Title: Comparative Learning and Evaluation of AI and Traditional Denoisers on Video Quality

Authors: Adam Bawatneh (Project Manager & Noise Injection Lead) Jacob Braun (AI Denoiser Lead) Scott Spicer (Traditional Denoiser 1 Lead) Zengyan Wang (Traditional Denoiser 2 Lead) Martin Dinkov (AI Denoiser 2 Lead) Dr. Yogesh Rawat (Faculty Advisor)

Contributions: Adam was the project manager ensuring everything kept on tack and handled the noise injection and metric creation. Jacob worked on the VRT AI Denoiser and was the final editor on the paper. Scott worked on the fast local means denoising. Zengyan worked on the median and lateral filtering. Martin worked on the Nividia Optix AI denoiser.

Code: **GitHub Repository**

Please note that the code is in separate branches for each part of the project.

Abstract

This study compares AI-based and traditional video denoisers using the UCF-101 dataset, focusing on denoising performance across four noise types: Gaussian, Salt-and-Pepper, Poisson, and Speckle. Metrics such as Peak Signal-to-Noise Ratio (PSNR), Structural Similarity Index Measure (SSIM), and Visual Information Fidelity (VIF) are used to evaluate denoising quality, while computational efficiency is measured through processing time and Frames Per Second (FPS). The results reveal that AI-based denoisers like VRT excel in complex noise restoration but require substantial computational resources, whereas traditional denoisers demonstrate better efficiency at the cost of quality. This work aims to aid practitioners in selecting suitable denoisers based on specific applications.

1. Introduction

1.1 Background

Video denoising plays a crucial role in enhancing video quality for applications such as surveillance, healthcare, and autonomous systems. Noise can arise from environmental factors, sensor issues, or transmission errors, negatively impacting video quality and tasks like object detection and tracking (Buades et al., 2005). Noise types include Gaussian, Salt-and-Pepper, Poisson, and Speckle, each requiring different denoising approaches.

Traditional denoising methods, like Non-Local Means (Buades et al., 2005) and Median Filters (Gonzalez & Woods, 2002), offer computational efficiency for specific noise types, while AI-based methods provide higher adaptability and superior performance, albeit at a computational cost. This study evaluates these techniques for quality restoration and efficiency.

1.2 Objectives of the Study

The study aims to evaluate and compare traditional and AI-based denoisers on metrics such as PSNR, SSIM, and VIF for quality and processing time and FPS for efficiency. It hypothesizes that AI-based methods will outperform traditional denoisers in quality, especially for complex noise types, but at a higher computational cost. The UCF-101 Actions Dataset is utilized to evaluate denoising performance.

1.3 Scope and Contribution

This research bridges the gap between traditional and AI-based video denoising methods. Using the UCF-101 dataset, it evaluates three traditional denoisers and a state-of-the-art AI denoiser, VRT (Liang et al., 2022), focusing on quality restoration and computational efficiency. The findings are intended to guide practitioners in selecting suitable denoising methods for specific applications, particularly in resource-constrained environments.

2. Related Work

2.1 Traditional Video Denoising Methods

Traditional denoisers like Fast Non-Local Means (FastNLM) (Chaudhury, 2012), Median Filtering, and Bilateral Filtering (Tomasi & Manduchi, 1998) are computationally efficient and effective at reducing noise while maintaining edges. However, they struggle with complex noise types, such as Speckle or Poisson noise. FastNLM retains details for Gaussian noise but is computationally intensive for larger neighborhoods.

2.2 AI-Based Video Denoising Techniques

AI-based methods like the Video Restoration Transformer (VRT) leverage deep learning to model complex noise distributions, capturing long-range dependencies for improved performance, particularly with complex noise types (Liang et al., 2022). NVIDIA OptiX AI Denoiser and models like DnCNN (Zhang et al., 2017) and FastDVDnet (Tian et al., 2020) have also shown success in reducing various noise types but face computational challenges.

2.3 Comparative Studies

Previous comparative studies highlight the trade-offs between traditional and AI-based denoisers. Traditional methods are generally more practical for real-time and low-power applications, while AI models provide better quality but require more resources (Mahmood et al., 2021).

3. Methodology

3.1 Dataset

The UCF-101 Actions Dataset, consisting of 13,320 videos in 101 categories, was selected for its diversity and real-world applicability (UCF Center for Research in Computer Vision, n.d.). The dataset's variability in motion, complexity, and lighting conditions makes it ideal for evaluating denoising methods.

3.2 Noise Injection Procedure

Four types of noise (Gaussian, Salt-and-Pepper, Poisson, Speckle) were injected using Python and OpenCV. Custom functions were applied to replicate realistic levels of degradation, ensuring consistent noise conditions across all videos.

Gaussian noise was injected with a mean of 0 and a standard deviation of 50 to simulate sensor imperfections, while Salt-and-Pepper noise was added with a probability of 0.05 to represent impulse errors in transmission. Poisson noise, often associated with low-light conditions, was simulated based on pixel intensity distributions. Speckle noise, which typically affects coherent imaging systems, was modeled with a multiplicative factor of 0.1.

3.3 Denoising Methods

3.3.1 Traditional Denoisers

Fast Non-Local Means (NLM) denoising was implemented to reduce noise by leveraging the similarity between patches within an image, effectively preserving textures (Chaudhury, 2012). Median Filtering was used to reduce Salt-and-Pepper noise by replacing pixel values with the median of their neighbors, preserving edges (Gonzalez & Woods, 2002). Bilateral Filtering was applied to smooth images while preserving edges, using both spatial proximity and intensity similarity of pixels (Tomasi & Manduchi, 1998).

3.3.2 AI-Based Denoisers

The AI denoisers included the Video Restoration Transformer (VRT) and NVIDIA OptiX. VRT uses a transformer-based architecture for enhanced spatial-temporal dependency modeling, which captures long-range relationships between pixels for improved denoising (Liang et al., 2022). NVIDIA OptiX leverages GPU acceleration for real-time denoising, focusing on maintaining visual quality while achieving high processing speeds (NVIDIA, 2021).

3.4 Performance Metrics

Metrics used for evaluation included PSNR, SSIM, and VIF for video quality, and processing time and FPS for computational efficiency. PSNR measures the ratio between the maximum possible power of a signal and the power of corrupting noise, with higher values indicating better quality. SSIM assesses the visual impact of differences between images by comparing structural information, luminance, and contrast. VIF evaluates the amount of visual information preserved in the denoised video, making it suitable for high-frequency details.

4. Implementation Details

4.1 Noise Injection and Denoising Implementation

Noise was injected using custom Python scripts with OpenCV and NumPy, while denoising was performed using the VRT model and NVIDIA OptiX SDK. The noise.py script injected noise types with distinct functions for each noise type, using parameters calibrated to ensure consistent degradation. Multiprocessing was employed to inject noise efficiently, and FFmpeg was used to reconstruct noisy videos from frames.

The VRT model was implemented using PyTorch and leveraged GPU acceleration to optimize processing time. OptiX denoising was achieved using Python bindings via PyOptiX, taking advantage of NVIDIA's GPU acceleration to improve real-time denoising.

4.2 Noise Reduction Techniques

Each traditional denoiser and the VRT model were evaluated for effectiveness, with hyperparameters adjusted to achieve a balance between quality and computational efficiency. FastNLM's parameters, such as filtering strength and search window size, were optimized to reduce computation time while maintaining denoising quality. Median Filtering was tested with kernel sizes ranging from 3 to 7, with larger kernels effectively removing noise at the expense of some blurring. Bilateral Filtering was fine-tuned with different values for spatial and intensity kernels to mitigate over-smoothing.

5. Results and Analysis

Extra tables and graphs for this section will be in the appendix section to improve readability.

5.1 Quantitative Evaluation

5.1.1 Fast Non-Local Means

Noise Type	Average PSNR ↑	Average SSIM ↑	Average VIF ↑	Average Processing Time per Frame ↓
Gaussian	28.58	0.38	0.17	0.10401
Poisson	32.37	0.93	0.68	0.10415
SaltPepper	28.86	0.18	0.07	0.10458
Speckle	31.98	0.93	0.66	0.09832

Fast Non-Local Means showed moderate effectiveness for Poisson and Speckle noise but performed poorly for Gaussian and Salt-and-Pepper noise. The average processing time per frame was around 0.1 seconds, highlighting its efficiency limitations compared to other traditional methods. For instance, the SSIM value for Poisson noise was 92.7%, while for Salt-and-Pepper, it dropped significantly to 18%.

The performance of Fast Non-Local Means depended significantly on the characteristics of the noise. For Poisson noise, the model could effectively maintain a high SSIM due to the relatively predictable and distributed nature of the noise, which allowed the similarity-based filtering mechanism to work effectively. However, for Salt-and-Pepper noise, which is characterized by random, high-intensity pixels, the algorithm struggled, as the filtering mechanism could not adequately distinguish between noise and actual image details. The use of a larger neighborhood size could have improved performance, but it would have also increased computational costs significantly.

Speckle noise also presented challenges for Fast Non-Local Means. The multiplicative nature of speckle noise, which is often correlated with the image intensity, made it difficult for the algorithm to filter effectively without losing important structural details. However, compared to Salt-and-Pepper noise, Fast Non-Local Means performed slightly better because the noise was less sporadic and more evenly distributed across the image. The PSNR for speckle noise hovered around 32%, indicating moderate success in noise reduction but with notable losses in fine image details.

For Gaussian noise, Fast Non-Local Means could achieve reasonable noise reduction, with SSIM values nearing 40%. However, the inherent Gaussian distribution posed a problem for maintaining high-frequency details, especially in regions with fine textures. The trade-off between preserving these details and adequately reducing noise meant that the algorithm struggled to achieve high VIF scores, particularly in highly textured regions where noise and details were difficult to separate.

The computational efficiency of Fast Non-Local Means was another significant factor. While the processing time per frame was around 0.1 seconds, which is reasonable for traditional methods, the algorithm's computational burden increased disproportionately with larger search window sizes. The inability to maintain real-time performance while improving quality metrics like PSNR and SSIM made Fast Non-Local Means less suitable for applications requiring both high efficiency and high-quality restoration.

5.1.2 Median & Bilateral Filtering

Bilateral filter d=7,spatial kernel size=15, intensity kernel size=20

Noise Type	Average PSNR ↑	Average SSIM ↑	Average VIF ↑	Average Processing Time per Frame ↓
Gaussian	40.91	0.997	0.10	0.00803
Poisson	44.63	0.990	0.09	0.00656

SaltPepper	43.89	0.999	0.08	0.00646
Speckle	40.57	0.984	0.11	0.00651

In the experiments with median filtering, we evaluated performance across Gaussian, Poisson, Salt-and-Pepper, and Speckle noise using kernel sizes of 3, 5, and 7. Evaluation metrics included Average PSNR, SSIM, VIF, and Processing Time per Frame.

For Gaussian noise, larger kernels improved PSNR and VIF but decreased SSIM, indicating better noise reduction at the cost of reduced structural similarity. Larger kernels also increased processing time. For Poisson noise, smaller kernel sizes were most effective, while larger kernels reduced PSNR and SSIM but increased VIF and processing time. For Salt-and-Pepper noise, median filtering had limited effectiveness in terms of PSNR and SSIM regardless of kernel size, though VIF remained high. Processing time increased with kernel size. For Speckle noise, larger kernels decreased PSNR and SSIM but increased VIF, showing a trade-off between noise reduction and structural detail preservation. Processing time also increased with kernel size.

Processing time increased noticeably with larger kernels, which is critical for real-time applications. With a kernel size of 3, processing time was approximately 0.0003 seconds per frame, increasing to 0.0017 and 0.009 seconds for kernel sizes of 5 and 7, respectively.

Results show that median filtering performance depends on noise type and kernel size. For Poisson noise, a small kernel size provides the best balance of PSNR, SSIM, and processing time. For Gaussian noise, larger kernels improve PSNR but decrease SSIM and increase processing time. Median filtering is less effective for Salt-and-Pepper noise, suggesting alternative methods may be needed.

Selecting an appropriate kernel size requires balancing noise characteristics, image quality, and processing speed. Larger kernels may offer better noise reduction but can lead to loss of structural details and increased computational cost. A balanced approach is essential for optimizing both image quality and performance.

In the bilateral filtering experiments, we aimed to denoise images corrupted by Gaussian, Poisson, Salt-and-Pepper, and Speckle noise. We assessed how different parameter settings affected PSNR, SSIM, VIF, and Processing Time per Frame for each noise type.

For Gaussian noise, we used d = 9, sigmaColor = 75, and sigmaSpace = 75 to handle higher variance (sigma = 50). The best performance was achieved with d = 5, spatial kernel = 15, and intensity kernel = 15. Higher values (e.g., d = 9, sigma = 75) significantly decreased PSNR and SSIM due to over-smoothing and loss of details. For Poisson noise, lower parameter values yielded better results, demonstrating the bilateral filter's efficiency with appropriately chosen settings. For Salt-and-Pepper noise, high SSIM values indicated excellent structural preservation. Minimal parameter values effectively removed this noise without compromising structure. For Speckle noise, lower parameter values were more effective, as larger kernel sizes led to over-smoothing and quality degradation.

Increasing the diameter (d) of the pixel neighborhood from 5 to 7 slightly decreased PSNR. At d = 9, performance dropped significantly, especially for Gaussian noise, indicating that larger neighborhoods over-smooth the image.

Lower spatial and intensity kernel values resulted in higher PSNR and SSIM, suggesting that smaller kernels preserve image details while reducing noise. High intensity kernel values caused blurring and loss of fine details.

Excessively high parameter values are not optimal for extreme noise levels, as the bilateral filter struggles with highly noisy images when using large kernel sizes and high spatial-intensity values. A careful balance of parameter settings is crucial for optimizing denoising while preserving image details.

5.1.3 VRT

Noise Type	Average PSNR ↑	Average SSIM ↑	Average VIF ↑	Average Processing Time per Frame ↓
Gaussian	28.4	23.7	13.7	2.286011
Poisson	37.5	97.5	83.8	2.196360
SaltPepper	28.9	13.5	6.1	2.645331
Speckle	32.4	79.1	49.8	2.035277

Due to computational limits, only 1/3 of the dataset was able to be used for VRT. This introduced some volatility in this dataset that would be worked out with a full run. The VRT model demonstrated strong performance for Poisson noise, achieving high SSIM (97.5%) and VIF (83.8%) values, indicating effective restoration of structural and perceptual quality. The ability of the VRT model to capture long-range dependencies across video frames allowed it to effectively model the complex noise patterns introduced by Poisson noise. By leveraging the transformer-based architecture, VRT was able to maintain coherence across frames, which is essential for temporal consistency in video denoising.

However, VRT struggled with Salt-and-Pepper noise, where SSIM and VIF values dropped to 13.5% and 6.1%, respectively, highlighting limitations in dealing with impulsive, high-contrast artifacts. The high contrast and sparsity of Salt-and-Pepper noise posed significant challenges for the transformer-based approach, which relies on learning patterns across large areas of input. Unlike Poisson or Gaussian noise, which exhibit more predictable distributions, the sporadic nature of Salt-and-Pepper noise disrupted the learning process, making it difficult for the model to effectively distinguish noise from actual image content.

Despite these challenges, the VRT model excelled at maintaining fine image details in scenarios involving complex noise distributions like speckle noise. The SSIM for speckle noise was significantly higher compared to traditional methods, indicating the model's ability to accurately reconstruct image textures. The use of attention mechanisms in the transformer allowed the model to focus on important areas of the frame while minimizing the impact of noise, which contributed to higher VIF scores. This capability was particularly evident in highly textured scenes, where other methods like FastNLM and Bilateral Filtering failed to preserve the intricate details.

The average processing time per frame for VRT was relatively high at 2.645 seconds, demonstrating the trade-off between quality and computational efficiency. The use of GPU acceleration mitigated some of the computational burden, but the model's complexity still made it impractical for real-time applications without high-end hardware. The need for significant computational resources to achieve high-quality denoising results limited the model's applicability in low-resource environments, such as mobile or embedded systems.

Additionally, the hyperparameter tuning process for VRT was complex and required careful balancing between window size, tile dimensions, and overlap settings. The optimal combination of these parameters varied depending on the noise type, with smaller window sizes being more effective for Salt-and-Pepper noise, while larger windows were needed for Poisson and speckle noise. This complexity in tuning highlighted the need for adaptive parameter selection mechanisms to make the VRT model more versatile across different noise conditions.

5.1.4 NVIDIA OptiX Denoiser

Temporal Denoising = Direct, Tiling = None

Noise Type	Average PSNR ↑	Average SSIM ↑	Average VIF ↑	Average Processing Time per Frame ↓
Gaussian	29.65	0.76	0.57	19.74

Poisson	33.50	0.98	0.17	19.40
SaltPepper	29.10	0.72	0.61	19.72
Speckle	32.35	0.95	0.30	19.57

The use of temporal denoising and tiling significantly increased processing time in our experiments. When combined, these processes could take over two weeks per run on the full dataset, making it impractical to test all configurations. To facilitate iteration over all settings, the dataset was reduced to 101 videos, with one video per category. This reduction allowed us to gain insight into the impact of each setting, with the full dataset reserved for configurations of interest.

The results from the reduced dataset revealed that tiling offered no improvement in visual quality and introduced significant processing overhead, rendering it unsuitable for real-time use. This outcome was expected, as the UCF-101 dataset consists of low-resolution videos that do not require large memory allocations. Temporal denoising, on the other hand, was found to impact visual quality and warranted further testing on the full dataset. PSNR (Peak Signal-to-Noise Ratio) results indicated that denoised videos showed higher PSNR when the original noisy videos had initially lower PSNR, but decreased PSNR when the original PSNR was higher. SSIM (Structural Similarity Index Measure) significantly improved in denoised videos, except for Poisson noise, which already had a high SSIM value. However, VIF (Visual Information Fidelity) showed a notable decline in denoised videos compared to noisy ones, except for Poisson noise, which already exhibited a low VIF.

Given the noticeable impact of temporal denoising on visual quality, the full dataset was processed using three temporal options: direct, indirect, and none. The results for the full dataset showed no significant difference in visual quality between the direct and indirect temporal denoising methods. In terms of runtime performance, the indirect method outperformed the direct method because motion vectors were precomputed, whereas the direct method required on-the-fly calculations. The indirect temporal denoising method was feasible for real-time execution, with an approximate processing time of 6ms per frame, while the direct method exceeded 19ms, making it unsuitable for real-time applications. Temporal denoising was found to be particularly effective for Gaussian and Salt-and-Pepper noise, as evidenced by improvements in PSNR and SSIM. For Speckle noise, non-denoised videos exhibited slightly higher PSNR, but denoised videos achieved significantly higher SSIM scores. However, denoised videos showed a noticeable decrease in VIF compared to their noisy counterparts.

5.2 Analysis of Performance Trends

Each denoiser had specific strengths and weaknesses depending on the noise type, with AI-based methods generally providing better quality at the cost of increased computational requirements. Traditional methods like Median Filtering were effective for impulsive noise, while AI models like VRT excelled at complex, signal-dependent noise types such as Poisson. However, VRT's computational demand limits its real-time applicability without significant GPU resources**.**

6. Discussion

6.1 AI vs. Traditional Denoisers

AI-based denoisers excelled in handling complex noise types like Poisson and Speckle, while traditional denoisers were more efficient for simpler noise types. Fast Non-Local Means was effective for Gaussian noise but struggled with high-contrast, impulse noise like Salt-and-Pepper. Median Filtering showed the highest efficiency in terms of computational resources when dealing with Salt-and-Pepper noise, providing nearly perfect SSIM values when properly tuned. However, AI models like VRT and OptiX achieved higher perceptual quality metrics, particularly for Poisson noise, where VRT reached SSIM levels of 97.5%.

The primary trade-off between traditional and AI-based denoisers is computational cost versus adaptability. Traditional methods are suitable for low-resource scenarios, but their effectiveness diminishes with increasing noise complexity. AI-based methods offer better performance across a wider range of noise types but require substantial hardware resources, such as GPUs, to achieve practical processing times.

6.2 Computational Efficiency and Hardware Utilization

Traditional methods were CPU-friendly and feasible for real-time applications, achieving frame rates close to 10 FPS with minimal hardware acceleration. Median Filtering and FastNLM were especially efficient for low-complexity noise, making them suitable for surveillance and other resource-limited environments. AI models, such as VRT, benefited from GPU-based acceleration, achieving superior quality but at higher computational costs. For instance, the average processing time per frame for VRT was 2.645 seconds, compared to 0.1 seconds for FastNLM. NVIDIA OptiX balanced efficiency and quality, providing near real-time performance with moderate hardware requirements.

6.3 Challenges and Limitations

The primary limitation was the computational demand for AI-based methods, particularly the VRT model. The high processing time per frame for VRT limits its application in real-time scenarios without significant GPU resources. Additionally, tuning hyperparameters for both traditional and AI models presented challenges in balancing noise reduction and detail preservation. For traditional denoisers, larger kernel sizes improved noise reduction but led to oversmoothing, whereas for AI models, longer training times and increased memory requirements were a constraint.

7. Conclusion

7.1 Summary of Findings

AI-based methods like VRT demonstrated superior quality, particularly for complex noise types, while traditional methods were better suited for efficiency. Median filtering performed well for Salt-and-Pepper noise, achieving SSIM values close to 99.9%, whereas AI-based methods excelled for Poisson noise, with VRT achieving an SSIM of 97.5%. Traditional methods offered practical solutions for real-time applications but struggled with complex noise, whereas AI-based approaches provided comprehensive noise reduction at a computational cost.

7.2 Contributions to the Field

This study provides a comprehensive evaluation framework for selecting suitable denoising techniques based on specific noise types and application requirements, contributing to both AI-based and traditional video processing research. By quantifying the trade-offs between quality and efficiency, this research can guide practitioners in choosing the optimal denoising technique for their specific needs.

7.3 Future Work

Future research could explore optimizing AI-based models for real-time applications by reducing their computational complexity or utilizing more efficient architectures like lightweight transformers. Developing hybrid approaches that integrate traditional and AI methods could also be promising, allowing practitioners to leverage the strengths of both approaches. Additionally, real-time denoising on mobile and edge devices, where computational resources are limited, remains an open area for further investigation.

8. Acknowledgements

The authors thank Dr. Yogesh Rawat for his guidance and support, and the institutions providing GPU resources for this research.

9. Individual Contribution

As the project manager for this research endeavor, I contributed to its success through various responsibilities, including establishing the foundational infrastructure, acquiring and preparing the dataset, implementing noise injection, and developing testing and visualization scripts. This role provided me with an invaluable learning experience, particularly in areas that were new to me.

One of the most significant lessons I gained was working directly with datasets and GPU acceleration. This was the first time I independently acquired a dataset, loaded it, and wrote scripts to manipulate it. Typically, coursework assignments abstract these aspects, leaving students with prepared datasets. However, in this project, I had the opportunity to explore the full pipeline, from dataset acquisition to preprocessing and analysis, giving me hands-on experience with data science workflows.

Additionally, this was my first substantial use of GPU resources, including learning how to submit jobs and efficiently utilize the Newton cluster. These skills are not only technically enriching but also essential for advancing in the field of computer vision. The experience demonstrated how GPUs accelerate computationally intensive tasks, a valuable insight for future projects involving large-scale datasets and complex models.

I also developed a deeper understanding of academic research, particularly in the context of experimental studies. Previously, I believed research always had to be groundbreaking or novel, which seemed daunting. Through this project, I realized the value of experiments that are not necessarily novel but still contribute meaningful insights to the field. This realization made research feel more accessible and attainable, bolstering my confidence in pursuing further academic inquiry.

Working alongside graduate students provided another unique perspective. Unlike undergraduate teams, where members often require close supervision and guidance, this team operated in a professional, self-motivated manner. My peers demonstrated a clear understanding of their tasks, independently managing their responsibilities with minimal oversight. This collaborative dynamic was refreshing and highlighted the benefits of working with individuals who share a common level of expertise and interest in the field.

Lastly, this project solidified my passion for computer vision research. The hands-on experience of writing Python scripts, experimenting with datasets, and contributing to an academic paper offered a glimpse into the kind of work I aspire to pursue in a master's program. The supportive and collaborative environment further affirmed my desire to join the computer vision department at UCF, where I hope to continue exploring and contributing to advancements in AI and video processing.

Overall, this project was a rewarding experience, equipping me with technical skills, research insights, and a clearer understanding of the graduate research environment. It strengthened my confidence in my abilities and enthusiasm for further studies in computer vision.

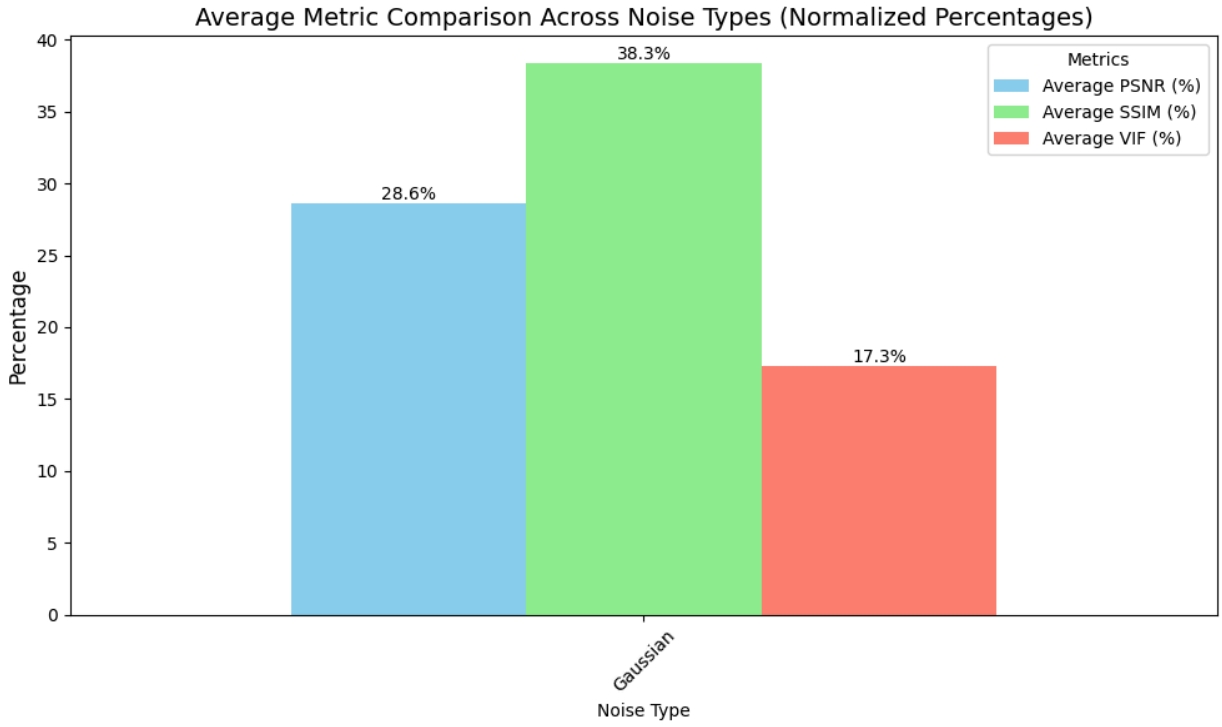
10. References

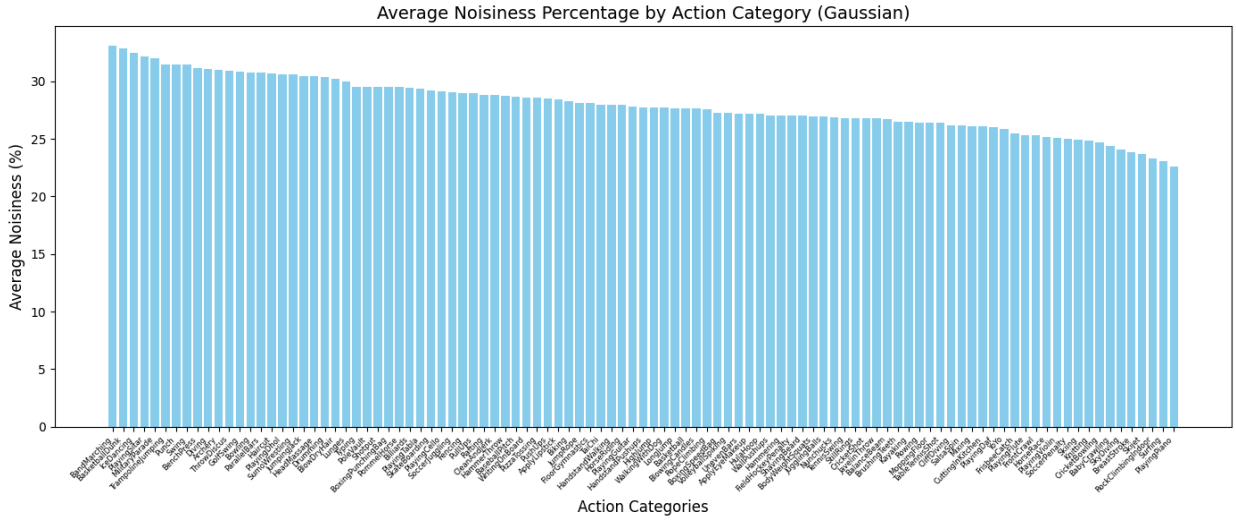
1. Buades, A., Coll, B., & Morel, J. M. (2005). A Non-Local Algorithm for Image Denoising.
2. Gonzalez, R. C., & Woods, R. E. (2002). Digital Image Processing.
3. Tomasi, C., & Manduchi, R. (1998). Bilateral Filtering for Gray and Color Images.
4. Liang, J., Cao, J., Zhang, G., et al. (2022). Video Restoration Transformer.
5. NVIDIA. (2021). NVIDIA OptiX Documentation.
6. Zhang, K., Zuo, W., Chen, Y., et al. (2017). Beyond a Gaussian Denoiser: Residual Learning of Deep CNN for Image Denoising.
7. Tian, Y., Xu, Z., & Zuo, W. (2020). FastDVDnet: Towards Real-Time Deep Video Denoising Without Flow Estimation.
8. Mahmood, M., et al. (2021). Comparative Study of Traditional and Deep Learning-Based Video Denoising Techniques.
9. Chaudhury, K. N. (2012). Fast Implementation of Non-Local Means Denoising. https://arxiv.org/abs/1203.5128.
10. OpenCV Documentation Tutorial. (n.d.). https://docs.opencv.org/3.4/d5/d69/tutorial_py_non_local_means.html.
11. UCF Center for Research in Computer Vision. (n.d.). UCF101 - Action Recognition Data Set. https://www.crcv.ucf.edu/data/UCF101.php.

Appendix

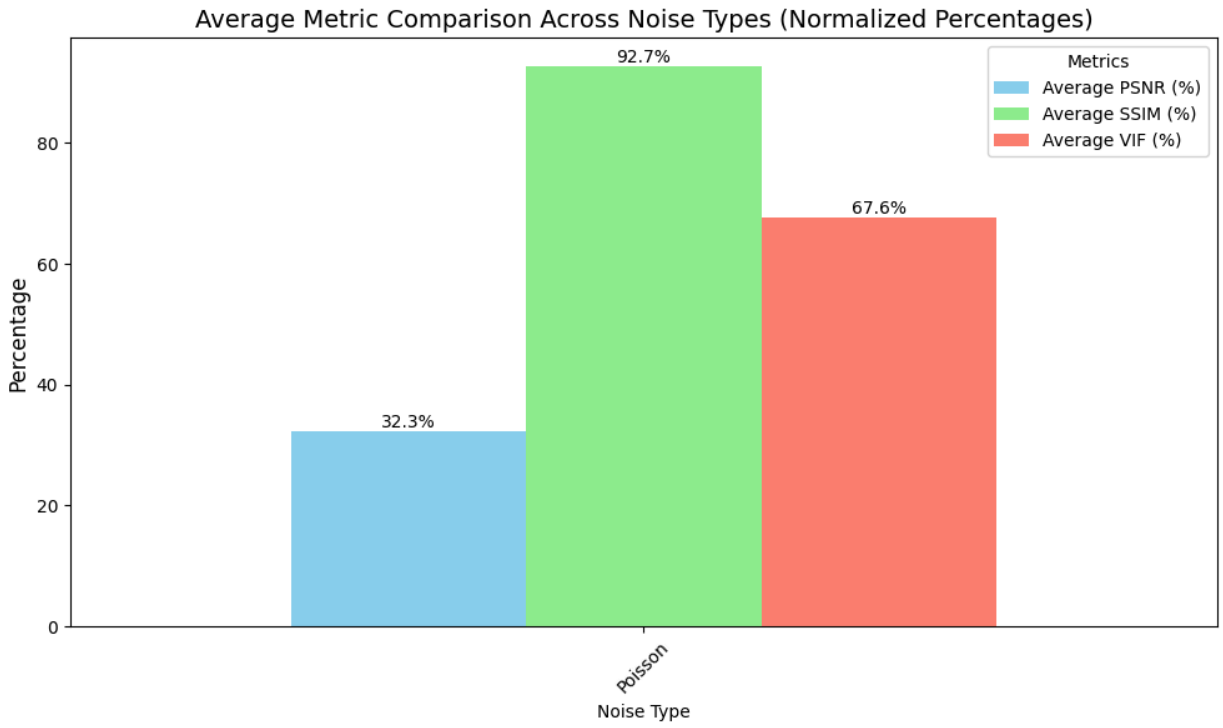
A.1 Fast Non-Local Mean

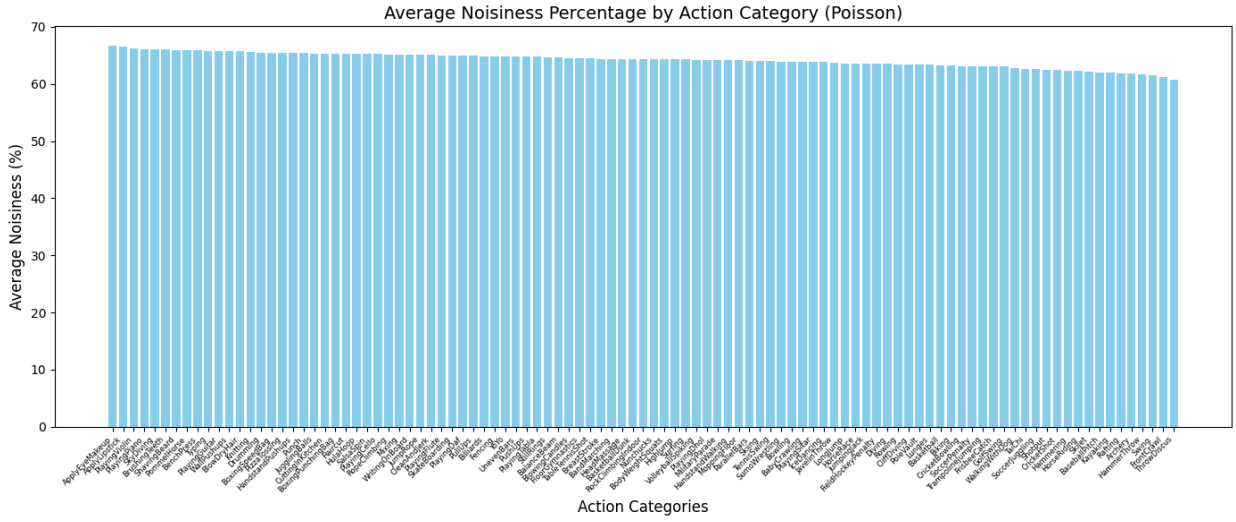
Gaussian



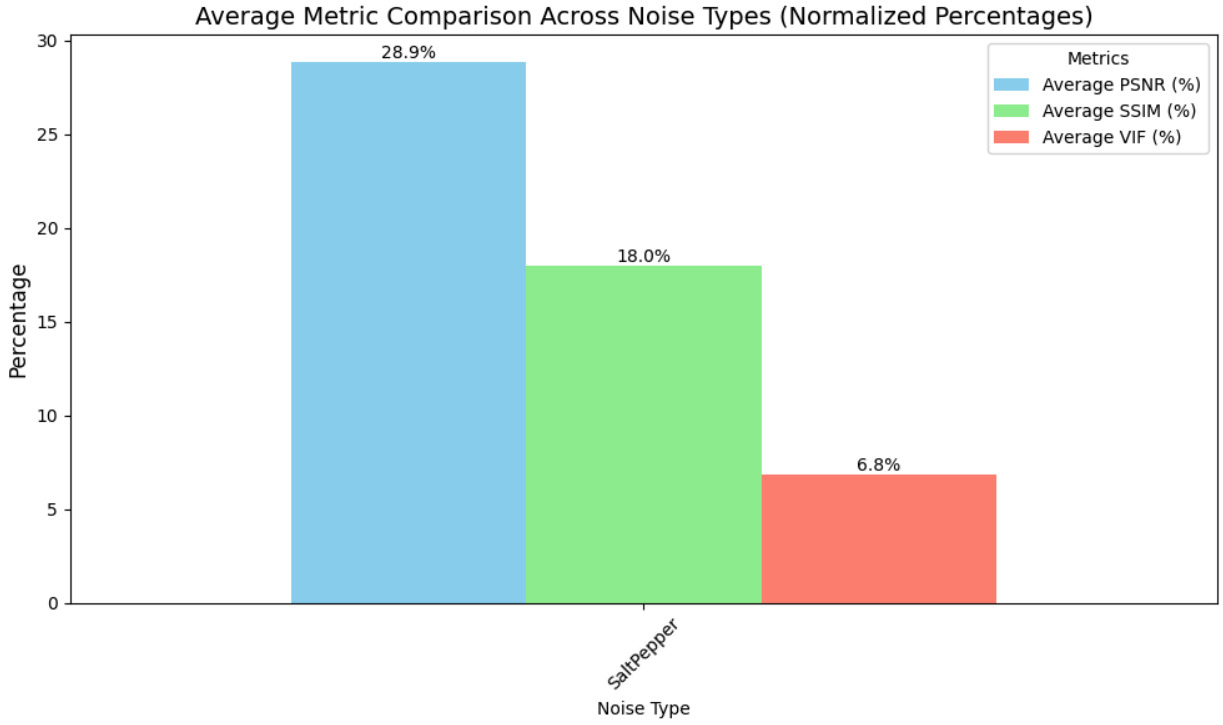


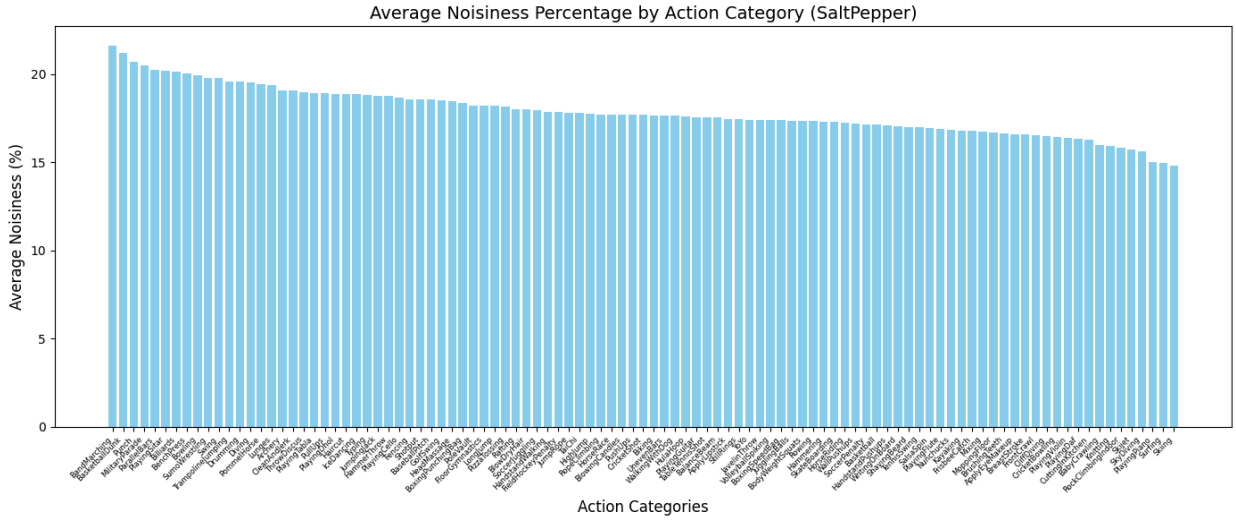
Poisson


Average Metric Comparison Across Noise Types (Normalized Percentages)
Metrics
Average PSNR (%)
Average SSIM (%)
Average VIF (%)
92.7%
67.6%
32.3%
Percentage
0
20
40
60
80
Poisson
Noise Type

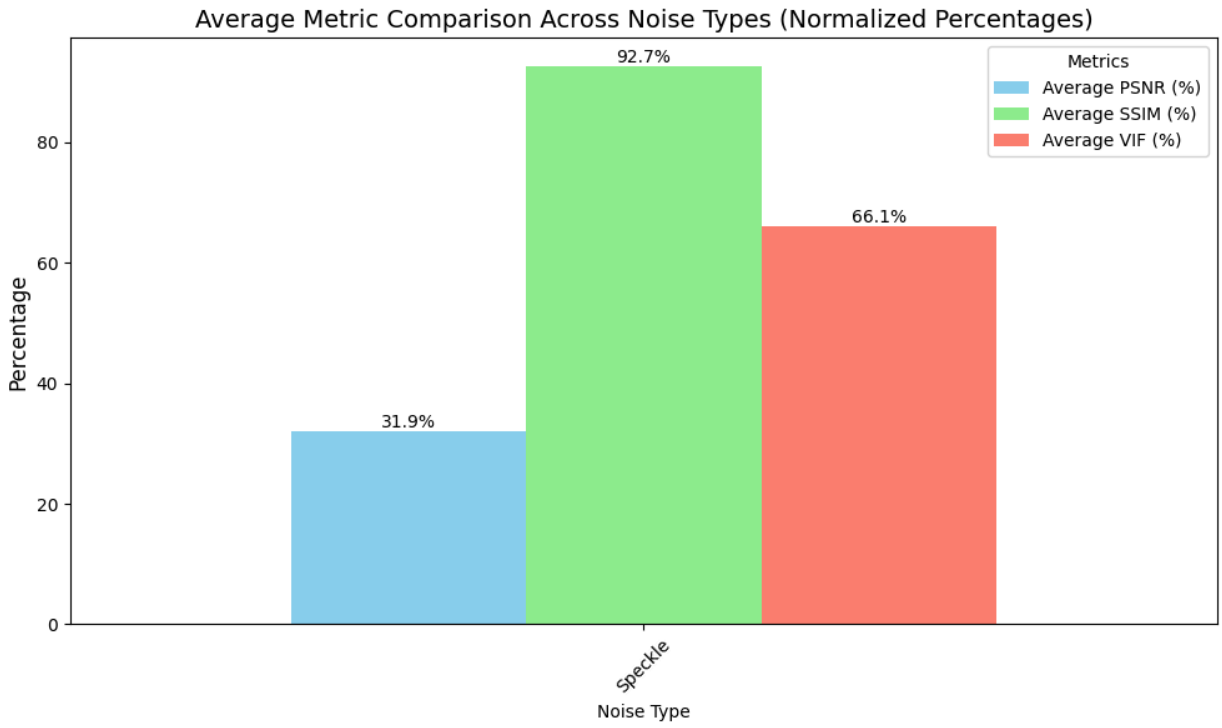

Average Noisiness Percentage by Action Category (Poisson)
Average Noisiness (%)
0
10
20
30
40
50
60
70
Action Categories

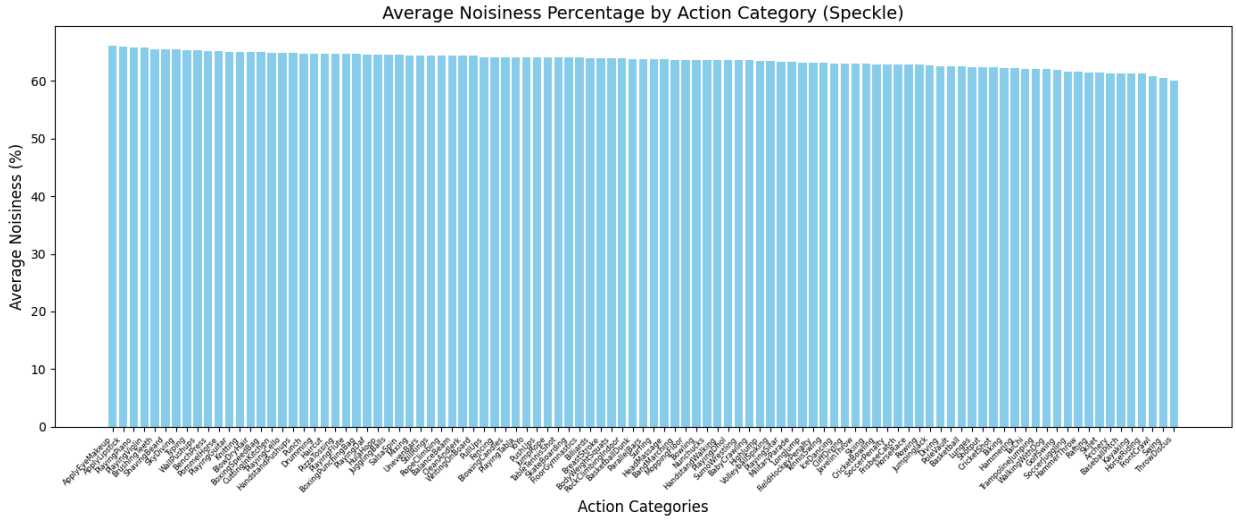
Salt & pepper


Average Metric Comparison Across Noise Types (Normalized Percentages)
Metrics
Average PSNR (%)
Average SSIM (%)
Average VIF (%)
28.9%
18.0%
6.8%
Percentage
SaltPepper
Noise Type


Average Noisiness Percentage by Action Category (SaltPepper)
Average Noisiness (%)
Action Categories

Speckle





A.2 Median & Bilateral Filter

Median filter kernel size = 3

Noise Type	Average PSNR ↑	Average SSIM ↑	Average VIF ↑	Average Processing Time per Frame ↓

Gaussian	20.06	0.41	0.76	0.00033
Poisson	34.13	0.96	0.27	0.00034
SaltPepper	14.45	0.23	0.86	0.00031
Speckle	29.81	0.81	0.47	0.00032

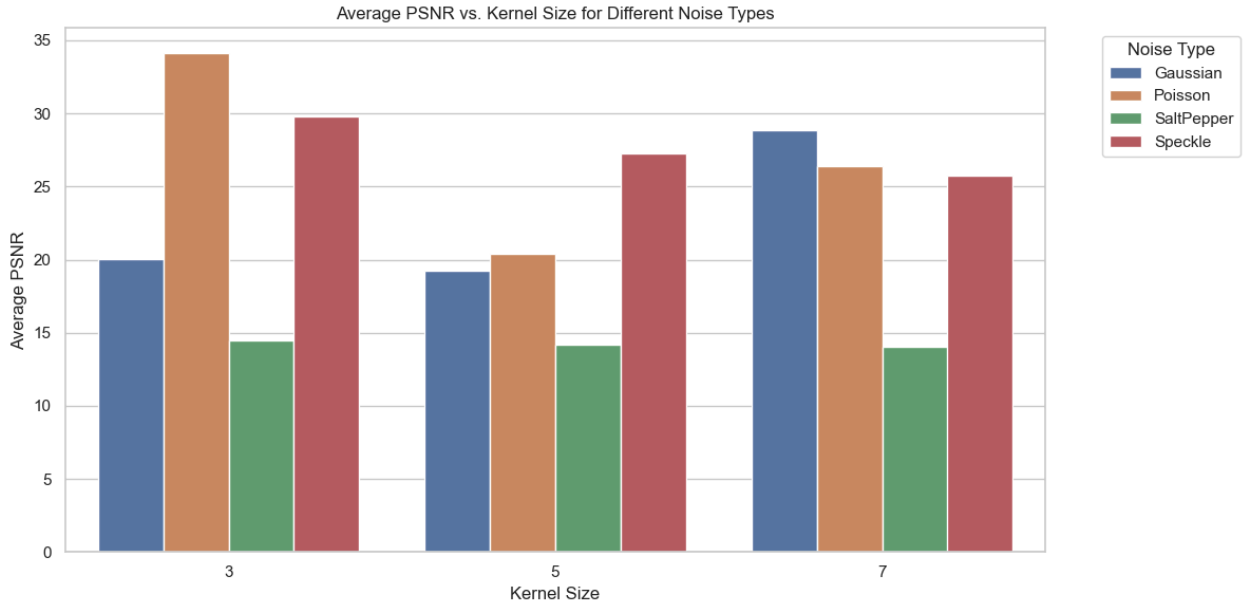
Median filter kernel size = 5

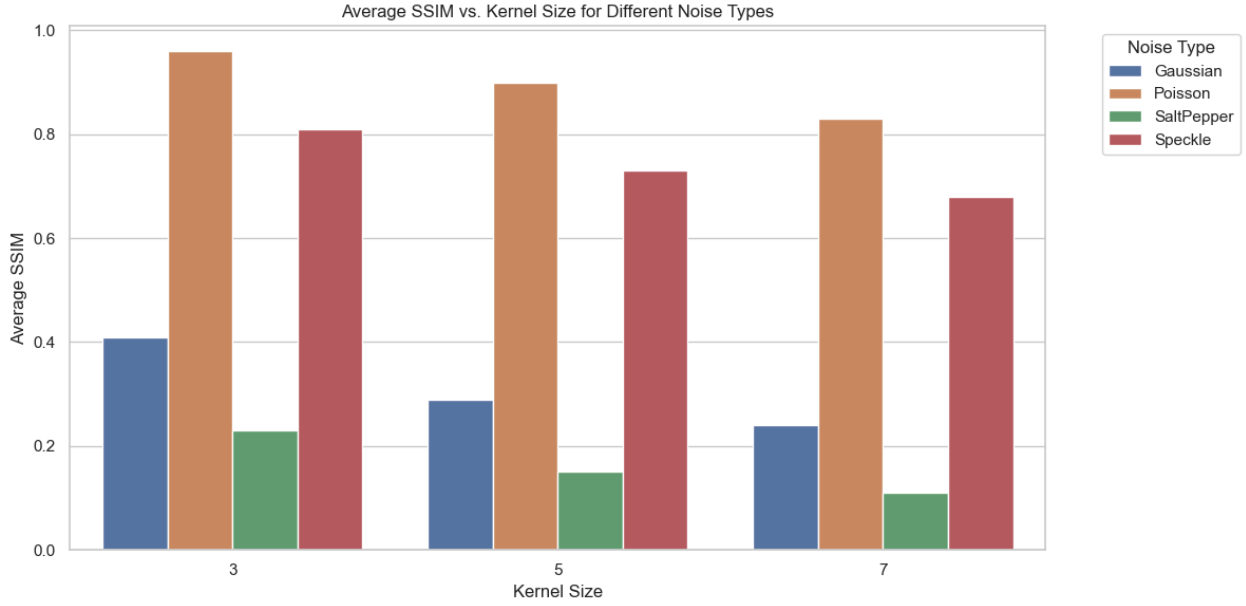
Noise Type	Average PSNR ↑	Average SSIM ↑	Average VIF ↑	Average Processing Time per Frame ↓
Gaussian	19.25	0.29	0.78	0.00171
Poisson	20.40	0.90	0.39	0.00170
SaltPepper	14.15	0.15	0.87	0.00171
Speckle	27.24	0.73	0.54	0.00173

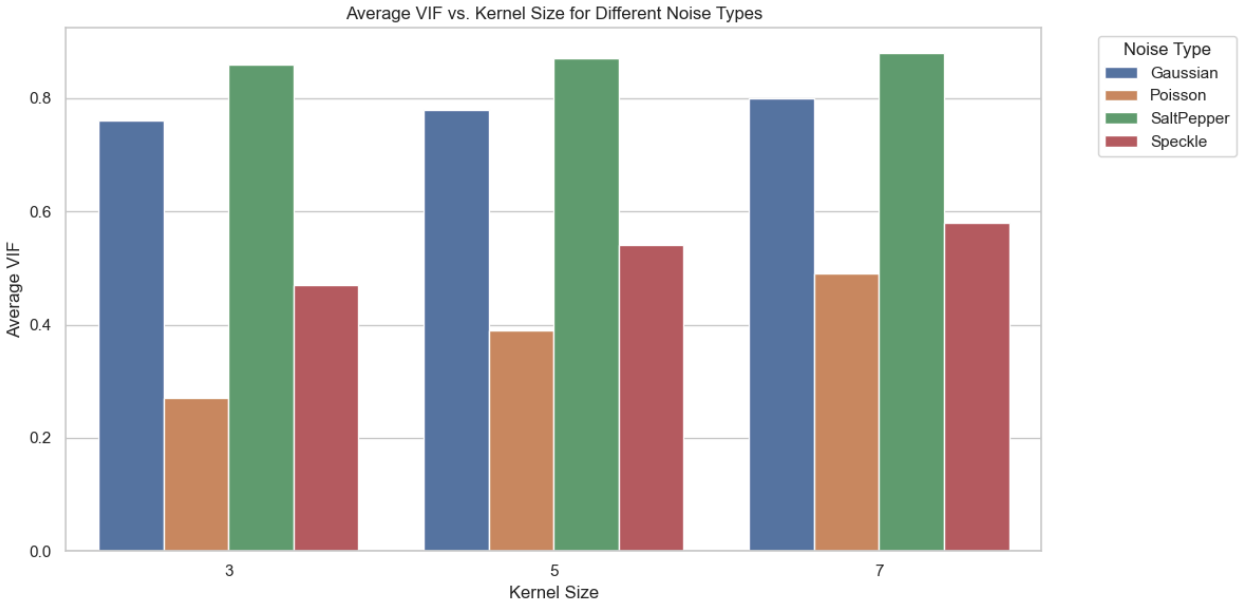
Median filter kernel size = 7

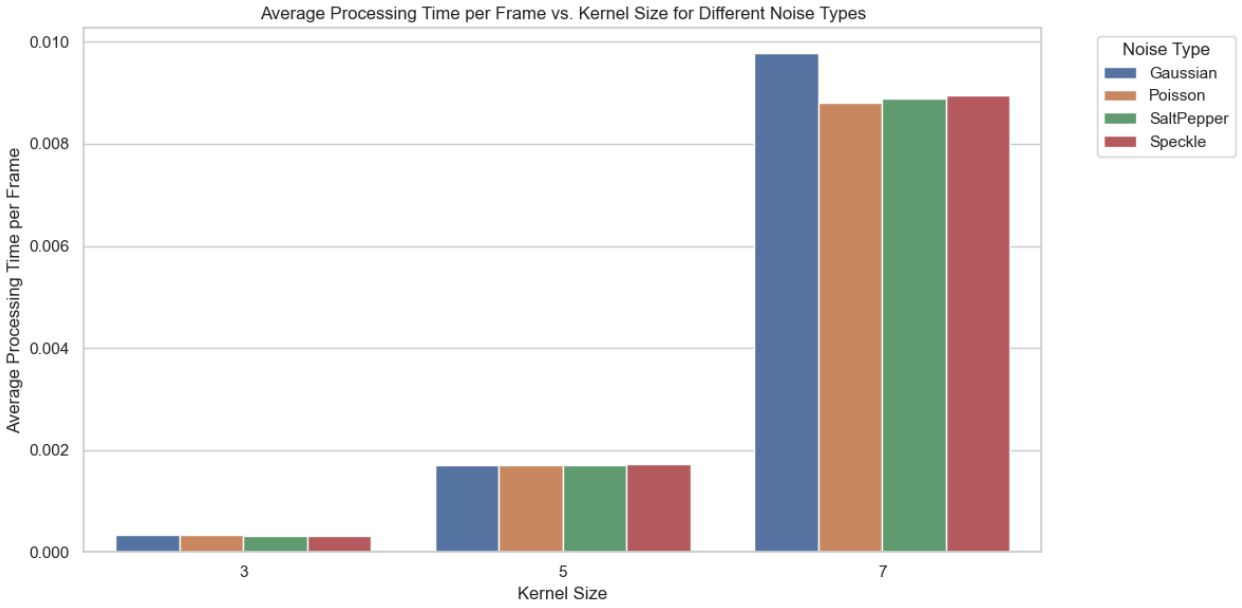
Noise Type	Average PSNR ↑	Average SSIM ↑	Average VIF ↑	Average Processing Time per Frame ↓
Gaussian	28.87	0.24	0.80	0.00978
Poisson	26.43	0.83	0.49	0.00880
SaltPepper	14.07	0.11	0.88	0.00889

Speckle	25.78	0.68	0.58	0.00895









Bilateral filter d=5,spatial kernel size=15, intensity kernel size=15

Noise Type	Average PSNR ↑	Average SSIM ↑	Average VIF ↑	Average Processing Time per Frame ↓
Gaussian	42.76	0.998	0.09	0.00181

Poisson	46.14	0.994	0.07	0.00189
SaltPepper	44.33	0.999	0.08	0.00184
Speckle	42.07	0.987	0.11	0.00183

Bilateral filter d=5,spatial kernel size=20, intensity kernel size=15

Noise Type	Average PSNR ↑	Average SSIM ↑	Average VIF ↑	Average Processing Time per Frame ↓
Gaussian	39.36	0.995	0.15	0.00183
Poisson	44.41	0.991	0.09	0.00189
SaltPepper	41.66	0.998	0.13	0.00182
Speckle	39.11	0.977	0.16	0.00186

Bilateral filter d=5,spatial kernel size=25, intensity kernel size=15

Noise Type	Average PSNR ↑	Average SSIM ↑	Average VIF ↑	Average Processing Time per Frame ↓
Gaussian	36.84	0.991	0.19	0.00184
Poisson	43.13	0.989	0.12	0.00189
SaltPepper	39.54	0.998	0.17	0.00184
Speckle	37.32	0.959	0.21	0.00189

Bilateral filter d=5,spatial kernel size=15, intensity kernel size=20

Noise Type	Average PSNR ↑	Average SSIM ↑	Average VIF ↑	Average Processing Time per Frame ↓
Gaussian	42.75	0.997	0.09	0.00190
Poisson	46.14	0.993	0.07	0.00192
SaltPepper	44.31	0.999	0.08	0.00189
Speckle	41.95	0.986	0.11	0.00192

Bilateral filter d=5,spatial kernel size=15, intensity kernel size=25

Noise Type	Average PSNR ↑	Average SSIM ↑	Average VIF ↑	Average Processing Time per Frame ↓
Gaussian	42.74	0.998	0.09	0.00182
Poisson	46.14	0.993	0.07	0.00188
SaltPepper	44.32	0.999	0.08	0.00186
Speckle	42.06	0.987	0.11	0.00186

Bilateral filter d=5,spatial kernel size=20, intensity kernel size=20

Noise Type	Average PSNR ↑	Average SSIM ↑	Average VIF ↑	Average Processing Time per Frame ↓
Gaussian	37.35	0.995	0.15	0.00186
Poisson	44.41	0.991	0.09	0.00193

SaltPepper	41.65	0.999	0.13	0.00181
Speckle	39.57	0.976	0.16	0.00181

Bilateral filter d=5,spatial kernel size=20, intensity kernel size=25

Noise Type	Average PSNR ↑	Average SSIM ↑	Average VIF ↑	Average Processing Time per Frame ↓
Gaussian	36.83	0.992	0.19	0.00181
Poisson	43.12	0.989	0.12	0.00189
SaltPepper	39.53	0.998	0.17	0.00181
Speckle	37.77	0.962	0.21	0.00181

Bilateral filter d=5,spatial kernel size=25, intensity kernel size=20

Noise Type	Average PSNR ↑	Average SSIM ↑	Average VIF ↑	Average Processing Time per Frame ↓
Gaussian	36.83	0.991	0.19	0.00184
Poisson	43.12	0.989	0.12	0.00187
SaltPepper	39.54	0.998	0.17	0.00181
Speckle	37.28	0.962	0.21	0.00183

Bilateral filter d=5,spatial kernel size=25, intensity kernel size=25

Noise Type	Average PSNR ↑	Average SSIM ↑	Average VIF ↑	Average Processing Time per Frame ↓

Gaussian	36.83	0.992	0.19	0.00181
Poisson	43.12	0.989	0.12	0.00188
SaltPepper	39.53	0.998	0.17	0.00182
Speckle	37.74	0.962	0.21	0.00182

Bilateral filter d=7,spatial kernel size=15, intensity kernel size=15

Noise Type	Average PSNR ↑	Average SSIM ↑	Average VIF ↑	Average Processing Time per Frame ↓
Gaussian	40.99	0.997	0.10	0.00727
Poisson	44.64	0.991	0.08	0.00762
SaltPepper	43.91	0.999	0.08	0.00769
Speckle	40.14	0.981	0.11	0.00769

Bilateral filter d=7,spatial kernel size=15, intensity kernel size=20

Noise Type	Average PSNR ↑	Average SSIM ↑	Average VIF ↑	Average Processing Time per Frame ↓
Gaussian	40.91	0.997	0.10	0.00803
Poisson	44.63	0.990	0.09	0.00656
SaltPepper	43.89	0.999	0.08	0.00646

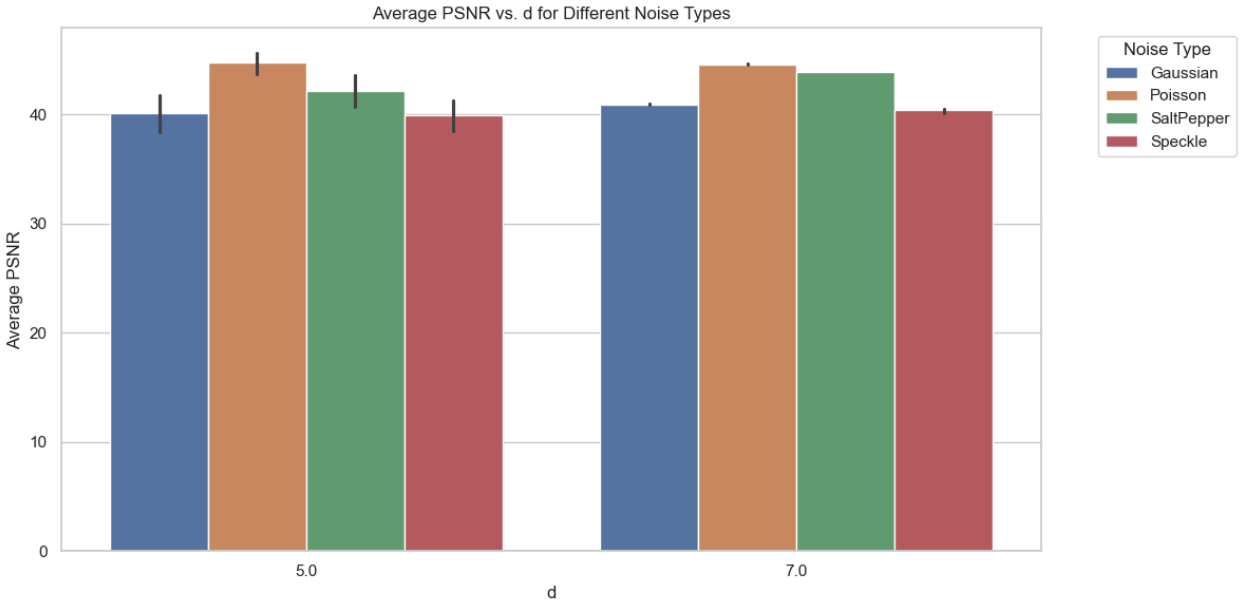
Speckle	40.57	0.984	0.11	0.00651

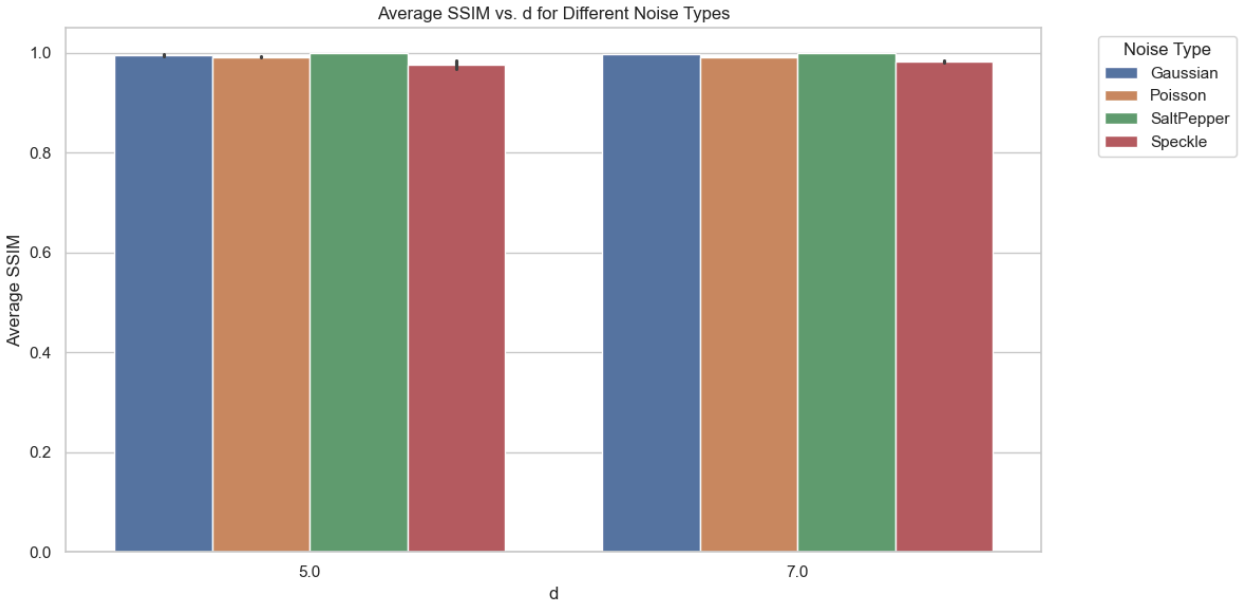
Bilateral filter d=7,spatial kernel size=15, intensity kernel size=25

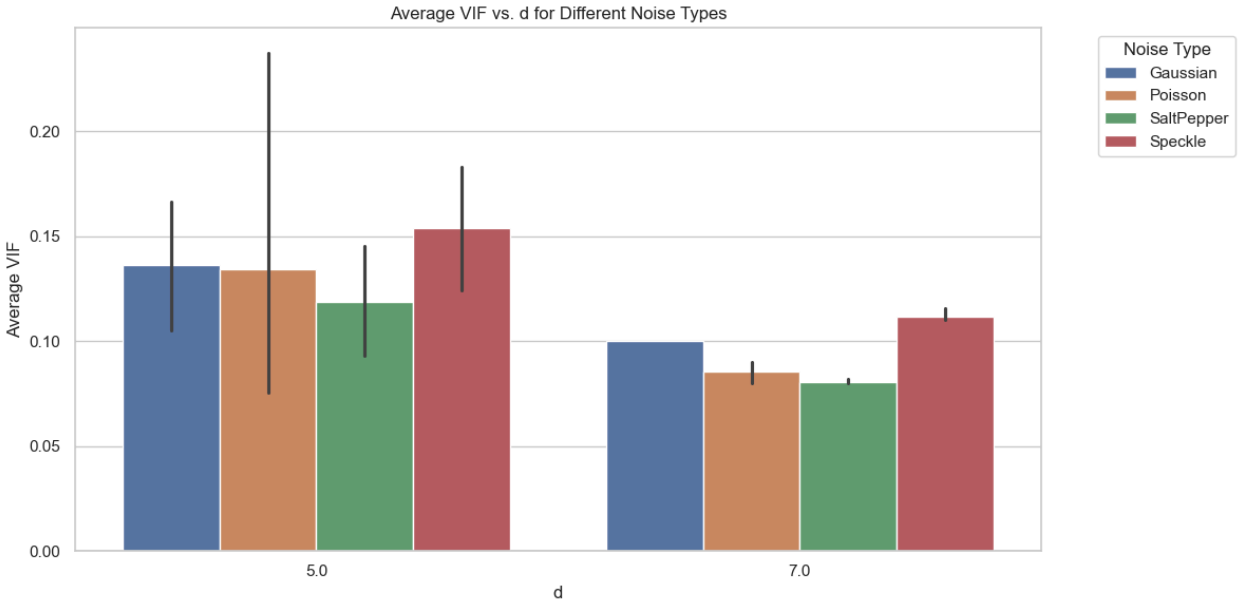
Noise Type	Average PSNR ↑	Average SSIM ↑	Average VIF ↑	Average Processing Time per Frame ↓
Gaussian	40.91	0.997	0.10	0.00745
Poisson	44.63	0.991	0.09	0.00753
SaltPepper	43.89	0.999	0.08	0.00745
Speckle	40.50	0.983	0.12	0.00746

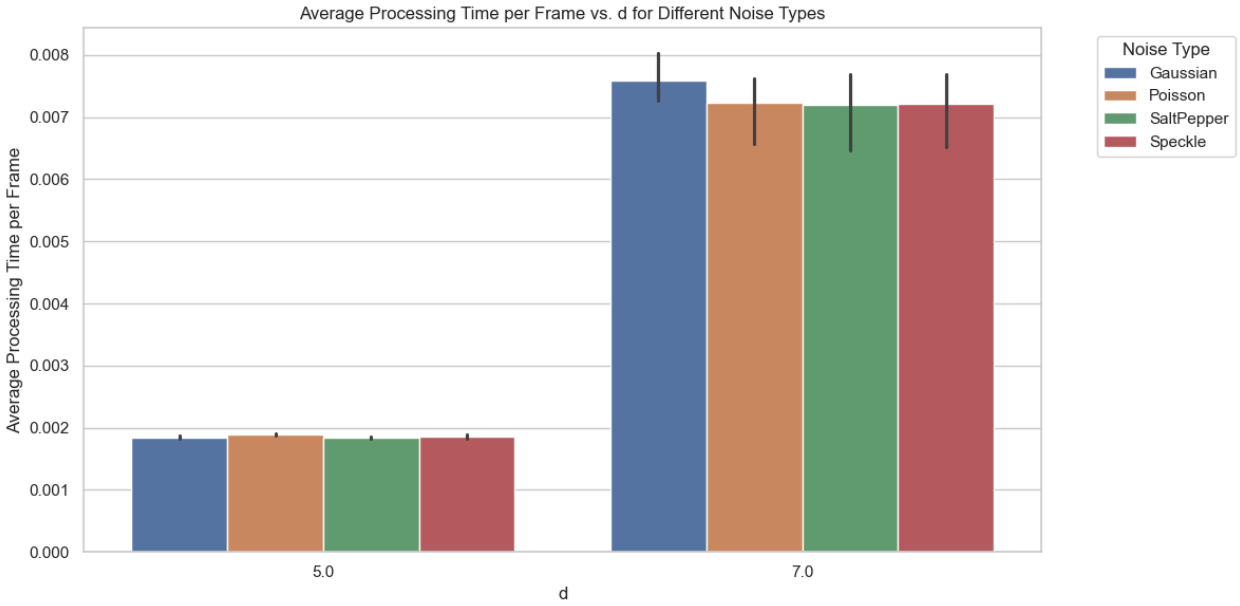
Bilateral filter d=9,spatial kernel size=75, intensity kernel size=75

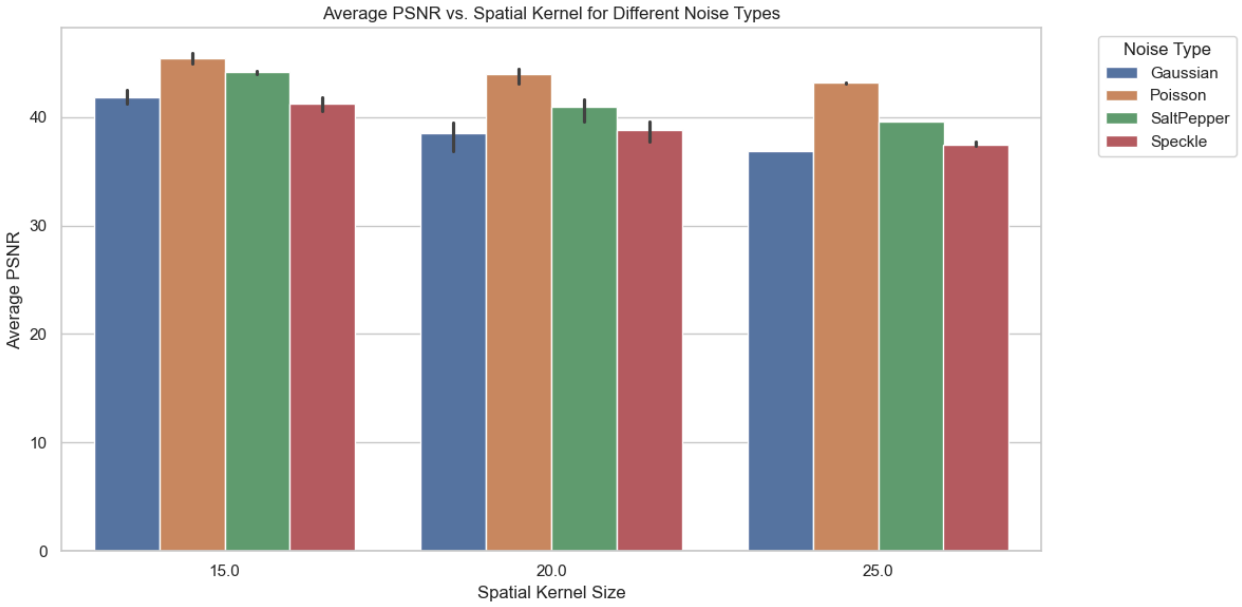
Noise Type	Average PSNR ↑	Average SSIM ↑	Average VIF ↑	Average Processing Time per Frame ↓
Gaussian	25.31	0.842	0.39	0.01137
Poisson	34.23	0.931	0.31	0.01149
SaltPepper	27.76	0.965	0.45	0.01134
Speckle	30.74	0.817	0.40	0.01154

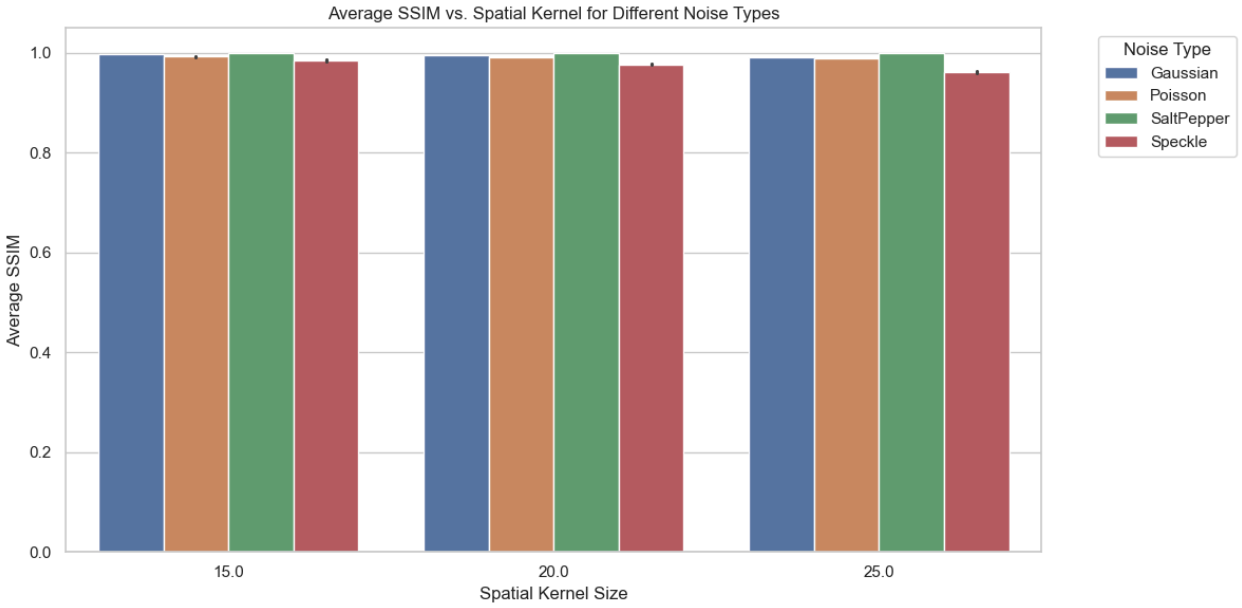

Average PSNR vs. d for Different Noise Types
Average PSNR
0
10
20
30
40
5.0
7.0
d
Noise Type
Gaussian
Poisson
SaltPepper
Speckle

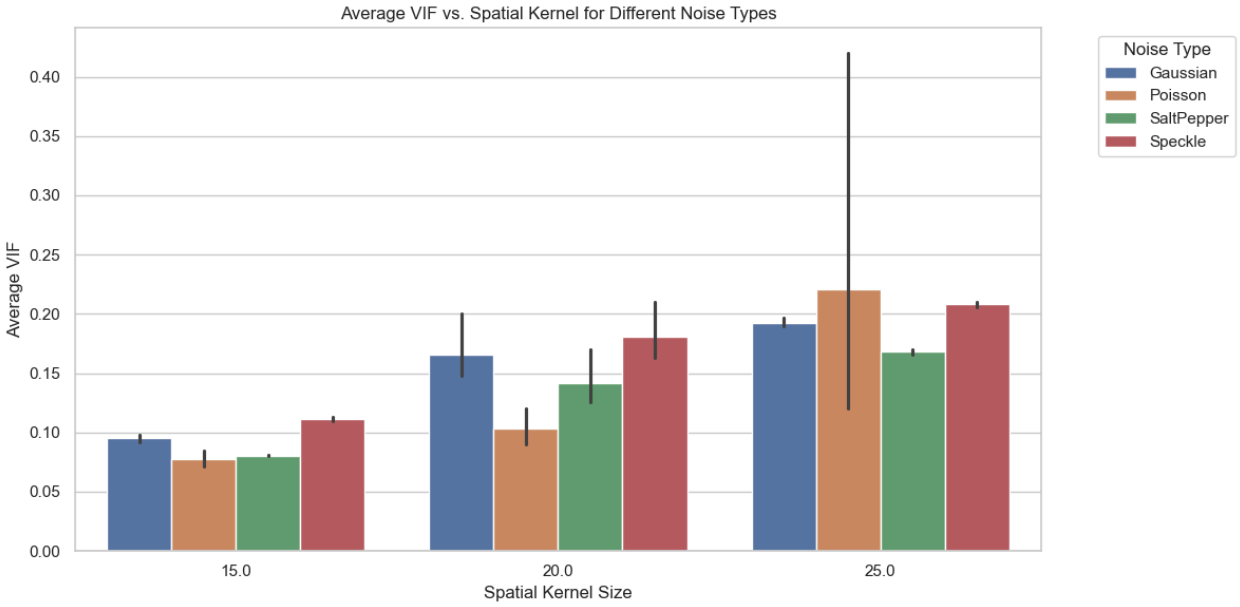

Average SSIM vs. d for Different Noise Types
Average SSIM
0.0
0.2
0.4
0.6
0.8
1.0
5.0
7.0
d
Noise Type
Gaussian
Poisson
SaltPepper
Speckle

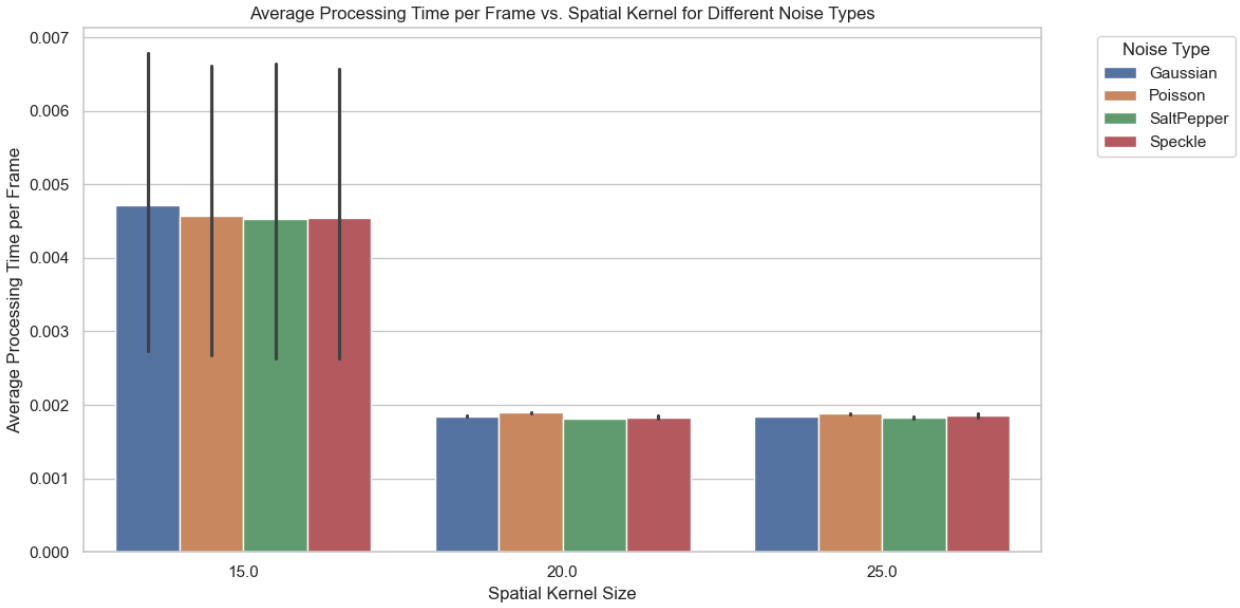

Average VIF vs. d for Different Noise Types
Average VIF
0.00
0.05
0.10
0.15
0.20
5.0
7.0
d
Noise Type
Gaussian
Poisson
SaltPepper
Speckle

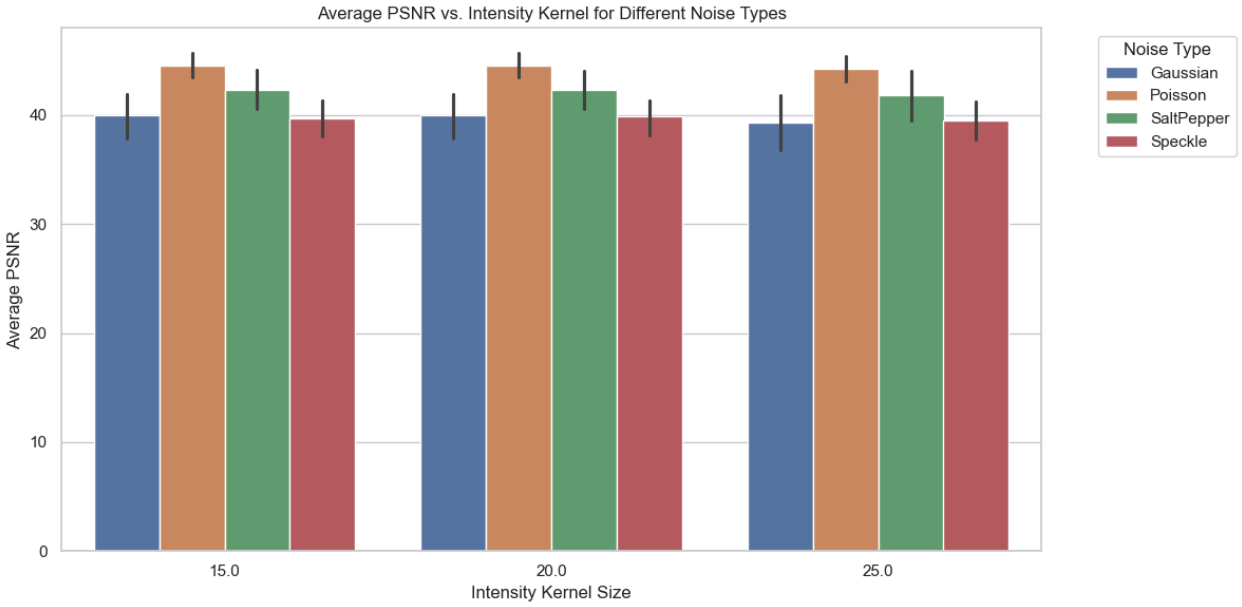

Average Processing Time per Frame vs. d for Different Noise Types
Average Processing Time per Frame
0.000
0.001
0.002
0.003
0.004
0.005
0.006
0.007
0.008
5.0
7.0
d
Noise Type
Gaussian
Poisson
SaltPepper
Speckle

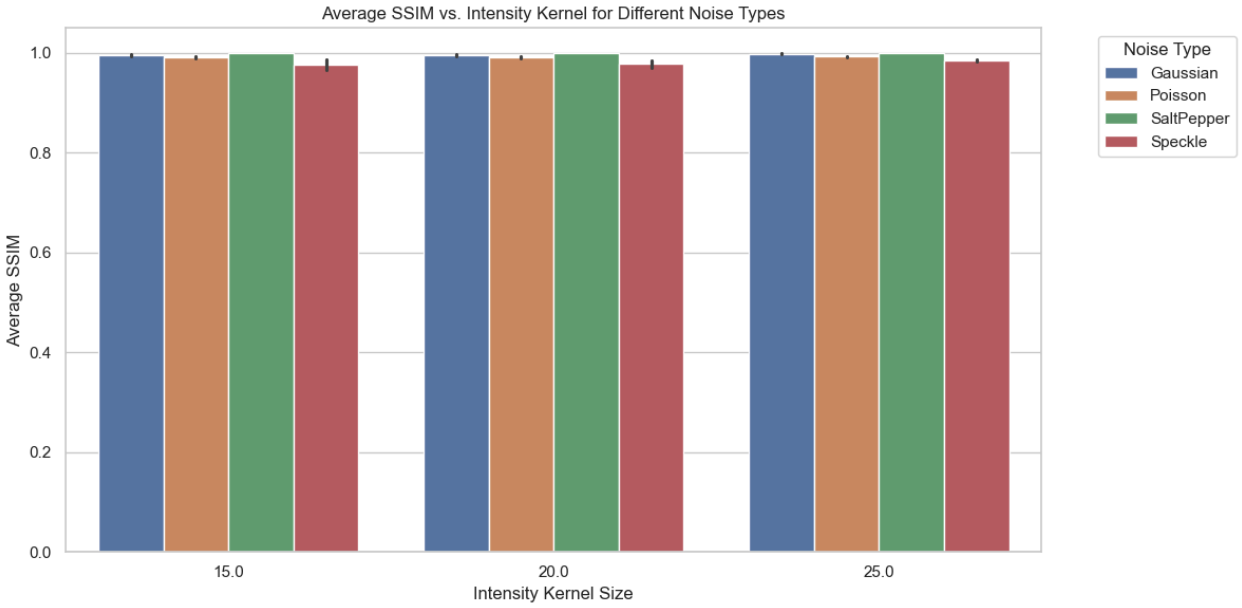

Average PSNR vs. Spatial Kernel for Different Noise Types
Average PSNR
0
10
20
30
40
15.0
20.0
25.0
Spatial Kernel Size
Noise Type
Gaussian
Poisson
SaltPepper
Speckle

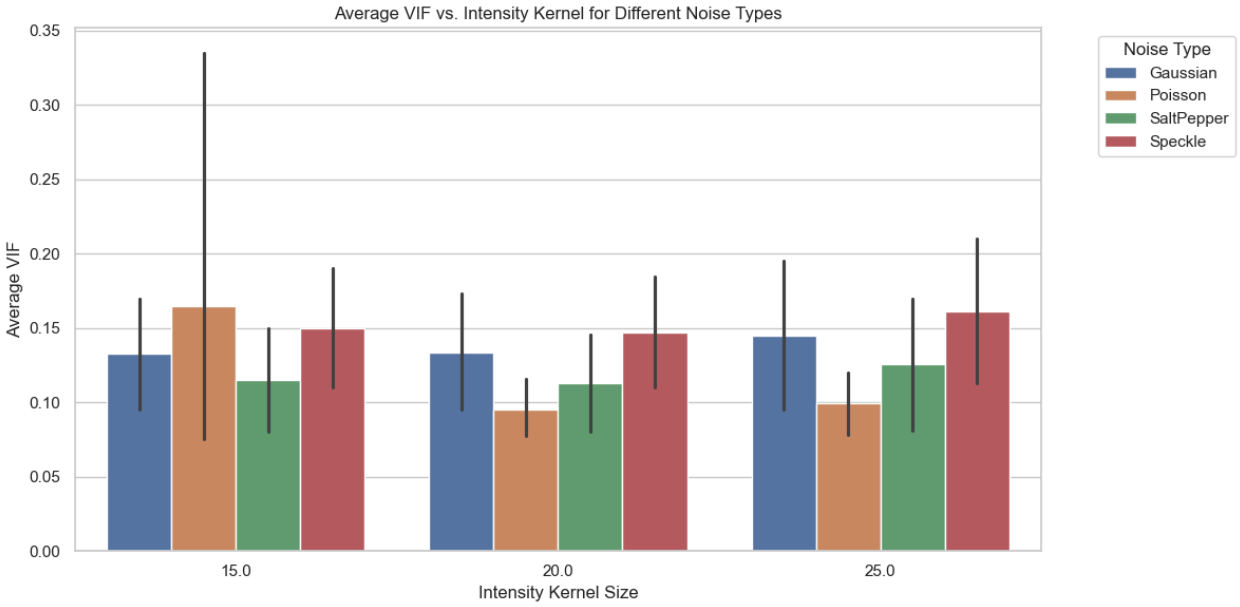

Average SSIM vs. Spatial Kernel for Different Noise Types
Average SSIM
0.0
0.2
0.4
0.6
0.8
1.0
15.0
20.0
25.0
Spatial Kernel Size
Noise Type
Gaussian
Poisson
SaltPepper
Speckle

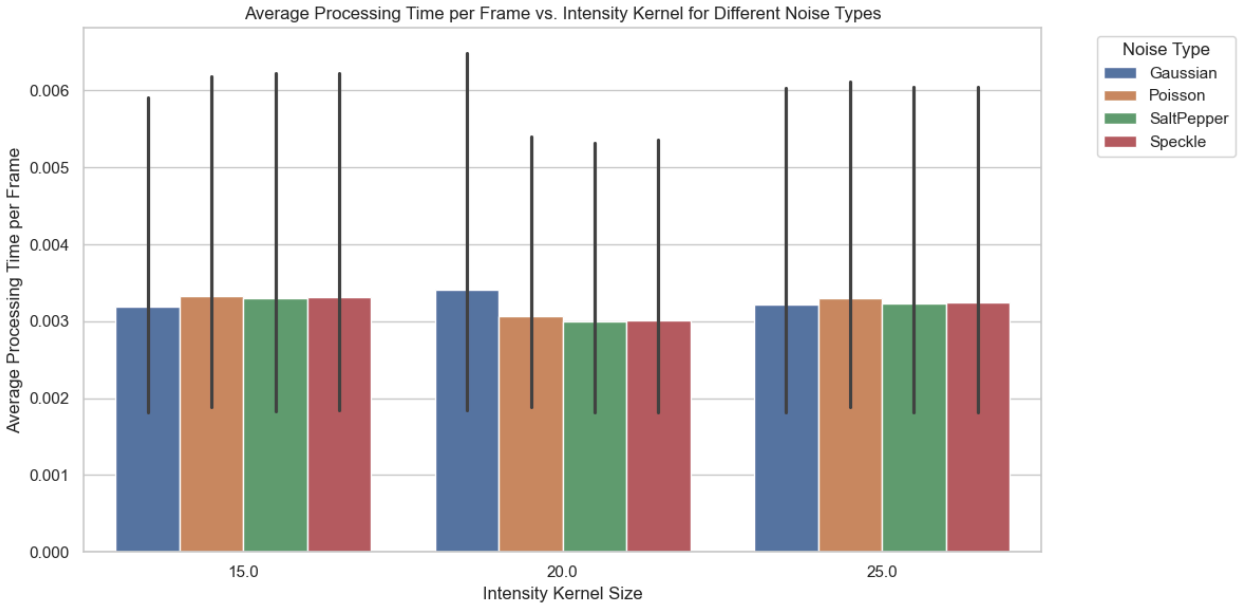

Average VIF vs. Spatial Kernel for Different Noise Types
Average VIF
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
15.0
20.0
25.0
Spatial Kernel Size
Noise Type
Gaussian
Poisson
SaltPepper
Speckle


Average Processing Time per Frame vs. Spatial Kernel for Different Noise Types
Average Processing Time per Frame
0.000
0.001
0.002
0.003
0.004
0.005
0.006
0.007
15.0
20.0
25.0
Spatial Kernel Size
Noise Type
Gaussian
Poisson
SaltPepper
Speckle

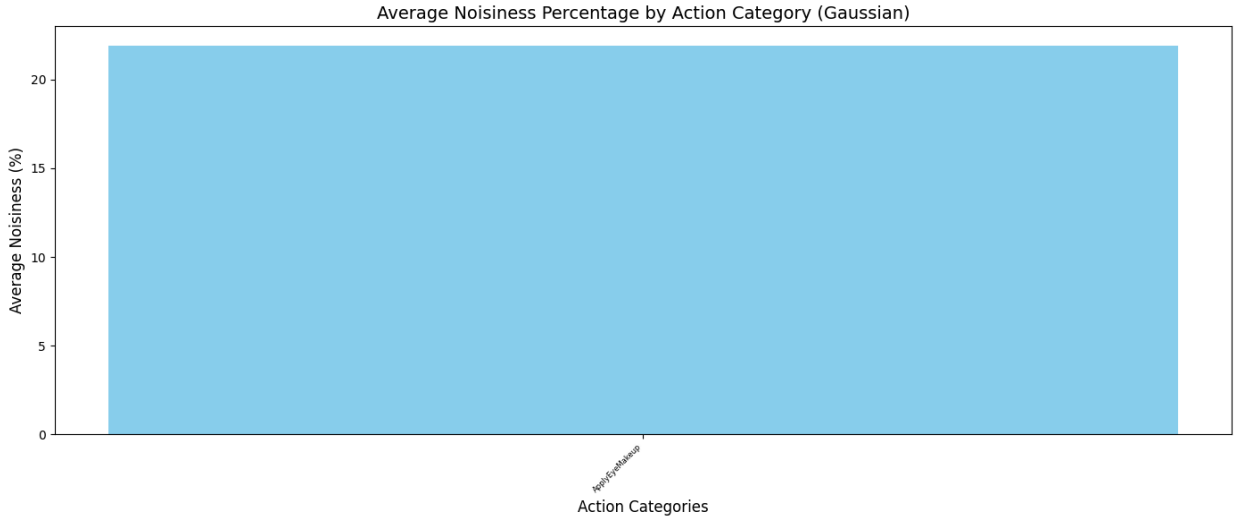

Average PSNR vs. Intensity Kernel for Different Noise Types
Average PSNR
0
10
20
30
40
15.0
20.0
25.0
Intensity Kernel Size
Noise Type
Gaussian
Poisson
SaltPepper
Speckle

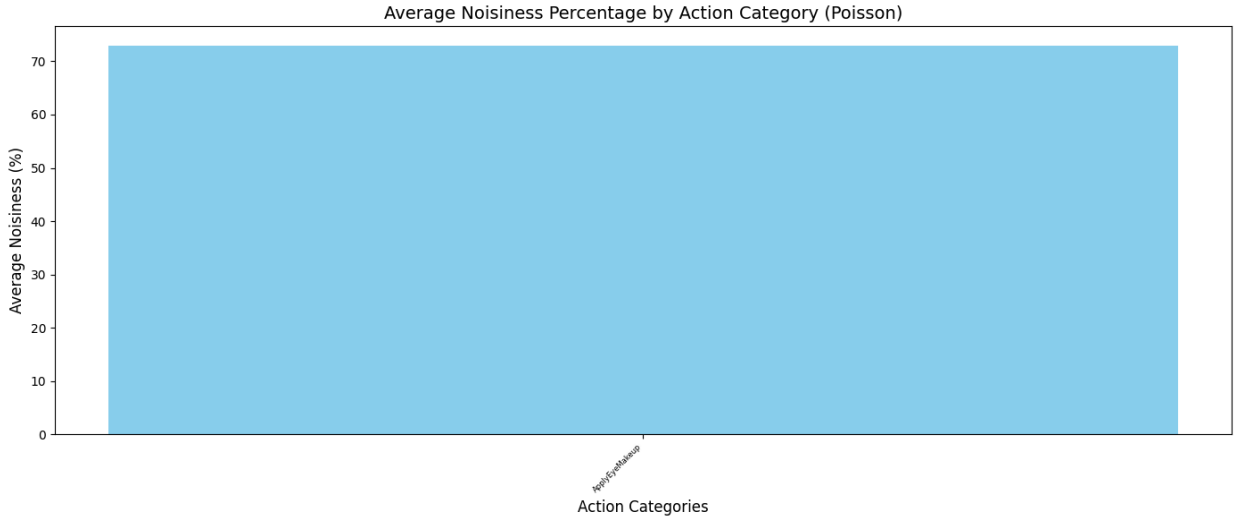

Average SSIM vs. Intensity Kernel for Different Noise Types
Average SSIM
0.0
0.2
0.4
0.6
0.8
1.0
15.0
20.0
25.0
Intensity Kernel Size
Noise Type
Gaussian
Poisson
SaltPepper
Speckle

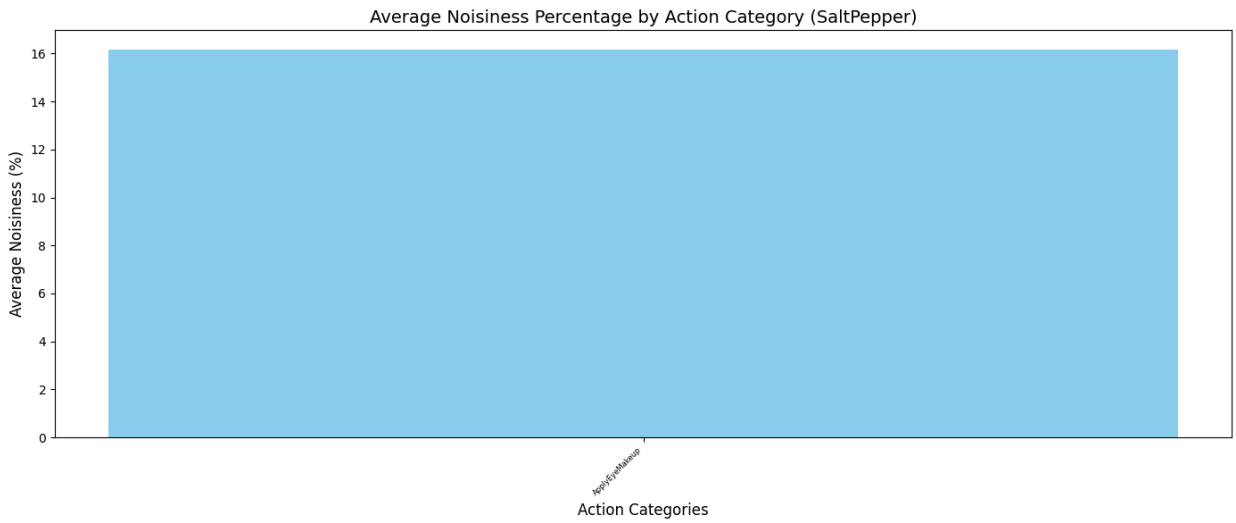

Average VIF vs. Intensity Kernel for Different Noise Types
Average VIF
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
15.0
20.0
25.0
Intensity Kernel Size
Noise Type
Gaussian
Poisson
SaltPepper
Speckle

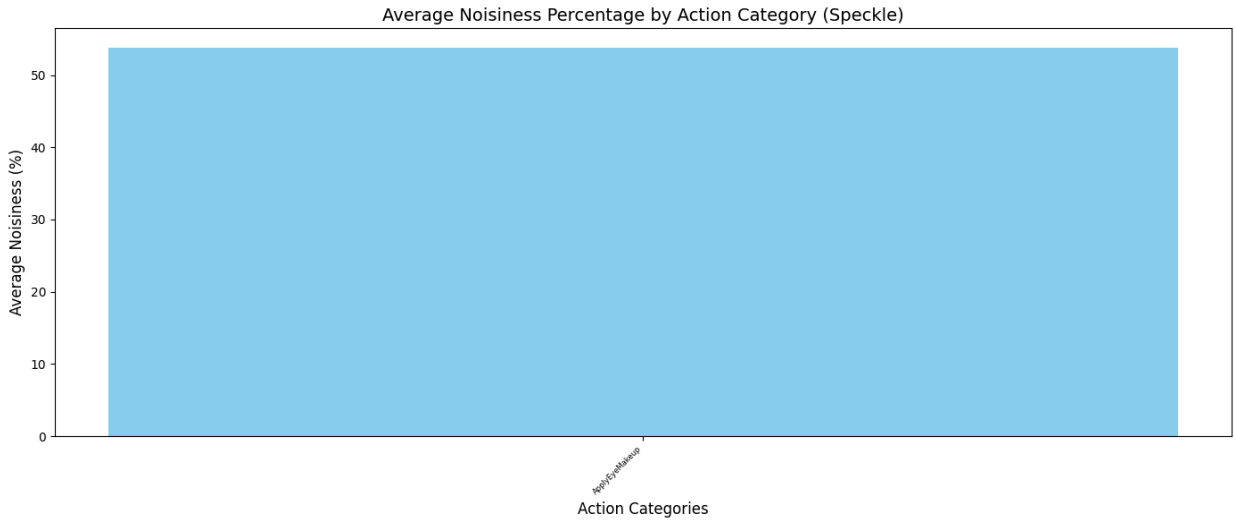

Average Processing Time per Frame vs. Intensity Kernel for Different Noise Types
Average Processing Time per Frame
0.006
0.005
0.004
0.003
0.002
0.001
0.000
15.0
20.0
25.0
Intensity Kernel Size
Noise Type
Gaussian
Poisson
SaltPepper
Speckle

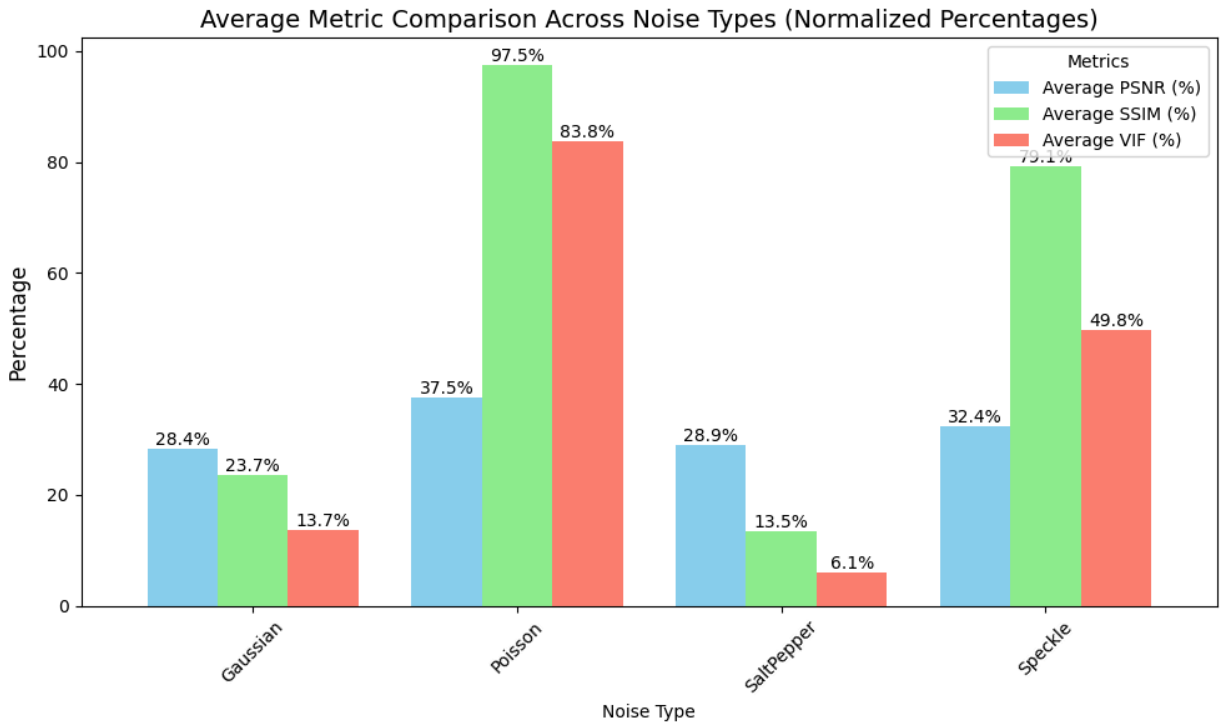
A.3 VRT

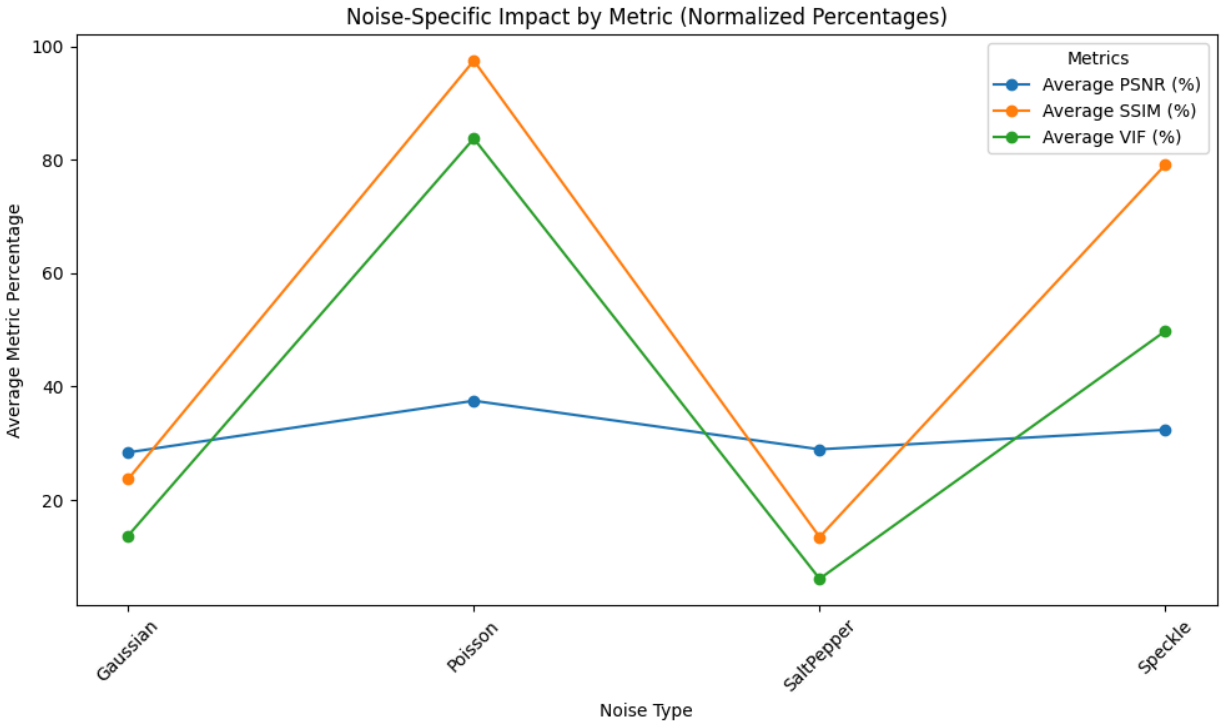

Average Noisiness Percentage by Action Category (Gaussian)
Average Noisiness (%)
0
5
10
15
20
/ApplyEyeMakeup
Action Categories


Average Noisiness Percentage by Action Category (Poisson)
Average Noisiness (%)
0
10
20
30
40
50
60
70
/ApplyEyeMakeup
Action Categories


Average Noisiness Percentage by Action Category (SaltPepper)
Average Noisiness (%)
0
2
4
6
8
10
12
14
16
ApplyEyeMakeup
Action Categories


Average Noisiness Percentage by Action Category (Speckle)
Average Noisiness (%)
0
10
20
30
40
50
ApplyEyeMakeup
Action Categories





A.4 NVIDIA Optix Denoiser

The tables and figures below present the results of running each configuration on the reduced dataset.

Temporal Denoising = None, Tiling = None

Noise Type	Average PSNR ↑	Average SSIM ↑	Average VIF ↑	Average Processing Time per Frame ↓
Gaussian	29.22	0.70	0.66	3.89
Poisson	33.49	0.98	0.17	3.60
SaltPepper	29.09	0.70	0.65	3.79
Speckle	32.23	0.95	0.32	3.72

Temporal Denoising = None, Tiling = 64 x 64

Noise Type	Average PSNR ↑	Average SSIM ↑	Average VIF ↑	Average Processing Time per Frame ↓
Gaussian	29.23	0.70	0.66	15.57
Poisson	33.50	0.98	0.17	15.29
SaltPepper	29.10	0.70	0.65	15.47
Speckle	32.23	0.95	0.32	15.40

Temporal Denoising = None, Tiling = 32 x 32

Noise Type	Average PSNR ↑	Average SSIM ↑	Average VIF ↑	Average Processing Time per Frame ↓
Gaussian	29.22	0.70	0.66	52.31
Poisson	33.50	0.98	0.17	52.02

SaltPepper	29.09	0.70	0.65	52.22
Speckle	32.23	0.95	0.32	52.13

Temporal Denoising = None, Tiling = 16 x 16

Noise Type	Average PSNR ↑	Average SSIM ↑	Average VIF ↑	Average Processing Time per Frame ↓
Gaussian	29.23	0.70	0.66	178.17
Poisson	33.47	0.98	0.18	177.90
SaltPepper	29.10	0.70	0.65	178.19
Speckle	32.22	0.95	0.32	178.00

Temporal Denoising = Direct, Tiling = None

Noise Type	Average PSNR ↑	Average SSIM ↑	Average VIF ↑	Average Processing Time per Frame ↓
Gaussian	29.72	0.76	0.57	19.80
Poisson	33.46	0.98	0.17	19.54
SaltPepper	29.15	0.72	0.61	19.80
Speckle	32.35	0.96	0.30	19.67

Temporal Denoising = Direct, Tiling = 64 x 64

Noise Type	Average PSNR ↑	Average SSIM ↑	Average VIF ↑	Average Processing Time per Frame ↓

Gaussian	29.71	0.76	0.57	32.84
Poisson	33.46	0.98	0.17	32.39
SaltPepper	29.15	0.72	0.61	32.86
Speckle	32.34	0.96	0.30	32.66

Temporal Denoising = Direct, Tiling = 32 x 32

Noise Type	Average PSNR ↑	Average SSIM ↑	Average VIF ↑	Average Processing Time per Frame ↓
Gaussian	29.71	0.76	0.57	75.16
Poisson	33.46	0.98	0.17	74.30
SaltPepper	29.15	0.72	0.61	75.28
Speckle	32.34	0.96	0.30	74.85

Temporal Denoising = Direct, Tiling = 16 x 16

Noise Type	Average PSNR ↑	Average SSIM ↑	Average VIF ↑	Average Processing Time per Frame ↓
Gaussian	29.70	0.76	0.58	216.25
Poisson	33.46	0.98	0.17	214.46
SaltPepper	29.16	0.72	0.62	216.66
Speckle	32.34	0.96	0.30	215.70

Temporal Denoising = Indirect, Tiling = None

Noise Type	Average PSNR ↑	Average SSIM ↑	Average VIF ↑	Average Processing Time per Frame ↓
Gaussian	29.72	0.76	0.57	6.37
Poisson	33.46	0.98	0.17	6.11
SaltPepper	29.15	0.72	0.61	6.34
Speckle	32.35	0.96	0.30	6.25

Temporal Denoising = Indirect, Tiling = 64 x 64

Noise Type	Average PSNR ↑	Average SSIM ↑	Average VIF ↑	Average Processing Time per Frame ↓
Gaussian	29.71	0.76	0.57	19.89
Poisson	33.46	0.98	0.17	19.49
SaltPepper	29.15	0.72	0.61	19.89
Speckle	32.34	0.96	0.30	19.72

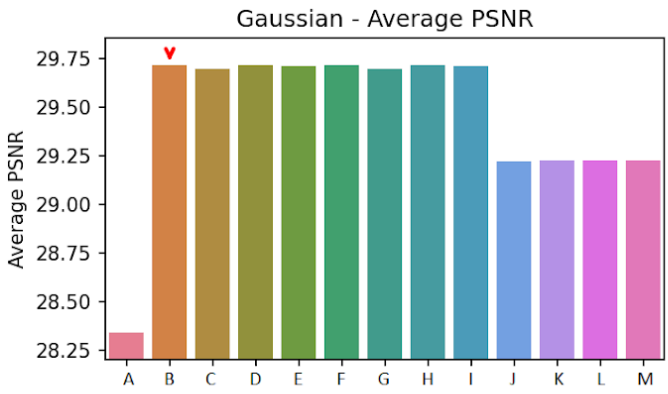
Temporal Denoising = Indirect, Tiling = 32 x 32

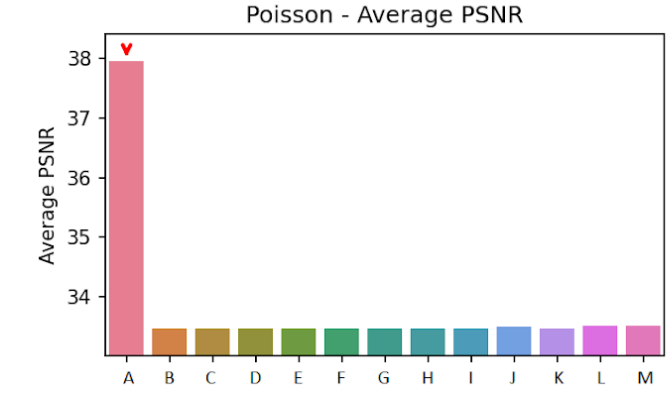
Noise Type	Average PSNR ↑	Average SSIM ↑	Average VIF ↑	Average Processing Time per Frame ↓
Gaussian	29.71	0.76	0.57	62.23
Poisson	33.46	0.98	0.17	61.42
SaltPepper	29.15	0.72	0.61	62.34

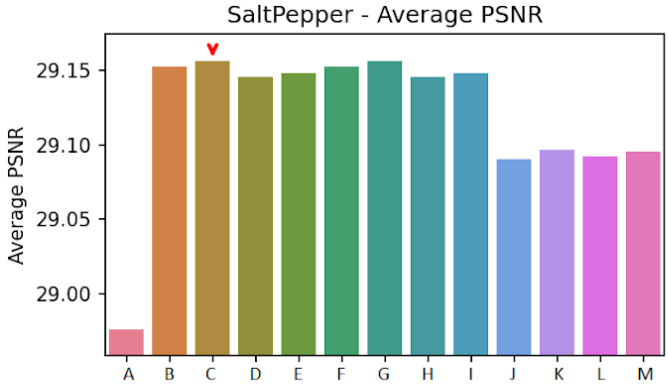
Speckle	32.34	0.96	0.30	61.93

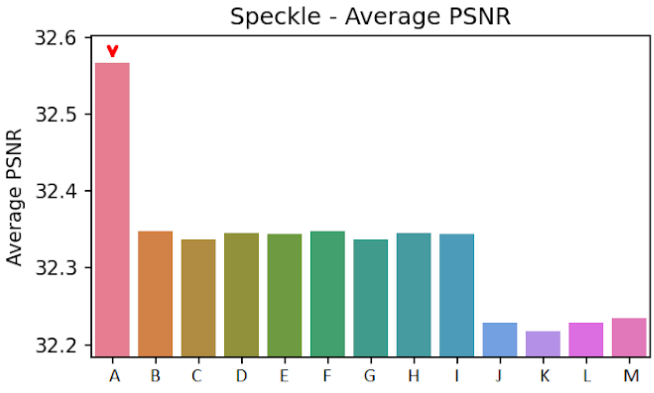
Temporal Denoising = Indirect, Tiling = 16 x 16

Noise Type	Average PSNR ↑	Average SSIM ↑	Average VIF ↑	Average Processing Time per Frame ↓
Gaussian	29.70	0.76	0.58	207.70
Poisson	33.46	0.98	0.17	205.85
SaltPepper	29.16	0.72	0.62	208.19
Speckle	32.34	0.96	0.30	207.16









Key:

A - **Noisy**

B - Temporal = **Indirect**, Tiling = **16 x 16**

C - Temporal = **Indirect**, Tiling = 3**2 x 32**

D - Temporal = **Indirect**, Tiling = **64 x 64**

E - Temporal = **Indirect**, Tiling = **None**

F - Temporal = **Direct**, Tiling = **16 x 16**

G - Temporal = **Direct**, Tiling = 3**2 x 32**

H - Temporal = **Direct**, Tiling = **64 x 64**

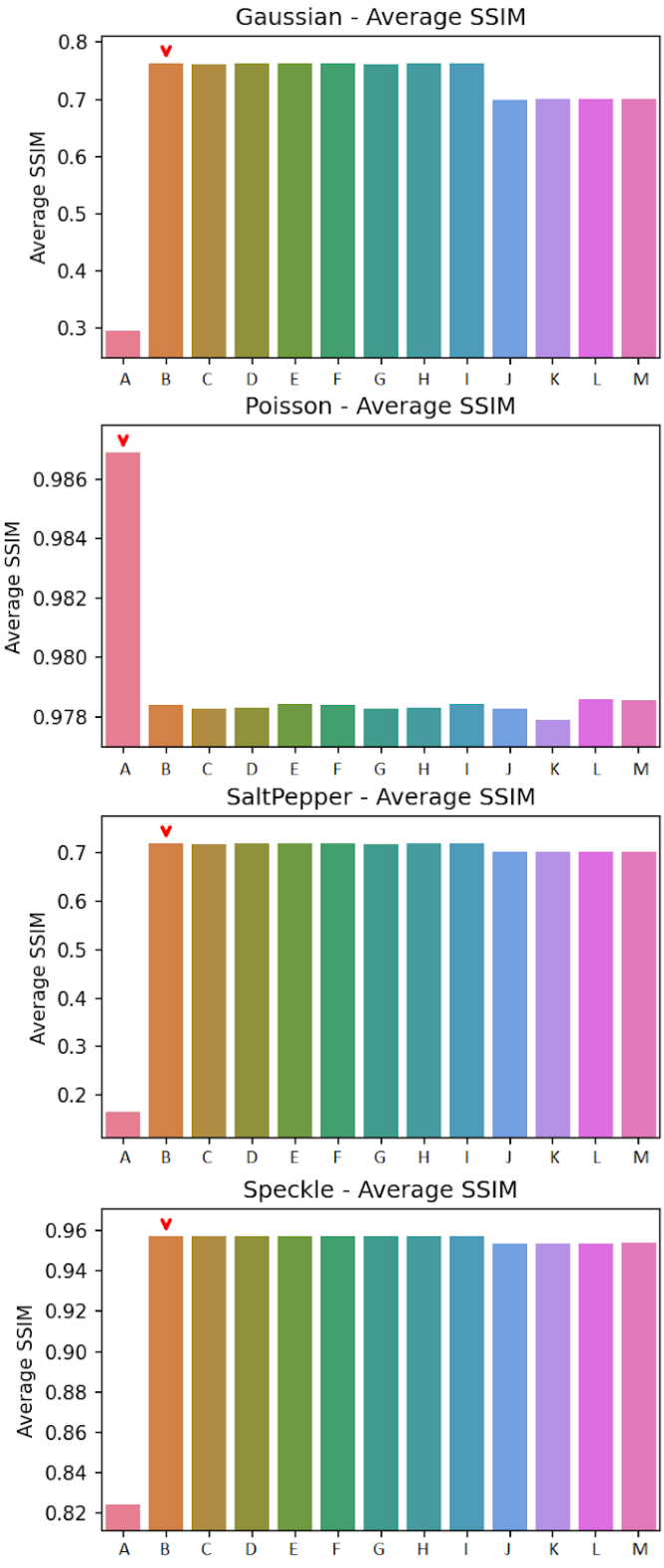
I - Temporal = **Direct**, Tiling = **None**

J - Temporal = **None**, Tiling = **16 x 16**

K - Temporal = **None**, Tiling = 3**2 x 32**

L - Temporal = **None**, Tiling = **64 x 64**

M - Temporal = **None**, Tiling = **None**



Key:

A - **Noisy**

B - Temporal = **Indirect**, Tiling = **16 x 16**

C - Temporal = **Indirect**, Tiling = 3**2 x 32**

D - Temporal = **Indirect**, Tiling = **64 x 64**

E - Temporal = **Indirect**, Tiling = **None**

F - Temporal = **Direct**, Tiling = **16 x 16**

G - Temporal = **Direct**, Tiling = 3**2 x 32**

H - Temporal = **Direct**, Tiling = **64 x 64**

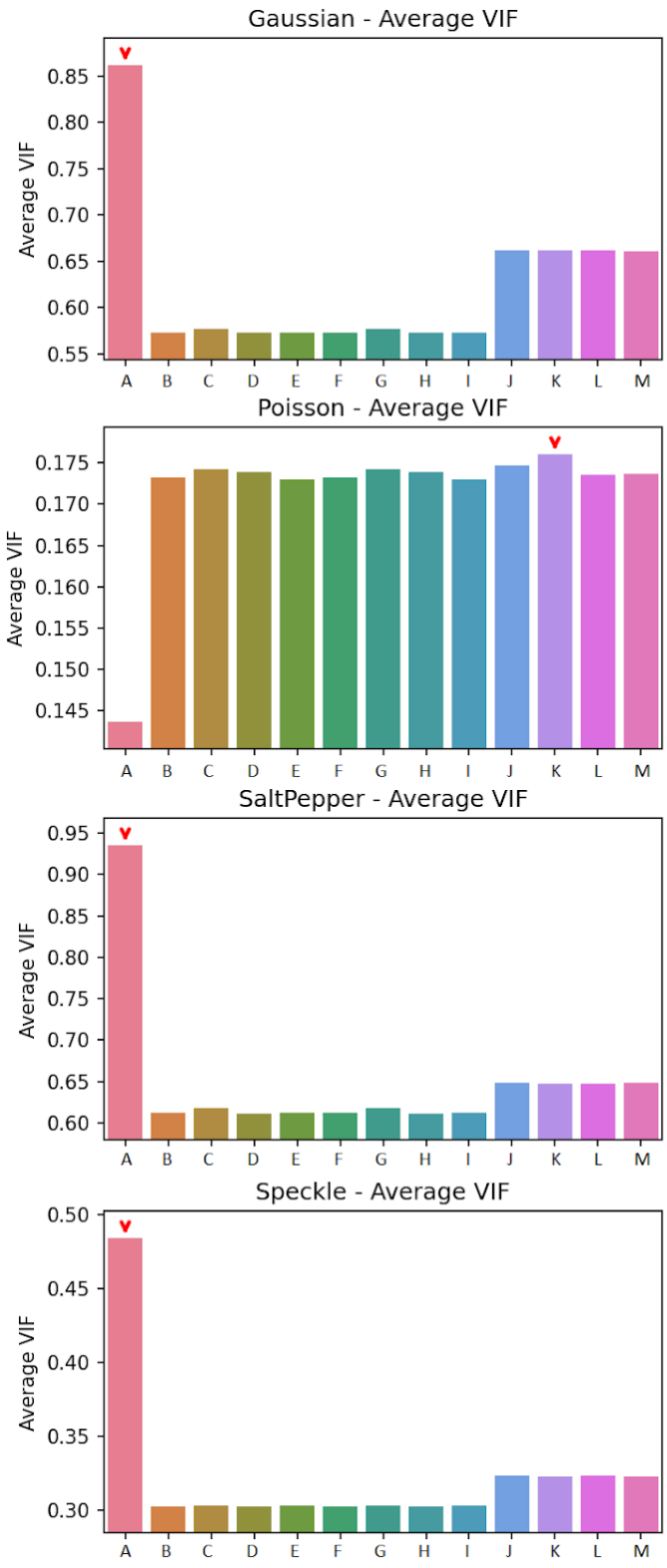
I - Temporal = **Direct**, Tiling = **None**

J - Temporal = **None**, Tiling = **16 x 16**

K - Temporal = **None**, Tiling = 3**2 x 32**

L - Temporal = **None**, Tiling = **64 x 64**

M - Temporal = **None**, Tiling = **None**



Key:

A - **Noisy**
B - Temporal = **Indirect**, Tiling = **16 x 16**
C - Temporal = **Indirect**, Tiling = 3**2 x 32**
D - Temporal = **Indirect**, Tiling = **64 x 64**
E - Temporal = **Indirect**, Tiling = **None**
F - Temporal = **Direct**, Tiling = **16 x 16**
G - Temporal = **Direct**, Tiling = 3**2 x 32**
H - Temporal = **Direct**, Tiling = **64 x 64**
I - Temporal = **Direct**, Tiling = **None**
J - Temporal = **None**, Tiling = **16 x 16**
K - Temporal = **None**, Tiling = 3**2 x 32**
L - Temporal = **None**, Tiling = **64 x 64**
M - Temporal = **None**, Tiling = **None**

Full dataset

Temporal Denoising = None, Tiling = None

Noise Type	Average PSNR ↑	Average SSIM ↑	Average VIF ↑	Average Processing Time per Frame ↓
Gaussian	29.19	0.70	0.66	3.90
Poisson	33.56	0.98	0.17	3.60
SaltPepper	29.04	0.70	0.65	3.79
Speckle	32.24	0.95	0.33	3.73

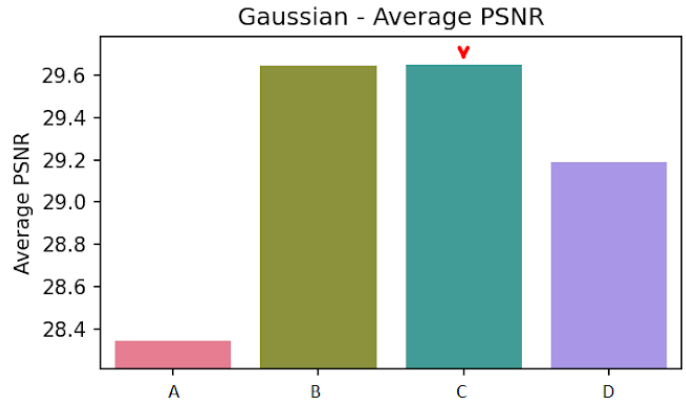
Temporal Denoising = Direct, Tiling = None

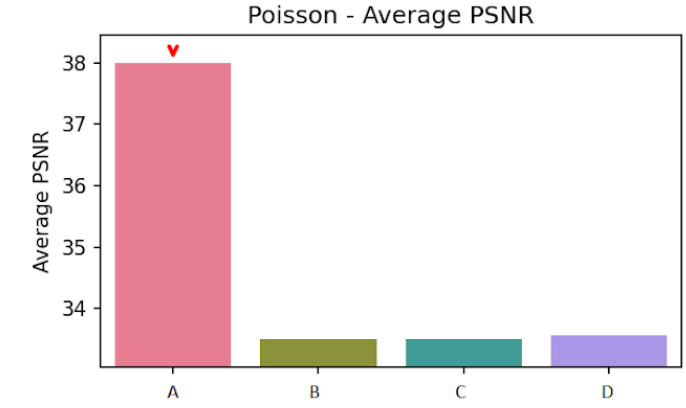
Noise Type	Average PSNR ↑	Average SSIM ↑	Average VIF ↑	Average Processing Time per Frame ↓
Gaussian	29.65	0.76	0.57	19.74
Poisson	33.50	0.98	0.17	19.40
SaltPepper	29.10	0.72	0.61	19.72
Speckle	32.35	0.95	0.30	19.57

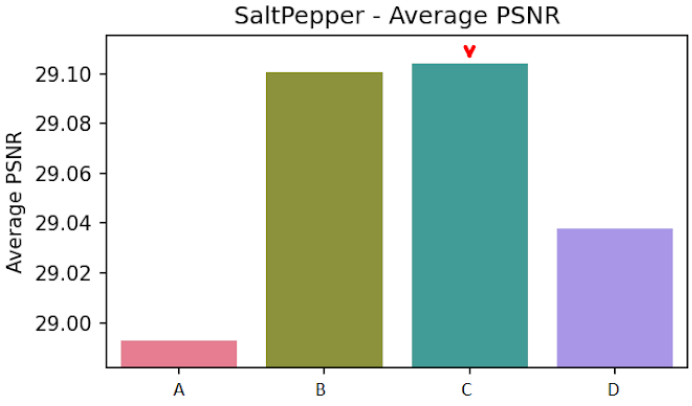
Temporal Denoising = Indirect, Tiling = None

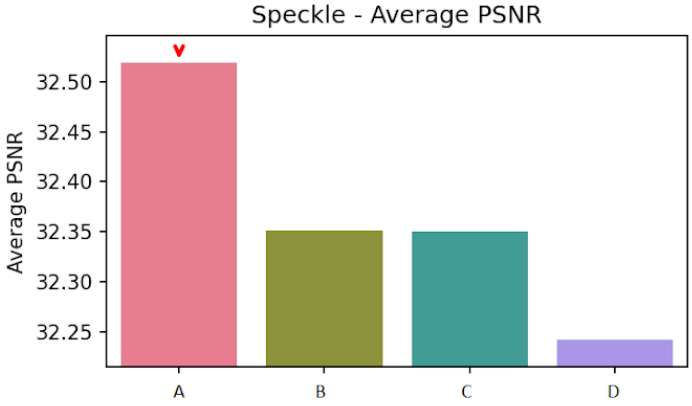
Noise Type	Average PSNR ↑	Average SSIM ↑	Average VIF ↑	Average Processing Time per Frame ↓
Gaussian	29.65	0.76	0.57	6.36
Poisson	33.51	0.98	0.17	6.09
SaltPepper	29.10	0.71	0.62	6.32

Speckle	32.35	0.95	0.30	6.22









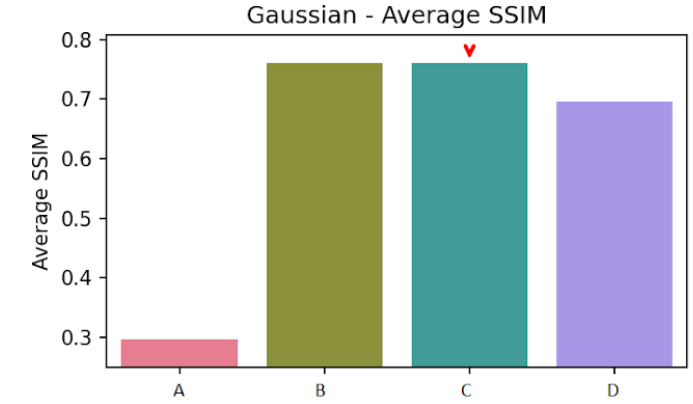
Key:

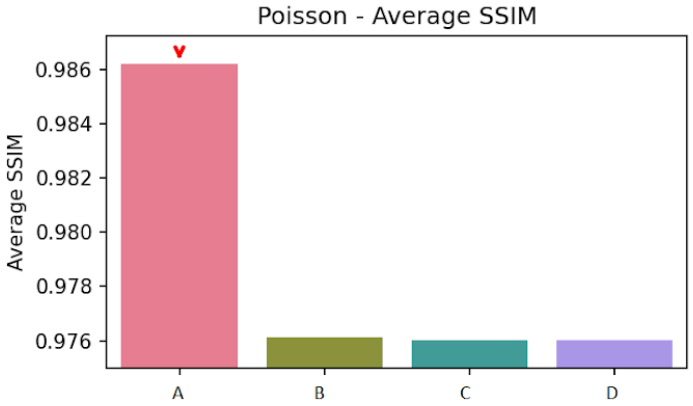
A - **Noisy**

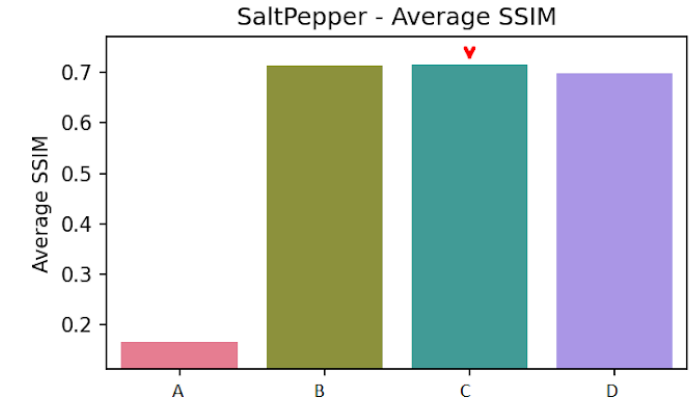
B - Temporal = **Indirect**, Tiling = **None**

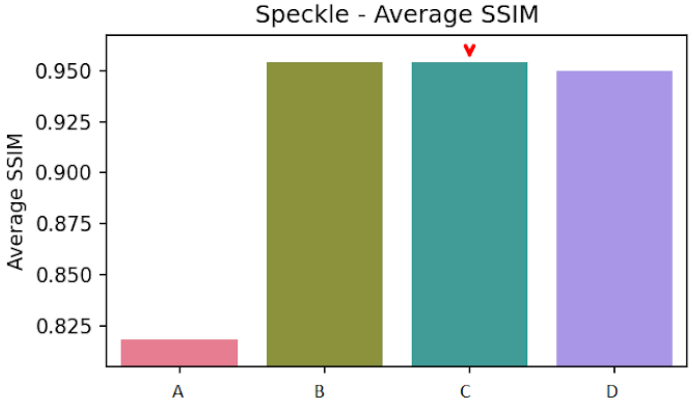
C - Temporal = **Direct**, Tiling = **None**

D - Temporal = **None**, Tiling = **None**









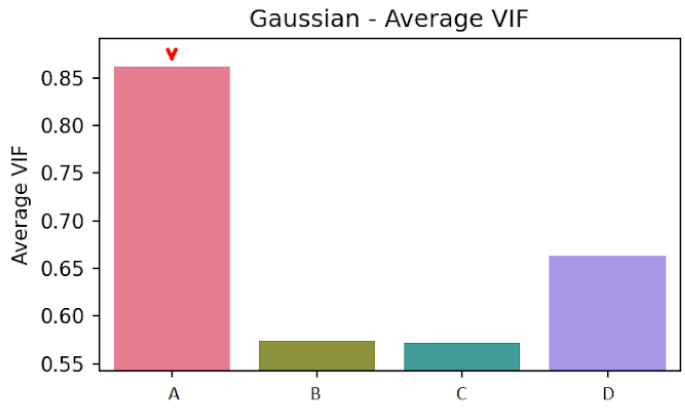
Key:

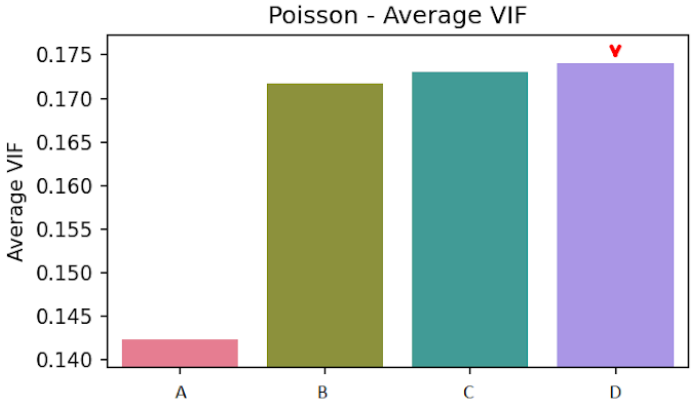
A - **Noisy**

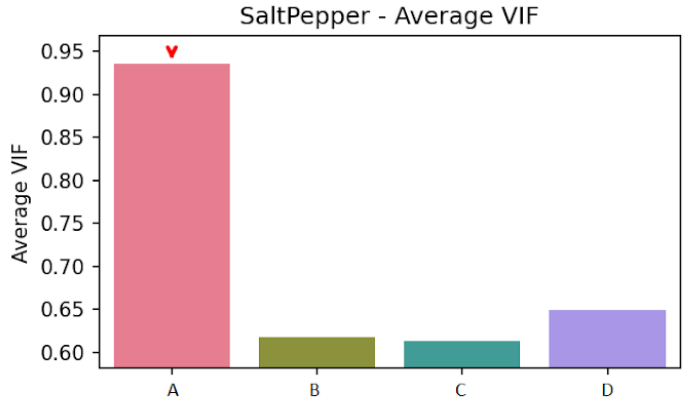
B - Temporal = **Indirect**, Tiling = **None**

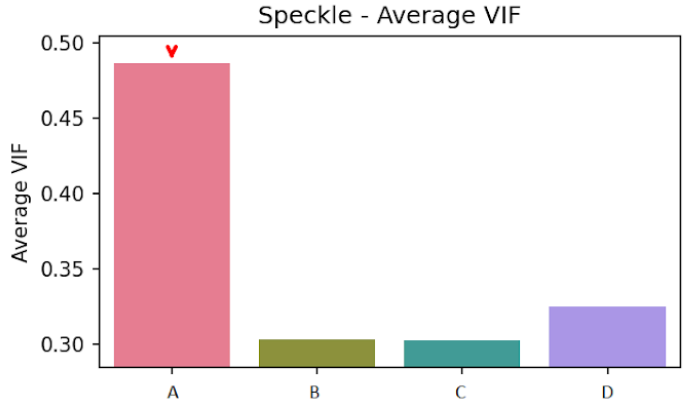
C - Temporal = **Direct**, Tiling = **None**

D - Temporal = **None**, Tiling = **None**









Key:

A - **Noisy**

B - Temporal = **Indirect**, Tiling = **None**

C - Temporal = **Direct**, Tiling = **None**

D - Temporal = **None**, Tiling = **None**