A Python Toolkit for Classical Digital Image Processing: Spatial-Domain Enhancement, Filtering, and Edge Detection

Shahir Abdullah
shahir2561@gmail.com

***Abstract*—**This report documents an open-source Python toolkit implementing sixteen classical digital image processing techniques spanning intensity transformations, histogram-based processing, spatial (neighborhood) filtering, and edge detection. Each technique is implemented as a self-contained Jupyter notebook using NumPy, OpenCV, and Matplotlib, with several notebooks pairing a from-scratch, pixel-loop implementation against OpenCV's built-in equivalent for validation. Techniques covered include image negatives, log and power-law (gamma) transformations, contrast stretching, intensity-level slicing, bit-plane slicing, histogram equalization and histogram matching (specification), averaging and geometric-mean smoothing filters of multiple mask sizes, median filtering, Laplacian sharpening, Sobel and Laplacian edge detection, unsharp masking and high-boost filtering, and a multi-stage combined spatial-enhancement pipeline applied to a medical X-ray image. This report describes the implemented algorithms, the exact formulas and parameters used in the codebase, the tools and sample images involved, and observed implementation notes. As of writing, the repository has been starred 20 times and forked 8 times on GitHub, indicating reuse by other learners as a reference implementation for standard digital image processing coursework.

Index Terms*—digital image processing, spatial filtering, histogram equalization, edge detection, image enhancement, OpenCV, Python.*

I. INTRODUCTION

Digital image processing courses and textbooks typically build up a standard toolbox of spatial-domain techniques: point-wise intensity transformations, histogram-based enhancement, neighborhood (convolution) filtering, and edge detection. This report documents a Python repository that implements this toolbox as sixteen independent Jupyter notebooks, each demonstrating one technique end-to-end on a real or synthetic sample image, from loading and displaying the input, through applying the transformation, to visualizing the histogram or output where relevant.

The repository is written entirely in Python using OpenCV for image I/O and select built-in operators, NumPy for array manipulation and manual convolution, and Matplotlib for visualization. Several notebooks implement a filter from scratch with explicit nested pixel loops and then cross-check the result against the corresponding OpenCV function, which makes the underlying mathematics explicit rather than hidden behind a library call. As of writing, the repository has received 20 stars and 8 forks on GitHub, indicating that it has been useful to other learners as a reference implementation.

II. BACKGROUND

The techniques implemented here follow the standard spatial-domain image enhancement and filtering curriculum described in Gonzalez and Woods' Digital Image Processing [2], which is the de facto reference text for this material and whose terminology (e.g., the power-law transform, the discrete Laplacian mask, the combined spatial-enhancement example applied to an X-ray image) is mirrored closely in the codebase. The implementation relies on three widely used Python libraries: OpenCV [3] for image I/O, color-space conversion, and reference filter implementations; NumPy [4] for array-based pixel manipulation; and Matplotlib [5] for displaying images and histograms.

III. REPOSITORY OVERVIEW

The repository consists of sixteen Jupyter notebooks, an images/ directory of sample and test images (including a chest X-ray, a kidney scan, a breast-cyst image, several natural photographs, and synthetic test patterns), and a top-level README. Table I summarizes the implemented techniques by category.

Category	Techniques
Intensity transformations	Image negative; log transform; power-law (gamma) transform; contrast stretching; intensity-level slicing
Histogram processing	Histogram equalization; histogram matching (specification)
Spatial filtering	Averaging (low-pass) filter, 3×3/5×5/9×9; high-pass averaging filter; geometric mean filter, 3×3/5×5/9×9; median filter
Sharpening & edge detection	Laplacian sharpening; Sobel & Laplacian edge detection; unsharp masking & high-boost filtering; combined multi-stage enhancement (Gonzalez & Woods X-ray example); bit-plane slicing

TABLE I. Implemented techniques grouped by category.

IV. IMPLEMENTED TECHNIQUES

A. Intensity Transformations

Image Negative computes s = 255 − r independently on each of the R, G, B channels via a per-pixel loop; the accompanying notebook notes this is suited to enhancing detail embedded in dark regions of an image and

demonstrates it on a breast-cyst image. Log Transformation applies $s = c \cdot \log(1 + r)$ with $c = 255 / \log(1 + \max(r))$, which expands dark pixel values while compressing bright ones, demonstrated on a Fourier-spectrum image. Power-Law (Gamma) Transformation applies $s = 255 \cdot (r/255)^{\gamma}$ and is demonstrated across five gamma values (0.1, 0.5, 1.2, 2.2, 3.2) on a photograph to illustrate gamma correction for display. Contrast Stretching applies the piecewise-linear mapping $pout = (pin - minI) \cdot (maxO - minO)/(maxI - minI) + minO$ per channel, demonstrated with example bounds minI = 3, maxI = 250, minO = 0, maxO = 155 on a photograph. Intensity-Level Slicing highlights a target intensity band by thresholding each channel against a min/max range.

B. Histogram-Based Processing

Histogram Equalization is implemented from scratch per color channel: a histogram and cumulative distribution function (CDF) are computed for each of the B, G, R channels, zero bins are masked out, the CDF is min–max normalized to [0, 255], and the result is used as a per-channel lookup table; the notebook additionally validates this against OpenCV's cv2.equalizeHist. Histogram Matching (Specification) maps the cumulative distribution of a source image onto that of a reference ("specified") image via nearest-quantile lookup between the two images' cumulative histograms; the notebook explicitly credits an external tutorial (TheAILearner [6]) as the source of this algorithm.

C. Spatial (Neighborhood) Filtering

Averaging (Low-Pass) Filters of size 3×3, 5×5, and 9×9 apply a uniform box-filter convolution via nested pixel/kernel loops to a synthetic striped test pattern, illustrating progressively stronger blurring as the mask grows. A High-Pass Averaging Filter uses a 3×3 mask with center weight 8/9 and surrounding weights −1/9, sharpening high-frequency detail while suppressing the low-frequency background. The Geometric Mean Filter (3×3, 5×5, 9×9) replaces each pixel with the N-th root of the product of its N-pixel neighborhood, $s = (\Pi \text{ neighborhood})^{(1/N)}$, an alternative to the arithmetic mean that tends to preserve edge detail better for certain noise types. The Median Filter (3×3) replaces each pixel with the median of its neighborhood to remove salt-and-pepper noise, demonstrated on a noisy test image.

D. Sharpening and Edge Detection

Laplacian Sharpening convolves an explicit discrete Laplacian mask (center weight −4, four-neighbor weight 1, corner weight 0) over the image and forms the sharpened result as the original image minus the filtered response, matching the classical $g(x, y) = f(x, y) - \nabla^2 f(x, y)$ formulation, and is cross-checked against OpenCV's cv2.Laplacian. Sobel and Laplacian Edge Detection applies a Gaussian blur for noise suppression followed by 5×5 Sobel gradients in x and y (via cv2.Sobel) and a Laplacian response (via cv2.Laplacian), displayed side-by-side for comparison. Unsharp Masking and High-Boost Filtering computes mask = original − blurred(3×3 average) and forms sharpened = original + k·mask, with k = 1 for classical unsharp masking and k > 1 (k = 2 demonstrated) for high-boost filtering; a second variant using cv2.GaussianBlur and cv2.addWeighted is also included.

E. Combined Enhancement and Bit-Plane Slicing

Combining Spatial Enhancement Methods reproduces the classical multi-stage enhancement example (Gonzalez & Woods [2]) on a chest X-ray image: a Laplacian-sharpened version of the image is computed, a Sobel gradient image is computed and smoothed with a 5×5 averaging filter, the sharpened and smoothed-gradient images are multiplied together to form a mask that emphasizes edges near high-gradient regions while suppressing it elsewhere, the mask is added back to the original image for a second sharpening pass, and a final power-law adjustment ($\gamma = 1.1$) is applied for display contrast — progressively enhancing fine bone detail while preserving the smooth background. Bit-Plane Slicing decomposes an 8-bit grayscale image into its eight individual bit planes using each pixel's binary representation, and recombines the four most-significant planes to approximate the original image using substantially less information than the full 8-bit image.

V. IMPLEMENTATION NOTES

Most spatial filters are implemented with explicit nested Python for-loops over image rows, columns, and kernel offsets rather than vectorized NumPy operations or cv2.filter2D, which keeps the underlying convolution mathematics explicit and readable at the cost of runtime performance — each such filter has $O(\text{rows} \times \text{cols} \times \text{kernel}^2)$ time complexity and is not intended for large images or real-time use. Where an OpenCV equivalent exists (histogram equalization, Laplacian, Sobel, Gaussian blur), several notebooks include it as a validation or alternative-approach cell alongside the manual implementation.

A minor implementation inconsistency was observed in the Intensity-Level Slicing notebook: the thresholding condition intended for the blue channel (pixel[1] in the code as written) is applied twice, while the red-channel condition operates on an already 255-inverted value; this does not affect the other fifteen notebooks and is noted here for the repository owner's awareness rather than as a criticism of the overall codebase.

VI. ADOPTION

As of writing, the repository has received 20 stars and 8 forks on GitHub. Given that the repository consists of standalone, well-commented notebooks mapped directly onto standard digital image processing course topics, this level of adoption suggests it has been used by other students and practitioners as a working reference implementation alongside textbook study.

VII. CONCLUSION

This report described a Python toolkit of sixteen Jupyter notebooks implementing the core spatial-domain techniques of classical digital image processing — intensity transformations, histogram processing, neighborhood filtering, and edge detection — each demonstrated on a real or synthetic sample image and, where applicable, validated against OpenCV's built-in equivalents. The toolkit's direct mapping onto standard course material, combined with its GitHub adoption (20 stars, 8 forks), supports its continued use as an educational reference implementation.

REFERENCES

[1] S. Abdullah, "Digital-Image-Processing," GitHub repository, 2020. [Online]. Available: https://github.com/Shahir-Abdullah/Digital-Image-Processing

[2] R. C. Gonzalez and R. E. Woods, Digital Image Processing, 4th ed. New York, NY, USA: Pearson, 2018.

[3] G. Bradski, "The OpenCV Library," Dr. Dobb’s Journal of Software Tools, 2000.

[4] C. R. Harris et al., "Array programming with NumPy," Nature, vol. 585, no. 7825, pp. 357–362, 2020.

[5] J. D. Hunter, "Matplotlib: A 2D graphics environment," Computing in Science & Engineering, vol. 9, no. 3, pp. 90–95, 2007.

[6] TheAILearner, "Histogram Matching (Specification)," 2019. [Online]. Available: https://theailearner.com/2019/04/10/histogram-matching-specification/