Technical Report on Image Conversion Program with Wavelet Transform Integration

Zhangjinrui

August 2, 2025

1 Overview

1.1 Background and Objectives

In the fields of digital image processing and computer vision, standardizing image dimensions is a fundamental requirement in the preprocessing stage. This program aims to achieve the following core objectives:

1. Convert any input image into a grayscale image of size $\mathbf{2^N} \times \mathbf{2^N}$ pixels (where N is an integer specified by the user). This meets the need for uniform dimensions in specific scenarios (e.g., inputs for deep learning models, academic experiment comparisons). It also lays a foundation for subsequent wavelet transform applications, as standardized dimensions enable more efficient and accurate wavelet-based processing.

2. Support the export and visualization of image matrices. This facilitates in-depth image data processing for numerical analysis and machine learning tasks. It also provides a structured format for wavelet transform operations, simplifying the extraction of frequency domain features.

1.2 Dependency Library Description

The program relies on the following Python libraries. Their functions and installation commands are as follows:

2 Program Design and Implementation

2.1 Core Functional Modules

The program implements the complete process via the `convert_image` function, consisting of 5 core steps. These steps not only enable basic image conversion but also prepare images for potential wavelet transform applications:

1. **Image Reading and Grayscale Conversion**: Use `Image.open` from the Pillow library to open an image, then convert it to a single-channel grayscale image via `convert('L')` (pixel values range from 0 to 255, with 0 as black and 255 as white). Grayscale images simplify wavelet transform calculations by reducing data complexity while preserving essential structural information.

Library Name	Function Description	Installation Command
Pillow (PIL)	Image reading, format conversion, scaling, and saving. Provides basic image processing capabilities essential for pre-wavelet-transform image transformation.	`pip install pillow`
NumPy	Matrix-based conversion and numerical operations for image data. Critical for representing images as matrices (a prerequisite for wavelet transform calculations).	`pip install numpy`
math	Built-in Python mathematical operations (used for power calculations to determine dimensions). Assists in standardizing image size, which impacts wavelet transform performance.	No additional installation required

Table 1: Dependency Library Details

2. **Target Size Calculation**: Calculate the target output image side length as `target_size = 2 ** N` based on the user-input N (e.g., $N = 3$ yields an 8×8 pixel image). Standardized dimensions align with wavelets' multi-resolution analysis characteristics, enabling consistent decomposition and reconstruction.

3. **Aspect-Ratio-Preserving Scaling**: Compare the original image's width and height, dynamically compute the scaling ratio to ensure at least one side reaches the target size, and scale using the LANCZOS interpolation algorithm (a high-fidelity method that preserves image details). Maintaining details is critical for accurate wavelet transform results, as wavelets are sensitive to fine-scale features.

4. **Centered Cropping**: Compute cropping coordinates $(\text{left}, \text{top}, \text{right}, \text{bottom})$ and perform centered cropping on the scaled image to ensure a strict $\mathbf{2^N} \times \mathbf{2^N}$ output size. This step ensures image dimensions suit wavelet transform requirements (wavelet operations typically need consistent input sizes).

5. **Result Saving and Output**: Save the cropped image file and convert the image to a 2D matrix using NumPy, supporting matrix data saving and console output. The matrix format is compatible with wavelet transform processing, enabling seamless integration of subsequent wavelet-based algorithms.

2.2 Complete Code Implementation

The following Python code (runnable as-is) performs basic image conversion and prepares images for wavelet transform via standardized matrix representation: LINK HERE

Code Key Logic Supplementary Explanation:

- **LANCZOS Interpolation**: Invoked via `Image.Resampling.LANCZOS`, it ensures high-fidelity scaling—critical for preserving wavelet-transform-relevant features (e.g., edges, textures).

- **Matrix Saving**: The `np.save` function stores the image matrix as a `.npy` file, enabling reuse in wavelet transform workflows (e.g., import into MATLAB/Python analysis scripts).

3 User Guide

3.1 Parameter Configuration

When calling `convert_image`, configure the following parameters:

Parameter Name	**Type**	**Description**
`image_path`	String	Input image path (e.g., `"input.jpg"`).
`output_path`	String	Output image save path (e.g., `"output.png"`).
`N`	Integer	Controls output size as $\mathbf{2^N} \times \mathbf{2^N}$ (e.g., $N = 5$ for 32×32). Critical for defining wavelet transform resolution.
`save_matrix`	Boolean	Whether to save the image matrix (default: `False`). Saves time in wavelet transform preprocessing.

Table 2: Function Parameter Description

3.2 Example Call

The following example demonstrates `convert_image` usage, producing wavelet-transform-ready images: LINK HERE

Running Process:

1. Enter N in the terminal (e.g., `3` for an 8×8 image). This N influences wavelet transform granularity (different sizes enable multi-resolution analysis).
2. The program converts the image, outputting size info and the matrix. The matrix serves as direct input for wavelet transform functions.
3. If `save_matrix` is enabled, the matrix saves as a `.npy` file for wavelet transform tasks.

4 Application Scenarios and Expansion Suggestions

4.1 Typical Application Scenarios

1. **Academic Research**: Standardizes image sizes for uniform input in image recognition, style transfer, etc. Wavelet transform can further extract multi-scale features, aiding algorithm research.
2. **Teaching Demonstration**: Visualizes RGB-to-grayscale conversion and matrix representation. Can be extended to show wavelet transform's frequency-domain decomposition.
3. **Data Preprocessing**: Prepares standardized image data for machine learning (e.g., MNIST-like datasets). Wavelet-based feature extraction enhances model input quality.

4.2 Function Expansion Suggestions

1. **Multi-Channel Support**: Add RGB processing to retain color information, enabling wavelet transform on color images.

2. **Batch Processing**: Automate multi-image conversion via folder traversal, streamlining large-scale wavelet transform workflows.

3. **Visualization Enhancement**: Integrate Matplotlib to compare original and converted images. Extend to visualize wavelet decomposition/reconstruction for intuitive validation.

5 Summary

This program converts images to $\mathbf{2^N} \times \mathbf{2^N}$ grayscale format, balancing practicality and extensibility. Leveraging Pillow and NumPy, it meets basic image processing needs and provides standardized input for deep learning (e.g., TensorFlow, PyTorch). By outputting wavelet-transform-compatible matrices, it supports advanced image analysis. It serves as a foundation for entry-level image processing practice, academic experiments, and wavelet-integrated preprocessing pipelines.