1. Executive Summary

This project implements a Python-based demonstration of the **Convolution Theorem**, a fundamental principle in signal and image processing. The system applies Gaussian blurring to digital images using two distinct methods: spatial domain convolution and frequency domain multiplication. By comparing the outputs of these two methods, the project mathematically and visually validates that convolution in space is equivalent to multiplication in the frequency domain.

2. Theoretical Framework

The core premise of this application is the **Convolution Theorem**, which establishes the relationship between the spatial and frequency domains.

2.1 Mathematical Definition

The theorem states that the convolution of two functions, $f(x,y)$ (the image) and $h(x,y)$ (the kernel), in the spatial domain is equivalent to the element-wise multiplication of their Fourier Transforms.

$$f(x,y) * h(x,y) \iff F(u,v) \cdot H(u,v)$$

Where:

- $*$ denotes the convolution operation.
- $\cdot$ denotes element-wise multiplication.
- $F(u,v)$ and $H(u,v)$ are the Fourier Transforms of the image and kernel, respectively.

3. Methodology

The application processes input images using two parallel pipelines to generate comparable results.

3.1 Spatial Domain Filtering (Method A)

- **Kernel Generation:** A 2D Gaussian kernel is generated based on a user-defined sigma (σ) and kernel size.
- **Convolution:** The kernel is convolved with the input image using a sliding window approach (via OpenCV's filter2D).
- **Outcome:** A blurred image representing traditional spatial smoothing.

3.2 Frequency Domain Filtering (Method B)

- **Transformation:** The Discrete Fourier Transform (DFT) is computed for the input image.
- **Padding:** The Gaussian kernel is padded to match the exact dimensions of the input image.
- **Multiplication:** The transformed image and transformed kernel are multiplied element-wise.

- **Inversion:** An Inverse DFT is applied to the product to recover the spatial representation.
- **Outcome:** A blurred image reconstructed from frequency data.

4. System Architecture

The project is structured to ensure modularity and ease of verification.

Component	Description
src/main.py	Core logic for image loading, processing, and mathematical verification.
Input Images/	Repository for raw source images (supports JPG, PNG, BMP, TIFF).
Results/	Destination for processed outputs (_spatial.png, _fourier.png, _diff.png).
run_project.bat	Automation script for environment setup and execution.

5. Verification & Results

To validate the Convolution Theorem, the system employs both statistical and visual metrics.

5.1 Statistical Validation (MSE)

The **Mean Squared Error (MSE)** is calculated between the spatial output (I_s) and the frequency output (I_f):

$$MSE = \frac{1}{N} \sum (I_s - I_f)^2$$

- **Success Criterion:** An MSE < 1.0 confirms the mathematical equivalence of the two methods, accounting for minor floating-point precision errors.

5.2 Visual Validation (Difference Maps)

A "Difference Map" is generated for every processed image.

- **Black Pixels:** Indicate perfect identity between methods.
- **Bright Pixels:** Indicate discrepancies.
- **Expected Result:** A predominantly black image, visually confirming the theorem.

6. Operational Guide

6.1 Requirements

- **OS:** Windows (for batch automation)
- **Runtime:** Python 3.7+
- **Libraries:** numpy, opencv-python, matplotlib

6.2 Execution

The project includes a run_project.bat file for automated setup.

1. Place target images in the Input Images/ folder.
2. Run the batch file to initialize the virtual environment and execute the script.

3. Review console output for MSE scores and the Results/ folder for visual outputs.

6.3 Customization

Users can adjust the blur intensity via command-line arguments:

- --sigma: Controls the variance of the Gaussian distribution.
- --kernel-size: Sets the aperture size of the filter.

7. Project Access & Source Code

The complete source code, including the implementation of the Fourier Transform pipeline and the automated test suite, is available on GitHub.

- **Repository URL:** https://github.com/DhruvilKachchhi/Computer-Vision/tree/main/Image%20Blurring