

Adaptive Restoration Before Edge Detection Using Image Condition Features

Gopal Borakar
Department of E&CE
KLS VBIT, Haliyal
Email: gopal.borakar@gmail.com

Dr. Plasin Francis Dias
Department of E&CE
KLS VBIT, Haliyal
Email: plasindias@gmail.com

Abstract—Edge detection gives poor results when the input image contains noise, blur, or weak contrast. In the developed prototype, the same edge detector did not behave in the same way for all uploaded images. Some noisy images produced many false edge pixels, while blurred images lost useful object boundaries. To handle this problem, this paper describes a Flask-based system in which image restoration is carried out before edge detection. The uploaded image is first examined through variance, standard deviation, impulse-noise ratio, and Laplacian variance. These values guide the restoration choice through learned bucket mapping, fuzzy rules, or a simple fallback decision. The restoration stage uses Gaussian, median, non-local means, bilateral filtering, and an optional DeblurGAN-v2 path. After restoration, Canny, Sobel, Prewitt, or adaptive thresholding is applied to obtain the edge output. The web interface shows the original image, restored image, edge map, and evaluation values. The experiments suggest that choosing the filter from the image condition is more practical than applying one fixed preprocessing method to every image.

Index Terms—image restoration, edge detection, adaptive filtering, fuzzy rules, learned mapping, Flask, Canny detector

I. INTRODUCTION

Edges describe sudden changes in intensity and usually indicate object outlines, surface changes, or region boundaries. They are used in medical image analysis, document processing, industrial inspection, surveillance, remote sensing, and robotic vision. In clean images, a common detector such as Canny or Sobel can give a meaningful edge map. The difficulty appears when the uploaded image is not clean. Real images may contain blur, sensor noise, compression marks, poor lighting, or salt-and-pepper noise. These degradations alter the gradient response and make the final edge map less reliable.

During testing of the application, direct edge detection gave uneven results. Salt-and-pepper noise appeared as small isolated edges. Blur removed fine details and made some boundaries incomplete. Low contrast reduced the separation between foreground and background regions. These observations show that edge detection should not always be treated as a direct one-step operation. A small restoration stage before edge extraction can improve the input quality, but that restoration stage must be selected carefully.

Classical detectors such as Canny and Prewitt are useful because they are simple and fast [1], [2]. However, they are affected by noise and threshold choice. Restoration methods

also have different behavior. Gaussian smoothing is suitable for mild noise, but it can soften real edges. Median filtering works well for impulse noise. Non-local means can reduce random noise using patch similarity, although it takes more time. Bilateral filtering smooths the image while preserving stronger intensity changes [3], [4]. Since each method has a different strength, one common filter cannot be expected to work well on every uploaded image.

The proposed system uses image features to select the restoration method before edge detection. The image is profiled first, and the selected restoration method is then applied before generating the edge map. The implementation is made as a Flask web application so that the user can upload an image and compare the original, restored, and edge-detected outputs on the same page.

The main contributions are:

- extraction of variance, standard deviation, impulse-noise ratio, and Laplacian variance from the uploaded image;
- restoration selection using learned bucket mapping, fuzzy rule selection, and fallback logic;
- use of Gaussian, median, non-local means, bilateral, and optional DeblurGAN-v2 restoration methods;
- generation of Canny, Sobel, Prewitt, and adaptive-threshold based edge outputs;
- a simple browser interface for result viewing, metric display, and output download.

II. RELATED WORK

Canny proposed a multi-stage edge detector that performs smoothing, gradient computation, non-maximum suppression, and hysteresis thresholding [1]. The method remains popular because it gives good localization in many images. Its output, however, changes when the image is noisy, blurred, or when threshold values are not suitable.

Prewitt introduced a gradient-based method for object enhancement and edge extraction [2]. It is easy to implement and fast, but it responds to sudden intensity changes. If the input contains noise, these noise pixels may also be detected as edges.

Tomasi and Manduchi introduced bilateral filtering for edge-preserving smoothing [3]. The method considers both spatial distance and intensity similarity. This helps reduce noise while retaining strong boundaries. It is still not a complete solution

TABLE I
SUMMARY OF RELATED WORK AND ITS RELEVANCE

Ref.	Method	Main Idea	Observation for the Proposed System
[1]	Canny edge detector	Uses smoothing, gradient estimation, non-maximum suppression, and hysteresis thresholding.	Gives good edges on clean images, but noise and threshold changes affect the output.
[2]	Prewitt operator	Uses simple gradient masks for edge and object enhancement.	Fast and simple, but noisy pixel changes may be treated as real edges.
[3]	Bilateral filter	Smooths an image using spatial closeness and intensity similarity.	Useful for edge-preserving smoothing, but not ideal for heavy impulse noise.
[4]	Non-local means	Denoises by averaging pixels from similar patches across the image.	Handles random noise better than simple averaging, but increases processing time.
[5]	SSIM	Measures structural similarity between reference and restored images.	Useful for restoration quality, but not enough for boundary overlap checking.
[8]	DeblurGAN-v2	Uses deep learning for motion deblurring.	Powerful when trained weights are available, but heavier for a basic Flask system.

for all cases, especially when the image has strong impulse noise.

Buades, Coll, and Morel proposed non-local means denoising [4]. Instead of depending only on nearby pixels, it searches for similar patches in the image. It can produce smoother results for random noise, but the processing time is higher than simple local filters.

Structural similarity index measure is widely used to compare restored images with reference images [5]. It is useful for measuring structural quality, but it does not directly check whether detected edge pixels match the true boundary locations. Therefore, boundary-oriented measures such as precision, recall, F1-score, and IoU are also used in this paper.

Deep deblurring models such as DeblurGAN and DeblurGAN-v2 can improve motion-blurred images [7], [8]. These methods need trained weights and higher computing support. For that reason, the present system keeps DeblurGAN-v2 as an optional path and uses classical restoration methods for normal execution.

III. PROPOSED SYSTEM

The system follows a restoration-assisted edge detection workflow. It accepts an image from the user, studies the image condition, chooses a restoration method, applies the selected method, generates the edge map, and presents the result in the browser.

A. System Overview

The user uploads an image from the Flask page. The backend reads the file and converts it into a processing format.

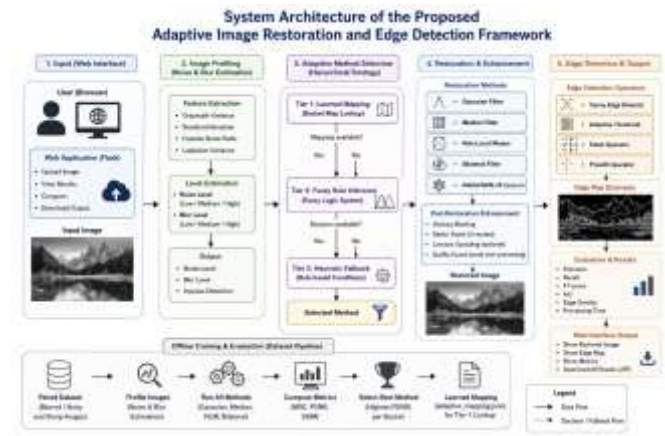


Fig. 1. System architecture of the adaptive image restoration and edge detection workflow.

The profiling block calculates variance, standard deviation, impulse-noise ratio, and Laplacian variance. These values are sent to the adaptive selector. The selected restoration method is applied, and the restored image is passed to the edge detection module. The result page then displays the original image, restored image, edge map, and metric values.

B. System Architecture

Fig. 1 shows the proposed architecture. The figure is included as an external image file named `system_architecture.png`. Keep this PNG file in the same Overleaf folder as the LaTeX source.

C. Processing Flow

After uploading, the image is converted into grayscale for feature calculation. The measured values are used to estimate the noise and blur condition. If the condition matches an existing learned bucket, the stored method for that bucket is selected. If no bucket is available, fuzzy rules are used. If the fuzzy result is still uncertain, fallback rules select a practical method. Finally, edge detection is applied and the output is shown with evaluation values.

IV. METHODOLOGY

A. Image Characterization

The uploaded image is first converted to grayscale. This reduces computation and keeps the feature calculation simple. Variance and standard deviation are used to understand how widely the pixel values are distributed. A low spread may indicate a smooth or low-contrast image, while a high spread may come from texture, contrast, or noise. Laplacian variance is used as a blur cue because sharp images normally produce stronger Laplacian responses. The impulse-noise ratio is estimated by observing abnormal bright and dark pixels compared with local median behavior.

TABLE II
RESTORATION METHODS USED IN THE SYSTEM

Method	Purpose in the Application
Gaussian filter	Mild smoothing when the measured noise is low or moderate.
Median filter	Removal of salt-and-pepper noise and isolated impulse pixels.
Non-local means	Random-noise reduction using similar image patches.
Bilateral filter	Smoothing while keeping stronger edge structures visible.
DeblurGAN-v2	Optional deblurring path when model weights and resources are available.

B. Noise and Blur Profiling

The extracted values provide only an approximate condition. A real image may contain more than one degradation. For example, the same image may be slightly blurred and also contain random noise. Therefore, the selector does not depend on a single hard threshold. It uses learned mapping first, fuzzy rules next, and fallback rules when the earlier stages do not give a clear result.

C. Hierarchical Restoration Selection

The first stage is learned bucket mapping. During training or calibration, images are grouped by noise and blur ranges. Several restoration methods are tested for each group, and the method with better output is stored for that bucket. During testing, an uploaded image that falls into a known bucket receives the stored restoration method.

The second stage is fuzzy selection. Fuzzy logic is useful when image conditions are not sharply separated [6]. An image may be moderately noisy rather than clearly noisy or clean. Fuzzy rules help in such intermediate situations.

The third stage is heuristic fallback. If the earlier stages do not give a confident decision, simple rules are used. Median filtering is selected for strong impulse noise. Gaussian filtering is used for mild smoothing. Bilateral filtering is selected when smoothing is needed but strong boundaries must remain visible. This fallback stage keeps the application usable even when learned bucket data are limited.

D. Restoration and Enhancement

After selection, the chosen restoration operation is applied to the uploaded image. Optional enhancement may be used when boundaries remain weak after restoration. Examples include unsharp masking or controlled deblur boosting. These steps are applied carefully because excessive sharpening can create artificial edges.

E. Edge Detection

The restored image is passed to the edge detection block. Canny is used as the main detector because it includes smoothing, gradient calculation, non-maximum suppression, and hysteresis thresholding. Sobel, Prewitt, and adaptive thresholding are also included so that alternative outputs can be generated when required.

V. Algorithm

Algorithm 1 Adaptive Restoration and Edge Detection

```

1: Start
2: Upload input image
3: Convert image into processing format
4: Calculate variance, standard deviation, impulse-noise ratio, and Laplacian variance
5: Estimate approximate noise and blur condition
6: if matching learned bucket exists then
7:   Select method stored for that bucket
8: else
9:   Apply fuzzy rule-based selection
10: end if
11: if method is still not clear then
12:   Apply fallback restoration rule
13: end if
14: Restore image using selected method
15: Apply optional enhancement if required
16: Generate edge output using selected detector
17: Compute precision, recall, F1-score, and IoU
18: Display restored image, edge image, and metric values
19: Stop

```

Algorithm 1 presents the complete workflow of the proposed adaptive restoration and edge detection system. The process begins when the user uploads an input image. The uploaded image is first converted into a suitable processing format so that image features can be extracted and analyzed.

In the next step, important image characteristics such as variance, standard deviation, impulse-noise ratio, and Laplacian variance are calculated. These values help the system understand the condition of the image. For example, variance and standard deviation indicate the intensity variation in the image, the impulse-noise ratio helps identify salt-and-pepper noise, and Laplacian variance is useful for estimating the level of blur.

Based on these extracted features, the system estimates whether the image is affected by noise, blur, or both. If the image condition matches an already learned bucket, the system directly selects the restoration method stored for that bucket. This helps improve processing speed and consistency. When no suitable learned bucket is found, the system uses fuzzy rule-based selection to choose an appropriate restoration technique.

VI. SYSTEM IMPLEMENTATION

The application was implemented in Python using the Flask framework. Flask manages the upload route, result rendering, and download options. OpenCV, NumPy, and scikit-image are used for image loading, feature extraction, filtering, edge generation, and metric calculation [11], [12]. The output page places the original image, restored image, and edge map together. This layout made manual testing easier because the effect of the selected restoration method could be checked visually.

TABLE IV
EXPERIMENTAL SETUP

Item	Description
Dataset layout	Training, validation, and testing folders with matching ground-truth files.
Ground truth	Boundary annotations stored in MAT format.
Main pipeline	Adaptive restoration followed by Canny edge detection.
Metrics	Precision, recall, F1-score, and IoU.
Platform	Flask-based web application.

TABLE V
EDGE DETECTION EVALUATION RESULTS

Split	Samples	Precision	Recall	F1	IoU
Train	200	0.2434	0.1600	0.1768	0.0981
Validation	100	0.2407	0.1666	0.1748	0.0967
Test	200	0.2472	0.1604	0.1761	0.0975

VII. EXPERIMENTAL RESULTS

A. Experimental Setup

The dataset is arranged into train, validation, and test folders. Each image has a corresponding ground-truth boundary annotation in MAT format. The pipeline applies adaptive restoration before Canny edge detection. Precision, recall, F1-score, and IoU are used to evaluate the detected boundaries.

B. Quantitative Results

Table V reports the boundary detection results. The values for train, validation, and test splits are close to each other. This indicates that the pipeline behaves consistently across the available data. The F1-score and IoU are moderate because the images contain weak edges, textured regions, and boundary locations that may not perfectly align with the annotation.

C. Practical Testing Notes

Several degraded images were uploaded through the web page during manual testing. Direct Canny detection was not reliable for all inputs. Noisy images produced scattered edge responses, whereas blurred images produced broken or missing boundaries. Median filtering improved salt-and-pepper cases. Bilateral filtering reduced noise while keeping visible boundaries. Non-local means produced smoother results for random noise, but it required more processing time. In some cases, the visual edge map looked acceptable even when the score was low because the detected pixels did not exactly coincide with the annotated boundary.

VIII. DISCUSSION

The results support the need for condition-based preprocessing. A fixed restoration filter is too limited for mixed image degradations. Median filtering is more suitable for impulse noise. Non-local means is useful for random noise but slower. Bilateral filtering is helpful when the image needs smoothing without losing major boundaries.

The selector also makes the output more explainable. The chosen method can be linked to measured features, learned bucket mapping, fuzzy output, or fallback rules. This is easier to justify than applying an unknown or fixed preprocessing step. The Flask result page also supports interpretation by showing the original image, restored result, and edge map side by side.

A. Limitations

The learned bucket table depends on representative training samples. If the stored mapping is small, the system depends more on fuzzy rules and fallback decisions. The DeblurGAN-v2 path needs model weights and better hardware support. The present scores are also affected by weak boundaries, textured backgrounds, and mismatch between detected edge pixels and ground-truth annotations.

B. Future Scope

Future improvements can include automatic Canny threshold tuning, learned denoising, deep edge detection, and testing with a larger dataset. The interface can also show the selected method, measured noise level, blur level, and reason for selection so that users can understand why a particular filter was applied.

IX. CONCLUSION

This paper described a condition-aware image restoration and edge detection system built using Flask. The image is examined before edge extraction using variance, standard deviation, impulse-noise ratio, and Laplacian variance. Based on these values, the system selects a restoration method using learned buckets, fuzzy rules, or fallback logic. Testing shows that different degradations need different preprocessing choices. Median filtering helps in impulse-noise cases, bilateral filtering preserves strong boundaries during smoothing, and non-local means improves random-noise cases at the cost of extra processing time. The current metric values can be improved further, but the system provides a practical starting point for restoration-assisted edge extraction in a web application.

REFERENCES

- [1] J. Canny, "A computational approach to edge detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-8, no. 6, pp. 679–698, 1986.
- [2] J. M. S. Prewitt, "Object enhancement and extraction," in *Picture Processing and Psychopictories*. New York, NY, USA: Academic Press, 1970, pp. 75–149.
- [3] C. Tomasi and R. Manduchi, "Bilateral filtering for gray and color images," in *Proc. IEEE International Conference on Computer Vision*, 1998, pp. 839–846.
- [4] A. Buades, B. Coll, and J. M. Morel, "A non-local algorithm for image denoising," in *Proc. IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2005, pp. 60–65.
- [5] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: From error visibility to structural similarity," *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, 2004.
- [6] L. A. Zadeh, "Fuzzy sets," *Information and Control*, vol. 8, no. 3, pp. 338–353, 1965.

- [7] O. Kupyn, V. Budzan, M. Mykhailych, D. Mishkin, and J. Matas, "DeblurGAN: Blind motion deblurring using conditional adversarial networks," in *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 8183–8192.
- [8] O. Kupyn, T. Martyniuk, J. Wu, and Z. Wang, "DeblurGAN-v2: Deblurring orders-of-magnitude faster and better," in *Proc. IEEE International Conference on Computer Vision*, 2019, pp. 8878–8887.
- [9] D. Martin, C. Fowlkes, D. Tal, and J. Malik, "A database of human segmented natural images and its application to evaluating segmentation algorithms," in *Proc. IEEE International Conference on Computer Vision*, 2001, pp. 416–423.
- [10] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, 4th ed. New York, NY, USA: Pearson, 2018.
- [11] G. Bradski, "The OpenCV library," *Dr. Dobbs's Journal of Software Tools*, 2000.
- [12] S. van der Walt et al., "scikit-image: Image processing in Python," *PeerJ*, vol. 2, p. e453, 2014.
- [13] W. McKinney, *Python for Data Analysis*, 2nd ed. Sebastopol, CA, USA: O'Reilly Media, 2018.
- [14] N. Otsu, "A threshold selection method from gray-level histograms," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 9, no. 1, pp. 62–66, 1979.
- [15] M. Sokolova and G. Lapalme, "A systematic analysis of performance measures for classification tasks," *Information Processing and Management*, vol. 45, no. 4, pp. 427–437, 2009.