Analytical Global Median Filtering Forensics

Shivangi Gaur (12241720)
Shubham Mahajan (12241730)
Vedansh Sharma (12242000

May 6, 2025

1 Introduction

In recent years, the field of digital image forensics has drawn attention due to its role in verifying the authenticity of images.

This paper presents a new approach to detecting median filtering. It introduces a unique set of features based on the skewness and kurtosis of small image regions. These features help highlight subtle changes introduced by the filtering process.

- Passive image forensics focuses on identifying signs of tampering without needing access to the original image. These methods typically rely on detecting statistical inconsistencies that hint at common types of forgery.
- Lately, researchers have been working on detecting specific image editing operations such as median filtering, contrast adjustment, and resampling. Among these, median filtering is particularly tricky—it's a popular nonlinear filter that smooths images while preserving edges, which makes it a go-to method for hiding traces of manipulation.
- Compared to existing techniques, the proposed method achieves better accuracy, especially in challenging conditions like compressed JPEG images or low-resolution formats. It performs well in terms of standard metrics, including lower error rates and higher scores on ROC curves.
- The paper introduced a novel feature set derived from skewness and kurtosis histograms of image blocks for detecting median filtering.
- **Skewness**:
  - Represents the asymmetry of the distribution.
  - Zero skewness indicates symmetry, while positive and negative skewness denote right-tail and left-tail distributions, respectively.
- **Kurtosis**:
  - Describes the distribution's peakedness or flatness.
  - Absolute kurtosis can be positive or undefined, indicating the distribution's tails' weight relative to the center.

2 Feature Set Construction

The proposed 19-dimensional feature vector is built by examining statistical patterns that emerge when an image undergoes median filtering. These patterns manifest in terms of changes in skewness and kurtosis—two statistical moments—computed over small sliding blocks within the image. The process involves dividing both the original and median-filtered images into overlapping $\xi \times \xi$ blocks (commonly 3×3), then calculating local and global moment-based statistics.

From an implementation standpoint, the feature vector is formed by aggregating four types of information:

1. **Skewness Histogram Bin Heights (SBH):** These are 9 values indicating the frequency of specific skewness levels across blocks. Each bin corresponds to a predefined skewness range. Higher counts in some bins, especially extreme ones, reveal the presence of median filtering, which tends to shift or reduce asymmetry in pixel intensities.

2. **Kurtosis Histogram Bin Heights (KBH):** Similar to SBH, these 4 values capture how often certain kurtosis values appear across blocks. Median filtering often results in increased peakedness (leptokurtic distributions), making these values significant in distinguishing between original and filtered images.

3. **Statistical Moments of Skewness and Kurtosis ($\mathbf{f^{MOM}}$):** A 4-element vector computed using the normalized variance and mean of block skewness and kurtosis values. This statistical summary provides robust descriptors of shape distribution changes caused by filtering.

4. **Same Intensity Block (SIB) Count ($\mathbf{f^{SIB}}$):** A single scalar indicating how many blocks consist of identical pixel values. Median filtering typically increases these uniform blocks, especially in flat or textured areas.

Putting it together, the full feature vector **SK** is defined as:

$$\mathbf{SK} = \left[h^{SBH},\ h^{KBH},\ f^{MOM},\ f^{SIB}\right] \tag{1}$$

This yields a total of 19 features: 9 from SBH, 4 from KBH, 4 from f^{MOM}, and 1 from f^{SIB}. In code, this feature extraction can be modularized into four steps—each returning part of the final feature vector—and then concatenated into one array for model training or analysis.

3 Our Implementation

Our Python code constructs feature vectors from a grayscale image and its median-filtered version to capture texture and structural patterns. The feature extraction proceeds as follows:

1. **Image Preprocessing:** The input image is first converted to grayscale (if not already). A median filter is applied to create a smoothed version of the image. (Window Size: 3x3 5x5)

2. **Sliding Window Blocks:** Both the original and median-filtered images are scanned using a 3×3 sliding window with a stride of 1. This creates overlapping local patches throughout the image.

3. **Feature Extraction per Block:** For each 3×3 block:

   - Flatten the block to a 1D vector of 9 pixel values.
   - Compute the histogram of pixel intensities with 256 bins (for values 0 to 255).
   - Normalize the histogram to form a probability distribution.
   - Extract the probabilities at four specific intensity values: 0, 64, 128, and 192.
   - Calculate skewness and kurtosis of the block's pixel distribution.

4. **Statistical Aggregation:** For all blocks in the image:

   - Compute the mean skewness and mean kurtosis.
   - Compute the mean histogram values at 0, 64, 128, and 192.
   - Repeat the above for the variance and entropy of each block.
   - Compute the average local contrast across all blocks.

5. **Feature Vector Formation:** A 19-dimensional feature vector is formed for each image (original and filtered). Each vector includes:

   - 6 features from skewness (mean, histogram stats),
   - 6 from variance,
   - 6 from entropy,
   - 1 from contrast.

   Concatenating the vectors from both images gives the final 38-dimensional feature vector.

4 Experiment Setup

4.1 Dataset and Attacks

To evaluate the performance of the proposed median filtering forensic detector, we form the following training-testing pairs:

1. All images in Ψ_{UCID} are median filtered using window sizes 3×3 and 5×5, resulting in Ψ_{UCID}^{MF3} and Ψ_{UCID}^{MF5} respectively. To analyze detection performance against other attacks, the following databases are created:

   - Ψ_{UCID}^{AVG}: Average filtering with a 3×3 window.
   - Ψ_{UCID}^{GAU}: Gaussian filtering with $\sigma = 0.5$.
   - Ψ_{UCID}^{RES}: Upscaling by a factor of 1.5 using bicubic interpolation (cropped back to original size).
   - Ψ_{UCID}^{J90}: JPEG compression with quality factor $Q = 90$.

   The above operations are also applied to lower-resolution versions of the original images at sizes 256×256 (cropped from the center) along with the original size 512×384 .

2. To validate detection of median filtering from other attacks, the training is performed using the union set $\{\Psi_{UCID}^{MF3}(I), \Psi_{UCID}^{MF5}(I), \Psi_{UCID}(I), \Psi_{UCID}^{AVG}(I), \Psi_{UCID}^{GAU}(I), \Psi_{UCID}^{RES}(I), \Psi_{UCID}^{J90}(I)\}$, and testing is performed on the complementary set.

3. Ψ_{UCID}^{MF35} is created by randomly selecting 50% of images from both Ψ_{UCID}^{MF3} and Ψ_{UCID}^{MF5}. Ψ_{UCID}^{ALL} is formed by randomly selecting 25% of images from each of Ψ_{UCID}^{AVG}, Ψ_{UCID}^{GAU}, Ψ_{UCID}^{RES}, and Ψ_{UCID}. The training set consists of $\{\Psi_{UCID}^{MF35}(I), \Psi_{UCID}^{ALL}(I)\}$ and testing is done on $\{\Psi_{UCID}^{MF35}(I'), \Psi_{UCID}^{ALL}(I')\}$. This pair is constructed to check robustness of the detector against other image processing operations.

4.2 Experimental Procedure

All experiments are framed as two-class classification problems using supervised learning. Since many image processing operations (AVG, GAU, RES, J90) introduce new SIB blocks into the image, features calculated directly from original and processed images may not be comparable. To address this, SIB blocks from both original and median-filtered images are excluded before computing skewness and kurtosis histograms. This allows for relative histogram comparison and better classification training. The training set comprises 75% of the dataset, while testing is conducted on the remaining 25%.

4.3 Support Vector Machine (SVM) Classification

For classification, a Support Vector Machine (SVM) with a polynomial kernel of degree 3 was employed. The training set consists of features extracted from original images along with various filtered versions: average filtering, Gaussian filtering, JPEG compression, rescaling, and median filtering (with 3×3 and 5×5 windows). The classifier is trained using:

- A cubic kernel function (degree = 3),
- Regularization parameter $C = 1.0$,
- Probability estimates enabled for AUC calculation.

To evaluate model performance, pairwise comparisons are made between median-filtered images and those processed by other operations. For each pair (e.g., `mf3` vs `avg`), the corresponding test sets are extracted using previously saved indices. Each test instance is labeled: 1 for the first type (e.g., `mf3`), and 0 for the second (e.g., `avg`).

The following metrics are computed for each pair:

- **Probability of Error (PoE)**: Computed as $1 -$ Accuracy,
- **Area Under the Curve (AUC)**: Represents the classifier's ability to distinguish between the two image types.

These values are tabulated to evaluate how reliably the classifier can distinguish median filtering from other operations.

5 Experimental Results

5.1 Median Filtering Against Other Manipulations for Varying Resolution Uncompressed Images

Table 1: Performance comparison for uncompressed images

Implementation	Resolution	Metric	MF3		MF3		MF5		MF35		
			AVG	GAU	JPEG	RES	AVG	GAU	JPEG	RES	ALL
Research paper	512x384	Pe	0.009	0.0041	0.0078	0.0011	0.0067	0.0019	0.0034	0.0015	0.0075
		AUC	0.9951	0.9986	0.9958	1	0.9981	0.9999	0.9971	1	0.9973
Grp-15	512x384	Pe	0.0075	0.0075	0.009	0.009	0.006	0.006	0.0075	0.0075	0.0075
		AUC	0.9952	0.9992	0.995	0.9958	0.9949	0.9977	0.9941	0.9953	0.9942
Research paper	256x256	Pe	0.006	0.006	0.0075	0.0037	0.0022	0.003	0.0056	0.0011	0.0101
		AUC	0.999	0.9966	0.9957	0.9976	0.9987	0.9991	0.9984	0.9986	0.9954
Grp-15	256x256	Pe	0.4343	0.4388	0.4239	0.4493	0.4358	0.4403	0.4254	0.4507	0.4222
		AUC	0.7434	0.7149	0.7481	0.6095	0.8041	0.7812	0.8179	0.6834	0.7836

5.2 Post-JPEG Compressed Results

Table 2: Post-JPEG compressed median filtered vs original images

Implementation	Resolution	ORI vs MF3+Q		ORI vs MF5+Q	
		Pe	AUC	Pe	AUC
Research paper	512x384	0.0135	0.9932	0.006	0.9956
Grp-15	512x384	0.2791	0.9858	0.0657	0.9886
Research paper	256x256	0.0594	0.9637	0.0299	0.9825
Grp-15	256x256	0.5418	0.4496	0.5269	0.42

Table 3: Median filtered images post-JPEG against other manipulations

Implementation	Resolution	Metric	MF3+Q		MF5+Q		MF35+Q		
			AVG	GAU	RES	AVG	GAU	RES	ALL
Research paper	512x384	Pe	0.0105	0.0138	0.0123	0.0071	0.0093	0.0056	0.0179
		AUC	0.994	0.9901	0.9909	0.9958	0.9934	0.9961	0.9882
Grp-15	512x384	Pe	0.2776	0.2776	0.2791	0.0642	0.0642	0.0657	0.0479
		AUC	0.9579	0.9908	0.9883	0.9849	0.993	0.9898	0.9891
Research paper	256x256	Pe	0.0213	0.0389	0.0426	0.0265	0.0176	0.0217	0.0407
		AUC	0.9884	0.9791	0.975	0.98	0.9875	0.9852	0.9754
Grp-15	256x256	Pe	0.5015	0.506	0.5164	0.4866	0.491	0.5015	0.482
		AUC	0.6482	0.6157	0.492	0.6078	0.5775	0.4598	0.5922

Table 4: Post-JPEG compressed median filtered vs post-JPEG original images

Implementation	Resolution	ORI+Q vs MF3+Q		ORI+Q vs MF5+Q	
		Pe	AUC	Pe	AUC
Research paper	512x384	0.0086	0.9962	0.003	0.9994
Grp-15	512x384	0.2791	0.9647	0.0657	0.9855
Research paper	256x256	0.0116	0.994	0.0037	0.9981
Grp-15	256x256	0.491	0.6504	0.4761	0.6082

- For uncompressed images (Table 1), our model does a solid job, especially with higher-resolution images like those at 512×384. In several comparisons, such as MF3 vs JPEG and MF5 vs RES, we see high AUC

values and low Pe scores. This shows that our system is generally able to pick up on the traces median filtering leaves behind and tell it apart from other kinds of manipulations, at least under clean, high-quality image conditions.

- However, performance starts to dip a bit at lower resolutions like 256×256. That actually makes sense—our features are statistical in nature and depend heavily on local texture patterns. When image resolution drops, those fine-grained textures start to blur together, and the differences between manipulations become harder to detect. So the drop isn't surprising; it's a limitation that naturally comes from working with smaller image patches.
- JPEG results in (Tables 2, 3, and 4) tend to mask or even distort the small statistical footprints left behind by earlier processing like median filtering. Since our features weren't specifically crafted to handle compressed images, the classifier struggles more here—Pe goes up, and AUC comes down a bit. This is particularly noticeable when trying to distinguish between compressed median-filtered images and other compressed manipulations, which all start to look somewhat similar once JPEG artifacts kick in.

Now, when comparing to the results in the research paper, we do notice that their system performs better in several scenarios, especially under JPEG compression. A few reasons can help explain that.

1. Our training was done only on the UCID dataset, while the research paper uses a much broader training setup across multiple datasets and resolutions, which likely gives their model a stronger foundation. They used a much broader training setup that pulls from multiple datasets like BOSSBase, BOWS2, RAISE, and more. They also experimented with different image resolutions and post-processing scenarios, which makes their model more robust in a wider range of conditions.
2. There's the difference in implementation platforms—our code was built in Python, while theirs is in MATLAB. Even when using the same logic, different platforms can handle image processing slightly differently—things like filtering, padding, and rounding behave subtly in distinct ways, which can affect the exact features extracted. These small technical differences add up.

So overall, while our implementation doesn't outperform the research work, it shows sound and interpretable behavior. With broader training data and compression-aware feature tuning, there's definitely potential for improvement.