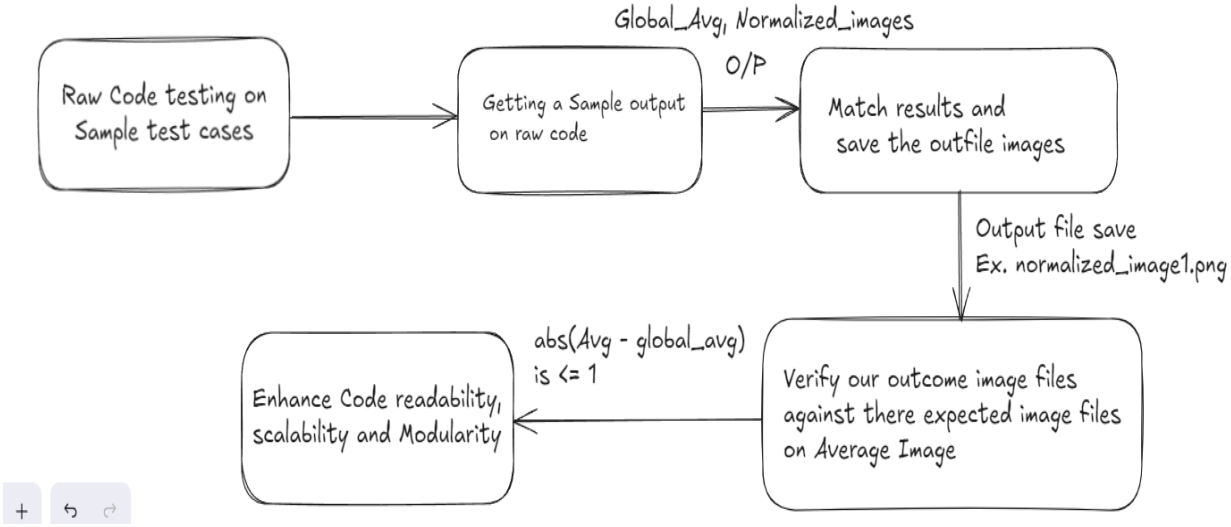
**Step 5:** Save Images: Save each normalized array as a PNG with required naming.

**Step 6:** Validate - Check if output images fall within the ±1 tolerance range.

## Extended Pipeline Steps:

- **Step 1:** Generate row code to test sample test case result.
- **Step 2:** Testing of results by using Python libraries i.e. OpenCV.
- **Step 3:** Match results to desired outcomes and save outputs.
- **Step 4:** Verify expected outcomes of test case results against predicted outcomes.
- **Step 5:** Refactor and enhance code modularity, structure, and readability for scalability.
- **Step 6:** Document the results and maintain logs for traceability and reproducibility.

# Flowchart




## Pseudo Code

```
1. Extract ZIP
Extract all files from satellite_images.zip

2. Compute Global Average
For i in 1 to 10:
    Read image{i}.png in grayscale
    If found, add to image list and track index
    Else, track as missing
    Flatten all pixels, calculate global_avg
    Return: global_avg, images, indices, missing list

3. Normalize Images
For each image:
    Compute factor = global_avg / image_avg
    Multiply image by factor, clip to [0,255], convert to uint8
    Save each normalized image as normalized_image{i}.png

For each image:
    If abs(image_avg - global_avg) ≤ 1, count as correct
    Return count of correct images

4. Calculate Score
    If all 10 images are correct → score = 10
    Else → score = 0

5. Main
Call global average function
Normalize images and check correctness
Calculate and print final score
```

# 3. Implementation Details

## Programming Language & Libraries:

**- Language:** Python 3

**- Libraries & Tools:** NumPy, OpenCV, zipfile, Git/Github

## Code Structure:

Structure.txt

```
kvgce-hackwise-problem1
├── myenv/
├── NormalizedImages/
│   ├── HiddenTestCase1/
│   ├── HiddenTestCase2/
│   ├── SampleOutputTestCase/
│   └── Satellite_images_Output/
├── satellite_images/
├── test cases/
│   ├── sample_expected/
│   └── sample_input/
├── compare_values.py
├── flowchart.excalidraw
├── main.py
├── problem1.pdf
├── requirements.txt
└── satellite_images.zip
```

## How to Run the Code:

main.py | flowchart.excalidraw | command_instructions.txt | compare_values.py

command_instructions.txt

```
Command Instructions to run the Code file

Step 1 - Activate Virtual env
    command - .\myvenv\Scripts\activate      ---> for windows
              source myvenv/bin/activate     ---> for Linux


    To Deactivate,
    command - deactivate

Step 2 - Install Necessary Libraries using
    command - pip install -r requirements.txt

Step 3 - Run the main.py file after Activation of venv
    command - python main.py

Step 4 - If want to check the that we have got same mean for Expected Outcome and Our result Outcome
    Run the compare_values.py

    command - python compare_values.py
```

# 4. Dataset handling:

- **Input**: satellite_images.zip from GitHub repository.
- **Processing**:
  - Unzipped using Python's zipfile inbuilt function.
  - Images loaded using OpenCV in grayscale mode.
- **Validation**:
  - Average intensity of each normalized image checked against computed global average which is between ±1.
  - Ensured that the number of images is exactly 10 and in expected naming format.

**Additional Measures Taken:**

- Corrupt or missing files trigger early exit with informative logs.
- Tested across multiple ZIP archives (sample datasets).

# 5. Output Format:

- Filename is created using variable name filename and stored filename format in it as f"normalized_image{i}.png".
- Error handling is done for Incorrect filename as per required.
- Finally, file is created using cv.imwrite() passing path where images should be created.

```
for i, img in zip(image_indices, normalized_images):
    filename = f"normalized_image{i}.png"

    if not filename.startswith("normalized_image") and not filename.endswith("f{i}.png"):
        return 0                                # If name does not follow the name of img

    cv.imwrite(f"NormalizedImages/SampleOutputTestCase/{filename}", img)
```

# 6. Challenges and Solutions:

### Challenges Faced:

- To save the result image file in correct required format as normalized_images1.png for each 10 images
- Error handling for images which are missing in given dataset
- Getting the required output as per guidelines

### Solutions Taken:

- Applied for loop from 1 to 10 for storing each normalized image file.
- If any particular image is missing, it is not passed to images list to compute Global Average.
- Re-Implementation of Normalization algorithm with error checking simultaneously

# 7. Test Case Results:

Below are the sample test case result having score 10 with 10/10 png images file correct satisfying the conditions as abs(avg - global_avg) <= 1

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  SEARCH ERROR

(myvenv) PS S:\Shivanjal\Hackathons\KVGC Hackathons\problem 1> python main.py
Missing Images: []
GLobal Average = 130.20
image1.png: avg=129.71 (within ±1 of 130.20)
image2.png: avg=129.73 (within ±1 of 130.20)
image3.png: avg=129.71 (within ±1 of 130.20)
image4.png: avg=129.70 (within ±1 of 130.20)
image5.png: avg=129.69 (within ±1 of 130.20)
image6.png: avg=129.69 (within ±1 of 130.20)
image7.png: avg=129.69 (within ±1 of 130.20)
image8.png: avg=129.70 (within ±1 of 130.20)
image9.png: avg=129.71 (within ±1 of 130.20)
image10.png: avg=129.67 (within ±1 of 130.20)
Score for Sample Test Case = 10
(myvenv) PS S:\Shivanjal\Hackathons\KVGC Hackathons\problem 1>
```

# 8. Future Improvements:

1. Robust Global Average Calculation:
    - Use a **trimmed mean** or **median** to reduce the influence of very dark or very bright pixels.
2. Noise Reduction Before Mean Calculation:
    - Apply **Gaussian blur** or **median filtering** before averaging.
3. Contrast Normalization Instead of Mean-Based Scaling:
    - Use **histogram equalization** to normalize brightness/contrast across images.
4. Adaptive Mean Targeting:
    - Use of **local group-wise averages**

# 9. References:


- OpenCV Library: https://opencv.org
- Zipping in Python: https://docs.python.org/3/library/zipfile.html
- Numpy Library: https://numpy.org
- GitHub Dataset: https://github.com/arshad-muhammad/kvgce-hackwise