

EVALUASI GAUSSIAN BLUR DALAM MEREDUKSI NOISE CITRA WAJAH PADA SISTEM PRESENSI MULTIFAKTOR

Tugas Akhir Skripsi



oleh:

GIAN PRADIPTA GUNAWAN

71210689

**PROGRAM STUDI INFORMATIKA FAKULTAS TEKNOLOGI INFORMASI
UNIVERSITAS KRISTEN DUTA WACANA**

TAHUN 2025

EVALUASI GAUSSIAN BLUR DALAM MEREDUKSI NOISE CITRA WAJAH PADA SISTEM PRESENSI MULTIFAKTOR

Tugas Akhir Skripsi



Diajukan kepada Program Studi Informatika Fakultas Teknologi Informasi

Universitas Kristen Duta Wacana

Sebagai Salah Satu Syarat dalam Memperoleh Gelar

Sarjana Komputer

Disusun oleh

GIAN PRADIPTA GUNAWAN

71210689

PROGRAM STUDI INFORMATIKA FAKULTAS TEKNOLOGI INFORMASI

UNIVERSITAS KRISTEN DUTA WACANA

TAHUN 2025

PERNYATAAN KEASLIAN TUGAS AKHIR

Saya menyatakan dengan sesungguhnya bahwa tugas akhir dengan judul:

EVALUASI GAUSSIAN BLUR DALAM MEREDUKSI *NOISE* CITRA WAJAH PADA SISTEM PRESENSI MULTIFAKTOR

yang saya kerjakan untuk melengkapi sebagian persyaratan menjadi Sarjana Komputer pada pendidikan Sarjana Program Studi Informatika Fakultas Teknologi Informasi Universitas Kristen Duta Wacana, bukan merupakan tiruan atau duplikasi dari tugas akhir kesarjanaan di lingkungan Universitas Kristen Duta Wacana maupun di Perguruan Tinggi atau instansi manapun, kecuali bagian yang sumber informasinya dicantumkan sebagaimana mestinya.

Jika dikemudian hari didapati bahwa hasil tugas akhir ini adalah hasil plagiasi atau tiruan dari tugas akhir lain, saya bersedia dikenai sanksi yakni pencabutan gelar kesarjanaan saya.

Yogyakarta, 30 Juni 2025

MATERAI Rp.10,000

GIAN PRADIPTA GUNAWAN

71210689

HALAMAN PERSETUJUAN

Judul Tugas Akhir : EVALUASI GAUSSIAN BLUR DALAM MEREDUKSI
NOISE CITRA WAJAH PADA SISTEM PRESENSI
MULTIFAKTOR

Nama Mahasiswa : Gian Pradipta Gunawan

NIM : 71210689

Mata Kuliah : Tugas Akhir

Kode : TI0366

Semester : Genap

Tahun Akademik : 2024/2025

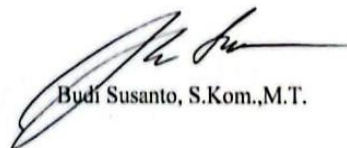
Telah diperiksa dan disetujui di
Yogyakarta,
Pada tanggal 20 Mei 2025

Dosen Pembimbing I

Dosen Pembimbing II



I Kadek Dendy Senapartha. S. T., M. Eng.



Budi Susanto, S.Kom.,M.T.

HALAMAN PENGESAHAN

EVALUASI GAUSSIAN BLUR DALAM MEREDUKSI *NOISE* CITRA WAJAH PADA SISTEM PRESENSI MULTIFAKTOR

Oleh: GIAN PRADIPTA GUNAWAN/ 71210689

Dipertahankan di depan Dewan Penguji Tugas Akhir

Program Studi Informatika Fakultas Teknologi Informasi

Universitas Kristen Duta Wacana - Yogyakarta

Dan dinyatakan diterima untuk memenuhi salah satu syarat memperoleh gelar

Sarjana Komputer

pada tanggal

Yogyakarta, -----

Mengesahkan,

Dewan Penguji:

1. Dosen 1

2. Dosen 2

3.

4.

Dekan

Ketua Program Studi

(Restyandito,S.Kom.,MSIS.,Ph.D.)

(Joko Purwadi, S.Kom., M.Kom.)

**HALAMAN PERNYATAAN PERSETUJUAN PUBLIKASI
TUGAS AKHIR UNTUK KEPENTINGAN AKADEMIS
SECARA ONLINE
UNIVERSITAS KRISTEN DUTA WACANA YOGYAKARTA**

Saya yang bertanda tangan di bawah ini:

| | | |
|-------------------|---|---|
| NIM | : | 71210689 |
| Nama | : | Gian Pradipta Gunawan |
| Fakultas / Prodi | : | Teknologi Informasi / Informatika |
| Judul Tugas Akhir | : | EVALUASI GAUSSIAN BLUR DALAM MEREDUKSI NOISE CITRA WAJAH PADA SISTEM PRESENSI MULTIFAKTOR |

bersedia menyerahkan Tugas Akhir kepada Universitas melalui Perpustakaan untuk keperluan akademis dan memberikan **Hak Bebas Royalti Non Eksklusif** (*Non-exclusive Royalty-free Right*) serta bersedia Tugas Akhirnya dipublikasikan secara online dan dapat diakses secara lengkap (*full access*).

Dengan Hak Bebas Royalti Non Eksklusif ini Perpustakaan Universitas Kristen Duta Wacana berhak menyimpan, mengalih media/formatkan, mengelola dalam bentuk *database*, merawat, dan mempublikasikan Tugas Akhir saya selama tetap mencantumkan nama saya sebagai penulis/pencipta dan sebagai pemilik Hak Cipta. Demikian pernyataan ini saya buat dengan sebenar-benarnya.

Yogyakarta, 20 Mei 2025

Yang menyatakan,

(71210689 – GIAN PRADIPTA GUNAWAN)

Karya sederhana ini dipersembahkan
kepada Tuhan, Keluarga Tercinta,
dan Kedua Orang Tua

*Dia juga memelihara aku, sehingga tidak sehelai rambut pun jatuh
dari kepalaku di luar kehendak Bapa yang ada di sorga*

Katekismus Heidelberg, 1563

Perjalanan ribuan mil dimulai dari langkah satu mil (Pepatah Kuno)

KATA PENGANTAR

Segala puji dan syukur kepada Tuhan yang maha kasih, karena atas segala rahmat, bimbingan, dan bantuan-Nya maka akhirnya Tugas Akhir Skripsi dengan judul EVALUASI GAUSSIAN BLUR DALAM MEREDUKSI *NOISE* CITRA WAJAH PADA SISTEM PRESENSI MULTIFAKTOR ini telah selesai disusun.

Penulis memperoleh banyak bantuan dari kerja sama baik secara moral maupun spiritual dalam penulisan Tugas Akhir ini, untuk itu tak lupa penulis ucapkan terima kasih yang sebesar-besarnya kepada:

1. Tuhan yang maha kasih,
2. Orang tua yang selama ini telah sabar membimbing dan mendoakan penulis tanpa kenal untuk selama-lamanya,
3. I Kadek Dendy Senapatha. S. T., M. Eng. selaku Dosen Pembimbing 1,
4. Budi Susanto, S.Kom.,M.T, selaku Dosen Pembimbing 2,
5. Kakak saya, Ivan Pradipta Gunawan, yang sudah membantu penulisan laporan ini,
6. Teman – teman seperjuangan saya, Stefanus Yandi Prakosa, Cahaya Sampebua, dan Michael Goland.
7. Kawan – kawan PPLK

Laporan proposal/tugas akhir ini tentunya tidak lepas dari segala kekurangan dan kelemahan, untuk itu segala kritikan dan saran yang bersifat membangun guna kesempurnaan skripsi ini sangat diharapkan. Semoga proposal/tugas akhir ini dapat bermanfaat bagi pembaca semua dan lebih khusus lagi bagi pengembangan ilmu komputer dan teknologi informasi.

Yogyakarta, 20 Mei 2025

GIAN PRADIPTA GUNAWAN

DAFTAR ISI

| | |
|---|----|
| PERNYATAAN KEASLIAN TUGAS AKHIR..... | 3 |
| HALAMAN PERSETUJUAN..... | 4 |
| HALAMAN PENGESAHAN..... | 5 |
| HALAMAN PERNYATAAN PERSETUJUAN PUBLIKASI TUGAS AKHIR UNTUK KEPENTINGAN AKADEMIS SECARA ONLINE..... | 6 |
| KATA PENGANTAR | 9 |
| DAFTAR ISI..... | 10 |
| DAFTAR TABEL..... | 12 |
| DAFTAR GAMBAR | 13 |
| INTISARI..... | 16 |
| ABSTRACT..... | 17 |
| PENDAHULUAN | 17 |
| 1.1. Latar Belakang Masalah..... | 17 |
| 1.2. Perumusan Masalah | 20 |
| 1.3. Batasan Masalah..... | 20 |
| 1.4. Tujuan Penelitian | 20 |
| 1.5. Manfaat Penelitian | 20 |
| 1.6. Sistematika Penulisan | 21 |
| TINJAUAN PUSTAKA DAN DASAR TEORI | 22 |
| 2.1 Tinjauan Pustaka | 22 |
| 2.2 Landasan Teori..... | 24 |
| METODOLOGI PENELITIAN..... | 34 |
| 3.1. Alur Penelitian | 34 |

| | |
|--|-----|
| IMPLEMENTASI DAN PEMBAHASAN..... | 40 |
| 4.1. Pengumpulan Data | 40 |
| 4.2. Penambahan <i>Noise</i> Aditif..... | 41 |
| 4.3. Pengujian..... | 43 |
| 4.4. Evaluasi Greyscale | 51 |
| 4.5. Evaluasi RGB..... | 68 |
| 4.6. Pembahasan..... | 85 |
| KESIMPULAN DAN SARAN..... | 87 |
| 5.1. Kesimpulan | 87 |
| 5.2. Saran..... | 88 |
| DAFTAR PUSTAKA | 89 |
| LAMPIRAN A | 92 |
| LAMPIRAN B | 109 |
| LAMPIRAN C | 110 |

DAFTAR TABEL

| | |
|---|----|
| Tabel 4. 1 Tabel PSNR dan SSIM untuk denosing noise Gaussian Greyscale..... | 51 |
| Tabel 4. 2 Tabel PSNR dan SSIM untuk denoising Poisson Greyscale | 58 |
| Tabel 4. 3 Tabel PSNR dan SSIM untuk denoising gabungan noise Poisson dan Gaussian Greyscale | 61 |
| Tabel 4. 4 Tabel PSNR dan SSIM denoising noise Gaussian RGB..... | 68 |
| Tabel 4. 5 Tabel PSNR dan SSIM denoising noise Poisson RGB..... | 75 |
| Tabel 4. 6 Tabel PSNR dan SSIM denoising noise Poisson dan Gaussian RGB . | 77 |

DAFTAR GAMBAR

| | |
|--|----|
| Gambar 2. 1 Visualisasi Convolution 3 channel (RGB) | 28 |
| Gambar 2. 2 Visualisasi Convolution Greyscale | 29 |
| Gambar 3. 1 Diagram Alir Metodologi Penelitian..... | 34 |
| Gambar 4. 1 Diagram batang perbandingan nilai PSNR dalam mereduksi noise Gaussian rendah berdasarkan nilai sigma dan ukuran kernel | 53 |
| Gambar 4. 2 Diagram batang perbandingan nilai SSIM dalam mereduksi noise Gaussian rendah berdasarkan nilai sigma dan ukuran kernel | 53 |
| Gambar 4. 3 Diagram batang perbandingan nilai PSNR dalam mereduksi noise Gaussian sedang berdasarkan nilai sigma dan ukuran kernel | 55 |
| Gambar 4. 4 Diagram batang perbandingan nilai SSIM dalam mereduksi noise Gaussian sedang berdasarkan nilai sigma dan ukuran kernel | 55 |
| Gambar 4. 5 Diagram batang perbandingan nilai PSNR dalam mereduksi noise Gaussian tinggi berdasarkan nilai sigma dan ukuran kernel | 56 |
| Gambar 4. 6 Diagram batang perbandingan nilai PSNR dalam mereduksi noise Gaussian tinggi berdasarkan nilai sigma dan ukuran kernel | 57 |
| Gambar 4. 7 Diagram batang perbandingan nilai PSNR dalam mereduksi noise Poisson berdasarkan nilai sigma dan ukuran kernel | 59 |
| Gambar 4. 8 Diagram batang perbandingan nilai SSIM dalam mereduksi noise Poisson berdasarkan nilai sigma dan ukuran kernel | 59 |
| Gambar 4. 9 Diagram batang perbandingan nilai PSNR dalam mereduksi noise Poisson dan Gaussian rendah berdasarkan nilai sigma dan ukuran kernel | 63 |
| Gambar 4. 10 Diagram batang perbandingan nilai SSIM dalam mereduksi noise Poisson dan Gaussian rendah berdasarkan nilai sigma dan ukuran kernel | 63 |
| Gambar 4. 11 Diagram batang perbandingan nilai PSNR dalam mereduksi noise Poisson dan Gaussian sedang berdasarkan nilai sigma dan ukuran kernel | 65 |

| | |
|---|----|
| Gambar 4. 12 Diagram batang perbandingan nilai SSIM dalam mereduksi noise Poisson dan Gaussian sedang berdasarkan nilai sigma dan ukuran kernel..... | 65 |
| Gambar 4. 13 Diagram batang perbandingan nilai PSNR dalam mereduksi noise Poisson dan Gaussian tinggi berdasarkan nilai sigma dan ukuran kernel..... | 66 |
| Gambar 4. 14 Diagram batang perbandingan nilai SSIM dalam mereduksi noise Poisson dan Gaussian tinggi berdasarkan nilai sigma dan ukuran kernel..... | 67 |
| Gambar 4. 15 Diagram batang perbandingan nilai PSNR dalam mereduksi noise Gaussian rendah berdasarkan nilai sigma dan ukuran kernel (RGB) | 70 |
| Gambar 4. 16 Diagram batang perbandingan nilai SSIM dalam mereduksi noise Gaussian rendah berdasarkan nilai sigma dan ukuran kernel RGB | 70 |
| Gambar 4. 17 Diagram batang perbandingan nilai PSNR dalam mereduksi noise Gaussian sedang berdasarkan nilai sigma dan ukuran kernel (RGB) | 71 |
| Gambar 4. 18 Diagram batang perbandingan nilai SSIM dalam mereduksi noise Gaussian sedang berdasarkan nilai sigma dan ukuran kernel (RGB) | 72 |
| Gambar 4. 19 Diagram batang perbandingan nilai PSNR dalam mereduksi noise Gaussian tinggi berdasarkan nilai sigma dan ukuran kernel (RGB)..... | 73 |
| Gambar 4. 20 Diagram batang perbandingan nilai SSIM dalam mereduksi noise Gaussian tinggi berdasarkan nilai sigma dan ukuran kernel (RGB)..... | 74 |
| Gambar 4. 21 Diagram batang perbandingan nilai PSNR dalam mereduksi noise Poisson berdasarkan nilai sigma dan ukuran kernel (RGB)..... | 76 |
| Gambar 4. 22 Diagram batang perbandingan nilai SSIM dalam mereduksi noise Poisson berdasarkan nilai sigma dan ukuran kernel (RGB)..... | 76 |
| Gambar 4. 23 Diagram batang perbandingan nilai PSNR dalam mereduksi noise Gaussian dan Poisson rendah berdasarkan nilai sigma dan ukuran kernel (RGB) | 79 |
| Gambar 4. 24 Diagram batang perbandingan nilai SSIM dalam mereduksi noise Gaussian dan Poisson rendah berdasarkan nilai sigma dan ukuran kernel (RGB) | 80 |
| Gambar 4. 25 Diagram batang perbandingan nilai PSNR dalam mereduksi noise Gaussian dan Poisson sedang berdasarkan nilai sigma dan ukuran kernel (RGB) | 81 |
| Gambar 4. 26 Diagram batang perbandingan nilai SSIM dalam mereduksi noise Gaussian dan Poisson sedang berdasarkan nilai sigma dan ukuran kernel (RGB) | 82 |

Gambar 4. 27 Diagram batang perbandingan nilai PSNR dalam mereduksi noise Gaussian dan Poisson tinggi berdasarkan nilai sigma dan ukuran kernel (RGB). 83

Gambar 4. 28 Diagram batang perbandingan nilai SSIM dalam mereduksi noise Gaussian dan Poisson tinggi berdasarkan nilai sigma dan ukuran kernel (RGB). 84

INTISARI

EVALUASI GAUSSIAN BLUR DALAM MEREDUKSI *NOISE* CITRA WAJAH PADA SISTEM PRESENSI MULTIFAKTOR

Oleh

GIAN PRADIPTA GUNAWAN

71210689

Penelitian ini bertujuan untuk mengevaluasi efektivitas metode Gaussian Blur dalam mereduksi *noise* pada citra digital, khususnya jenis *noise* Gaussian dan Poisson, yang umum dijumpai pada citra hasil tangkapan kamera ponsel. Studi ini merupakan pengembangan dari Sistem Presensi Multifaktor berbasis Android, di mana kualitas citra wajah sangat mempengaruhi kinerja sistem pengenalan wajah, termasuk tahap pendeteksian keburaman yang sensitif terhadap *noise*. Gaussian Blur digunakan sebagai metode penyaringan low-pass untuk meredam komponen frekuensi tinggi, seperti *noise* acak, tanpa menghilangkan detail penting dalam citra. Evaluasi dilakukan menggunakan dua metrik, yaitu Peak Signal-to-Noise Ratio (PSNR) dan Structural Similarity Index Measure (SSIM), guna menilai kualitas citra hasil pemrosesan baik dari segi kuantitatif maupun persepsi visual. Berdasarkan pengujian, konfigurasi Gaussian Blur dengan kernel 9×9 dan sigma 1.5 memberikan hasil terbaik dengan nilai PSNR di atas 30 dB dan SSIM di atas 0.9 untuk tingkat *noise* rendah hingga sedang. Hasil ini menunjukkan bahwa Gaussian Blur efektif dan layak diterapkan dalam sistem yang dikembangkan untuk meningkatkan kualitas citra melalui pereduksian *noise*.

Kata kunci: Gaussian Blur, *noise* citra, PSNR, SSIM, pemrosesan citra digital

ABSTRACT

EVALUATION OF GAUSSIAN BLUR IN IMAGE DENOISING IN MULTIFACTOR ATTENDANCE SYSTEMS

By

GIAN PRADIPTA GUNAWAN

7121068

This study aims to evaluate the effectiveness of the Gaussian Blur method in reducing *noise* in digital images, particularly Gaussian and Poisson *noise*, which are commonly found in images captured by mobile phone cameras. The research is a development of the existing Multi-Factor Attendance System Android application, where image quality significantly affects the performance of the facial recognition system, including the blur detection stage that is highly sensitive to *noise*. Gaussian Blur is employed as a low-pass filtering technique to suppress high-frequency components such as random *noise* while preserving important image details. The evaluation was conducted using two metrics: Peak Signal-to-Noise Ratio (PSNR) and Structural Similarity Index Measure (SSIM), to assess both quantitative quality and visual perception of the processed images. Experimental results show that a Gaussian Blur configuration with a 9×9 kernel and a sigma value of 1.5 yields the best performance, achieving PSNR values above 30 dB and SSIM values above 0.9 for low to moderate *noise* levels. These findings indicate that Gaussian Blur is effective and suitable for application in the developed system to enhance image quality through *noise* reduction.

Keywords: Gaussian Blur, image *noise*, PSNR, SSIM, digital image processing.

BAB I

PENDAHULUAN

1.1. Latar Belakang Masalah

Penelitian ini merupakan pengembangan dari aplikasi yang telah dibuat dalam penelitian sebelumnya oleh (Kornelius, 2024), yaitu Sistem Presensi Multifaktor, sebuah aplikasi Android yang memanfaatkan lokasi perangkat, QR Code, dan pengenalan wajah untuk melakukan presensi. Dalam proses pengenalan wajah, citra yang ditangkap melalui kamera akan diproses melalui beberapa tahap, seperti pendeteksian tepi dan keburaman. Penelitian oleh (Senapatha & Tamtama, 2023) juga telah mengembangkan sistem deteksi wajah palsu (*anti-spoofing*) untuk meningkatkan keamanan. Namun, kualitas citra wajah yang digunakan sangat bergantung pada kondisi lingkungan dan perangkat, yang dapat memunculkan gangguan berupa *noise*, seperti *noise* Gaussian dan Poisson. Kehadiran *noise* ini dapat mengganggu akurasi pengenalan wajah, sehingga berdampak pada keandalan sistem presensi secara keseluruhan.

Permasalahan *noise* pada citra menjadi semakin penting untuk diperhatikan dalam konteks ini, karena secara langsung mempengaruhi kualitas dan akurasi hasil analisis citra yang dilakukan oleh sistem. *Noise* muncul sebagai gangguan acak pada nilai piksel yang dapat disebabkan oleh berbagai faktor, seperti kondisi pencahayaan yang buruk, gangguan elektronik pada sensor kamera, atau kesalahan transmisi data. Akibatnya, citra dapat menjadi buram, berbintik, atau kehilangan detail penting, yang menyulitkan proses-proses lanjutan seperti segmentasi, deteksi tepi, maupun pengenalan pola (Siregar et al., 2024). Oleh karena itu, penanganan *noise* menjadi aspek krusial dalam meningkatkan keandalan sistem presensi berbasis pengenalan wajah. Contoh permasalahan yang dapat ditimbulkan oleh *noise* adalah pada proses *anti-spoofing* dan pendeteksian keburaman.

Noise dapat menyebabkan terganggunya proses *anti-spoofing*, yaitu upaya sistem untuk membedakan antara wajah asli dan wajah tiruan, seperti foto, video, atau topeng. Kehadiran *noise* dapat menutupi atau mengubah karakteristik mikrotekstur dan pola detail pada wajah yang biasanya menjadi indikator utama dalam mendeteksi serangan spoofing. Berdasarkan penelitian yang dilakukan oleh (Cai et al., 2024), akurasi sistem *anti-spoofing* dapat menurun hingga 8% ketika citra wajah yang dianalisis mengandung *noise* dalam tingkat tertentu. Penurunan ini menunjukkan bahwa sistem menjadi lebih rentan terhadap serangan pemalsuan, sehingga memperlemah aspek keamanan dari sistem presensi berbasis pengenalan wajah secara keseluruhan.

Noise juga berpengaruh pada metode pendeteksi keburaman pada aplikasi ini. Salah satu kelemahan pada metode pendeteksi keburaman yang digunakan oleh aplikasi ini adalah kepekaannya yang tinggi terhadap *noise*. Saat ini, aplikasi menggunakan metode berbasis Laplacian Filtering, yaitu metode yang memanfaatkan operator turunan kedua untuk menonjolkan tepi pada citra. Berdasarkan buku *Digital Image Processing* (Gonzalez & Woods, 2018), operator turunan kedua sangat sensitif terhadap perubahan intensitas nilai piksel antara satu piksel dengan yang lainnya. Contohnya, dua buah piksel yang bertetangga, yang satu nilainya 0 dan yang lainnya bernilai 150. Adanya transisi nilai dari satu piksel ke piksel yang lain ini menyebabkan operator ini akan menonjolkan daerah ini sebagai tepi. Permasalahan muncul ketika *noise* muncul pada citra. *Noise* muncul sebagai fluktuasi acak pada nilai intensitas piksel yang tidak mewakili struktur nyata dalam gambar. Akibatnya, operator ini dapat salah menonjolkan *noise* sebagai tepi (Ha & Shin, 2021).

Dalam pengolahan citra digital, khususnya pada gambar yang diambil menggunakan kamera ponsel, dua jenis *noise* yang paling umum dijumpai adalah *noise* Gaussian dan *noise* Poisson (Igual, 2019). *Noise* Gaussian biasanya muncul akibat gangguan acak dari sensor kamera, terutama dalam kondisi pencahayaan rendah atau saat ISO tinggi digunakan. *Noise* ini bersifat aditif dan tersebar secara merata, sehingga mempengaruhi seluruh piksel dalam citra dengan intensitas yang

bervariasi secara acak mengikuti distribusi normal. Sementara itu, *noise* Poisson (juga dikenal sebagai *noise* foton) berkaitan erat dengan sifat kuantum cahaya yang diterima oleh sensor. Jenis *noise* ini bersifat bergantung pada sinyal, artinya tingkat gangguannya meningkat pada area gambar yang lebih terang. Karena kedua jenis *noise* ini berasal dari sifat fisik proses penangkapan gambar, maka pemahaman dan penanganannya sangat penting dalam proses peningkatan kualitas citra digital dari kamera ponsel.

Salah satu pendekatan yang umum digunakan untuk mereduksi *noise* dalam pemrosesan citra digital adalah dengan menggunakan metode Gaussian Blur. Gaussian Blur merupakan teknik penyaringan low-pass yang efektif dalam mengurangi komponen frekuensi tinggi, seperti *noise* acak, dengan cara menghaluskan citra melalui konvolusi dengan kernel berbentuk distribusi Gaussian. Hal ini memungkinkan perataan berbobot terhadap nilai-nilai piksel di sekitar suatu titik, sehingga fluktuasi acak yang disebabkan oleh *noise* dapat diminimalkan tanpa mengorbankan detail penting dalam citra (Marr & Hildreth, 1980).

Oleh karena itu, tujuan dari penelitian ini adalah untuk mengevaluasi performa Gaussian Blur dalam mereduksi *noise* pada citra digital, khususnya Gaussian *noise* dan Poisson *noise*, dengan menggunakan metrik Peak Signal-to-Noise Ratio (PSNR) dan Structural Similarity Index Measure (SSIM) sebagai alat ukur. PSNR dipilih karena mampu memberikan gambaran kuantitatif mengenai tingkat kemiripan antara citra hasil pemrosesan dengan citra aslinya dalam aspek intensitas piksel secara keseluruhan. Sementara itu, SSIM digunakan untuk mengukur kemiripan struktural antara dua citra, termasuk informasi luminansi, kontras, dan struktur lokal, yang lebih mencerminkan persepsi visual manusia. Kombinasi kedua metrik ini memungkinkan penilaian yang lebih menyeluruh terhadap efektivitas Gaussian Blur, baik dalam mempertahankan detail citra maupun dalam mereduksi *noise*. Evaluasi ini bertujuan untuk menentukan sejauh mana Gaussian Blur cocok diterapkan dalam aplikasi yang sedang dikembangkan, baik dari segi efisiensi pengurangan *noise* maupun kualitas visual hasil akhirnya.

1.2. Perumusan Masalah

Berdasarkan latar belakang yang sudah dijabarkan, rumusan masalah yang dapat disimpulkan adalah, Bagaimana efektivitas metode Gaussian Blur dalam mereduksi *noise* Gaussian dan Poisson pada citra wajah digital untuk meningkatkan kinerja deteksi *anti-spoofing* dan pendeteksi keburaman dalam sistem presensi multifaktor berbasis pengenalan wajah?

1.3. Batasan Masalah

Penelitian memiliki batasan masalah sebagai berikut:

1. Citra yang dipakai diambil dari Labeled Faces in the Wild (LFW)
2. Jenis *noise* yang diteliti adalah *noise Gaussian* dan Poisson.
3. Pengujian dilakukan dengan menggunakan teknik Peak Signal-to-Noise Ratio (PSNR) dan Structural Similarity Index Measure (SSIM)
4. Pengujian dilakukan dengan memanipulasi ukuran kernel dan nilai sigma pada Gaussian Blur.
5. Pengujian dilakukan dengan penambahan *noise* aditif terkontrol pada citra wajah yang bersih.

1.4. Tujuan Penelitian

Tujuan dari penelitian ini adalah untuk mengevaluasi keefektifan Gaussian Blur dalam mereduksi *noise* Gaussian Poisson.

1.5. Manfaat Penelitian

Manfaat dari penelitian ini adalah,

1. Meningkatkan Kualitas Citra sebelum diolah oleh Sistem Presensi Multifaktor

2. Mengetahui konfigurasi ukuran kernel dan nilai sigma Gaussian Blur yang optimal dalam pereduksian *noise* Gaussian dan Poisson untuk diterapkan pada Sistem Presensi Multifaktor.

1.6. Sistematika Penulisan

Laporan/Proposal tugas akhir ini disusun dengan sistematika bagian pertama, terdiri dari empat bab:

- Bab I Pendahuluan
Bab ini berisi latar belakang masalah, perumusan masalah, batasan masalah. Tujuan penelitian, manfaat penelitian, metode penelitian, dan sistematika penelitian. Bab ini menjelaskan persoalan yang dihadapi pada sistem pengenalan wajah, yaitu gambar masukan yang memiliki *noise*, dan solusi yang bisa dipakai, yaitu Gaussian Filter dalam melakukan *denoising*.
- Bab II Tinjauan Pustaka dan Landasan Teori
Bab II berisi tinjauan pustaka dan landasan teori. Pada bab ini dijabarkan hasil – hasil dari penelitian sebelumnya yang dinilai relevan sehingga dapat menjelaskan lebih lanjut tentang pentingnya penelitian ini. Bab ini juga akan menjelaskan mengenai Gaussian Blur, Laplacian of Gaussian, Convolution Filtering, *noise*, *noise* Gaussian, *noise* Poisson, PSNR, dan SSIM.
- Bab III Metodologi Penelitian
Bab ini membahas proses penelitian dilakukan, dimulai dari pengumpulan data, penambahan *noise* aditif, pengujian data, dan evaluasi hasil.
- Bab IV Implementasi dan Pembahasan
Pembahasan mengenai cara penerapan penelitian, library yang digunakan, algoritma yang dipakai, dan pemaparan hasil penelitian.
- Bab V Kesimpulan dan Saran
Rangkuman dari hasil pembahasan serta saran untuk penelitian yang selanjutnya.

BAB II

TINJAUAN PUSTAKA DAN DASAR TEORI

2.1 Tinjauan Pustaka

Penelitian yang dilakukan oleh (Dharavath, 2014) menunjukkan bahwa tahap *preprocessing* citra wajah memiliki dampak yang signifikan terhadap peningkatan akurasi dalam pengenalan wajah. Dalam penelitian tersebut, mereka menguji berbagai teknik *preprocessing* yang dilakukan sebelum tahap ekstraksi fitur. Teknik-teknik tersebut meliputi *face detection and cropping*, pengubahan ukuran (*resizing*), normalisasi histogram, dan *denoising*. Hasil pengujian menunjukkan bahwa penerapan tahap *preprocessing* menghasilkan akurasi pengenalan wajah yang lebih tinggi dibandingkan dengan pengolahan tanpa *preprocessing*.

Berdasarkan penelitian oleh (B & K, 2016), proses *denoising* merupakan tahap penting yang perlu dilakukan karena citra yang ditangkap oleh sensor hampir selalu terkontaminasi oleh *noise*. Hal ini disebabkan oleh berbagai faktor, seperti ketidaksempurnaan perangkat keras, gangguan selama proses akuisisi data, serta pengaruh lingkungan sekitar. Oleh karena itu, sebelum citra dianalisis lebih lanjut, perlu dilakukan proses *denoising* untuk meningkatkan kualitas citra. Penelitian ini juga menekankan pentingnya pemilihan metode *denoising* yang tepat, disesuaikan dengan karakteristik jenis *noise* yang ada pada citra.

Dalam penelitian yang dilakukan oleh (Peltoketo, 2014a), ditemukan bahwa pengaturan ISO memiliki pengaruh signifikan terhadap tingkat keberadaan *noise* pada citra digital. Penelitian ini melibatkan pengujian terhadap 20 kamera ponsel yang berbeda, dan hasilnya menunjukkan bahwa semakin tinggi nilai ISO yang digunakan, maka tingkat *noise* yang muncul juga akan semakin meningkat. Temuan ini menegaskan bahwa kondisi pencahayaan rendah yang memerlukan ISO tinggi dapat memperburuk kualitas citra akibat peningkatan *noise*.

Menurut (Igual, 2019), jenis *noise* yang paling umum ditemukan pada citra hasil tangkapan kamera ponsel adalah *noise* Gaussian dan *noise* Poisson. Kedua jenis *noise* ini berkaitan erat dengan proses pengambilan gambar menggunakan sensor digital. *Noise* Gaussian biasanya muncul akibat gangguan acak dari sensor, sedangkan *noise* Poisson berkaitan dengan fluktuasi kuantum dalam proses penangkapan cahaya. Penelitian ini juga menegaskan bahwa peningkatan nilai ISO, yang umumnya digunakan dalam kondisi pencahayaan rendah, secara signifikan memperbesar kemungkinan munculnya kedua jenis *noise* ini pada citra.

Penelitian yang dilakukan oleh (Nasution, 2021) menemukan bahwa metode Gaussian Blur efektif dalam menghilangkan *noise* pada citra USG. Penelitian tersebut menguji dampak dari tiga metode filter: Mean Filter, Gaussian Filter, dan Median Filter. Hasil penelitian menunjukkan bahwa citra USG yang diolah menggunakan Gaussian Blur menghasilkan citra dengan nilai Peak Signal-to-Noise Ratio (PSNR) di atas 30, yang menunjukkan efektivitas metode ini dalam melakukan *denoising* dan meningkatkan kualitas citra.

Pada penelitian lain oleh (Bharati et al., 2021) menyatakan bahwa Gaussian Blur efektif untuk menghilangkan *noise* Gaussian. Selain dari jenis *noise* tersebut, metode ini juga dapat digunakan untuk mereduksi jenis *noise* lainnya, seperti *noise* Salt-and-Pepper, Speckle, dan Uniform, meskipun performanya tidak sebagus dalam pengurangan *noise* Gaussian.

Salah satu kelebihan dari Gaussian Blur adalah kompleksitas algoritmanya yang rendah, sehingga memungkinkan pemrosesan citra dilakukan dengan cepat dan efisien (Nainggolan & Khair, 2020). Selain itu, Gaussian Blur juga cukup efektif dalam mereduksi *noise* Gaussian, yaitu jenis *noise* yang memiliki distribusi normal dan sering muncul pada citra digital akibat gangguan elektronik dari sensor kamera. Karena Gaussian Blur menggunakan kernel berbobot yang mengikuti distribusi Gaussian, filter ini secara alami selaras dengan karakteristik *noise* Gaussian, sehingga mampu meredam fluktuasi intensitas yang tidak diinginkan tanpa menghapus terlalu banyak informasi penting dari citra.

Dalam penelitian oleh Raghav Bansal, Gaurav Raj, dan Tanupriya Choudhury, mereka meneliti penerapan Laplacian Filter untuk pendeteksian keburaman citra. Mereka menerapkan Laplacian Filtering pada citra dan menghitung variansinya. Semakin tinggi variansinya, semakin tajam citra tersebut. Berdasarkan penelitian ini, metode ini terbukti efektif dalam mendeteksi blur pada citra. Penelitian lain oleh (Pagaduan et al., 2020) juga menunjukkan bahwa metode ini memiliki tingkat akurasi mencapai 85 persen. Meskipun begitu, Operator Laplacian bersifat sensitif terhadap *noise*. Sebagaimana dijelaskan dalam buku oleh (Gonzalez & Woods, 2018), Operator Laplacian merupakan operator turunan kedua yang bekerja dengan cara menonjolkan transisi nilai piksel pada citra. Karena *noise* kerap muncul dalam bentuk transisi acak pada nilai piksel, metode ini akan menonjolkan *noise* pada citra keluarannya.

Penelitian oleh (Aripin et al., 2020) meneliti penerapan Gaussian Blur pada citra gigi berlubang sebelum dimasukkan ke dalam metode Laplacian. Hasil penelitian mereka menunjukkan bahwa penerapan Gaussian Blur menghasilkan keluaran pendeteksi tepi yang baik dan meningkatkan akurasi ketebalan citra gigi berlubang.

2.2 Landasan Teori

2.2.1. Noise

Noise adalah gangguan pada gambar yang mempengaruhi nilai piksel - pikselnya. Faktor - faktor yang menyebabkan hal ini bisa terjadi karena lingkungan sekitar, *transmission channel*, dan lain - lain. *Noise* sangat sering terjadi dan hampir tidak bisa dihindari. Pada proses pengenalan gambar, *noise* dapat mengurangi hasil keakuratannya (Fan et al., 2019). Salah satu penyebab *noise* adalah pencahayaan yang rendah, yang memaksa kamera menaikkan nilai ISO. ISO yang tinggi berbanding lurus dengan peningkatan *noise*, sehingga dapat mempengaruhi kualitas gambar (Peltoketo, 2014b).

2.2.1.1 *Noise Gaussian*

Noise Gaussian adalah jenis gangguan acak yang sering muncul dalam citra digital dan sinyal, yang mengikuti distribusi probabilitas normal (kurva Gaussian). Karakteristik utama dari *noise Gaussian* adalah fluktuasi nilai piksel yang terjadi secara acak, dengan rata-rata nilai nol dan distribusi nilai *noise* yang simetris di sekitar nilai rata-rata tersebut. *Noise Gaussian* sering kali muncul pada citra yang diambil dalam kondisi pencahayaan rendah atau dengan pengaturan ISO tinggi, yang membuat sensor lebih sensitif terhadap gangguan acak. Dalam pemrosesan citra, *noise Gaussian* dapat merusak kualitas gambar, dan seringkali perlu diatasi dengan teknik-teknik *denoising*, seperti Gaussian Blur, untuk meningkatkan ketajaman dan kualitas visual citra (Wibowo & Susanto, 2017).

2.2.1.2 *Noise Poisson*

Noise Poisson, atau sering disebut juga sebagai *noise foton*, adalah jenis gangguan yang umum terjadi pada citra digital yang dihasilkan dari proses penangkapan cahaya, terutama pada perangkat seperti kamera ponsel. *Noise* ini berasal dari sifat kuantum cahaya, di mana jumlah foton yang mencapai sensor selama proses pencitraan bersifat acak dan mengikuti distribusi Poisson. Akibatnya, variasi jumlah foton yang diterima oleh sensor menyebabkan fluktuasi intensitas piksel pada citra yang dihasilkan. Karakteristik utama dari Poisson *noise* adalah sifatnya yang *signal-dependent*, artinya tingkat *noise* akan meningkat seiring dengan bertambahnya intensitas cahaya yang diterima — semakin terang suatu area pada citra, semakin besar kemungkinan *noise* muncul (Igual, 2019).

2.2.2. *Denoising*

Proses *denoising* adalah proses untuk menghilangkan *Noise* pada gambar sehingga diperoleh struktur gambar yang sebenarnya. Proses ini tidak bisa dengan sempurna mengembalikan gambar tanpa *noise* dikarenakan sulitnya membedakan detail nilai *piksel* yang asli dan yang bukan. Pada kenyataannya, proses ini masih menjadi tantangan dalam dunia matematika (Fan et al., 2019).

$$y = x + n \quad (2.1)$$

Secara matematis, proses *denoising* dapat digambarkan dituliskan sebagai $y = x + n$. Penjelasannya sebagai berikut:

y = adalah gambar masukan yang mengandung *noise*.

x = adalah hasil yang ingin kita cari, yaitu gambar murni tanpa *noise*.

n = adalah *noise* yang ada pada gambar.

Pada kenyataannya, solusi untuk nilai x adalah unik, artinya setiap metode *denoising* yang dipakai akan menghasilkan hasil x yang berbeda pula (Fan et al., 2019). Dengan begitu, dapat disimpulkan teknik – teknik yang ada tidak bertujuan untuk menghilangkan semua *noise* pada gambar, tetapi mengurangi sebanyak mungkin *noise* tanpa mengurangi fitur – fitur pada gambar yang asli.

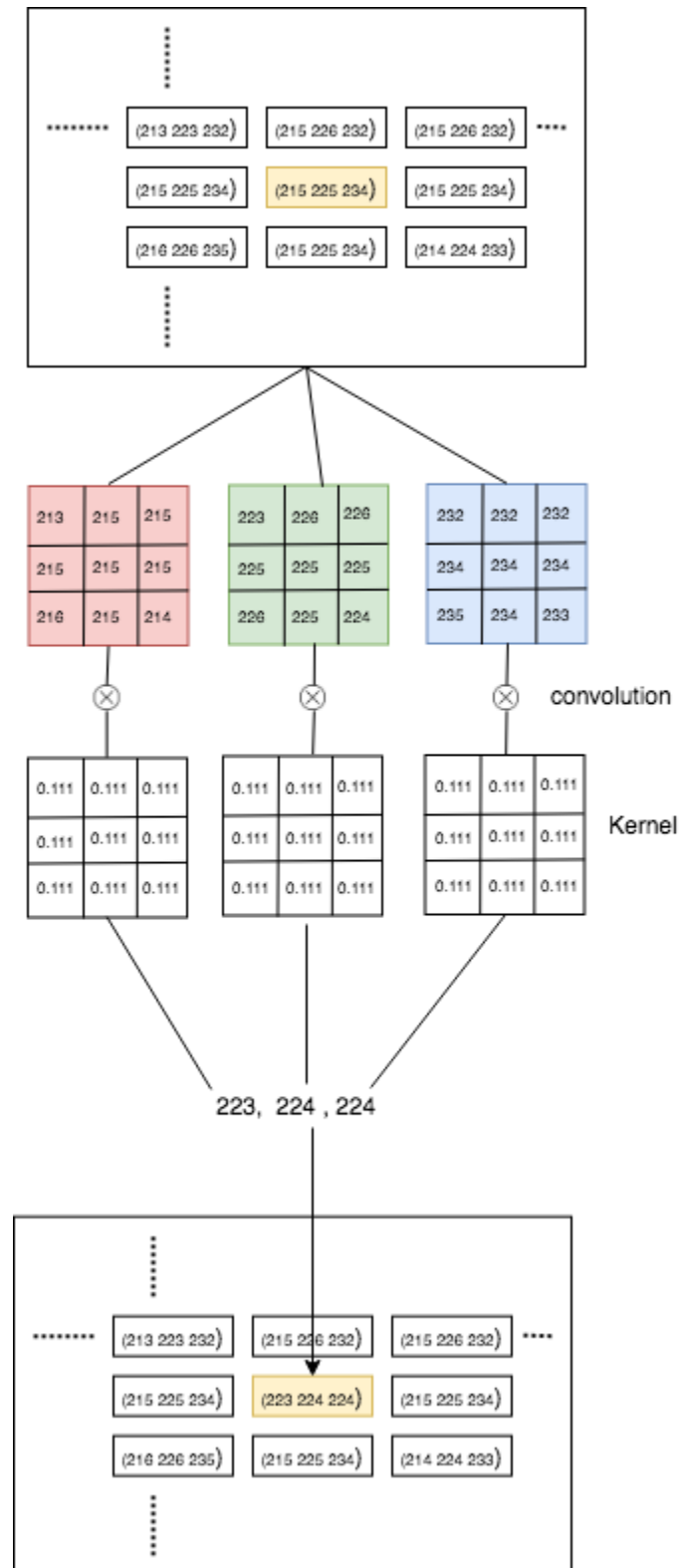
2.2.3. Convolution Filtering

Convolution Filtering adalah metode pemodifikasian citra dengan cara menyesuaikan nilai suatu piksel berdasarkan nilai-nilai piksel tetangganya. Proses ini menggunakan kernel, yaitu sebuah matriks kecil berbentuk grid (biasanya 3×3 , 5×5 , dan seterusnya) yang berisi bobot. Secara visual, kernel akan "digeser" ke seluruh posisi dalam citra, dan pada setiap posisi, ia akan mencakup piksel-piksel di sekitar piksel pusat. Piksel-piksel yang tercakup inilah yang disebut piksel tetangga. Setiap piksel tetangga akan dikalikan dengan bobot yang sesuai di dalam kernel, lalu hasil-hasil perkalian tersebut akan dijumlahkan. Hasil akhir ini kemudian menjadi nilai baru dari piksel pusat tempat kernel berada. Proses ini disebut dengan *konvolusi*. Proses ini berguna untuk penghalusan (*smoothing*), penajaman (*sharpening*), pendeteksian tepi (*edge detection*), dan lain sebagainya (Gazali et al., 2012).

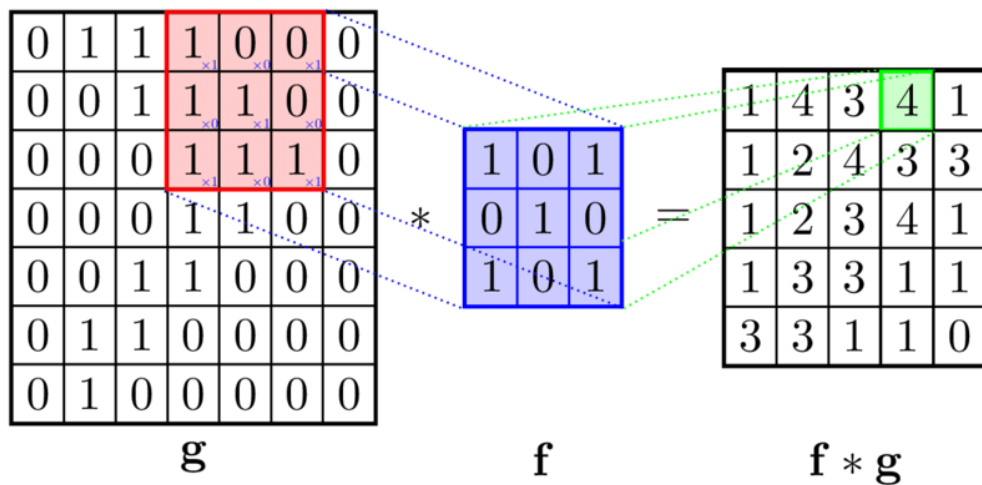
Pada citra *greyscale*, setiap piksel hanya memiliki satu nilai intensitas yang merepresentasikan tingkat kecerahan, biasanya dalam rentang 0 (hitam) hingga 255 (putih). Dalam hal ini, proses konvolusi cukup dilakukan langsung pada satu saluran, seperti pada Gambar 2.2, karena tidak ada informasi warna yang perlu diproses secara terpisah. Kernel akan diterapkan langsung pada matriks nilai intensitas, menghasilkan matriks baru dengan nilai-nilai piksel yang telah dimodifikasi.

Sementara itu, pada citra RGB, setiap piksel memiliki tiga saluran warna (Merah, Hijau, Biru). Oleh karena itu, proses konvolusi dilakukan secara terpisah untuk setiap channel warna. Artinya, kernel akan diterapkan masing-masing pada saluran merah, hijau, dan biru dari gambar, seperti pada Gambar 2.1. Setelah konvolusi selesai pada ketiga channel, hasilnya akan digabung kembali untuk membentuk citra berwarna yang telah dimodifikasi. Pendekatan ini memastikan bahwa detail warna tetap terjaga, sambil memungkinkan efek visual seperti *blur*, *sharpening*, atau *edge detection* diterapkan secara menyeluruh.

Dengan kata lain, meskipun mekanisme dasar konvolusi serupa untuk grayscale dan RGB, kompleksitas meningkat pada citra berwarna karena setiap channel harus diproses secara individual sebelum disatukan kembali menjadi satu gambar RGB yang utuh.



Gambar 2. 1 Visualisasi Convolution 3 channel (RGB)



Gambar 2. 2 Visualisasi Convolution Greyscale

2.2.4. Laplacian of Gaussian

Laplacian of Gaussian (LoG) adalah metode deteksi tepi yang menggabungkan dua langkah utama: penghalusan citra menggunakan filter Gaussian dan kemudian penerapan operator Laplacian untuk mendeteksi perubahan intensitas. Pendekatan ini pertama kali diperkenalkan oleh Marr dan Hildreth dalam paper mereka yang berjudul *Theory of Edge Detection*. Filter Gaussian digunakan untuk mengurangi *noise* dan fluktuasi kecil yang tidak penting dalam citra, sementara operator Laplacian digunakan untuk menyoroti lokasi-lokasi dengan perubahan intensitas yang tajam, yang biasanya menunjukkan keberadaan tepi. Gabungan kedua filter ini menghasilkan deteksi tepi yang lebih stabil dan efektif, terutama dalam citra alami yang memiliki banyak variasi tekstur dan pencahayaan (Marr & Hildreth, 1980).

2.2.5. Gaussian Blur

Dalam paper (Marr & Hildreth, 1980), mereka menyarankan penggunaan penghalusan citra dengan filter yang memiliki kernel yang nilainya berdasarkan distribusi Gaussian sebelum citra dilakukan pendeteksian tepi. Penggunaan filter ini digunakan untuk mengurangi fluktuasi nilai piksel yang tajam. Kelebihan yang

dimiliki filter ini adalah kemampuannya menghaluskan citra secara halus, sehingga mampu mengurangi *noise* namun tetap menjaga detail tepi pada citra. Kelebihan lainnya adalah metode ini merupakan satu-satunya metode filtering yang menghasilkan kurva berbentuk Gaussian baik di domain spasial maupun di domain frekuensi, menjadikannya stabil dan konsisten dalam berbagai pendekatan pemrosesan citra.

$$G(r) = (\frac{1}{2}\pi\sigma^2) \exp(-r^2/2\sigma^2). \quad (2.2)$$

$$G_{\sigma}(x, y) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right) \quad (2.3)$$

Rumus 2.2 dan Rumus 2.3 merupakan notasi fungsi Gaussian pada nilai - nilai kernel. Variabel x dan y merupakan koordinat kartesian pada kernel 2 dimensi, dengan bagian tengah kernel sebagai koordinat (0, 0). Fungsi Gaussian menghasilkan nilai yang lebih tinggi untuk titik-titik yang lebih dekat ke pusat dan nilai yang lebih rendah untuk titik-titik yang lebih jauh, mengikuti distribusi berbentuk lonceng. Hal ini menyebabkan bobot pada kernel lebih besar di sekitar pusat dan semakin kecil seiring bertambahnya jarak dari pusat. Parameter σ (sigma) atau simpangan baku berperan dalam mengatur lebar penyebaran distribusi; nilai σ yang lebih besar menghasilkan kernel yang lebih lebar dan lebih halus dalam efek penghalusannya, sedangkan nilai σ yang lebih kecil menghasilkan kernel yang lebih sempit dengan efek penghalusan yang lebih tajam di area lokal.

2.2.6. Metode Evaluasi

2.2.6.1. Peak Signal-to-Noise Ration (PSNR)

Peak Signal-to-Noise Ratio (PSNR) adalah metrik yang sering digunakan untuk mengukur kualitas citra atau video yang telah terkompresi atau diproses,

dengan cara membandingkan citra asli dan citra hasil pemrosesan. PSNR memberikan indikasi seberapa banyak kerugian informasi yang terjadi selama pemrosesan, misalnya saat mengurangi *noise* atau mengompresi citra (Nasution, 2021)

PSNR dihitung dengan menggunakan Mean Squared Error (MSE), yang mengukur rata-rata kuadrat perbedaan antara nilai piksel citra asli dan citra yang diproses. PSNR dihitung dengan rumus 2.1 bersama rumus 2.2:

$$MSE = \frac{1}{M \times N} \sum_{i=1}^M \sum_{j=1}^N (f_a(i, j) - f_b(i, j))^2 \quad (2.4)$$

$$PSNR = 10 \log_{10} \frac{255^2}{MSE}. \quad (2.5)$$

$f_a(i, j)$ adalah nilai piksel citra asli di posisi i, j

$f_b(i, j)$ adalah nilai piksel citra yang telah diproses

M dan N adalah ukuran citra

Interpretasi Nilai PSNR menunjukkan sejauh mana kualitas citra hasil pemrosesan dibandingkan dengan citra aslinya. Secara umum, nilai PSNR yang lebih tinggi menunjukkan kualitas yang lebih baik. Jika nilai PSNR ≥ 30 dB, citra hasil dianggap berkualitas baik, karena perbedaannya dengan citra asli sangat kecil dan sulit dikenali oleh mata manusia. Jika PSNR berada di rentang 20–30 dB, kualitas citra dapat dikategorikan sebagai sedang, di mana degradasi visual mulai terlihat meskipun masih dapat diterima dalam banyak aplikasi. Sementara itu, jika nilai PSNR < 20 dB, maka kualitas citra dianggap buruk, karena perbedaan antara citra asli dan hasil pemrosesan sangat jelas secara visual.

2.2.6.2. Structural Similarity Index Measure (SSIM)

Structural Similarity Index Measure (SSIM) adalah sebuah metrik yang digunakan untuk mengukur tingkat kemiripan antara dua citra, dengan fokus pada aspek persepsi visual manusia. Berbeda dengan metrik konvensional seperti PSNR yang hanya membandingkan perbedaan nilai intensitas piksel, SSIM mempertimbangkan struktur lokal, kontras, dan luminansi dari citra, sehingga mampu memberikan penilaian yang lebih representatif terhadap kualitas visual. SSIM memiliki nilai antara -1 hingga 1, di mana nilai 1 menunjukkan kesamaan struktural yang sempurna antara citra yang dibandingkan. Dalam konteks pemrosesan citra, SSIM sering digunakan untuk mengevaluasi hasil dari teknik kompresi, denoising, atau rekonstruksi citra, karena lebih sesuai dengan cara manusia menilai kualitas gambar. Dengan demikian, SSIM menjadi alat ukur yang sangat berguna dalam menilai sejauh mana detail dan struktur asli citra dapat dipertahankan setelah mengalami proses pemrosesan atau modifikasi (Nilsson, Jim 2020).

$$SSIM(x,y) = (l(x,y))^{\alpha} (c(x,y))^{\beta} (s(x,y))^{\gamma} \quad (2.6)$$

Rumus 2.6 merepresentasikan perhitungan SSIM.

$l(x, y)$ merepresentasikan luminasi antara citra x dan y

$c(x, y)$ merepresentasikan kontras antara citra x dan y

$s(x, y)$ merepresentasikan kemiripan struktural antara citra x dan y

α , β , dan γ mengontrol derajat kepentingan antara ketiga pembanding

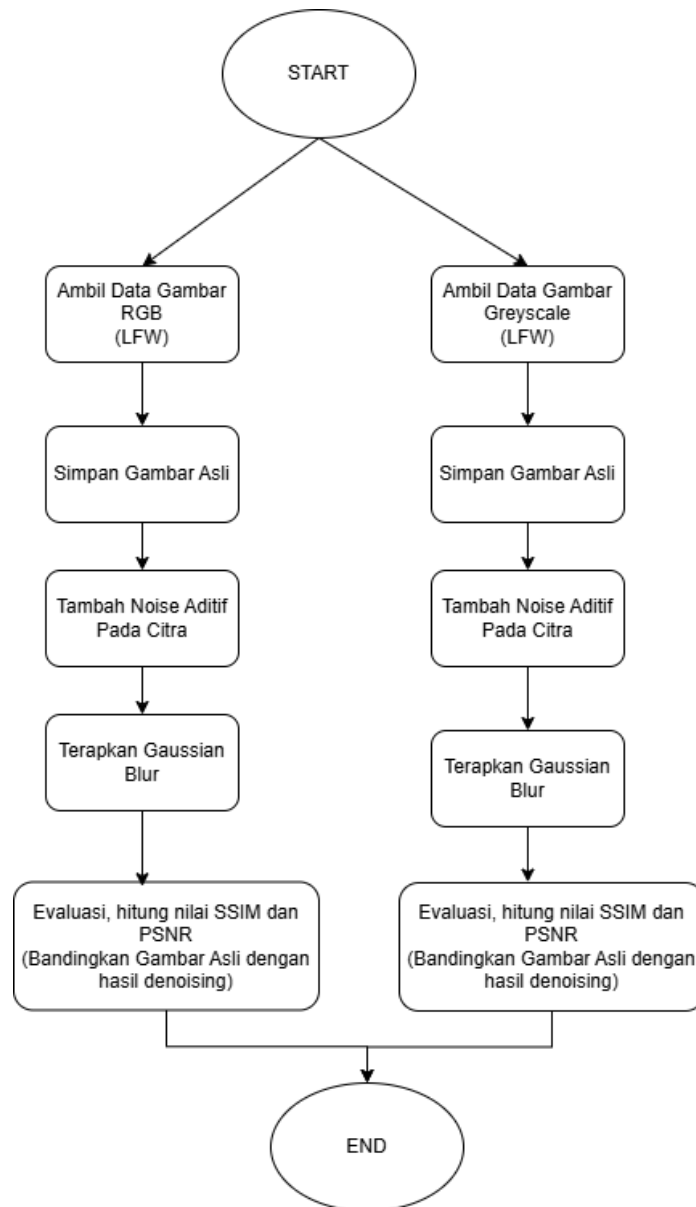
Interpretasi Nilai SSIM digunakan untuk menilai kemiripan struktural antara citra asli dan citra hasil pemrosesan. Nilai SSIM berada dalam rentang 0 hingga 1, di mana nilai yang mendekati 1 menunjukkan tingkat kesamaan struktural yang tinggi. Jika nilai SSIM berada di atas 0.9, citra hasil dianggap memiliki

kualitas sangat baik, karena struktur dan detail citra masih sangat mirip dengan citra aslinya. Nilai SSIM antara 0.6 hingga 0.9 menunjukkan kualitas sedang, di mana beberapa perbedaan mulai terlihat, namun citra masih mempertahankan struktur dasarnya. Jika nilai SSIM berada di bawah 0.6, maka kualitas citra dianggap buruk, karena banyak detail struktural penting telah hilang atau berubah signifikan dibandingkan dengan citra asli.

BAB III

METODOLOGI PENELITIAN

3.1. Alur Penelitian



Gambar 3. 1 Diagram Alir Metodologi Penelitian

Penelitian ini bertujuan untuk mengevaluasi efektivitas metode Gaussian Blur dalam mereduksi *noise* aditif pada citra wajah. Gambar 3.1 menunjukkan diagram alir metodologi yang digunakan. Penelitian dimulai dengan tahap

pengumpulan data. Data yang dipakai diambil dari dataset yang tersedia secara publik. Data yang akan dipakai adalah Labeled Faces in the Wild (LFW). Tahap selanjutnya adalah penambahan *noise* aditif terhadap gambar yang asli. Tahap berikutnya adalah pengujian data. Data – data yang sudah diberi *noise* aditif kemudian di-*denoise* menggunakan Gaussian Blur dengan konfigurasi ukuran kernel dan ukuran sigma yang berbeda-beda. Tahap evaluasi melakukan perhitungan evaluasi berdasarkan metrik PSNR: gambar asli dengan gambar hasil *denoised* akan dihitung kesamaannya.

3.1.1. Pengumpulan Data

Pada tahap pengumpulan data, penelitian ini menggunakan dataset Labeled Faces in the Wild (LFW) yang merupakan kumpulan citra wajah berlabel yang banyak digunakan dalam penelitian pengenalan wajah. Dataset ini berisi lebih dari 13.000 gambar wajah dari individu yang berbeda, diambil dari berbagai kondisi pencahayaan, pose, dan ekspresi wajah, sehingga mewakili variasi citra yang realistis. Dataset LFW dipilih karena memiliki kualitas dan kompleksitas visual yang sesuai untuk menguji efektivitas metode reduksi *noise*, khususnya Gaussian Blur. Seluruh citra dalam dataset ini akan digunakan sebagai data uji dengan menambahkan *noise* aditif secara terkontrol, guna mensimulasikan gangguan yang umum terjadi pada proses akuisisi citra menggunakan kamera ponsel. Data yang telah ditambahkan *noise* kemudian akan diproses menggunakan Gaussian Blur, dan hasilnya dievaluasi menggunakan metrik PSNR untuk mengukur efektivitas reduksi *noise*.

3.1.2. Pengujian Data

Setelah citra diperoleh dari dataset Labeled Faces in the Wild (LFW), langkah selanjutnya dalam penelitian ini adalah menambahkan *noise* Gaussian dan *noise* Poisson secara digital pada citra-citra tersebut. Penambahan *noise* bertujuan untuk mensimulasikan kondisi nyata yang kerap terjadi dalam proses akuisisi citra menggunakan kamera ponsel, seperti gangguan sensor, pencahayaan rendah, atau penggunaan ISO tinggi.

Citra yang digunakan dalam penelitian ini terdiri dari dua jenis format warna, yaitu RGB dan greyscale. Penggunaan kedua format ini bertujuan untuk mengevaluasi dan membandingkan efektivitas metode denoising pada citra berwarna dan citra hitam-putih. Format RGB mencerminkan situasi nyata dalam pengolahan citra digital secara umum, di mana informasi warna memainkan peran penting dalam persepsi visual dan identifikasi objek. Sementara itu, format greyscale menyederhanakan representasi citra hanya dalam satu kanal intensitas, yang sering digunakan dalam aplikasi pengolahan citra berbasis struktur dan tekstur.

Noise Gaussian ditambahkan dengan menggunakan distribusi normal bermean nol ($\mu = 0$) dan variansi (σ^2) yang dapat disesuaikan untuk merepresentasikan berbagai tingkat gangguan. Proses ini dilakukan dengan bantuan pustaka NumPy untuk menghasilkan distribusi acak, serta OpenCV (cv2) untuk manipulasi citra digital. Sementara itu, *noise* Poisson, yang merepresentasikan gangguan akibat fluktuasi jumlah foton yang diterima oleh sensor kamera, ditambahkan dengan memanfaatkan pustaka Scikit-Image, khususnya melalui fungsi `random_noise()` dengan mode `poisson`.

Dalam implementasinya, citra RGB dikonversi ke dalam format matriks numerik berdimensi tiga ($\text{tinggi} \times \text{lebar} \times 3$ kanal warna), sedangkan citra greyscale dikonversi menjadi matriks berdimensi dua ($\text{tinggi} \times \text{lebar}$). Untuk citra RGB, *noise* Gaussian ditambahkan secara independen ke setiap kanal (Red, Green, Blue), sehingga simulasi gangguan lebih realistis karena mempertimbangkan variasi *noise* pada tiap komponen warna. Begitu pula untuk *noise* Poisson, penambahan dilakukan per kanal warna. Sebaliknya, untuk citra greyscale, penambahan *noise* hanya dilakukan pada satu kanal intensitas, yang mempercepat proses komputasi namun tetap merepresentasikan degradasi kualitas visual yang signifikan.

Variansi *noise* Gaussian diatur secara bertahap (misalnya $\sigma^2 = 10, 20, 30$, dan seterusnya), guna menghasilkan berbagai versi citra dengan tingkat gangguan berbeda. Seluruh citra hasil modifikasi ini disimpan dan disiapkan untuk tahap pemrosesan berikutnya, yaitu penerapan filter Gaussian. Pendekatan ini

memungkinkan pelaksanaan eksperimen secara terkontrol dan berulang, sehingga hasil evaluasi menjadi lebih akurat dan dapat dipertanggungjawabkan.

Pada tahap pengujian, citra-citra yang telah diberi *noise* Gaussian maupun Poisson kemudian diproses menggunakan metode Gaussian Blur dengan berbagai konfigurasi kernel. Pengujian dilakukan dengan variasi ukuran kernel, seperti 3×3 , 5×5 , 7×7 , dan 9×9 , serta variasi nilai sigma (simpangan baku), untuk mengevaluasi pengaruh masing-masing parameter terhadap efektivitas reduksi *noise*. Implementasi filter dilakukan menggunakan pustaka OpenCV dalam bahasa pemrograman Python, melalui fungsi `cv2.GaussianBlur()`, yang memungkinkan pengaturan fleksibel terhadap parameter kernel dan sigma.

Setiap konfigurasi filter diterapkan pada citra RGB dan greyscale dengan tingkat *noise* yang berbeda, baik untuk *noise* Gaussian maupun Poisson. Untuk citra RGB, evaluasi dilakukan terhadap hasil denoising pada masing-masing kanal warna dan juga secara keseluruhan setelah penggabungan ulang kanal. Sementara itu, untuk citra greyscale, evaluasi dilakukan langsung terhadap hasil akhir karena hanya melibatkan satu kanal. Hasil pemrosesan kemudian dianalisis secara kuantitatif menggunakan metrik evaluasi seperti PSNR (Peak Signal-to-Noise Ratio) dan SSIM (Structural Similarity Index), untuk mengukur kualitas citra hasil denoising dibandingkan dengan citra asli tanpa *noise*.

Dengan membandingkan hasil dari berbagai konfigurasi terhadap kedua jenis *noise* dan dua jenis format citra (RGB dan greyscale), penelitian ini bertujuan menemukan parameter optimal yang mampu secara efektif mereduksi *noise* tanpa menghilangkan detail penting pada citra wajah. Selain itu, analisis juga ditujukan untuk mengidentifikasi perbedaan perilaku metode Gaussian Blur dalam menangani *noise* pada citra berwarna dan citra hitam-putih, yang dapat memberikan wawasan penting dalam pengembangan algoritma denoising yang lebih adaptif dan efisien.

3.1.3. Evaluasi

Setelah penerapan Gaussian Blur dengan berbagai konfigurasi parameter, langkah berikutnya dalam penelitian ini adalah melakukan evaluasi terhadap hasil

pemrosesan menggunakan dua metrik kuantitatif utama, yaitu PSNR dan SSIM. Evaluasi ini bertujuan untuk menilai sejauh mana metode Gaussian Blur mampu mereduksi *noise* tanpa mengorbankan kualitas visual citra asli.

PSNR digunakan untuk mengukur tingkat kemiripan piksel antara citra asli dan citra hasil pemrosesan. Semakin tinggi nilai PSNR, semakin kecil perbedaan antara keduanya, yang menunjukkan bahwa kualitas citra telah berhasil dipertahankan. Metrik ini sangat sensitif terhadap gangguan global, sehingga efektif untuk mengevaluasi *noise* Gaussian yang tersebar merata.

Sementara itu, SSIM digunakan untuk menilai kemiripan struktur, kontras, dan luminansi antara citra asli dan hasil pemrosesan. Tidak seperti PSNR yang hanya melihat kesamaan numerik antar piksel, SSIM mempertimbangkan aspek persepsi visual manusia, menjadikannya metrik yang lebih representatif dalam menilai kualitas citra secara subjektif. Oleh karena itu, kombinasi PSNR dan SSIM memberikan evaluasi yang lebih menyeluruh, baik secara objektif maupun subjektif.

Dalam konteks penelitian ini, evaluasi dilakukan terhadap dua jenis format citra, yaitu RGB dan greyscale. Untuk citra greyscale, perhitungan PSNR dan SSIM dilakukan secara langsung terhadap satu kanal intensitas. Namun, untuk citra RGB, evaluasi dilakukan secara per-kanal (Red, Green, Blue), kemudian dirata-ratakan untuk memperoleh nilai agregat yang merepresentasikan kualitas keseluruhan citra berwarna. Pendekatan ini bertujuan untuk menangkap degradasi yang mungkin terjadi secara berbeda pada setiap kanal warna akibat penambahan *noise* dan penerapan filter.

Proses perhitungan metrik dilakukan menggunakan pustaka Scikit-Image (skimage) dalam bahasa pemrograman Python. Nilai PSNR dihitung dengan fungsi `peak_signal_noise_ratio()`, sedangkan SSIM dihitung menggunakan fungsi `structural_similarity()`, yang juga mendukung pengolahan citra multikanal seperti RGB. Untuk RGB, argumen tambahan seperti `multichannel=True` atau

`channel_axis=-1` digunakan untuk memastikan fungsi memproses kanal warna secara tepat.

Setiap hasil citra dari kombinasi jenis *noise* (Gaussian atau Poisson), tingkat gangguan, dan konfigurasi Gaussian Blur (ukuran kernel dan sigma), dievaluasi menggunakan kedua metrik tersebut. Hasil evaluasi kemudian dibandingkan dan dianalisis guna mengetahui konfigurasi mana yang paling efektif dalam mereduksi *noise*, baik pada citra greyscale maupun RGB. Analisis ini mencakup perbandingan nilai PSNR dan SSIM antar variasi kernel (misalnya 3×3 , 5×5 , 7×7 , 9×9) dan sigma, serta pengaruh format citra terhadap hasil denoising.

Melalui pendekatan ini, penelitian tidak hanya menghasilkan temuan empiris mengenai parameter optimal Gaussian Blur, tetapi juga memberikan wawasan tentang perbedaan performa filter terhadap format citra yang berbeda. Dengan demikian, hasil penelitian diharapkan dapat menjadi acuan dalam pengembangan metode denoising yang lebih adaptif terhadap berbagai kondisi citra digital di dunia nyata..

BAB IV

IMPLEMENTASI DAN PEMBAHASAN

4.1. Pengumpulan Data

Pada penelitian ini, dataset yang digunakan adalah Labeled Faces in the Wild (LFW), yang merupakan kumpulan gambar wajah yang telah diberi label dan sering digunakan dalam penelitian terkait pengenalan wajah. Dataset ini dipilih karena representatif dan banyak digunakan dalam berbagai studi pengenalan wajah serta memiliki variasi kondisi yang sesuai untuk penelitian ini.

4.1.1. Sumber Dataset

Dataset LFW dapat diakses secara publik melalui LFW Website. Dataset ini awalnya dikumpulkan oleh University of Massachusetts Amherst untuk tujuan pengenalan wajah dalam situasi dunia nyata, yang mencakup wajah dari berbagai individu yang diambil dari foto-foto di media sosial, acara-acara publik, dan sumber lainnya.

4.1.2. Ukuran Dataset

Dataset LFW terdiri dari lebih dari 13.000 gambar wajah yang berasal dari 5.749 individu berbeda. Setiap gambar dalam dataset ini telah diberi label dengan nama orang yang ada pada gambar tersebut, sehingga dataset ini cocok untuk tugas pengenalan wajah berbasis klasifikasi.

- Jumlah Gambar: 13.000+
- Jumlah Individu: 5.749
- Jumlah Citra per Individu: Tergantung pada individu, bisa lebih dari satu citra.

4.1.3. Akuisisi Dataset

Pengambilan data dilakukan menggunakan pustaka scikit-learn, yang menyediakan dataset Labeled Faces in the Wild (LFW) secara langsung melalui

fungsi `fetch_lfw_people()`. Dataset ini kemudian dimuat sebagai array citra grayscale, dan setiap citra diubah ukurannya menjadi 224x224 piksel menggunakan pustaka pemrosesan citra seperti Pillow atau OpenCV, agar sesuai dengan kebutuhan sistem dan kompatibel dengan model pembelajaran mesin yang umum digunakan.

1. `data = ambil dari repositori LFW (RGB dan Greyscale)`
2. `daftar_gambar = []`
3. `untuk setiap gambar dalam data:`
4. `ubah gambar menjadi 224 x 224`
5. `simpan gambar pada daftar_gambar`

Gambar 4.1 menjelaskan alur penyiapan dataset citra. Dataset Labeled Faces in the Wild (LFW) diambil menggunakan fungsi `fetch_lfw_people()` dari pustaka `scikit-learn`, dengan parameter `min_faces_per_person=0` agar seluruh citra dalam dataset disertakan. Ukuran gambar awal dipertahankan pada ukuran aslinya (sekitar 250×250 piksel) dengan `resize=1`, kemudian diubah menjadi 224×224 piksel menggunakan fungsi `cv2.resize()` dari pustaka OpenCV. Ukuran 224×224 dipilih karena sesuai dengan resolusi standar yang digunakan dalam aplikasi sistem yang menjadi target pengujian dalam penelitian ini, sehingga hasil evaluasi menjadi lebih relevan dengan implementasi nyata.

4.2. Penambahan *Noise* Aditif

Dalam penelitian ini, dilakukan penambahan *noise* aditif pada citra untuk mensimulasikan gangguan yang sering terjadi pada proses akuisisi citra di dunia nyata, seperti akibat sensor kamera, pencahayaan buruk, atau transmisi sinyal. Penambahan *noise* ini bertujuan untuk menguji ketahanan dan performa metode pengolahan citra terhadap gangguan visual yang umum terjadi.

Jenis *noise* yang digunakan meliputi Gaussian *noise* dan Poisson *noise*, yang keduanya termasuk dalam kategori *noise* aditif. *Noise* tersebut ditambahkan ke setiap piksel citra dengan karakteristik distribusi probabilitas tertentu untuk menciptakan variasi kondisi input.

Penambahan *noise* dilakukan menggunakan fungsi `random_noise()` dari pustaka `scikit-image`, dengan parameter `mode='gaussian'` atau `mode='poisson'`. Proses ini menghasilkan citra baru dengan tingkat gangguan visual yang dapat dikontrol dan disesuaikan untuk keperluan eksperimen.

Jenis *noise* yang digunakan sebagai berikut:

1. *Noise Gaussian*

noise Gaussian merupakan *noise* aditif dengan distribusi normal (Gaussian) yang memiliki rata-rata (mean) nol dan variansi tertentu. *Noise* ini disimulasikan untuk merepresentasikan gangguan acak dari sensor atau lingkungan. Penambahan dilakukan dengan parameter `mode='gaussian'`, dan variansi (parameter `var`) dapat diatur untuk menentukan intensitas gangguan.

2. *Noise Poisson*

Noise Poisson muncul secara alami pada proses pencitraan berbasis foton, seperti pada kamera digital atau pemindaian medis. *Noise* ini bergantung pada nilai piksel itu sendiri dan tidak memiliki parameter varian eksplisit. Penambahan dilakukan dengan `mode='poisson'`.

4.2.1 Alur Penambahan *Noise* Aditif

Berikut alur penambahan *noise* aditif pada citra:

1. Konversi Citra ke dalam format `float32`

Konversi dilakukan dengan tujuan supaya bisa diterapkan penambahan *noise* aditif yang mana sesuai dengan persyaratan fungsi `random_noise()`. Pseudocode-nya sebagai berikut,

1. untuk setiap index dalam random indeks:
2. ambil gambar dari `images` pada posisi indeks
3. ubah gambar menjadi tipe angka `float32`
4. normalisasi piksel jadi 0 - 1
5. simpan hasil sebagai original

Pseudocode ini menjelaskan bagaimana melakukan konversi citra ke dalam format `float32`. Baris kode `original = images[idx].astype(np.float32) / 255.0` digunakan untuk melakukan *preprocessing* terhadap citra sebelum diberi gangguan

atau *noise*. Pada proses ini, citra diambil dari dataset berdasarkan indeks acak yang telah ditentukan sebelumnya, lalu diubah tipe datanya menjadi float32 untuk memungkinkan operasi matematis dengan presisi yang lebih tinggi. Selanjutnya, nilai piksel citra tersebut dinormalisasi ke rentang $[0,1]$ dengan membagi setiap piksel dengan 255.0, karena nilai piksel pada citra digital umumnya berada dalam rentang 0–255. Normalisasi ini penting karena sebagian besar metode penambahan *noise*, seperti Gaussian *noise*, bekerja secara optimal pada data yang berada dalam rentang nilai kontinu antara 0 dan 1. Dengan demikian, langkah ini memastikan bahwa penambahan *noise* dapat dilakukan secara akurat dan sesuai dengan asumsi fungsi-fungsi pemrosesan citra yang digunakan.

4.3. Pengujian

4.3.1. Penerapan Gaussian Blur

Gaussian Blur merupakan sebuah *low-pass* Filter, yaitu sebuah filter yang mereduksi frekuensi tinggi pada domain frekuensi pada sebuah citra digital. Meskipun filter ini bekerja pada domain frekuensi, filter ini dapat diterapkan pada domain spasial dan hasilnya akan tetap sama pada domain frekuensi. Hal ini disebabkan karena sifat distribusi Gaussian yang bentuknya sama pada domain spasial dan domain frekuensi, yaitu sebuah kurva lonceng. Oleh karena itu, penerapan filter ini dapat dengan mudah dilakukan dengan menggunakan teknik Convolution Filtering.

Garis besar penerapan metode ini dapat dinyatakan dalam beberapa langkah, yaitu:

1. Penerapan Convolution Filtering

```
1. function ConvolutionFiltering(image, kernel):  
2.   I = number of rows in image  
3.   J = number of columns in image  
4.   K = size of the kernel (assumed square)  
5.   offset = floor(K / 2)  
6.   outputRows = I - 2 * offset  
7.   outputCols = J - 2 * offset  
8.   outputImage = matrix[outputRows x outputCols]
```

```

9.     for i from offset to I - offset - 1:
10.        for j from offset to J - offset - 1:
11.            sum = 0
12.            for m from 0 to K - 1:
13.                for n from 0 to K - 1:
14.                    x = i + m - offset
15.                    y = j + n - offset
16.                    pixel = image[x][y]
17.                    weight = kernel[m][n]
18.                    sum += pixel * weight
19.            outputImage[i - offset][j - offset] = sum
20.    return outputImage

```

Pseudocode ini untuk Fungsi ConvolutionFiltering. Fungsi ConvolutionFilter(image, kernel) adalah sebuah algoritma untuk menerapkan proses konvolusi pada citra digital dua dimensi (grayscale) tanpa memperhitungkan bagian tepi (edge border). Fungsi ini menerima dua masukan, yaitu image, yang merupakan matriks dua dimensi berukuran $I \times J$, dan kernel, yaitu filter matriks berukuran $K \times K$ (biasanya ganjil, seperti 3×3 atau 5×5). Untuk menghindari akses di luar batas array saat proses konvolusi, fungsi ini menghitung offset sebesar setengah ukuran kernel ($\text{floor}(K / 2)$), lalu hanya melakukan perhitungan pada bagian tengah citra yang memiliki cukup ruang di sekelilingnya untuk menerima kernel secara penuh.

Proses utama dilakukan dengan menjelajah piksel-piksel citra dari baris dan kolom yang dimulai dari offset hingga $I - \text{offset} - 1$ dan $J - \text{offset} - 1$, sehingga tepi citra tidak ikut diproses. Untuk setiap piksel pusat di lokasi (i, j) , fungsi menghitung hasil konvolusi dengan menjumlahkan hasil perkalian elemen-elemen kernel dengan piksel citra yang bersesuaian. Nilai hasil penjumlahan ini kemudian disimpan di matriks outputImage pada posisi yang disesuaikan dengan pengurangan offset.

Hasil akhir dari fungsi ini adalah matriks outputImage yang berukuran lebih kecil dari citra awal, karena bagian tepi tidak dihitung. Fungsi ini cocok untuk diterapkan pada filter simetris seperti Gaussian blur, di mana pembalikan kernel (flip) tidak diperlukan.

1. Men-*generate* sebuah kernel Gaussian

```

1. function GenerateGaussianKernel(k_size, sigma)
2.     kernel = matrix of size k_size x k_size
3.     center = floor(k_size / 2)
4.     sum = 0
5.     for i from 0 to k_size - 1:
6.         for j from 0 to k_size - 1:
7.             x = i - center
8.             y = j - center
9.             distance = x*x + y*y
10.            denom = 2 * sigma * sigma
11.            exponent = -distance / denom
12.            normalizer = PI * denom
13.            kernel[i][j] = exp(exponent) / normalizer
14.            sum += kernel[i][j]
15.     for i from 0 to k_size - 1:
16.         for j from 0 to k_size - 1:
17.             kernel[i][j] = kernel[i][j] / sum
18.     return kernel

```

Pseudocode ini untuk Fungsi GenerateGaussianKernel. Fungsi GenerateGaussianKernel(k_size, sigma) digunakan untuk menghasilkan kernel Gaussian dua dimensi dengan ukuran $k_size \times k_size$ dan standar deviasi sigma. Kernel ini digunakan untuk melakukan Gaussian Blur pada citra digital. Langkah pertama dalam fungsi ini adalah membuat matriks kosong berukuran $k_size \times k_size$ dan menentukan pusat kernel dengan menghitung $center = \text{floor}(k_size / 2)$. Selanjutnya, fungsi menghitung setiap elemen kernel berdasarkan rumus Gaussian dua dimensi, yaitu $\exp(-(x^2 + y^2) / (2 * \sigma^2)) / (2 * \pi * \sigma^2)$, di mana x dan y adalah jarak dari pusat kernel ke posisi (i, j). Hasil perhitungan disimpan di dalam matriks kernel, dan semua nilainya dijumlahkan ke dalam variabel sum.

Setelah semua elemen kernel dihitung, langkah berikutnya adalah melakukan normalisasi agar jumlah seluruh elemen kernel sama dengan 1. Hal ini dilakukan dengan membagi setiap elemen kernel dengan nilai sum. Normalisasi ini penting agar proses konvolusi tidak mengubah tingkat kecerahan rata-rata citra. Fungsi kemudian mengembalikan matriks kernel yang sudah dinormalisasi. Kernel ini bisa langsung digunakan dalam proses convolution filtering untuk menghasilkan efek Gaussian Blur.

2. Melakukan konvolusi pada citra menggunakan kernel Gaussian yang sudah dibuat. Pseudocode-nya sebagai berikut,

```
1. Function DoGaussianBlur(image, k_size, sigma):  
2.   kernel = GenerateGaussianKernel(k_size, sigma)  
3.   result = ConvolutionFiltering(image, kernel)  
4.   return result
```

Pseudocode ini adalah untuk fungsi DoGaussianBlur. Fungsi DoGaussianBlur(image, k_size, sigma) adalah fungsi utama yang digunakan untuk menerapkan efek Gaussian Blur pada sebuah citra digital. Fungsi ini menerima tiga parameter: image berupa matriks dua dimensi yang merepresentasikan citra grayscale, k_size yaitu ukuran kernel Gaussian yang harus berupa bilangan ganjil, dan sigma yaitu nilai standar deviasi yang menentukan seberapa besar penyebaran blur pada citra. Di dalam fungsi ini, pertama-tama dipanggil fungsi GenerateGaussianKernel(k_size, sigma) untuk membangkitkan kernel Gaussian berdasarkan parameter yang diberikan. Setelah kernel terbentuk, fungsi kemudian memanggil ConvolutionFilter(image, kernel) untuk menerapkan kernel tersebut ke citra input melalui proses konvolusi. Hasil dari proses ini adalah citra baru yang telah mengalami pemburaman (blur) dengan pola distribusi Gaussian, yang kemudian dikembalikan sebagai output dari fungsi. Pendekatan ini modular dan efisien, sehingga mudah digunakan dalam berbagai aplikasi pemrosesan citra seperti penghalusan, reduksi *noise*, dan deteksi tepi.

4.3.2. Pengujian Terhadap *noise* Gaussian

Pengujian dilakukan setelah seluruh citra wajah dari dataset berhasil diolah dan disesuaikan ukurannya. Tahapan pengujian ini dimulai dengan menambahkan *noise* aditif pada citra-citra yang tersedia. *Noise* yang digunakan adalah *noise* Gaussian, yaitu jenis gangguan yang umum terjadi dalam akuisisi citra digital, terutama karena fluktuasi sensor atau kondisi pencahayaan. Penambahan *noise* ini dilakukan dalam tiga tingkat intensitas berbeda, yaitu tingkat rendah, sedang, dan tinggi, yang masing-masing direpresentasikan oleh nilai variansi sebesar 0.001, 0.01, dan 0.1. Tujuan dari variasi ini adalah untuk mengamati sejauh mana metode

denoising Gaussian Blur mampu memulihkan kualitas citra yang terganggu pada berbagai kondisi *noise*. Dengan demikian, dapat dievaluasi performa metode terhadap gangguan ringan hingga berat secara sistematis.

```
1. function test_gaussian(k_size, sigma, var, images, n) :
2.   psnr_list = empty list
3.   ssim_list = empty list
4.   indices = pick n random indices from images
5.   for each i in indices:
6.     img = images[i]
7.     img = normalize img to range 0-1
8.     noisy = add Gaussian noise to img using var
9.     denoised = DoGaussianBlur(noisy, k_size, sigma)
10.    ssim = compute SSIM(img, denoised)
11.    add ssim to ssim_list
12.    psnr = compute PSNR(img, denoised)
13.    add psnr to psnr_list
14.  avg_ssim = average of ssim_list
15.  avg_psnr = average of psnr_list
16.  return (avg_psnr, avg_ssim)
```

Fungsi `test_gaussian` digunakan untuk melakukan pengujian efektivitas metode *denoising* Gaussian Blur terhadap citra-citra wajah yang telah diberikan gangguan berupa *noise* Gaussian. Fungsi ini menerima beberapa parameter masukan, yaitu kumpulan citra (`images`), ukuran kernel (`ksize`), nilai deviasi standar Gaussian (`sigma`), besar varians *noise* (`var`), serta jumlah sampel citra yang akan diuji (`jumlah`). Proses pengujian diawali dengan pemilihan sejumlah citra secara acak dari dataset. Masing-masing citra dikonversi ke tipe data `float32` dan dinormalisasi ke rentang `[0, 1]` untuk keperluan perhitungan yang lebih presisi.

Selanjutnya, citra tersebut diberikan *noise* Gaussian dengan varians sesuai parameter `var`. Setelah itu, citra yang telah diberi *noise* diproses menggunakan Gaussian Blur dengan parameter kernel dan `sigma` yang ditentukan, menghasilkan citra hasil *denoising*. Untuk menilai kualitas hasil *denoising*, dua metrik evaluasi digunakan, yaitu PSNR dan SSIM. PSNR digunakan untuk mengukur tingkat kesamaan secara numerik antara citra asli dan citra hasil *denoising*, sementara SSIM mengukur kesamaan struktur secara visual yang meliputi luminansi, kontras, dan struktur tekstur citra.

Nilai PSNR dan SSIM untuk setiap citra disimpan dalam list, dan setelah seluruh sampel diuji, nilai rata-rata dari kedua metrik tersebut dihitung. Fungsi ini akhirnya mengembalikan nilai rata-rata SSIM dan PSNR dalam bentuk Dictionary. Dengan cara ini, performa metode Gaussian Blur dapat dibandingkan secara kuantitatif untuk berbagai kombinasi parameter kernel, sigma, dan tingkat *noise* yang ditambahkan pada citra.

4.3.2. Pengujian Terhadap *noise* Poisson

Pengujian dilakukan setelah seluruh citra wajah dari dataset berhasil diolah dan disesuaikan ukurannya. Pada tahap ini, dilakukan penambahan *noise* bertipe Poisson ke dalam citra-citra yang tersedia. *Noise* Poisson merupakan jenis *noise* yang berkaitan dengan fluktuasi alami dalam proses penghitungan foton selama akuisisi citra, sehingga umum dijumpai pada sistem pencitraan yang menggunakan cahaya rendah atau paparan singkat. Berbeda dengan *noise* Gaussian yang dikendalikan oleh variansi, *noise* Poisson bersifat tergantung pada nilai intensitas piksel itu sendiri. Oleh karena itu, pengujian dilakukan tanpa pengaturan variansi secara eksplisit, namun tetap menggunakan berbagai parameter filter Gaussian seperti ukuran kernel dan nilai sigma untuk mengamati efektivitas proses *denoising*. Tujuan dari pengujian ini adalah untuk mengevaluasi seberapa baik metode Gaussian Blur dapat mengurangi gangguan akibat *noise* Poisson dan mengembalikan kualitas citra secara struktural dan visual.

```
1. function test_poisson(k_size, sigma, images, n):
2.     psnr_list = empty list
3.     ssim_list = empty list
4.     indices = pick n random indices from images
5.     for each i in indices:
6.         img = images[i]
7.         img = normalize to range 0-1
8.         noisy = add Poisson noise to img
9.         denoised = DoGaussianBlur(noisy, k_size, sigma)
10.        ssim = compute SSIM(img, denoised)
11.        add ssim to ssim_list
12.        psnr = compute PSNR(img, denoised)
13.        add psnr to psnr_list
```

```
14. avg_ssim = average of ssim_list
15. avg_psnr = average of psnr_list
16. return (avg_psnr, avg_ssim)
```

Fungsi `test_poisson` bertujuan untuk menguji efektivitas filter Gaussian dalam mengurangi gangguan *noise* Poisson pada citra. Fungsi ini menerima beberapa parameter, yaitu ukuran kernel filter Gaussian (`ksize`), nilai sigma untuk filter Gaussian (`sigma`), kumpulan citra (`images`), dan jumlah citra yang akan diproses (`jumlah`).

Proses dimulai dengan memilih sejumlah citra secara acak dari kumpulan citra yang ada, sesuai dengan nilai jumlah yang diberikan. Untuk setiap citra yang dipilih, citra tersebut pertama-tama dikonversi ke tipe data `float32` dan dinormalisasi dengan membaginya dengan 255.0, sehingga nilai pixel citra berada dalam rentang `[0, 1]`.

Selanjutnya, citra yang sudah dinormalisasi diberi *noise* Poisson menggunakan fungsi `random_noise` dengan mode `'poisson'`. *Noise* Poisson sering muncul dalam citra yang diambil di lingkungan dengan cahaya rendah atau pada gambar yang memiliki banyak rincian tekstur halus.

Setelah *noise* Poisson ditambahkan, citra yang telah terkontaminasi *noise* tersebut diproses menggunakan filter Gaussian Blur dengan ukuran kernel (`ksize`) dan nilai sigma (`sigma`) yang ditentukan oleh parameter input. Filter ini bertujuan untuk mengurangi *noise* dan memperhalus citra.

Setelah proses *denoising*, citra yang telah di-*denoise* dibandingkan dengan citra asli menggunakan dua metrik kualitas citra: PSNR dan SSIM. Nilai PSNR dan SSIM untuk setiap citra yang diproses dihitung, kemudian dirata-ratakan untuk memberikan hasil keseluruhan dari pengujian. Fungsi ini akhirnya mengembalikan nilai rata-rata PSNR dan SSIM dalam bentuk Dictionary, yang memberikan gambaran tentang seberapa baik filter Gaussian dapat mengurangi *noise* Poisson pada citra.

4.3.3. Pengujian Terhadap Gabungan *Noise* Poisson dan Gaussian

Pengujian terhadap gabungan *noise* Poisson dan Gaussian dilakukan untuk mengevaluasi efektivitas filter Gaussian dalam mengurangi gangguan yang ditimbulkan oleh *noise* Poisson pada citra. *Noise* Poisson, yang sering ditemukan pada sistem pencitraan dengan cahaya rendah atau paparan singkat, memiliki sifat yang tergantung pada intensitas piksel itu sendiri, berbeda dengan *noise* Gaussian yang dikendalikan oleh variansi. Pada pengujian ini, citra-citra yang telah diolah dan disesuaikan ukurannya diberi tambahan *noise* Poisson dan Gaussian. Selanjutnya, filter Gaussian dengan berbagai parameter, seperti ukuran kernel dan nilai sigma, diterapkan untuk proses denoising. Tujuan dari pengujian gabungan ini adalah untuk mengevaluasi seberapa baik metode Gaussian Blur dapat mengurangi gabungan antara *noise* Poisson dan Gaussian, serta seberapa besar pengaruhnya terhadap kualitas citra baik secara struktural maupun visual.

```
1.function test_mixed(k_size,sigma,var,images,n):
2.    psnr_list = empty list
3.    ssim_list = empty list
4.    indices = pick n random indices from images
5.    for each i in indices:
6.        img = images[i]
7.        img = normalize to range 0-1
8.        noisy = img
9.        add Gaussian noise to noisy using var
10.       add Poisson noise to noisy
11.       denoised = DoGaussianBlur(noisy, k_size, sigma)
12.       ssim = compute SSIM(img, denoised)
13.       add ssim to ssim_list
14.       psnr = compute PSNR(img, denoised)
15.       add psnr to psnr_list
16.   avg_ssim = average of ssim_list
17.   avg_psnr = average of psnr_list
18.   return (avg_psnr, avg_ssim)
```

Fungsi `test_gaussian_poisson` bertujuan untuk menguji efektivitas metode filter Gaussian dalam mengurangi gangguan gabungan dari *noise* Poisson dan Gaussian pada citra. Fungsi ini menerima beberapa parameter, yaitu ukuran kernel filter Gaussian (`ksize`), nilai sigma untuk filter Gaussian (`sigma`), variansi untuk

noise Gaussian (var), kumpulan citra (images), dan jumlah citra yang akan diproses (jumlah).

Proses dimulai dengan memilih sejumlah citra secara acak dari kumpulan citra yang ada, sesuai dengan nilai jumlah yang ditentukan. Untuk setiap citra yang dipilih, citra tersebut terlebih dahulu dikonversi ke tipe data float32 dan dinormalisasi dengan membaginya dengan 255.0. Kemudian, citra tersebut diberi dua jenis gangguan: pertama, *noise* Poisson yang ditambahkan menggunakan mode 'poisson', diikuti dengan penambahan *noise* Gaussian menggunakan variansi (var) yang diberikan.

Setelah penambahan *noise*, filter Gaussian Blur diterapkan pada citra yang sudah terkontaminasi *noise* tersebut, dengan ukuran kernel (ksize) dan nilai sigma (sigma) yang diberikan. Hasil citra yang sudah di-*denoise* kemudian dibandingkan dengan citra asli menggunakan dua metrik kualitas citra: PSNR dan SSIM. Hasil dari PSNR dan SSIM untuk semua citra yang diproses kemudian dirata-ratakan dan dikembalikan sebagai output dalam bentuk Dictionary yang berisi nilai rata-rata PSNR dan SSIM. Fungsi ini berguna untuk mengevaluasi seberapa baik filter Gaussian dapat mengurangi *noise* gabungan dari Poisson dan Gaussian pada citra.

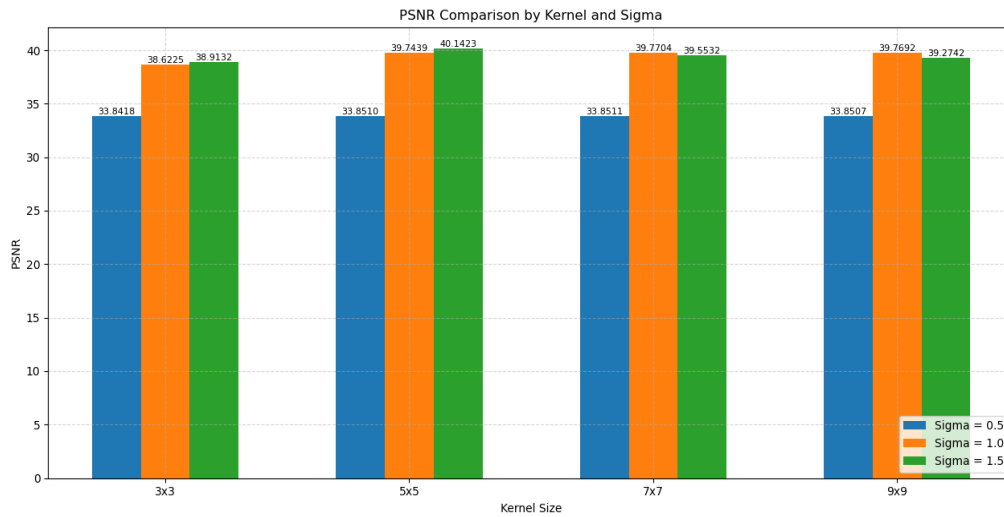
4.4. Evaluasi Greyscale

4.4.1. Evaluasi PSNR dan SSIM terhadap *denoising noise* Gaussian

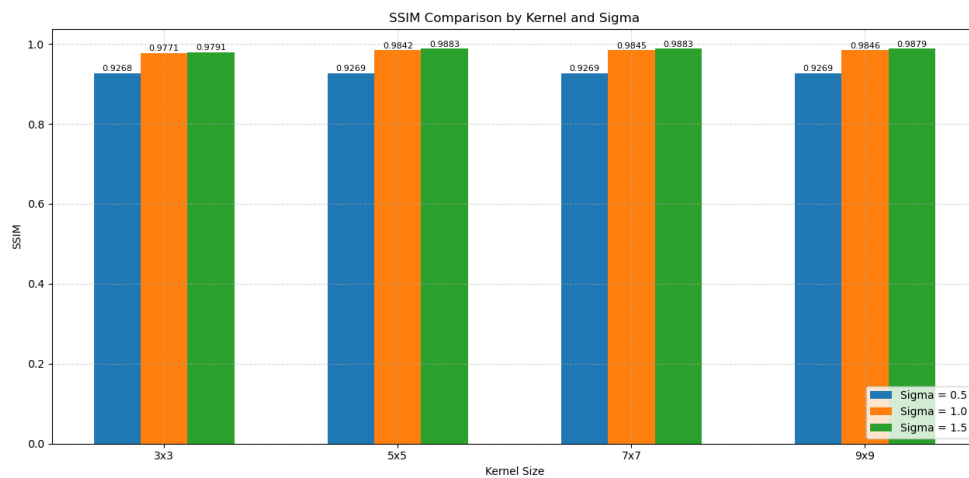
Tabel 4. 1 Tabel PSNR dan SSIM untuk denosing noise Gaussian Greyscale

| Kernel Size | Sigma | Variansi Noise | SSIM | PSNR |
|-------------|-------|----------------|----------|----------|
| 3 | 0.5 | 0.001 | 0.926772 | 33.84181 |
| 3 | 0.5 | 0.01 | 0.580676 | 23.9388 |
| 3 | 0.5 | 0.1 | 0.165325 | 15.08318 |
| 3 | 1 | 0.001 | 0.977118 | 38.62255 |
| 3 | 1 | 0.01 | 0.824245 | 28.99332 |
| 3 | 1 | 0.1 | 0.400394 | 19.87754 |
| 3 | 1.5 | 0.001 | 0.979074 | 38.91322 |
| 3 | 1.5 | 0.01 | 0.838774 | 29.3934 |

| | | | | |
|---|-----|-------|----------|----------|
| 3 | 1.5 | 0.1 | 0.425027 | 20.24873 |
| 5 | 0.5 | 0.001 | 0.92692 | 33.851 |
| 5 | 0.5 | 0.01 | 0.581144 | 23.94748 |
| 5 | 0.5 | 0.1 | 0.165621 | 15.09206 |
| 5 | 1 | 0.001 | 0.984198 | 39.74394 |
| 5 | 1 | 0.01 | 0.879989 | 30.68821 |
| 5 | 1 | 0.1 | 0.509966 | 21.4399 |
| 5 | 1.5 | 0.001 | 0.988266 | 40.14228 |
| 5 | 1.5 | 0.01 | 0.926659 | 32.50278 |
| 5 | 1.5 | 0.1 | 0.646076 | 23.14121 |
| 7 | 0.5 | 0.001 | 0.926919 | 33.8511 |
| 7 | 0.5 | 0.01 | 0.581135 | 23.94704 |
| 7 | 0.5 | 0.1 | 0.165618 | 15.09225 |
| 7 | 1 | 0.001 | 0.984544 | 39.77043 |
| 7 | 1 | 0.01 | 0.883712 | 30.81961 |
| 7 | 1 | 0.1 | 0.518798 | 21.56396 |
| 7 | 1.5 | 0.001 | 0.98828 | 39.55322 |
| 7 | 1.5 | 0.01 | 0.943168 | 33.25349 |
| 7 | 1.5 | 0.1 | 0.712683 | 23.96603 |
| 9 | 0.5 | 0.001 | 0.926918 | 33.85075 |
| 9 | 0.5 | 0.01 | 0.581124 | 23.94698 |
| 9 | 0.5 | 0.1 | 0.165616 | 15.09217 |
| 9 | 1 | 0.001 | 0.984552 | 39.76924 |
| 9 | 1 | 0.01 | 0.88383 | 30.8241 |
| 9 | 1 | 0.1 | 0.519048 | 21.56742 |
| 9 | 1.5 | 0.001 | 0.987927 | 39.27422 |
| 9 | 1.5 | 0.01 | 0.945719 | 33.35767 |
| 9 | 1.5 | 0.1 | 0.725033 | 24.12439 |



Gambar 4. 1 Diagram batang perbandingan nilai PSNR dalam mereduksi noise Gaussian rendah berdasarkan nilai sigma dan ukuran kernel



Gambar 4. 2 Diagram batang perbandingan nilai SSIM dalam mereduksi noise Gaussian rendah berdasarkan nilai sigma dan ukuran kernel

Tabel 4.1 berisi hasil perhitungan PSNR dan SSIM untuk *denoising noise* Gaussian dengan berbagai konfigurasi Gaussian Blur. Gambar 4.9 adalah perbandingan nilai PSNR dalam mereduksi *noise* tingkat rendah berdasarkan nilai sigma dan ukuran kernel. Peringkat nilai PSNR tertinggi, di antaranya:

1. Kernel 5x5 dengan sigma 1.5, dengan nilai PSNR 40.1423

2. Kernel 7x7 dengan sigma 1.0, dengan nilai PSNR 39.7704
3. Kernel 9x9 dengan sigma 1.0, dengan nilai PSNR 39.7692

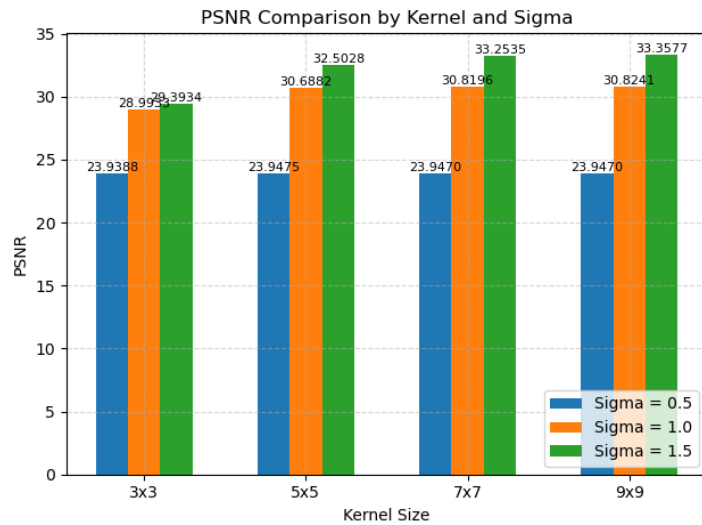
ketiga konfigurasi ini menghasilkan keluaran citra dengan hasil yang baik, dengan PSNR di atas 35 dB. Hal ini menunjukkan bahwa kombinasi ukuran kernel yang lebih besar dengan nilai sigma yang tepat mampu mereduksi *noise* tingkat rendah secara lebih efektif. Kernel 5x5 dengan sigma 1.5 memberikan hasil terbaik karena dapat menangkap detail sekaligus mereduksi *noise* secara optimal. Sementara itu, kernel 7x7 dan 9x9 dengan sigma 1.0 juga menunjukkan performa yang sangat baik, mendekati hasil dari konfigurasi terbaik.

Sebaliknya, konfigurasi dengan sigma 0.5 pada semua ukuran kernel menunjukkan performa yang lebih rendah, dengan PSNR berkisar di angka 33.8 dB. Ini mengindikasikan bahwa nilai sigma yang terlalu kecil kurang mampu mereduksi *noise* secara maksimal, bahkan dengan ukuran kernel yang besar sekalipun.

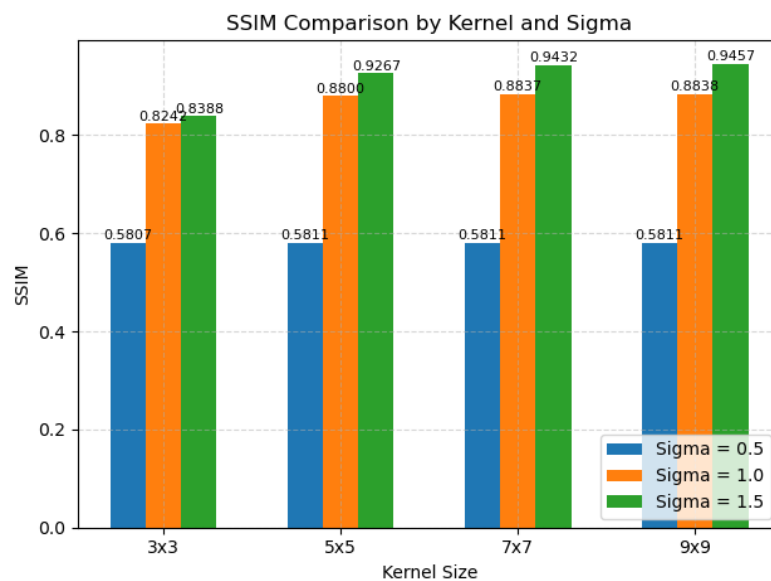
Gambar 4.10 adalah diagram batang perbandingan nilai SSIM dalam mereduksi *noise* tingkat rendah berdasarkan nilai sigma dan ukuran kernel. Seperti yang didapat dari pembahasan nilai PSNR sebelumnya, nilai sigma yang lebih tinggi (1.5) menghasilkan hasil SSIM yang lebih baik, dengan peringkat:

1. Kernel 7x7 dengan nilai sigma 1.5 menghasilkan SSIM dengan hasil 0.9882802339
2. Kernel 5x5 dengan nilai sigma 1.5 menghasilkan SSIM dengan nilai 0.9882660152
3. Kernel 9x9 dengan nilai sigma 1.5 menghasilkan SSIM dengan nilai 0.9879266412

Nilai SSIM tertinggi diperoleh oleh konfigurasi kernel 7x7 dengan sigma 1.5. Meskipun demikian, selisih nilai SSIM antara ketiga konfigurasi tersebut sangat kecil, menunjukkan bahwa semua konfigurasi tersebut memberikan kualitas struktur citra yang hampir setara setelah proses *denoising*.



Gambar 4. 3 Diagram batang perbandingan nilai PSNR dalam mereduksi noise Gaussian sedang berdasarkan nilai sigma dan ukuran kernel



Gambar 4. 4 Diagram batang perbandingan nilai SSIM dalam mereduksi noise Gaussian sedang berdasarkan nilai sigma dan ukuran kernel

Gambar 4.11 adalah diagram batang perbandingan nilai PSNR dalam mereduksi *noise* sedang berdasarkan nilai sigma dan ukuran kernel. Hasil sigma 1.5 menghasilkan nilai PSNR terbesar di seluruh ukuran kernel yang diuji, dengan peringkat:

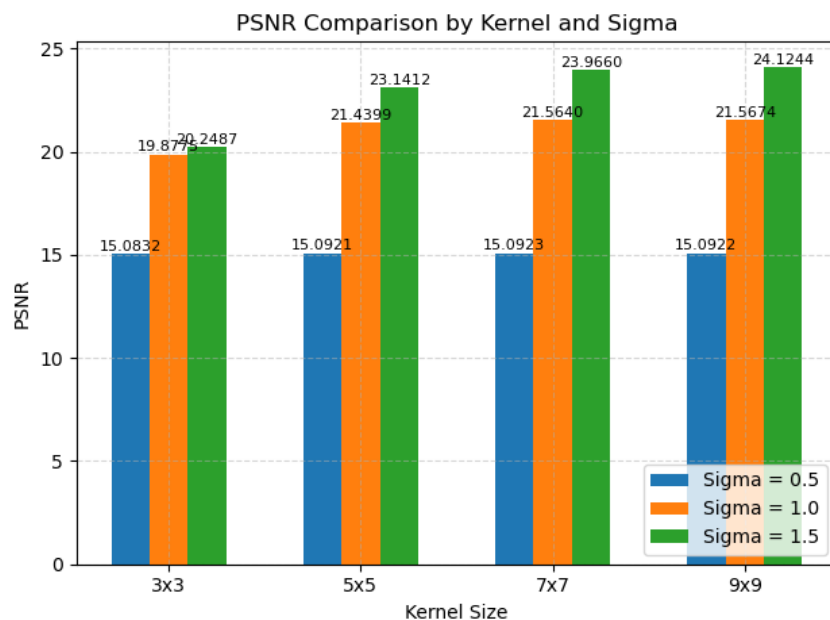
1. Kernel 9x9 dengan sigma 1.5 menghasilkan PSNR dengan nilai 33.357

2. Kernel 7x7 dengan sigma 1.5 menghasilkan PSNR dengan nilai 33.2335
3. Kernel 5x5 dengan sigma 1.5 menghasilkan PSNR dengan nilai 32.5028

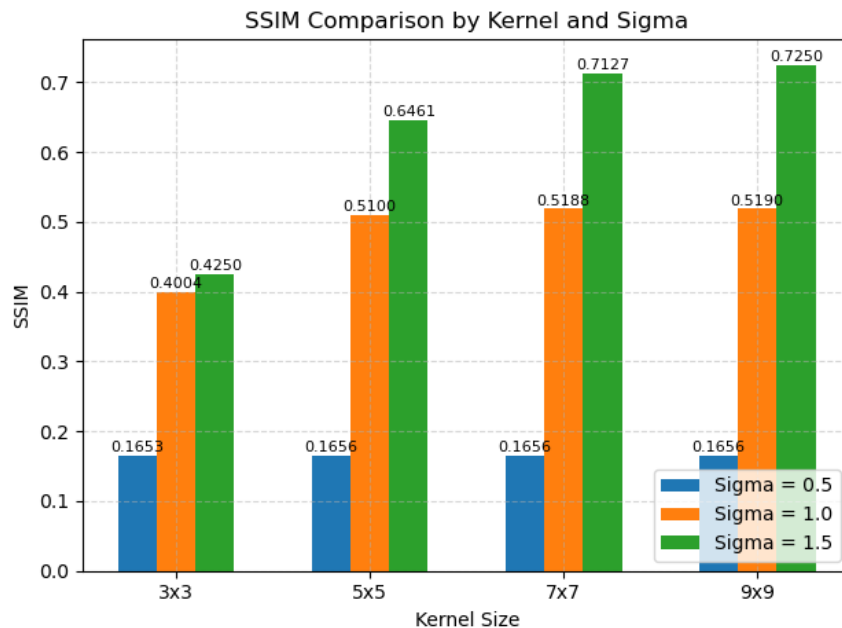
Kernel 9x9 dengan sigma 1.5 menghasilkan nilai PSNR dengan nilai tertinggi. Hal ini juga sama dengan nilai SSIM yang bisa dilihat pada Gambar 4.12, dengan peringkat:

1. Kernel 9x9 dengan sigma 1.5 menghasilkan PSNR dengan nilai 0.9457
2. Kernel 7x7 dengan sigma 1.5 menghasilkan PSNR dengan nilai 0.9432
3. Kernel 5x5 dengan sigma 1.5 menghasilkan PSNR dengan nilai 0.9267

Secara keseluruhan, konfigurasi-konfigurasi dengan sigma 1.5 menunjukkan performa yang sangat baik dalam mereduksi *noise* sedang, dengan nilai PSNR di atas 32 dan SSIM di atas 0.92. Hal ini mengindikasikan bahwa konfigurasi tersebut tidak hanya mampu mengurangi *noise*, tetapi juga mempertahankan kemiripan struktural terhadap citra asli secara efektif.



Gambar 4. 5 Diagram batang perbandingan nilai PSNR dalam mereduksi *noise* Gaussian tinggi berdasarkan nilai sigma dan ukuran kernel



Gambar 4. 6 Diagram batang perbandingan nilai PSNR dalam mereduksi noise Gaussian tinggi berdasarkan nilai sigma dan ukuran kernel

Gambar 4.13 adalah diagram batang perbandingan nilai PSNR dalam mereduksi noise sedang berdasarkan nilai sigma dan ukuran kernel. Hasil sigma 1.5 menghasilkan nilai PSNR terbesar di seluruh ukuran kernel yang diuji, dengan peringkat:

1. Kernel 9x9 dengan sigma 1.5 menghasilkan PSNR dengan nilai 24.1244
2. Kernel 7x7 dengan sigma 1.5 menghasilkan PSNR dengan nilai 23.9660
3. Kernel 5x5 dengan sigma 1.5 menghasilkan PSNR dengan nilai 23.1412

Kernel 9x9 dengan sigma 1.5 menghasilkan nilai PSNR dengan nilai tertinggi. Hal ini juga sama dengan nilai SSIM yang bisa dilihat pada Gambar 4.14, dengan peringkat:

1. Kernel 9x9 dengan sigma 1.5 menghasilkan PSNR dengan nilai 0.7250
2. Kernel 7x7 dengan sigma 1.5 menghasilkan PSNR dengan nilai 0.7127
3. Kernel 5x5 dengan sigma 1.5 menghasilkan PSNR dengan nilai 0.6461

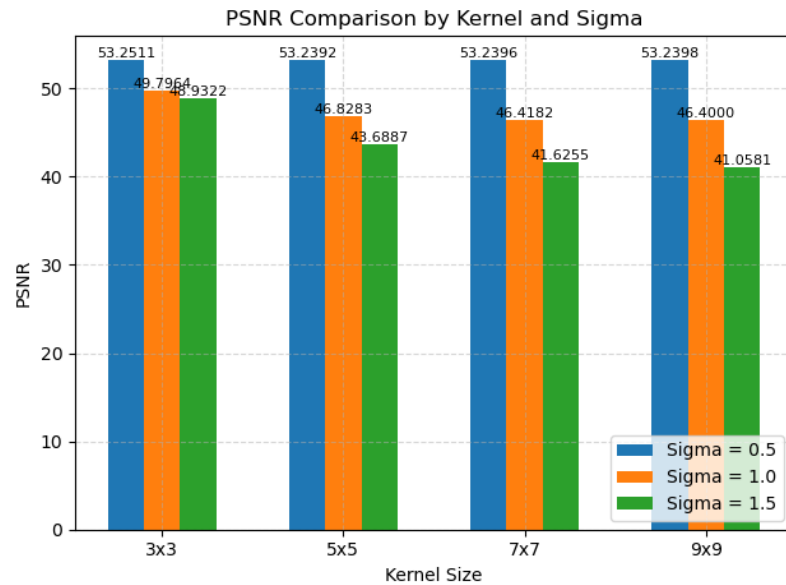
Secara keseluruhan, konfigurasi-konfigurasi dengan sigma 1.5 menunjukkan performa yang buruk dalam mereduksi noise tinggi. Nilai SSIM yang

relatif rendah ini menandakan bahwa meskipun struktur citra masih dapat dipertahankan sebagian, kualitas hasil pereduksian *noise* tinggi belum optimal. Secara keseluruhan, konfigurasi-konfigurasi dengan sigma 1.5 menunjukkan performa yang kurang memuaskan dalam menangani *noise* tinggi, ditandai dengan nilai PSNR yang berada di bawah 30 dan nilai SSIM yang tidak mencapai 0.75.

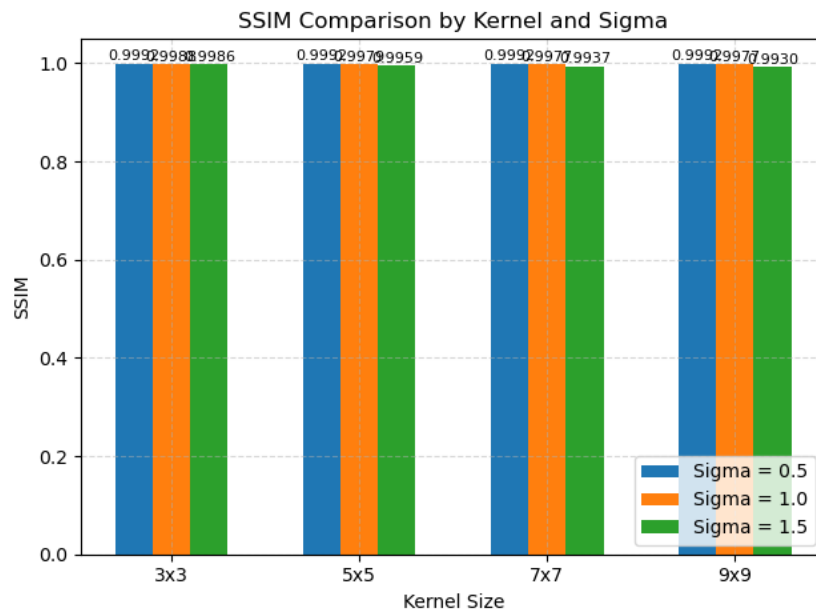
4.4.2. Evaluasi PSNR dan SSIM terhadap *denoising noise Poisson*

Tabel 4. 2 Tabel PSNR dan SSIM untuk *denoising Poisson Greyscale*

| Kernel Size | Sigma | SSIM | PSNR |
|-------------|-------|----------|----------|
| 3 | 0.5 | 0.999216 | 53.25106 |
| 3 | 1 | 0.998831 | 49.79636 |
| 3 | 1.5 | 0.998593 | 48.93223 |
| 5 | 0.5 | 0.999216 | 53.23925 |
| 5 | 1 | 0.997915 | 46.82828 |
| 5 | 1.5 | 0.995888 | 43.68868 |
| 7 | 0.5 | 0.999216 | 53.23963 |
| 7 | 1 | 0.997744 | 46.41819 |
| 7 | 1.5 | 0.99373 | 41.62551 |
| 9 | 0.5 | 0.999216 | 53.23976 |
| 9 | 1 | 0.997737 | 46.40002 |
| 9 | 1.5 | 0.993004 | 41.0581 |



Gambar 4. 7 Diagram batang perbandingan nilai PSNR dalam mereduksi noise Poisson berdasarkan nilai sigma dan ukuran kernel



Gambar 4. 8 Diagram batang perbandingan nilai SSIM dalam mereduksi noise Poisson berdasarkan nilai sigma dan ukuran kernel

Hasil pengujian performa filter berdasarkan variasi kernel dan nilai sigma terhadap citra dengan *noise* Poisson ditunjukkan pada tabel 4.2. Pada konfigurasi kernel 3x3, nilai PSNR tertinggi dicapai dengan sigma 0.5 sebesar 53.2511, diikuti oleh sigma 1.0 sebesar 49.7964 dan sigma 1.5 sebesar 48.9322. Nilai SSIM untuk

ketiga konfigurasi tersebut juga sangat tinggi, masing-masing sebesar 0.9992, 0.9988, dan 0.9986, menandakan bahwa struktur citra tetap sangat terjaga.

Untuk kernel 5x5, nilai PSNR tertinggi kembali diperoleh pada sigma 0.5 yaitu sebesar 53.2392, disusul oleh sigma 1.0 dengan PSNR sebesar 46.8283 dan sigma 1.5 sebesar 43.6887. Nilai SSIM berkisar antara 0.9959 hingga 0.9992, menunjukkan hasil yang sangat baik dalam mempertahankan detail citra.

Konfigurasi kernel 7x7 menunjukkan kecenderungan serupa, dengan PSNR tertinggi sebesar 53.2396 pada sigma 0.5, lalu turun menjadi 46.4182 pada sigma 1.0 dan 41.6255 pada sigma 1.5. Nilai SSIM yang dihasilkan berkisar antara 0.9937 hingga 0.9992, masih dalam rentang yang sangat baik.

Begitu pula pada kernel 9x9, sigma 0.5 menghasilkan PSNR sebesar 53.2398 dengan SSIM sebesar 0.9992, jauh lebih tinggi dibandingkan sigma 1.0 (PSNR 46.4000, SSIM 0.9977) dan sigma 1.5 (PSNR 41.0581, SSIM 0.9930).

Secara umum, seluruh konfigurasi dengan sigma 0.5 memberikan hasil PSNR dan SSIM tertinggi di masing-masing ukuran kernel. Hal ini menunjukkan bahwa pada citra tanpa *noise* atau dengan *noise* sangat rendah, penggunaan nilai sigma yang kecil lebih efektif dalam mempertahankan kualitas visual citra, dengan detail dan struktur yang tetap terjaga secara optimal.

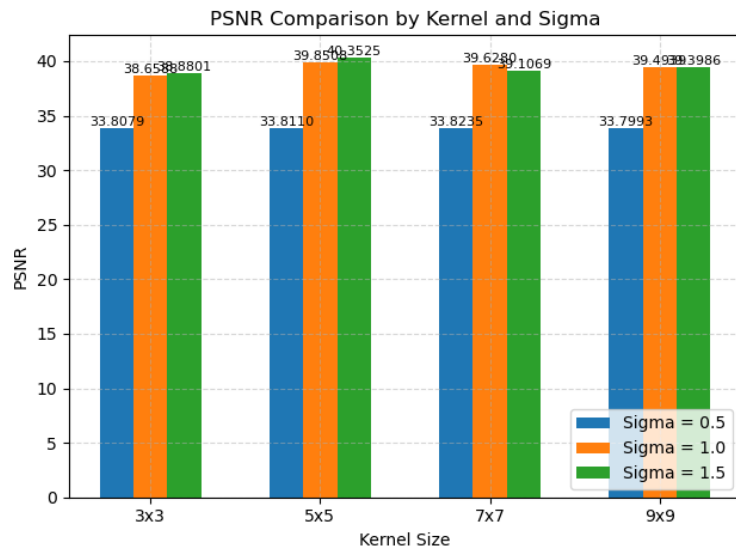
Terdapat perbedaan antara nilai PSNR dan SSIM pada hasil evaluasi tersebut. Nilai PSNR turun secara drastis diiringi dengan nilai sigma yang semakin turun. Nilai SSIM lebih stabil, dengan hampir semua konfigurasi menghasilkan nilai di atas 0.99. Hal ini menunjukkan bahwa meskipun secara matematis kualitas citra menurut PSNR menurun seiring meningkatnya tingkat blur (ditunjukkan oleh nilai sigma yang lebih tinggi), struktur visual citra yang diproses tetap relatif serupa dengan citra aslinya menurut persepsi manusia, sebagaimana direpresentasikan oleh SSIM. Dengan kata lain, PSNR lebih peka terhadap perbedaan piksel-per-piksel, sedangkan SSIM lebih menekankan pada kesamaan struktur dan tampilan umum citra.

4.4.3. Evaluasi PSNR dan SSIM terhadap *denoising noise* gabungan Poisson dan Gaussian

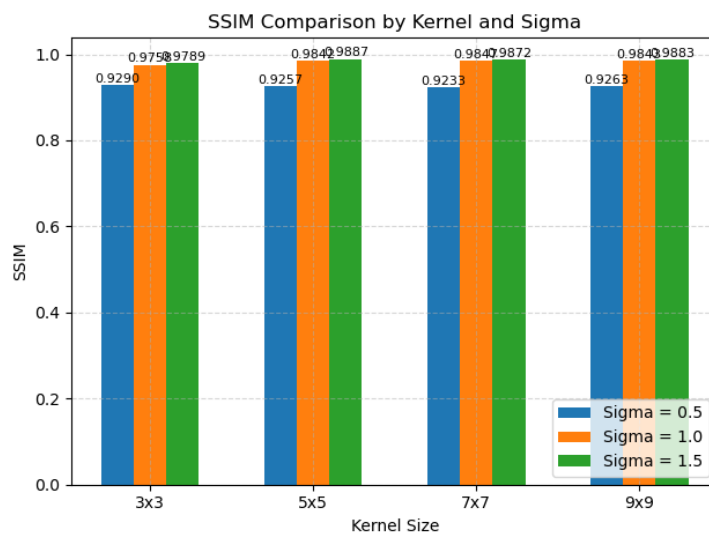
Tabel 4. 3 Tabel PSNR dan SSIM untuk denoising gabungan noise Poisson dan Gaussian Greyscale

| Kernel Size | Sigma | Variansi Noise | SSIM | PSNR |
|-------------|-------|----------------|----------|----------|
| 3 | 0.5 | 0.001 | 0.929042 | 33.80788 |
| 3 | 0.5 | 0.01 | 0.580842 | 23.93288 |
| 3 | 0.5 | 0.1 | 0.164614 | 15.10013 |
| 3 | 1 | 0.001 | 0.975813 | 38.65883 |
| 3 | 1 | 0.01 | 0.831001 | 29.03087 |
| 3 | 1 | 0.1 | 0.393111 | 19.89266 |
| 3 | 1.5 | 0.001 | 0.978933 | 38.88011 |
| 3 | 1.5 | 0.01 | 0.839961 | 29.40633 |
| 3 | 1.5 | 0.1 | 0.430989 | 20.19175 |
| 5 | 0.5 | 0.001 | 0.925694 | 33.81095 |
| 5 | 0.5 | 0.01 | 0.572452 | 23.91844 |
| 5 | 0.5 | 0.1 | 0.160614 | 15.06855 |
| 5 | 1 | 0.001 | 0.984228 | 39.85082 |
| 5 | 1 | 0.01 | 0.884336 | 30.69709 |
| 5 | 1 | 0.1 | 0.51713 | 21.40195 |
| 5 | 1.5 | 0.001 | 0.988716 | 40.35248 |
| 5 | 1.5 | 0.01 | 0.92802 | 32.48267 |
| 5 | 1.5 | 0.1 | 0.64962 | 23.1224 |
| 7 | 0.5 | 0.001 | 0.923346 | 33.82348 |
| 7 | 0.5 | 0.01 | 0.590208 | 23.94316 |
| 7 | 0.5 | 0.1 | 0.16804 | 15.06546 |
| 7 | 1 | 0.001 | 0.984665 | 39.62797 |
| 7 | 1 | 0.01 | 0.882877 | 30.83657 |
| 7 | 1 | 0.1 | 0.526873 | 21.59636 |
| 7 | 1.5 | 0.001 | 0.987184 | 39.10687 |
| 7 | 1.5 | 0.01 | 0.939324 | 32.94915 |

| | | | | |
|---|-----|-------|----------|----------|
| 7 | 1.5 | 0.1 | 0.717379 | 23.89681 |
| 9 | 0.5 | 0.001 | 0.926325 | 33.79926 |
| 9 | 0.5 | 0.01 | 0.565914 | 23.90401 |
| 9 | 0.5 | 0.1 | 0.16237 | 15.08985 |
| 9 | 1 | 0.001 | 0.984266 | 39.49393 |
| 9 | 1 | 0.01 | 0.886155 | 30.78157 |
| 9 | 1 | 0.1 | 0.517589 | 21.55108 |
| 9 | 1.5 | 0.001 | 0.988294 | 39.39864 |
| 9 | 1.5 | 0.01 | 0.943162 | 33.35686 |
| 9 | 1.5 | 0.1 | 0.725415 | 24.05321 |



Gambar 4. 9 Diagram batang perbandingan nilai PSNR dalam mereduksi noise Poisson dan Gaussian rendah berdasarkan nilai sigma dan ukuran kernel



Gambar 4. 10 Diagram batang perbandingan nilai SSIM dalam mereduksi noise Poisson dan Gaussian rendah berdasarkan nilai sigma dan ukuran kernel

Tabel 4.3 berisi hasil perhitungan PSNR dan SSIM untuk *denoising noise* Gaussian dan Poisson dengan berbagai konfigurasi Gaussian Blur. Gambar 4.17 adalah perbandingan nilai PSNR dalam mereduksi *noise* tingkat rendah berdasarkan nilai sigma dan ukuran kernel. Peringkat nilai PSNR tertinggi, di antaranya:

1. Kernel 5x5 dengan sigma 1.5, dengan nilai PSNR 40.3525
2. Kernel 7x7 dengan sigma 1.0, dengan nilai PSNR 39.6280

3. Kernel 9x9 dengan sigma 1.0, dengan nilai PSNR 39.4939

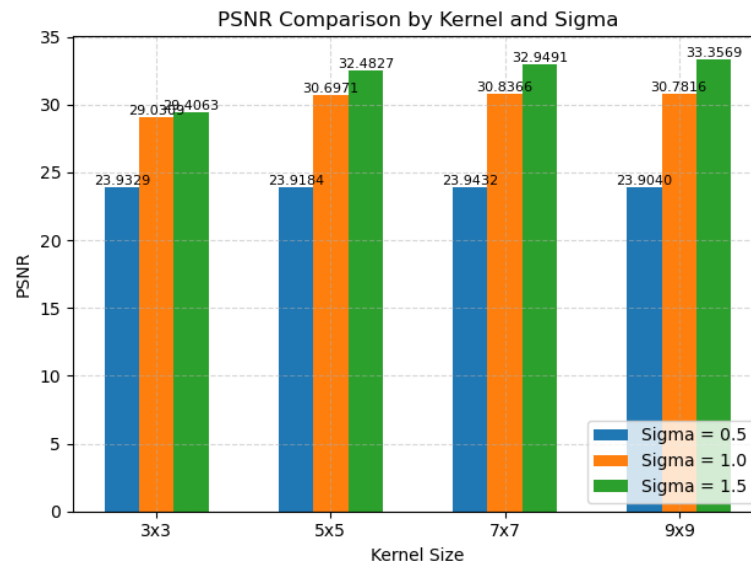
ketiga konfigurasi ini menghasilkan keluaran citra dengan hasil yang baik, dengan PSNR di atas 35 dB. Hal ini menunjukkan bahwa kombinasi ukuran kernel yang lebih besar dengan nilai sigma yang tepat mampu mereduksi *noise* tingkat rendah secara lebih efektif. Kernel 5x5 dengan sigma 1.5 memberikan hasil terbaik karena dapat menangkap detail sekaligus mereduksi *noise* secara optimal. Sementara itu, kernel 7x7 dan 9x9 dengan sigma 1.0 juga menunjukkan performa yang sangat baik, mendekati hasil dari konfigurasi terbaik.

Sebaliknya, konfigurasi dengan sigma 0.5 pada semua ukuran kernel menunjukkan performa yang lebih rendah, dengan PSNR berkisar di angka 33.8 dB. Ini mengindikasikan bahwa nilai sigma yang terlalu kecil kurang mampu mereduksi *noise* secara maksimal, bahkan dengan ukuran kernel yang besar sekalipun.

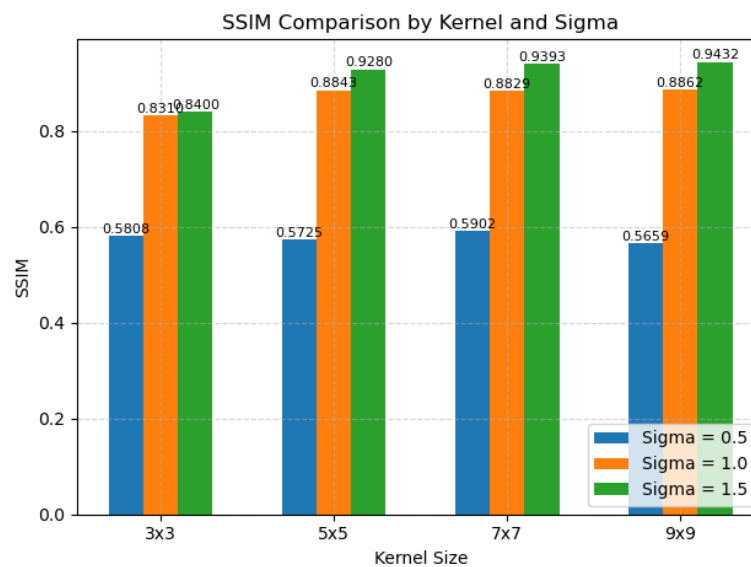
Gambar 4.18 adalah diagram batang perbandingan nilai SSIM dalam mereduksi *noise* tingkat rendah berdasarkan nilai sigma dan ukuran kernel. Seperti yang didapat dari pembahasan nilai PSNR sebelumnya, nilai sigma yang lebih tinggi (1.5) menghasilkan hasil SSIM yang lebih baik, dengan peringkat:

1. Kernel 5x5 dengan nilai sigma 1.5 menghasilkan SSIM dengan hasil 0.9887162393013712
2. Kernel 9x9 dengan nilai sigma 1.5 menghasilkan SSIM dengan nilai 0.9882938185665818
3. Kernel 7x7 dengan nilai sigma 1.5 menghasilkan SSIM dengan nilai 0.987183813825118

Nilai SSIM tertinggi diperoleh oleh konfigurasi kernel 5x5 dengan sigma 1.5. Meskipun demikian, selisih nilai SSIM antara ketiga konfigurasi tersebut sangat kecil, menunjukkan bahwa semua konfigurasi tersebut memberikan kualitas struktur citra yang hampir setara setelah proses *denoising*.



Gambar 4. 11 Diagram batang perbandingan nilai PSNR dalam mereduksi noise Poisson dan Gaussian sedang berdasarkan nilai sigma dan ukuran kernel



Gambar 4. 12 Diagram batang perbandingan nilai SSIM dalam mereduksi noise Poisson dan Gaussian sedang berdasarkan nilai sigma dan ukuran kernel

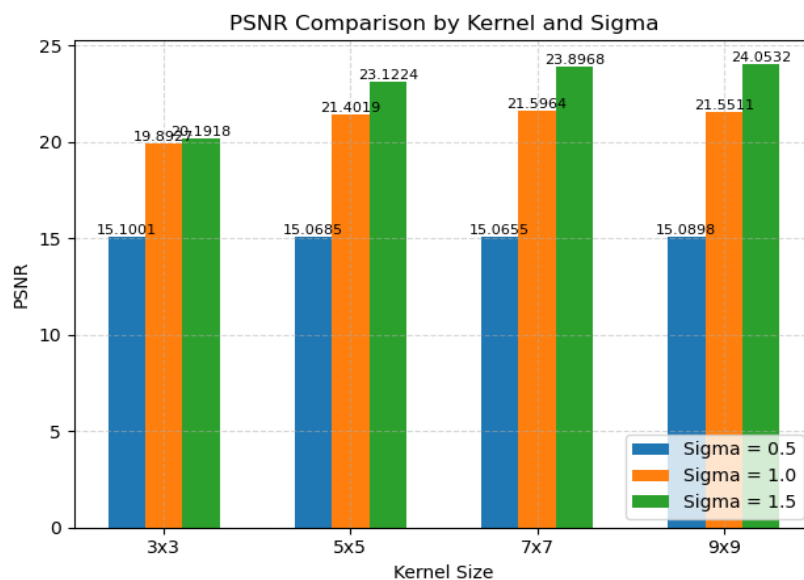
Gambar 4.19 adalah diagram batang perbandingan nilai PSNR dalam mereduksi *noise* sedang berdasarkan nilai sigma dan ukuran kernel. Hasil sigma 1.5 menghasilkan nilai PSNR terbesar di seluruh ukuran kernel yang diuji, dengan peringkat:

1. Kernel 9x9 dengan sigma 1.5 menghasilkan PSNR dengan nilai 33.3569
2. Kernel 7x7 dengan sigma 1.5 menghasilkan PSNR dengan nilai 32.9491
3. Kernel 5x5 dengan sigma 1.5 menghasilkan PSNR dengan nilai 32.4827

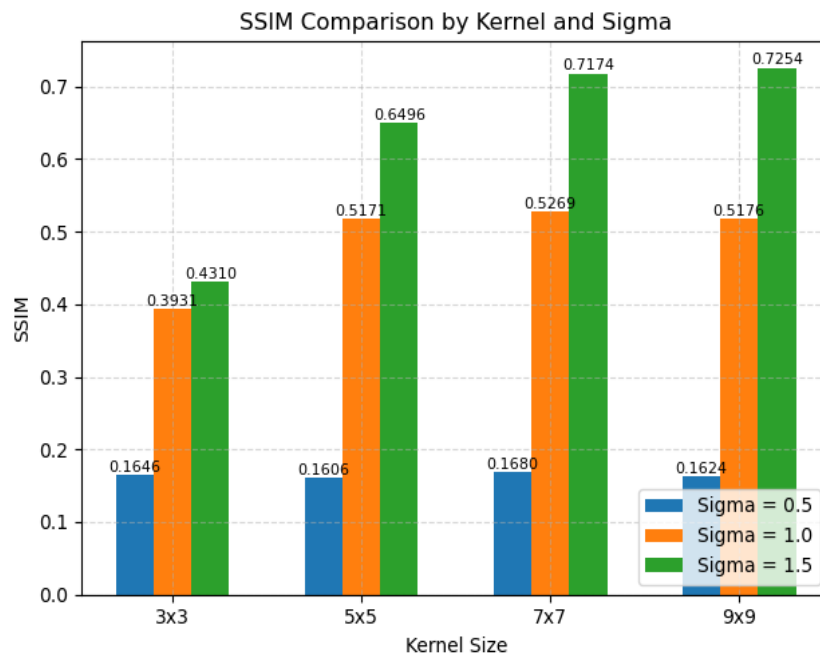
Kernel 9x9 dengan sigma 1.5 menghasilkan nilai PSNR dengan nilai tertinggi. Hal ini juga sama dengan nilai SSIM yang bisa dilihat pada Gambar 4.20, dengan peringkat:

1. Kernel 9x9 dengan sigma 1.5 menghasilkan PSNR dengan nilai 0.9432
2. Kernel 7x7 dengan sigma 1.5 menghasilkan PSNR dengan nilai 0.9393
3. Kernel 5x5 dengan sigma 1.5 menghasilkan PSNR dengan nilai 0.9280

Secara keseluruhan, konfigurasi-konfigurasi dengan sigma 1.5 menunjukkan performa yang sangat baik dalam mereduksi *noise* sedang, dengan nilai PSNR di atas 32 dan SSIM di atas 0.92. Hal ini mengindikasikan bahwa konfigurasi tersebut tidak hanya mampu mengurangi *noise*, tetapi juga mempertahankan kemiripan struktural terhadap citra asli secara efektif.



Gambar 4. 13 Diagram batang perbandingan nilai PSNR dalam mereduksi *noise* Poisson dan Gaussian tinggi berdasarkan nilai sigma dan ukuran kernel



Gambar 4. 14 Diagram batang perbandingan nilai SSIM dalam mereduksi noise Poisson dan Gaussian tinggi berdasarkan nilai sigma dan ukuran kernel

Gambar 4.21 adalah diagram batang perbandingan nilai PSNR dalam mereduksi *noise* tinggi berdasarkan nilai sigma dan ukuran kernel. Hasil sigma 1.5 menghasilkan nilai PSNR terbesar di seluruh ukuran kernel yang diuji, dengan peringkat:

1. Kernel 9x9 dengan sigma 1.5 menghasilkan PSNR dengan nilai 24.0532
2. Kernel 7x7 dengan sigma 1.5 menghasilkan PSNR dengan nilai 23.8968
3. Kernel 5x5 dengan sigma 1.5 menghasilkan PSNR dengan nilai 23.1224

Kernel 9x9 dengan sigma 1.5 menghasilkan nilai PSNR dengan nilai tertinggi. Hal ini juga sama dengan nilai SSIM yang bisa dilihat pada Gambar 4.22, dengan peringkat:

1. Kernel 9x9 dengan sigma 1.5 menghasilkan PSNR dengan nilai 0.7254
2. Kernel 7x7 dengan sigma 1.5 menghasilkan PSNR dengan nilai 0.7174
3. Kernel 5x5 dengan sigma 1.5 menghasilkan PSNR dengan nilai 0.6496

Secara keseluruhan, konfigurasi-konfigurasi dengan sigma 1.5 menunjukkan performa yang buruk dalam mereduksi *noise* tinggi. Nilai SSIM yang relatif rendah ini menandakan bahwa meskipun struktur citra masih dapat dipertahankan sebagian, kualitas hasil pereduksian *noise* tinggi belum optimal. Secara keseluruhan, konfigurasi-konfigurasi dengan sigma 1.5 menunjukkan performa yang kurang memuaskan dalam menangani *noise* tinggi, ditandai dengan nilai PSNR yang berada di bawah 30 dan nilai SSIM yang tidak mencapai 0.75.

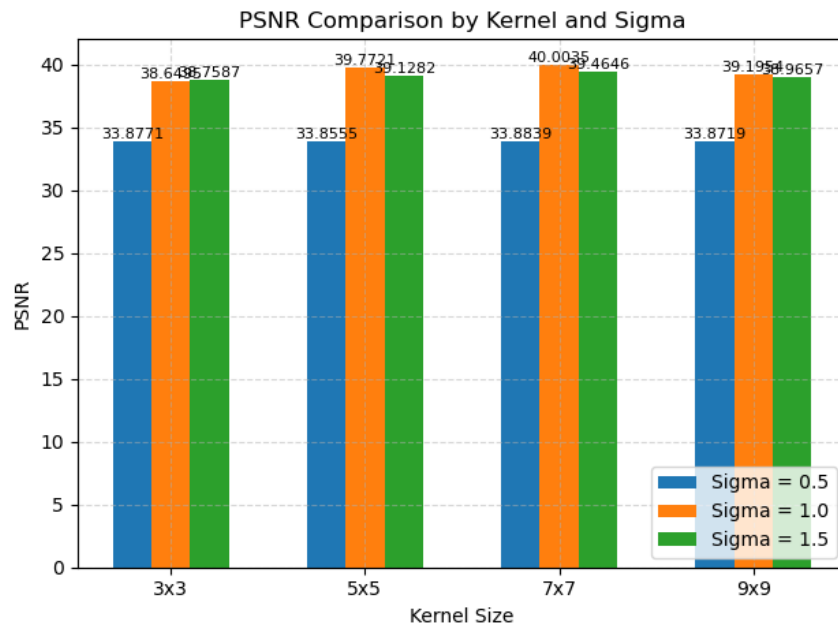
4.5. Evaluasi RGB

4.5.1. Evaluasi PSNR dan SSIM terhadap *denoising noise Gaussian*

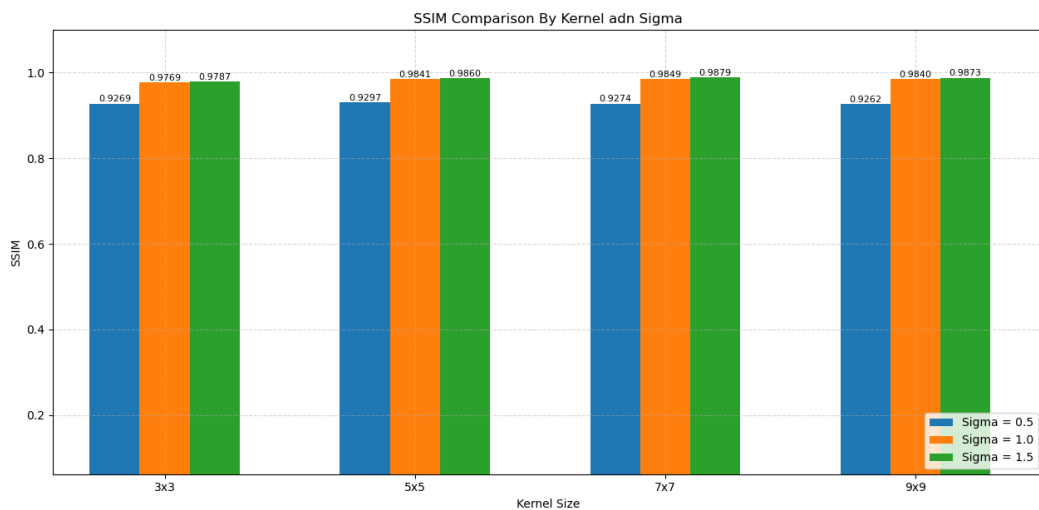
Tabel 4. 4 Tabel PSNR dan SSIM *denoising noise Gaussian RGB*

| Kernel Size | Variansi Noise | Sigma | SSIM | PSNR |
|-------------|----------------|-------|------------|---------|
| 3 | 0.001 | 0.5 | 0.9268705 | 33.8771 |
| 3 | 0.01 | 0.5 | 0.59476703 | 24.0491 |
| 3 | 0.1 | 0.5 | 0.16304693 | 15.1457 |
| 3 | 0.001 | 1.0 | 0.9768761 | 38.6495 |
| 3 | 0.01 | 1.0 | 0.8229659 | 29.0226 |
| 3 | 0.1 | 1.0 | 0.4070542 | 19.7451 |
| 3 | 0.001 | 1.5 | 0.97868854 | 38.7587 |
| 3 | 0.01 | 1.5 | 0.8351227 | 29.4133 |
| 3 | 0.1 | 1.5 | 0.43409163 | 20.0537 |
| 5 | 0.001 | 0.5 | 0.92968446 | 33.8555 |
| 5 | 0.01 | 0.5 | 0.57548416 | 24.0198 |
| 5 | 0.1 | 0.5 | 0.17449519 | 15.1647 |
| 5 | 0.001 | 1.0 | 0.9840681 | 39.7721 |
| 5 | 0.01 | 1.0 | 0.8849014 | 30.6710 |
| 5 | 0.1 | 1.0 | 0.52310926 | 21.1241 |
| 5 | 0.001 | 1.5 | 0.9860288 | 39.1282 |
| 5 | 0.01 | 1.5 | 0.9289444 | 32.4681 |

| | | | | |
|---|-------|-----|------------|---------|
| 5 | 0.1 | 1.5 | 0.6496011 | 22.6274 |
| 7 | 0.001 | 0.5 | 0.92737925 | 33.8839 |
| 7 | 0.01 | 0.5 | 0.5826428 | 24.0235 |
| 7 | 0.1 | 0.5 | 0.17213558 | 15.1496 |
| 7 | 0.001 | 1.0 | 0.98487586 | 40.0035 |
| 7 | 0.01 | 1.0 | 0.88239175 | 30.7670 |
| 7 | 0.1 | 1.0 | 0.5234269 | 21.3414 |
| 7 | 0.001 | 1.5 | 0.9879018 | 39.4646 |
| 7 | 0.01 | 1.5 | 0.9440977 | 33.0144 |
| 7 | 0.1 | 1.5 | 0.7129093 | 23.1204 |
| 9 | 0.001 | 0.5 | 0.9261905 | 33.8719 |
| 9 | 0.01 | 0.5 | 0.58450466 | 24.0399 |
| 9 | 0.1 | 0.5 | 0.16734125 | 15.1796 |
| 9 | 0.001 | 1.0 | 0.9840226 | 39.1954 |
| 9 | 0.01 | 1.0 | 0.8832364 | 30.8033 |
| 9 | 0.1 | 1.0 | 0.52310896 | 21.2453 |
| 9 | 0.001 | 1.5 | 0.98725206 | 38.9657 |
| 9 | 0.01 | 1.5 | 0.9457738 | 33.1816 |
| 9 | 0.1 | 1.5 | 0.7297225 | 23.6234 |



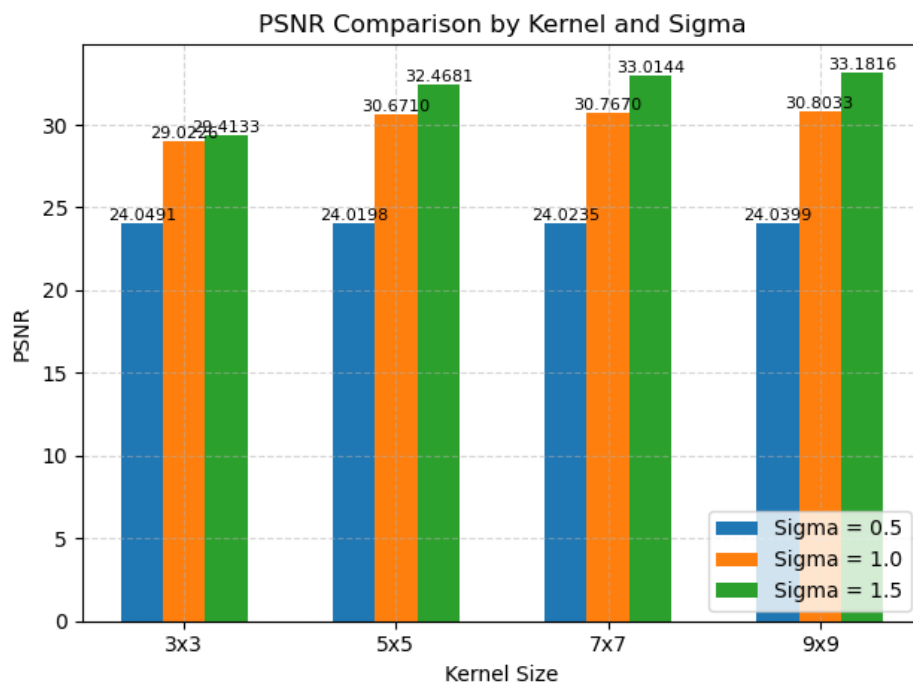
Gambar 4. 15 Diagram batang perbandingan nilai PSNR dalam mereduksi noise Gaussian rendah berdasarkan nilai sigma dan ukuran kernel (RGB)



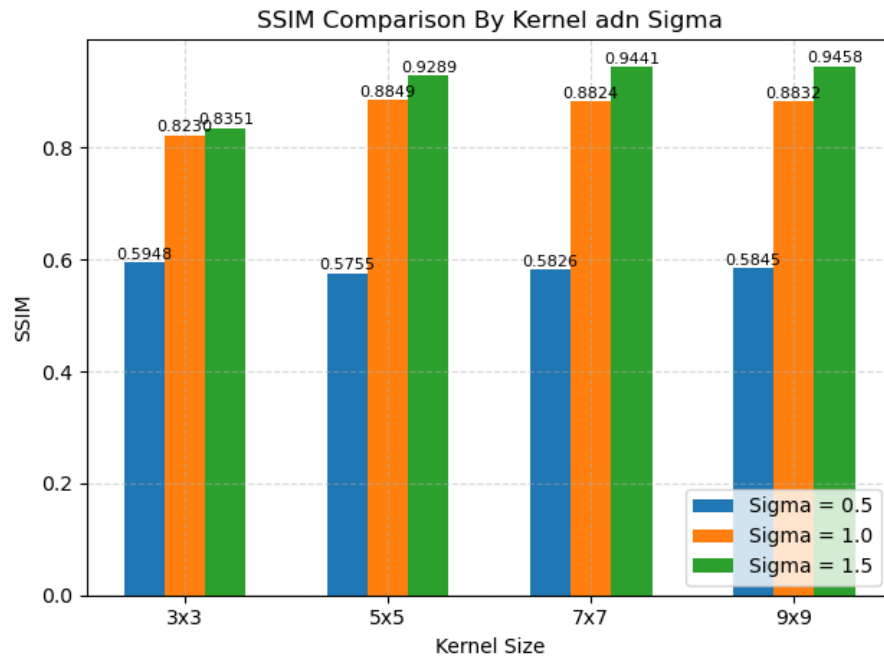
Gambar 4. 16 Diagram batang perbandingan nilai SSIM dalam mereduksi noise Gaussian rendah berdasarkan nilai sigma dan ukuran kernel RGB

Tabel 4.4 berisi hasil perhitungan PSNR dan SSIM untuk *denoising noise* Gaussian dengan berbagai konfigurasi Gaussian Blur untuk gambar RGB. Gambar 4.15 adalah diagram batang perbandingan nilai PSNR dalam mereduksi *noise*

sedang berdasarkan nilai sigma dan ukuran kernel. Untuk citra yang telah ditambahkan *noise* Gaussian dengan variansi sebesar 0.001, evaluasi menggunakan metrik PSNR dan SSIM menunjukkan bahwa konfigurasi Gaussian Blur dengan kernel berukuran 7×7 dan nilai sigma 1.0 menghasilkan performa terbaik dari sisi PSNR, dengan nilai tertinggi sebesar 40.00 dB, yang mengindikasikan tingkat kesamaan numerik yang sangat tinggi antara citra hasil dan citra asli. Di sisi lain, jika dilihat dari metrik SSIM, yang lebih merepresentasikan persepsi visual manusia, konfigurasi terbaik diperoleh pada kernel 7×7 dengan sigma 1.5, menghasilkan nilai SSIM tertinggi sebesar 0.9879. Hasil ini menunjukkan bahwa peningkatan nilai sigma secara umum mampu meningkatkan kualitas visual yang dirasakan, khususnya pada ukuran kernel yang lebih besar. Secara keseluruhan, kombinasi kernel besar (5×5 hingga 9×9) dan sigma antara 1.0 hingga 1.5 cenderung memberikan hasil terbaik baik dalam hal PSNR maupun SSIM, sementara konfigurasi dengan sigma 0.5 memberikan hasil yang relatif lebih rendah pada kedua metrik tersebut.



Gambar 4. 17 Diagram batang perbandingan nilai PSNR dalam mereduksi *noise* Gaussian sedang berdasarkan nilai sigma dan ukuran kernel (RGB)



Gambar 4. 18 Diagram batang perbandingan nilai SSIM dalam mereduksi noise Gaussian sedang berdasarkan nilai sigma dan ukuran kernel (RGB)

Gambar 4.17 merupakan diagram batang pereduksian *noise* sedang berdasarkan nilai sigma dan ukuran kernel. Hasil sigma 1.5 menghasilkan nilai PSNR terbesar di seluruh ukuran kernel yang diuji, dengan peringkat:

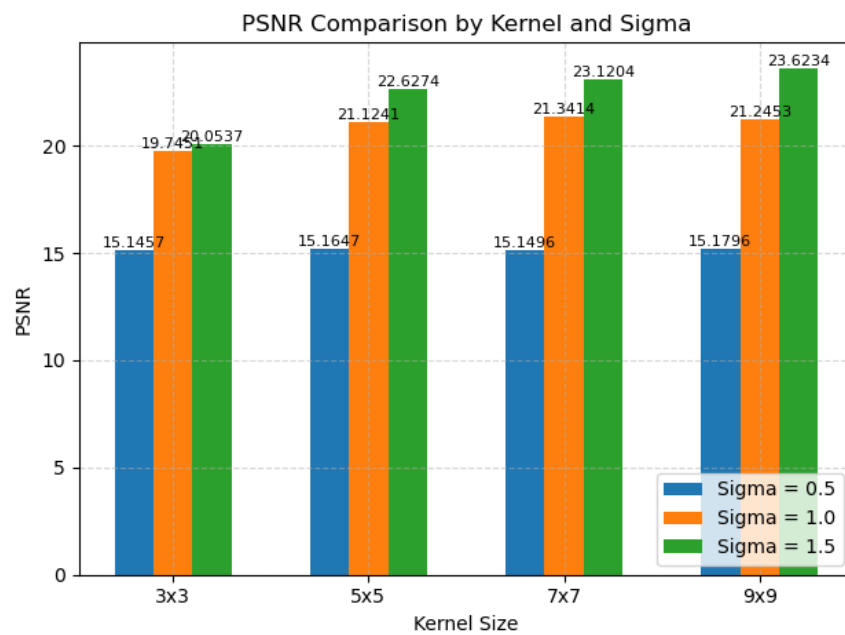
1. Kernel 9×9 dengan sigma 1.5 menghasilkan PSNR dengan nilai 33.1816
2. Kernel 7×7 dengan sigma 1.5 menghasilkan PSNR dengan nilai 33.0144
3. Kernel 5×5 dengan sigma 1.5 menghasilkan PSNR dengan nilai 32.4681

Kernel 9×9 dengan sigma 1.5 menghasilkan nilai PSNR tertinggi. Hal ini juga konsisten dengan hasil evaluasi menggunakan nilai SSIM yang dapat dilihat pada Gambar 4.18, dengan peringkat:

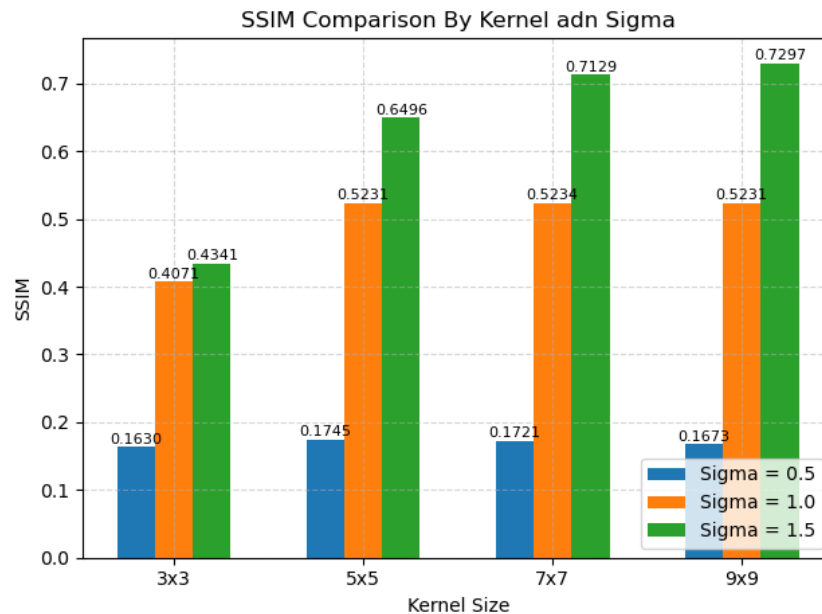
1. Kernel 9×9 dengan sigma 1.5 menghasilkan SSIM dengan nilai 0.9458
2. Kernel 7×7 dengan sigma 1.5 menghasilkan SSIM dengan nilai 0.9441
3. Kernel 5×5 dengan sigma 1.5 menghasilkan SSIM dengan nilai 0.9289

Secara keseluruhan, konfigurasi-konfigurasi dengan sigma 1.5 menunjukkan performa yang sangat baik dalam mereduksi *noise* sedang. Nilai PSNR yang

mendekati atau melebihi 33 dB, serta nilai SSIM yang berada di atas 0.92, menandakan bahwa kualitas visual citra hasil pemrosesan mendekati citra asli dan struktur wajah tetap terjaga. Dengan demikian, penggunaan kernel besar (7×7 atau 9×9) dan nilai sigma tinggi (1.5) menjadi pilihan optimal untuk menangani *noise* Gaussian dengan variansi 0.01.



Gambar 4. 19 Diagram batang perbandingan nilai PSNR dalam mereduksi noise Gaussian tinggi berdasarkan nilai sigma dan ukuran kernel (RGB)



Gambar 4. 20 Diagram batang perbandingan nilai SSIM dalam mereduksi noise Gaussian tinggi berdasarkan nilai sigma dan ukuran kernel (RGB)

Gambar 4.19 merupakan diagram batang pereduksian *noise* tinggi berdasarkan nilai sigma dan ukuran kernel dengan metrik PSNR. Hasil sigma 1.5 kembali menghasilkan nilai PSNR terbesar di seluruh ukuran kernel yang diuji, dengan peringkat:

1. Kernel 9×9 dengan sigma 1.5 menghasilkan PSNR dengan nilai 23.6234
2. Kernel 7×7 dengan sigma 1.5 menghasilkan PSNR dengan nilai 23.1204
3. Kernel 5×5 dengan sigma 1.5 menghasilkan PSNR dengan nilai 22.6274

Kernel 9×9 dengan sigma 1.5 menghasilkan nilai PSNR tertinggi. Hal ini juga selaras dengan hasil evaluasi menggunakan metrik SSIM yang dapat dilihat pada Gambar 4.20, dengan peringkat:

1. Kernel 9×9 dengan sigma 1.5 menghasilkan SSIM dengan nilai 0.7297
2. Kernel 7×7 dengan sigma 1.5 menghasilkan SSIM dengan nilai 0.7129
3. Kernel 5×5 dengan sigma 1.5 menghasilkan SSIM dengan nilai 0.6496

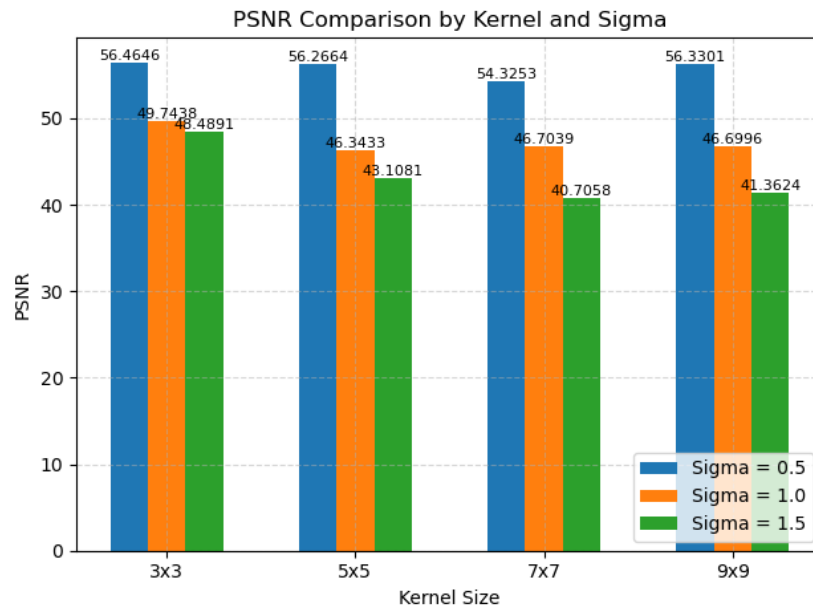
Secara keseluruhan, konfigurasi-konfigurasi dengan sigma 1.5 memberikan hasil paling baik dalam mereduksi *noise* tinggi, meskipun performanya masih

terbatas. Nilai PSNR yang berada di kisaran 23 dB dan SSIM yang tidak mencapai 0.75 menunjukkan bahwa meskipun *noise* dapat dikurangi secara signifikan, detail visual dan struktur citra wajah mengalami degradasi yang cukup besar. Dengan demikian, dalam kasus *noise* tinggi ($\text{var} = 0.1$), Gaussian Blur dengan kernel besar dan sigma tinggi adalah yang paling efektif, tetapi hasilnya tetap belum optimal untuk mempertahankan kualitas visual citra secara menyeluruh.

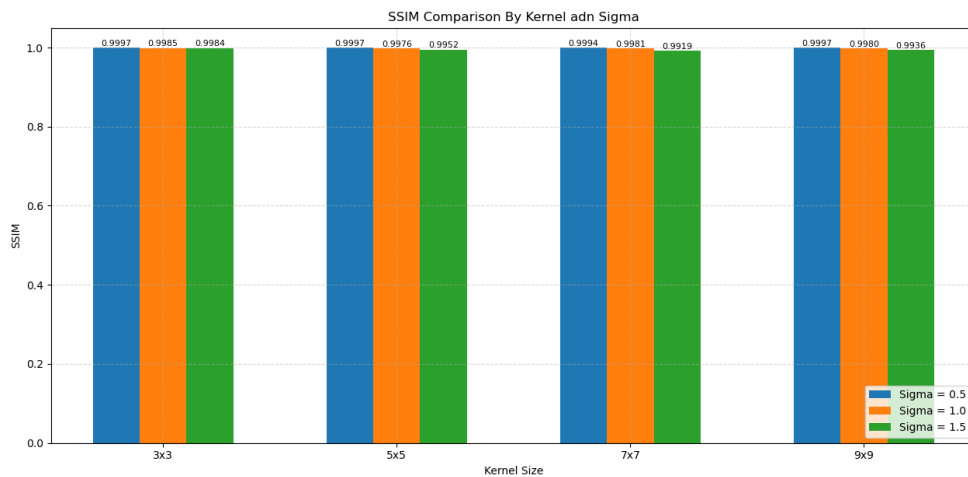
4.5.2. Evaluasi PSNR dan SSIM terhadap *denoising noise Poisson*

Tabel 4. 5 Tabel PSNR dan SSIM *denoising noise Poisson RGB*

| Kernel Size | Sigma | SSIM | PSNR |
|-------------|-------|------------|-------------------|
| 3 | 0.5 | 0.99970007 | 56.46464219618834 |
| 3 | 1.0 | 0.99850464 | 49.74380739153892 |
| 3 | 1.5 | 0.99839130 | 48.48910139007784 |
| 5 | 0.5 | 0.99969065 | 56.26639441772829 |
| 5 | 1.0 | 0.99762136 | 46.34334360519692 |
| 5 | 1.5 | 0.99515570 | 43.10808120698058 |
| 7 | 0.5 | 0.99944746 | 54.32532147291994 |
| 7 | 1.0 | 0.99809855 | 46.70390040700605 |
| 7 | 1.5 | 0.99194735 | 40.70581989422924 |
| 9 | 0.5 | 0.99969006 | 56.33010338608242 |
| 9 | 1.0 | 0.99795663 | 46.69962028330838 |
| 9 | 1.5 | 0.99364920 | 41.36244601776850 |



Gambar 4. 21 Diagram batang perbandingan nilai PSNR dalam mereduksi noise Poisson berdasarkan nilai sigma dan ukuran kernel (RGB)



Gambar 4. 22 Diagram batang perbandingan nilai SSIM dalam mereduksi noise Poisson berdasarkan nilai sigma dan ukuran kernel (RGB)

Tabel 4.5 berisi hasil perhitungan PSNR dan SSIM untuk *denoising noise* Poisson dengan berbagai konfigurasi Gaussian Blur untuk gambar RGB. Gambar 4.21 merupakan diagram batang pereduksian *noise* Poisson berdasarkan nilai sigma dan ukuran kernel berdasarkan metrik PSNR. Berbeda dengan *noise* Gaussian, pada

noise jenis Poisson hasil terbaik justru diperoleh pada sigma kecil, khususnya sigma 0.5, dengan peringkat PSNR tertinggi sebagai berikut:

1. Kernel 3×3 dengan sigma 0.5 menghasilkan PSNR sebesar 56.4646
2. Kernel 5×5 dengan sigma 0.5 menghasilkan PSNR sebesar 56.2664
3. Kernel 9×9 dengan sigma 0.5 menghasilkan PSNR sebesar 56.3301

Kernel 3×3 dengan sigma 0.5 memberikan hasil PSNR tertinggi, yang menunjukkan bahwa smoothing ringan sudah cukup efektif untuk mereduksi gangguan akibat fluktuasi foton tanpa merusak detail citra. Hal ini juga tercermin dari hasil evaluasi SSIM, dengan peringkat sebagai berikut:

1. Kernel 3×3 dengan sigma 0.5 menghasilkan SSIM sebesar 0.9997
2. Kernel 5×5 dengan sigma 0.5 menghasilkan SSIM sebesar 0.9997
3. Kernel 9×9 dengan sigma 0.5 menghasilkan SSIM sebesar 0.9997

Secara umum, semua konfigurasi dengan sigma 0.5 menunjukkan performa yang sangat tinggi dalam mereduksi *noise* Poisson. Namun, seiring dengan meningkatnya nilai sigma, baik nilai PSNR maupun SSIM mulai menurun. Contohnya, pada kernel 9×9 dengan sigma 1.5, PSNR turun menjadi 41.3624 dan SSIM menjadi 0.9936, menunjukkan bahwa smoothing yang terlalu agresif mulai mengaburkan detail citra penting.

Dengan demikian, untuk menangani *noise* Poisson, konfigurasi Gaussian Blur yang optimal adalah kernel kecil hingga sedang (3×3 hingga 5×5) dengan sigma rendah (0.5). Ini efektif mereduksi *noise* tanpa mengorbankan kualitas visual maupun struktur citra secara signifikan.

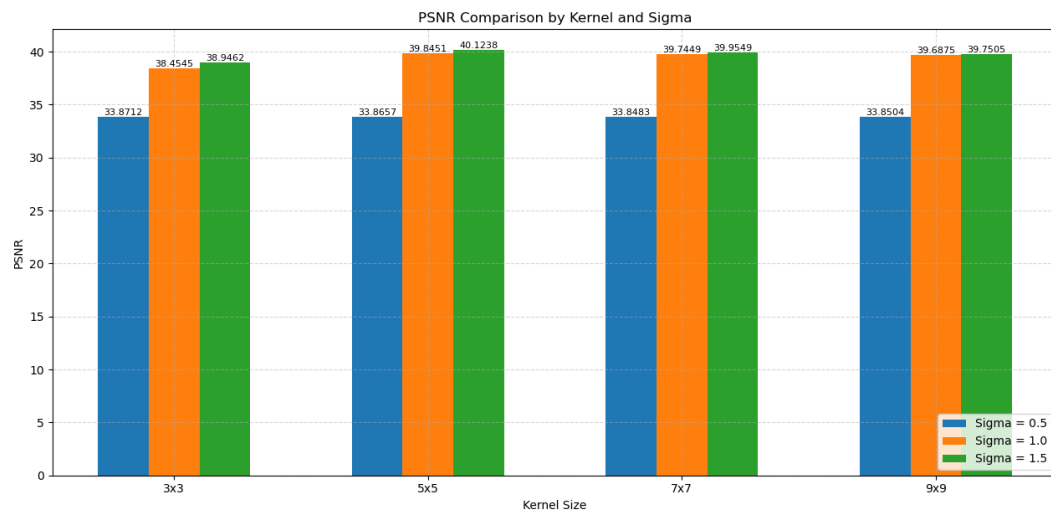
4.5.3. Evaluasi PSNR dan SSIM terhadap *denoising noise* gabungan Poisson dan Gaussian

Tabel 4. 6 Tabel PSNR dan SSIM *denoising noise* Poisson dan Gaussian RGB

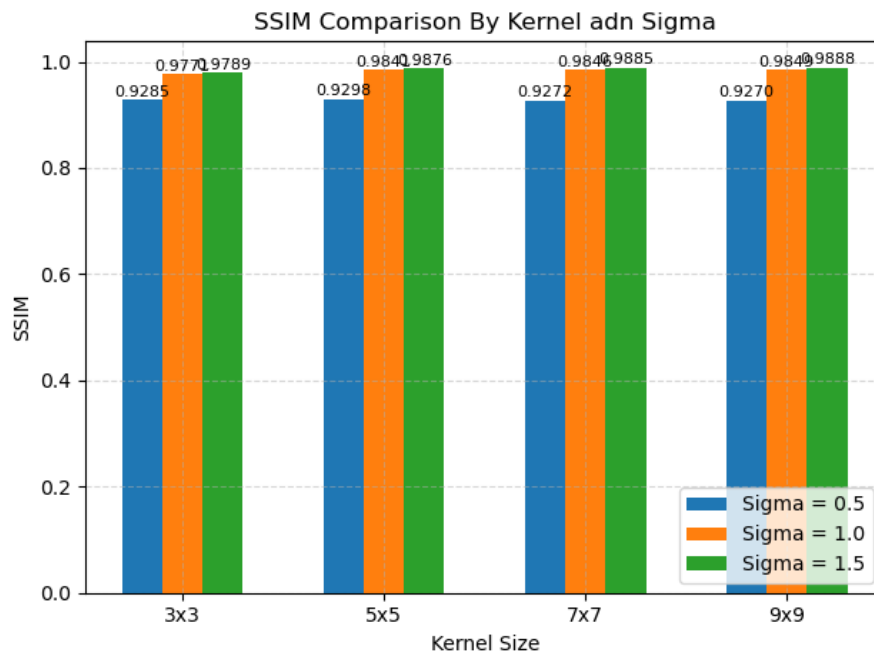
| Kernel Size | Variansi <i>Noise</i> | Sigma | SSIM | PSNR |
|-------------|-----------------------|-------|------------|---------|
| 3 | 0.001 | 0.5 | 0.92850435 | 33.8712 |
| 3 | 0.01 | 0.5 | 0.58904720 | 24.0410 |

| | | | | |
|---|-------|-----|------------|---------|
| 3 | 0.1 | 0.5 | 0.17538404 | 15.1363 |
| 3 | 0.001 | 1.0 | 0.97711990 | 38.4545 |
| 3 | 0.01 | 1.0 | 0.82591990 | 29.0082 |
| 3 | 0.1 | 1.0 | 0.42209990 | 19.7062 |
| 3 | 0.001 | 1.5 | 0.97890820 | 38.9462 |
| 3 | 0.01 | 1.5 | 0.83929574 | 29.3932 |
| 3 | 0.1 | 1.5 | 0.43037290 | 20.1032 |
| 5 | 0.001 | 0.5 | 0.92981005 | 33.8657 |
| 5 | 0.01 | 0.5 | 0.58903337 | 24.0513 |
| 5 | 0.1 | 0.5 | 0.19222179 | 15.2282 |
| 5 | 0.001 | 1.0 | 0.98414250 | 39.8451 |
| 5 | 0.01 | 1.0 | 0.88478720 | 30.6636 |
| 5 | 0.1 | 1.0 | 0.51012610 | 21.2803 |
| 5 | 0.001 | 1.5 | 0.98757110 | 40.1238 |
| 5 | 0.01 | 1.5 | 0.92300490 | 32.4323 |
| 5 | 0.1 | 1.5 | 0.64834320 | 22.5248 |
| 7 | 0.001 | 0.5 | 0.92722100 | 33.8483 |
| 7 | 0.01 | 0.5 | 0.59347534 | 24.0253 |
| 7 | 0.1 | 0.5 | 0.17315390 | 15.1833 |
| 7 | 0.001 | 1.0 | 0.98462164 | 39.7449 |
| 7 | 0.01 | 1.0 | 0.88682050 | 30.7886 |
| 7 | 0.1 | 1.0 | 0.52335405 | 21.3185 |
| 7 | 0.001 | 1.5 | 0.98853314 | 39.9549 |
| 7 | 0.01 | 1.5 | 0.94375370 | 32.7999 |
| 7 | 0.1 | 1.5 | 0.72188014 | 23.3819 |
| 9 | 0.001 | 0.5 | 0.92703930 | 33.8504 |
| 9 | 0.01 | 0.5 | 0.58681446 | 24.0080 |
| 9 | 0.1 | 0.5 | 0.17020957 | 15.1694 |
| 9 | 0.001 | 1.0 | 0.98491730 | 39.6875 |
| 9 | 0.01 | 1.0 | 0.88214284 | 30.8337 |

| | | | | |
|---|-------|-----|------------|---------|
| 9 | 0.1 | 1.0 | 0.53419507 | 21.2146 |
| 9 | 0.001 | 1.5 | 0.98875870 | 39.7505 |
| 9 | 0.01 | 1.5 | 0.94274586 | 32.9846 |
| 9 | 0.1 | 1.5 | 0.72580576 | 23.6332 |



Gambar 4. 23 Diagram batang perbandingan nilai PSNR dalam mereduksi noise Gaussian dan Poisson rendah berdasarkan nilai sigma dan ukuran kernel (RGB)



Gambar 4. 24 Diagram batang perbandingan nilai SSIM dalam mereduksi noise Gaussian dan Poisson rendah berdasarkan nilai sigma dan ukuran kernel (RGB)

Tabel 4.6 berisi hasil perhitungan PSNR dan SSIM untuk *denoising noise* Gaussian dan Poisson dengan berbagai konfigurasi Gaussian Blur untuk gambar RGB. Gambar 4.23 merupakan diagram batang dalam pereduksian *noise* gabungan (Gaussian + Poisson) ringan berdasarkan nilai sigma dan ukuran kernel berdasarkan PSNR. Hasil terbaik diperoleh pada konfigurasi sigma 1.5, dengan peringkat PSNR sebagai berikut:

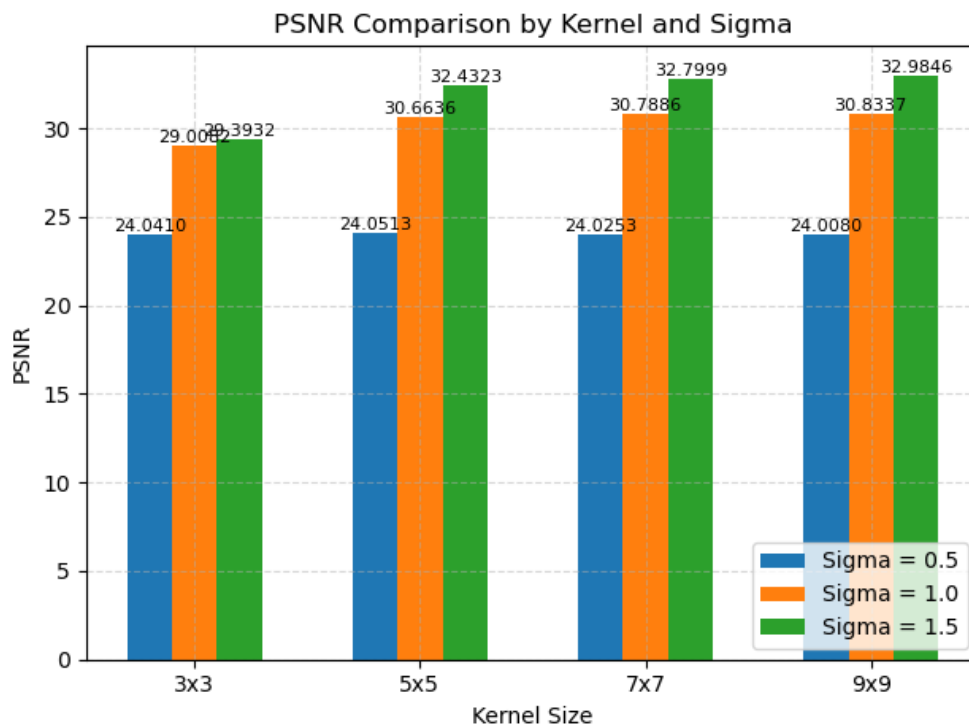
1. Kernel 5×5 dengan sigma 1.5 menghasilkan PSNR sebesar 40.1238
2. Kernel 7×7 dengan sigma 1.5 menghasilkan PSNR sebesar 39.9549
3. Kernel 3×3 dengan sigma 1.5 menghasilkan PSNR sebesar 38.9462

Kernel 5×5 dengan sigma 1.5 memberikan hasil PSNR tertinggi, menandakan bahwa konfigurasi ini sangat efektif dalam mereduksi gangguan dari *noise* ringan gabungan. Evaluasi SSIM juga memperkuat temuan ini, dengan peringkat tertinggi sebagai berikut:

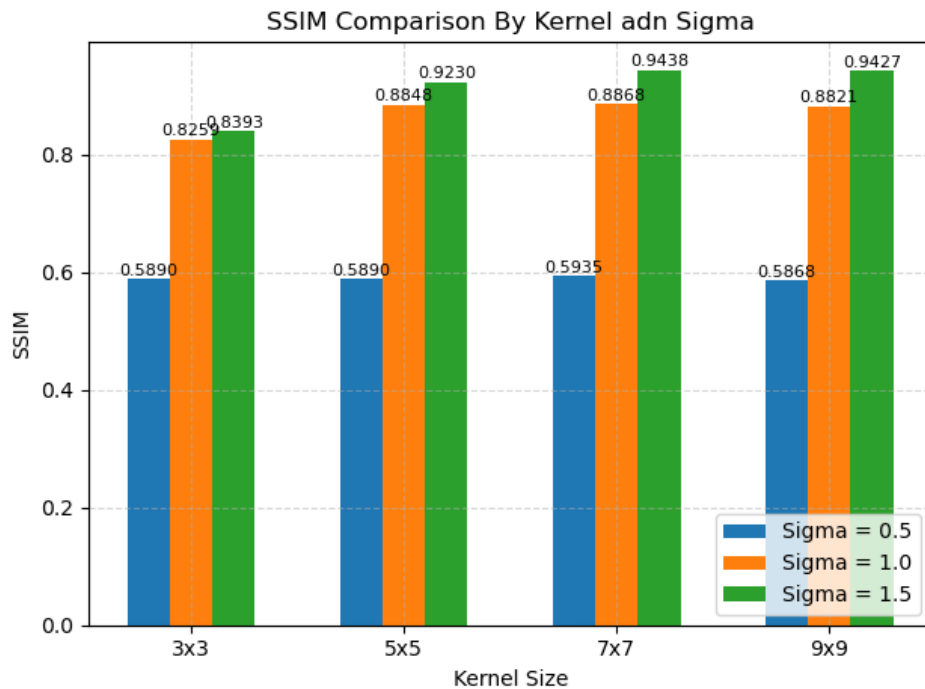
1. Kernel 9×9 dengan sigma 1.5 menghasilkan SSIM sebesar 0.9888
2. Kernel 7×7 dengan sigma 1.5 menghasilkan SSIM sebesar 0.9885

3. Kernel 5×5 dengan sigma 1.5 menghasilkan SSIM sebesar 0.9876

Secara keseluruhan, konfigurasi dengan sigma 1.5 memberikan hasil paling optimal dalam mereduksi *noise* gabungan dengan varian rendah. Nilai PSNR di atas 38 dB dan SSIM mendekati 1.0 menunjukkan bahwa detail penting citra tetap terjaga dengan baik, dan degradasi akibat *noise* dapat diminimalkan secara signifikan. Oleh karena itu, untuk menangani *noise* gabungan dengan intensitas ringan, konfigurasi kernel 5×5 atau lebih besar dan sigma tinggi (1.5) merupakan pilihan yang paling disarankan.



Gambar 4. 25 Diagram batang perbandingan nilai PSNR dalam mereduksi *noise* Gaussian dan Poisson sedang berdasarkan nilai sigma dan ukuran kernel (RGB)



Gambar 4. 26 Diagram batang perbandingan nilai SSIM dalam mereduksi noise Gaussian dan Poisson sedang berdasarkan nilai sigma dan ukuran kernel (RGB)

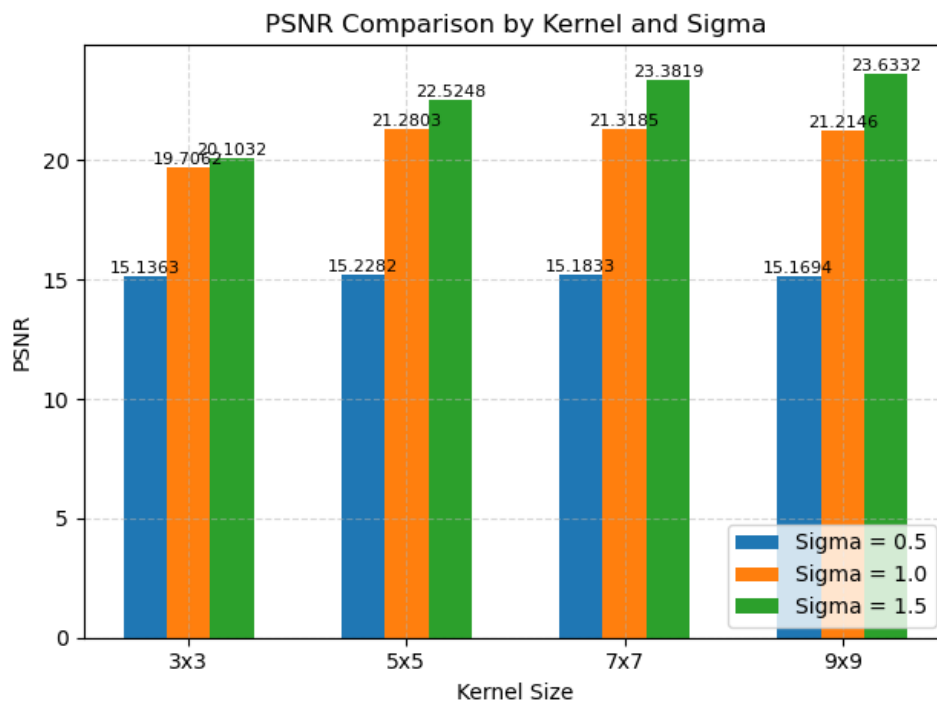
Gambar 4.25 merupakan diagram batang pereduksian *noise* gabungan (Gaussian + Poisson) dengan intensitas sedang, berdasarkan nilai sigma dan ukuran kernel berdasarkan PSNR. Hasil terbaik diperoleh pada konfigurasi sigma 1.5, dengan peringkat PSNR sebagai berikut:

1. Kernel 9×9 dengan sigma 1.5 menghasilkan PSNR sebesar 32.9846
2. Kernel 7×7 dengan sigma 1.5 menghasilkan PSNR sebesar 32.7999
3. Kernel 5×5 dengan sigma 1.5 menghasilkan PSNR sebesar 32.4323

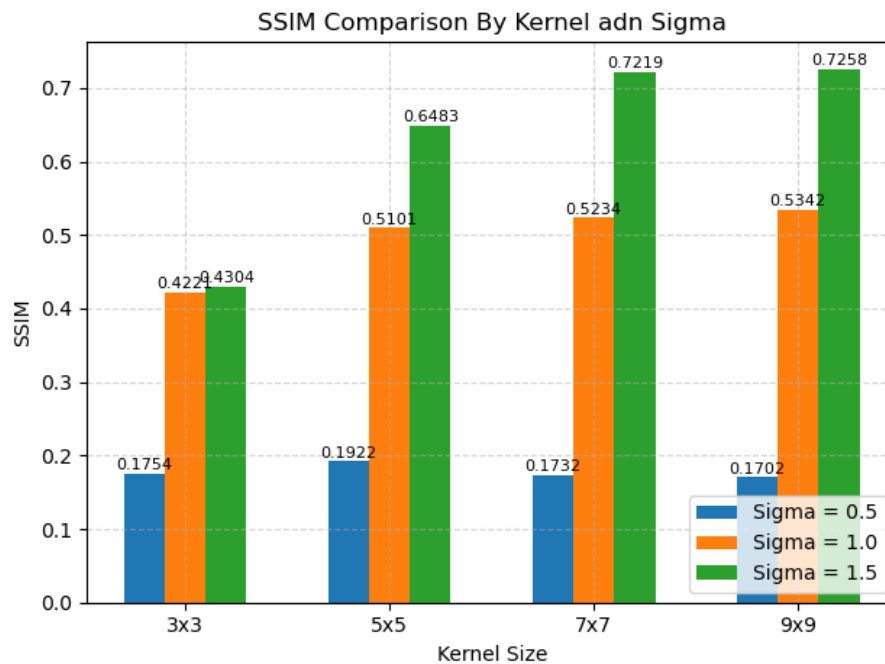
Kernel 9×9 dengan sigma 1.5 memberikan hasil PSNR tertinggi, menunjukkan bahwa penggunaan kernel besar dan sigma tinggi efektif dalam mereduksi *noise* gabungan pada tingkat sedang. Evaluasi berdasarkan SSIM juga menunjukkan konsistensi, dengan peringkat sebagai berikut:

1. Kernel 7×7 dengan sigma 1.5 menghasilkan SSIM sebesar 0.9438
2. Kernel 9×9 dengan sigma 1.5 menghasilkan SSIM sebesar 0.9427
3. Kernel 5×5 dengan sigma 1.5 menghasilkan SSIM sebesar 0.9230

Secara keseluruhan, konfigurasi-konfigurasi dengan sigma 1.5 dan kernel besar ($\geq 5 \times 5$) menunjukkan performa yang sangat baik dalam mereduksi *noise* gabungan sedang, ditandai dengan nilai PSNR di atas 32 dB dan SSIM di atas 0.92. Hal ini menunjukkan bahwa meskipun *noise* cukup signifikan, detail wajah tetap dapat dipertahankan dengan baik, dan kualitas visual citra yang dihasilkan tetap tinggi. Maka dari itu, untuk kondisi *noise* gabungan dengan varian sedang, Gaussian Blur dengan sigma 1.5 dan kernel 9×9 atau 7×7 adalah konfigurasi yang paling disarankan.



Gambar 4. 27Diagram batang perbandingan nilai PSNR dalam mereduksi noise Gaussian dan Poisson tinggi berdasarkan nilai sigma dan ukuran kernel (RGB)



Gambar 4. 28 Diagram batang perbandingan nilai SSIM dalam mereduksi noise Gaussian dan Poisson tinggi berdasarkan nilai sigma dan ukuran kernel (RGB)

Gambar 4.27 dan Gambar 4.28 merupakan diagram batang pereduksian *noise* gabungan (Gaussian + Poisson) dengan intensitas tinggi, berdasarkan nilai sigma dan ukuran kernel. Hasil terbaik kembali diperoleh pada konfigurasi sigma 1.5, dengan peringkat PSNR sebagai berikut:

1. Kernel 9×9 dengan sigma 1.5 menghasilkan PSNR sebesar 23.6332
2. Kernel 7×7 dengan sigma 1.5 menghasilkan PSNR sebesar 23.3819
3. Kernel 5×5 dengan sigma 1.5 menghasilkan PSNR sebesar 22.5248

Kernel 9×9 dengan sigma 1.5 memberikan nilai PSNR tertinggi, menunjukkan bahwa smoothing yang lebih luas mampu mereduksi *noise* tinggi secara lebih efektif. Namun demikian, nilai PSNR yang berada di kisaran 23 dB masih tergolong rendah, mencerminkan bahwa *noise* belum sepenuhnya tereliminasi. Evaluasi SSIM menunjukkan peringkat sebagai berikut:

1. Kernel 9×9 dengan sigma 1.5 menghasilkan SSIM sebesar 0.7258
2. Kernel 7×7 dengan sigma 1.5 menghasilkan SSIM sebesar 0.7219
3. Kernel 5×5 dengan sigma 1.5 menghasilkan SSIM sebesar 0.6483

Meskipun SSIM dari konfigurasi terbaik mendekati 0.73, angka ini tetap menandakan penurunan struktur dan kualitas visual yang cukup signifikan akibat *noise* yang tinggi. Secara keseluruhan, konfigurasi sigma 1.5 dan kernel besar (9×9 atau 7×7) merupakan pilihan paling efektif untuk mereduksi *noise* gabungan berat, namun hasilnya masih belum optimal untuk menghasilkan citra yang benar-benar bersih. Maka dari itu, pada kondisi *noise* berat seperti ini, dibutuhkan metode denoising tambahan atau gabungan teknik lain untuk memperoleh hasil visual yang lebih baik.

4.6. Pembahasan

Berdasarkan hasil pengujian terhadap citra grayscale yang telah ditambahkan *noise* aditif Gaussian, Poisson, dan gabungan Gaussian serta Poisson, diperoleh bahwa konfigurasi optimal untuk Gaussian Blur adalah penggunaan kernel berukuran 9×9 dengan nilai sigma sebesar 1.5. Pemilihan konfigurasi ini didasarkan pada performa PSNR dan SSIM tertinggi dalam mereduksi *noise* tingkat sedang hingga tinggi. Nilai PSNR dan SSIM yang dihasilkan dalam pereduksian *noise* Gaussian dan gabungan Poisson-Gaussian berada di atas 30 dB untuk *noise* tingkat rendah dan sedang, yang menunjukkan kualitas citra yang baik. Walaupun konfigurasi ini menghasilkan PSNR yang sedikit lebih rendah dibandingkan kernel 3×3 dengan sigma 0.5 dalam mereduksi *noise* Poisson, hasilnya tetap sangat baik dengan PSNR di atas 40 dB. Untuk *noise* tingkat tinggi, performa konfigurasi ini memang menurun dengan nilai PSNR sekitar 24 dB dan SSIM sekitar 0.72, namun tetap menjadi yang terbaik di antara seluruh konfigurasi yang diuji.

Ukuran kernel paling optimal kedua adalah 5×5 dan 7×7 dengan sigma masing-masing 1.5. Ukuran kernel yang lebih kecil ini dapat menjadi pilihan yang baik apabila ingin menghemat komputasi, karena memberikan performa yang cukup baik dengan waktu proses yang lebih singkat dibandingkan kernel 9×9 .

Hasil perbandingan antara metrik PSNR dan SSIM juga menunjukkan konsistensi yang baik, tanpa adanya kontradiksi signifikan di antara keduanya. Dengan kata lain, konfigurasi yang menghasilkan nilai PSNR tertinggi umumnya

juga menunjukkan nilai SSIM yang tinggi, sehingga validitas kualitas citra hasil pemrosesan dapat dipertanggungjawabkan secara objektif maupun perseptual. Hal ini menunjukkan bahwa kedua metrik tersebut saling mendukung dalam mengevaluasi efektivitas Gaussian Blur dalam mereduksi berbagai jenis *noise*, serta memperkuat keandalan konfigurasi yang terpilih sebagai yang paling optimal.

Pada pengujian terhadap citra RGB (berwarna), tren hasil menunjukkan kecenderungan yang serupa dengan citra grayscale. Namun, karena pengolahan dilakukan secara independen pada setiap kanal warna (merah, hijau, dan biru), nilai PSNR dan SSIM yang dihasilkan umumnya lebih tinggi dibandingkan dengan grayscale, terutama pada *noise* ringan hingga sedang. Hal ini disebabkan oleh fakta bahwa kanal warna cenderung memiliki informasi berbeda, sehingga *noise* pada satu kanal dapat dikompensasi oleh kanal lain dalam persepsi visual. Konfigurasi kernel 9×9 dengan sigma 1.5 tetap menunjukkan performa terbaik dalam mereduksi *noise* pada citra RGB, dengan nilai PSNR yang sering kali melebihi 40 dB dan SSIM yang mendekati 1.0 untuk *noise* ringan.

Namun, pada *noise* berat (varian tinggi), meskipun penurunan kualitas masih terjadi, warna dan struktur citra RGB tetap lebih terjaga dibandingkan grayscale, karena efek smoothing tersebar di tiga kanal dan tidak terfokus pada satu sumber informasi saja. Dengan demikian, konfigurasi optimal untuk citra RGB tetap sama, namun dengan tambahan keuntungan dari persepsi warna yang lebih toleran terhadap gangguan *noise*. Oleh karena itu, untuk implementasi Gaussian Blur pada citra wajah berwarna dalam kondisi *noise* nyata, pendekatan yang digunakan terbukti efektif dan konsisten baik secara numerik (PSNR) maupun secara visual (SSIM)..

BAB V

KESIMPULAN DAN SARAN

5.1. Kesimpulan

Berdasarkan hasil pengujian yang dilakukan terhadap citra wajah dari dataset LFW yang telah diberi noise Gaussian, Poisson, dan kombinasi keduanya, dapat disimpulkan bahwa metode Gaussian Blur mampu mereduksi noise secara efektif, terutama pada tingkat noise rendah hingga sedang. Konfigurasi yang terbukti paling optimal adalah kernel berukuran 9×9 dengan nilai sigma 1.5, yang secara konsisten menghasilkan nilai PSNR dan SSIM tertinggi pada berbagai jenis noise, baik pada citra grayscale maupun RGB.

Untuk citra grayscale, konfigurasi kernel 9×9 dan sigma 1.5 memberikan hasil terbaik pada:

- Noise Gaussian var = 0.001: PSNR = 38.97 dB, SSIM = 0.9873
- Noise Gaussian var = 0.01: PSNR = 33.18 dB, SSIM = 0.9458
- Noise Gaussian var = 0.1: PSNR = 23.62 dB, SSIM = 0.7297
- Gabungan Gaussian + Poisson var = 0.001: PSNR = 39.75 dB, SSIM = 0.9888
- Gabungan Gaussian + Poisson var = 0.01: PSNR = 32.98 dB, SSIM = 0.9427
- Gabungan Gaussian + Poisson var = 0.1: PSNR = 23.63 dB, SSIM = 0.7258

Pada citra RGB, hasilnya cenderung lebih tinggi, dengan PSNR yang dapat mencapai hingga 56.46 dB dan SSIM di atas 0.999, khususnya pada noise ringan. Ini menunjukkan bahwa smoothing pada tiga kanal warna memberikan efek pengurangan noise yang lebih efektif secara visual, tanpa mengorbankan terlalu banyak detail.

Untuk noise ringan, konfigurasi sigma = 0.5 dan kernel kecil seperti 3×3 atau 5×5 juga dapat menghasilkan PSNR tinggi (bahkan di atas 56 dB) dan SSIM

hampir sempurna (> 0.999). Namun, pada noise sedang hingga tinggi, konfigurasi sigma 1.5 dan kernel besar terbukti paling efektif menjaga struktur dan kualitas visual citra.

Secara keseluruhan, penelitian ini menunjukkan bahwa Gaussian Blur merupakan metode denoising yang andal dan efisien dalam berbagai skenario. Kombinasi kernel 9×9 dan sigma 1.5 direkomendasikan sebagai konfigurasi paling optimal untuk mereduksi noise Gaussian, Poisson, maupun gabungan keduanya, baik pada citra grayscale maupun RGB, dengan hasil PSNR ≥ 33 dB dan SSIM ≥ 0.94 pada noise sedang, serta tetap unggul pada noise tinggi dibanding konfigurasi lainnya.

5.2.Saran

Saran dari penelitian ini adalah sebagai berikut:

1. Penggunaan Adaptive Gaussian Blur

Penelitian selanjutnya disarankan untuk menerapkan metode Adaptive Gaussian Blur, di mana ukuran kernel dan nilai sigma disesuaikan secara lokal terhadap karakteristik citra. Pendekatan ini berpotensi meningkatkan efektivitas pereduksian *noise*, terutama pada citra dengan distribusi *noise* yang tidak merata, serta mempertahankan lebih banyak detail penting seperti tepi dan tekstur halus.

2. Pengujian pada Dataset yang Lebih Luas

Disarankan untuk menggunakan lebih banyak citra dari berbagai jenis (misalnya citra medis, satelit, atau citra alami) agar hasil evaluasi lebih general dan dapat diterapkan dalam konteks yang lebih luas.

DAFTAR PUSTAKA

- Aripin, S., Sarumaha, L., & Sinaga, M. N. (2020). *Implementasi Metode Laplacian of Gaussian Dalam Deteksi Tepi Citra Gigi Berlubang*.
- B, A. P., & K, G. S. (2016). *Denoising*. IOSR Journal of Electronics and Communication Engineering (IOSR-JECE).
- Bharati, S., Khan, T. Z., Podder, P., & Hung, N. Q. (2021). A Comparative Analysis of Image Denoising Problem: *Noise Models, Denoising Filters and Applications*. In A. E. Hassanien, A. Khamparia, D. Gupta, K. Shankar, & A. Slowik (Eds.), *Cognitive Internet of Medical Things for Smart Healthcare* (Vol. 311, pp. 49–66). Springer International Publishing. https://doi.org/10.1007/978-3-030-55833-8_3
- Cai, R., Zhang, L., Chen, C., Hu, Y., & Kot, A. (2024). Learning deep forest for face anti-spoofing: An alternative to the neural network against adversarial attacks. *Electronic Research Archive*, 32(10), 5592–5614. <https://doi.org/10.3934/era.2024259>
- Dharavath, K. (2014). Improving Face Recognition Rate with Image Preprocessing. *Indian Journal of Science and Technology*, 7(8), 1170–1175. <https://doi.org/10.17485/ijst/2014/v7i8.26>
- Fan, L., Zhang, F., Fan, H., & Zhang, C. (2019). Brief review of image denoising techniques. *Visual Computing for Industry, Biomedicine, and Art*, 2(1), 7. <https://doi.org/10.1186/s42492-019-0016-7>
- Gazali, W., Soeparno, H., & Ohliati, J. (2012). *PENERAPAN METODE KONVOLUSI DALAM PENGOLAHAN CITRA DIGITAL*.
- Gonzalez, R. C., & Woods, R. E. (2018). *Digital image processing*. Pearson.
- Ha, W., & Shin, C. (2021). Seismic Random *Noise* Attenuation in the Laplace Domain Using Singular Value Decomposition. *IEEE Access*, 9, 62029–62037. <https://doi.org/10.1109/ACCESS.2021.3074648>
- Igual, J. (2019). Photographic *Noise* Performance Measures Based on RAW Files Analysis of Consumer Cameras. *Electronics*, 8(11), 1284. <https://doi.org/10.3390/electronics8111284>

- Kornelius, Y. (2024). *Pengembangan Aplikasi Presensi Wajah Menggunakan Model MobileFaceNet*. Universitas Kristen Duta Wacana.
- Marr, D., & Hildreth, E. (1980). Theory of edge detection. *Proceedings of the Royal Society of London. Series B. Biological Sciences*, 207(1167), 187–217. <https://doi.org/10.1098/rspb.1980.0020>
- Nainggolan, S. Y., & Khair, U. (2020). *PENINGKATAN KUALITAS CITRA MRI SCAN DENGAN MENGGUNAKAN METODE GAUSSIAN FILTER*.
- Nasution, I. F. (2021). *PENINGKATAN KUALITAS CITRA DENGAN METODE FILTER GAUSSIAN, MEAN DAN MEDIAN UNTUK REDUKSI NOISE PADA CITRA ULTRASONOGRAPHY*.
- Pagaduan, R. A., R. Aragon, Ma. C., & Medina, R. P. (2020). iBlurDetect: Image Blur Detection Techniques Assessment and Evaluation Study: *Proceedings of the International Conference on Culture Heritage, Education, Sustainable Tourism, and Innovation Technologies*, 286–291. <https://doi.org/10.5220/0010307702860291>
- Peltoketo, V.-T. (2014a). *Mobile phone camera benchmarking: Combination of camera speed and image quality* (S. Triantaphillidou & M.-C. Larabi, Eds.; p. 90160F). <https://doi.org/10.1117/12.2034348>
- Peltoketo, V.-T. (2014b). *Mobile phone camera benchmarking: Combination of camera speed and image quality* (S. Triantaphillidou & M.-C. Larabi, Eds.; p. 90160F). <https://doi.org/10.1117/12.2034348>
- Senapatha, I. K. D., & Tamtama, G. I. W. (2023). *RANCANG BANGUN SISTEM ANTI-SPOOF WAJAH PADA PERANGKAT BERGERAK BERBASIS MOBILENET*. Institut Teknologi Indonesia. <https://www.researchgate.net/publication/368307840>
- Siregar, C. P. H., Syahputri, E. T., Hasibuan, N., Burhani, Y., & Widyawanti, T. (2024). *Penerapan Filter Adaptif Untuk Pengurangan Noise Pada Citra Digital*. 02(02).
- Wibowo, S. H., & Susanto, F. (2017). *PENERAPAN METODE GAUSSIAN SMOOTHING UNTUK MEREDUKSI NOISE PADA CITRA DIGITAL*.

JURNAL MEDIA INFOTAMA, 12(2).

<https://doi.org/10.37676/jmi.v12i2.416>

LAMPIRAN A

KODE SUMBER PROGRAM

a. Convolution.kt

```
package com.mfa.preprocessor

import kotlinx.coroutines.*
import kotlin.math.abs

class Convolution {

    fun convolve(
        input: Array<IntArray>,
        kernel: FloatArray,
        kernelRows: Int,
        kernelCols: Int
    ): Array<IntArray> = runBlocking {

        val inputRows = input.size
        val inputCols = input[0].size

        val outputRows = inputRows - kernelRows + 1
        val outputCols = inputCols - kernelCols + 1

        val output = Array(outputRows) { IntArray(outputCols) }

        // Use coroutines for actual parallelism
```

```

coroutineScope {

    (0 until outputRows).map { i ->

        launch(Dispatchers.Default) { // Launch parallel coroutines

            for (j in 0 until outputCols) {

                var sum = 0f

                for (ki in 0 until kernelRows) {

                    for (kj in 0 until kernelCols) {

                        sum += input[i + ki][j + kj] * kernel[ki * kernelCols + kj]

                    }

                }

                output[i][j] = abs(sum.toInt()).coerceIn(0, 255) // More readable

            }

        }.joinAll() // Wait for all coroutines to finish

    }

    output

}

```

b. PreprocessingUtils.kt

```
package com.mfa.preprocessor
```

```

import android.graphics.Bitmap

import android.widget.Toast

import kotlinx.coroutines.Dispatchers

import kotlinx.coroutines.async

import kotlinx.coroutines.awaitAll

import kotlinx.coroutines.runBlocking

import kotlinx.coroutines.withContext

import java.util.*

class PreprocessingUtils {

    private val conv : Convolution = Convolution();

    private fun gaussian(x: Int, y: Int, sigma: Double): Double {

        require(!(sigma <= 0)) { "Sigma must be positive." }

        val squaredDistance = (x * x + y * y).toDouble()

        val denominator = 2 * sigma * sigma

        val exponent = -squaredDistance / denominator

        return 1 / (2 * Math.PI * sigma * sigma) * Math.exp(exponent)

    }

    fun calculateVariance(array: Array<IntArray>): Double = runBlocking {

        val rows = array.size

```

```

val cols = array[0].size

val totalElements = rows * cols

if (totalElements == 0) return@runBlocking 0.0

// Step 1: Calculate the mean in parallel
val sum = withContext(Dispatchers.Default) {
    array.map { row ->
        async { row.sum() } // Each row is summed in parallel
    }.awaitAll().sum() // Combine results from all coroutines
}

val mean = sum.toDouble() / totalElements

// Step 2: Calculate the sum of squared differences in parallel
val sumSquaredDifferences = withContext(Dispatchers.Default) {
    array.map { row ->
        async {
            row.sumOf { value ->
                val diff = value - mean
                diff * diff
            }
        }
    }
}

```



```

        }.awaitAll().sum() // Combine results from all coroutines
    }

    // Step 3: Divide by the number of elements to get the variance
    sumSquaredDifferences / totalElements
}

fun histogramEqualization(image: Array<IntArray>, width: Int, height: Int):
Array<IntArray> {
    val histogram = IntArray(256) // Histogram for pixel values 0 to 255
    val cdf = IntArray(256) // Cumulative Distribution Function
    val equalizedImage = Array(height) { IntArray(width) }

    // Calculate the histogram
    for (y in 0 until height) {
        for (x in 0 until width) {
            // Clamp the pixel value between 0 and 255
            val pixelValue = image[y][x].coerceIn(0, 255)
            histogram[pixelValue]++
        }
    }

    // Calculate the CDF (Cumulative Distribution Function)

```

```

cdf[0] = histogram[0]

for (i in 1..255) {
    cdf[i] = cdf[i - 1] + histogram[i]
}

val cdfMin = cdf[0]

val totalPixels = width * height

val equalizationMap = IntArray(256)

for (i in 0..255) {
    equalizationMap[i] = Math.round((cdf[i] - cdfMin) / (totalPixels -
cdfMin).toFloat() * 255)

    if (equalizationMap[i] < 0) equalizationMap[i] = 0

    if (equalizationMap[i] > 255) equalizationMap[i] = 255
}

// Apply the equalization map to get the equalized image
for (y in 0 until height) {
    for (x in 0 until width) {
        val pixelValue = image[y][x].coerceIn(0, 255)

        equalizedImage[y][x] = equalizationMap[pixelValue]
    }
}

```

```

    return equalizedImage
}

```

```

fun generateGaussianKernel(size: Int, sigma: Float): FloatArray {

    val arr = Array(size * size) { FloatArray(2) }

    var x = 0f
    var y = 0f

    val offset = size / 2

    for (i in 0 until size * size) {

        arr[i] = floatArrayOf(x - offset, y - offset)

        y++

        if (y >= size) {

            y = 0f

            x++

        }

    }

    val doubleResult = Arrays.stream(arr)

        .mapToDouble { n: FloatArray -> gaussian(n[0].toInt(), offset,
sigma.toDouble()).toFloat().toDouble() }

        .toArray()

    val sum = Arrays.stream(doubleResult).sum()

    for (i in doubleResult.indices) {

```

```

        doubleResult[i] /= sum
    }

    val result = FloatArray(doubleResult.size)

    for (i in result.indices) {
        result[i] = doubleResult[i].toFloat()
    }

    return result
}

fun convolve(pixels: Array<IntArray>, kernel: FloatArray, ksize: Int):
Array<IntArray> {
    return conv.convolve(pixels, kernel, ksize, ksize);
}

fun isBlurry(pixels: Array<IntArray>): Boolean {
    val threshold = 500;

    val gaussianKernel = generateGaussianKernel(3, 3f/6f);

    val lKernel2 : FloatArray = arrayOf(
        1f, 1f, 1f,
        1f, -8f, 1f,
        1f, 1f, 1f
    ).toFloatArray();

```

```

var newPixels : Array<IntArray>;

val eqPixels = histogramEqualization(pixels, pixels[0].size, pixels.size);

newPixels = convolve(eqPixels, gaussianKernel, 3);

newPixels = convolve(newPixels, lKernel2, 3);

val variance = calculateVariance(newPixels);

println(variance);

//    Toast.makeText(this, "${variance}", Toast.LENGTH_SHORT).show();

return variance < threshold;

}

```

```

fun isBlurryD(pixels: Array<IntArray>): Double {

    val threshold = 500;

    val gaussianKernel = generateGaussianKernel(3, 3f/6f);

    val lKernel2 : FloatArray = arrayOf(

        1f, 1f, 1f,

        1f, -8f, 1f,

        1f, 1f, 1f

    ).toFloatArray();

    var newPixels : Array<IntArray>;

    val eqPixels = histogramEqualization(pixels, pixels[0].size, pixels.size);

    newPixels = convolve(eqPixels, gaussianKernel, 3);

    newPixels = convolve(newPixels, lKernel2, 3);

```

```

        val variance = calculateVariance(newPixels);

        println(variance);

//        Toast.makeText(this, "${variance}", Toast.LENGTH_SHORT).show();

        return variance;
    }

```

```

fun convertRawGreyImg(bitmap: Bitmap): Array<IntArray> {

    val w = bitmap.width

    val h = bitmap.height

    val pixels = IntArray(h * w)

    bitmap.getPixels(pixels, 0, w, 0, 0, w, h)

    val result = Array(h) { IntArray(w) }

    for (i in 0 until h) {

        for (j in 0 until w) {

            val data = pixels[w * i + j]

            val red = ((data shr 16) and 0xFF)

            val green = ((data shr 8) and 0xFF)

            val blue = (data and 0xFF)

```

```

        var grey = (red.toFloat() * 0.3 + green.toFloat() * 0.59 + blue.toFloat() *
0.11).toInt()

        result[i][j] = grey
    }
}

return result
}

```

```

fun convertGreyImg(bitmap: Bitmap): Array<IntArray> {

    val w = bitmap.width

    val h = bitmap.height

    val pixels = IntArray(h * w)

    bitmap.getPixels(pixels, 0, w, 0, 0, w, h)

    val result = Array(h) { IntArray(w) }

    val alpha = 0xFF shl 24

    for (i in 0 until h) {

        for (j in 0 until w) {

            val data = pixels[w * i + j]

            val red = ((data shr 16) and 0xFF)

            val green = ((data shr 8) and 0xFF)

```

```

        val blue = (data and 0xFF)

        var grey = (red.toFloat() * 0.3 + green.toFloat() * 0.59 + blue.toFloat() *
0.11).toInt()

        grey = alpha or (grey shl 16) or (grey shl 8) or grey

        result[i][j] = grey
    }
}

return result
}

```

```

fun convertArrayToBitmap(pixelArray: Array<IntArray>): Bitmap {

    val height = pixelArray.size

    val width = pixelArray[0].size

    val bitmap = Bitmap.createBitmap(width, height,
Bitmap.Config.ARGB_8888)

    val pixels = IntArray(width * height)

    for (y in pixelArray.indices) {

        for (x in pixelArray[0].indices) {

            val gray = pixelArray[y][x].coerceIn(0, 255) // Ensure valid grayscale
range

```



```

        pixels[y * width + x] = (0xFF shl 24) or (gray shl 16) or (gray shl 8) or
gray
    }
}

bitmap.setPixels(pixels, 0, width, 0, 0, width, height)

return bitmap
}
}

```

c. **TestingPython.py**

```

import numpy as np
from sklearn.datasets import fetch_lfw_people
from skimage.metrics import peak_signal_noise_ratio as psnr
from skimage.util import random_noise
from skimage.metrics import structural_similarity as ssim
import json
import cv2
import random
import sys

def test_gaussian(ksize, sigma, var, images, jumlah):
    psnr_values = []
    ssim_values = []
    random_indices = random.sample(range(len(images)), jumlah)
    for idx in random_indices:
        original = images[idx].astype(np.float32) / 255.0
        noisy = random_noise(original, mode='gaussian', var=var)

```

```

    denoised = cv2.GaussianBlur(noisy, (ksize,ksize), sigma)
    ssim_value, ssim_map = ssim(original, denoised, full=True)
    value = psnr(original, denoised)
    psnr_values.append(value)
    ssim_values.append(ssim_value)

return {
    'ssim': np.mean(ssim_values),
    'psnr': np.mean(psnr_values)
}

def test_poisson(ksize, sigma, images, jumlah):
    psnr_values = []
    ssim_values = []
    random_indices = random.sample(range(len(images)), jumlah)
    for idx in random_indices:
        original = images[idx].astype(np.float32) / 255.0
        noisy = random_noise(original, mode='poisson')
        denoised = cv2.GaussianBlur(noisy, (ksize,ksize), sigma)
        ssim_value, ssim_map = ssim(original, denoised, full=True)
        value = psnr(original, denoised)
        psnr_values.append(value)
        ssim_values.append(ssim_value)

    return {
        'ssim': np.mean(ssim_values),
        'psnr': np.mean(psnr_values)
    }

def test_gaussian_poisson(ksize, sigma, var, images, jumlah):
    psnr_values = []

```

```

ssim_values = []
random_indices = random.sample(range(len(images)), jumlah)
for idx in random_indices:
    original = images[idx].astype(np.float32) / 255.0
    noisy = random_noise(original, mode='poisson')
    noisy = random_noise(noisy, mode='gaussian', var=var)
    denoised = cv2.GaussianBlur(noisy, (ksize,ksize), sigma)
    ssim_value, ssim_map = ssim(original, denoised, full=True)
    value = psnr(original, denoised)
    psnr_values.append(value)
    ssim_values.append(ssim_value)

return {
    'ssim': np.mean(ssim_values),
    'psnr': np.mean(psnr_values)
}

```

```

def iterate_through_ksize_and_sigma(test_func, images, jumlah):
    hasil = []
    for k_size in [3, 5, 7, 9, 11, 13, 15]:
        for sigma in [0.5, 1.0, 1.5]:
            result = test_func(k_size, sigma, images, jumlah)
            hasil.append({
                'kernel': k_size,
                'sigma': sigma,
                'ssim': result['ssim'],
                'psnr': result['psnr']
            })
    return hasil

```

```

def iterate_through_ksize_and_sigma_and_var(test_func, images, jumlah) -> dict:

```

```

hasil = []
for k_size in [3, 5, 7, 9, 11, 13, 15]:
    for sigma in [0.5, 1.0, 1.5]:
        for var in [0.001, 0.01, 0.1]:
            result = test_func(k_size, sigma, var, images, jumlah)
            hasil.append({
                'kernel': k_size,
                'var': var,
                'sigma': sigma,
                'ssim': result['ssim'],
                'psnr': result['psnr']
            })
        print(hasil)
    return hasil

def test_all_images(mode):
    lfw_people = fetch_lfw_people(min_faces_per_person=0, resize=1)
    images = [cv2.resize(img, (224, 224)) for img in lfw_people.images]
    hasil_semua = []
    if mode == 'gaussian':
        hasil_semua = iterate_through_ksize_and_sigma_and_var(test_gaussian,
            images, len(images))
    elif mode == 'poisson':
        hasil_semua = iterate_through_ksize_and_sigma(test_poisson, images,
            len(images))
    elif mode == 'poisson_gaussian':
        hasil_semua =
            iterate_through_ksize_and_sigma_and_var(test_gaussian_poisson, images,
            len(images))
    return hasil_semua

```

```

def main():
    if (len(sys.argv) <= 1):
        print("option:")
        print("\tgaussian")
        print("\tpoisson")
        print("\tpoisson_gaussian")
        return

    if (sys.argv[1] not in ['gaussian', 'poisson', 'poisson_gaussian']):
        print("option:")
        print("\tgaussian")
        print("\tpoisson")
        print("\tpoisson_gaussian")
        return


    option = sys.argv[1]
    hasil = test_all_images(option)
    with open(f'{option}-1.json', 'w') as f:
        json.dump(hasil, f)

if __name__ == '__main__':
    main()


```

LAMPIRAN B







KARTU KONSULTASI DOSEN 1



Kartu Konsultasi Tugas Akhir
 Program Studi Informatika Fakultas Teknologi Informasi
 Universitas Kristen Duta Wacana Yogyakarta
 Dr. Wahidin Sudirahusada 5-25 Yogyakarta, 55224 Telp. (0274) 963929




NIM/NAMA : 71210689/Gian Pradipta Gunawan
 Judul : Penerapan Gaussian Blur Untuk Peningkatan Akurasi Sistem Presensi Multifaktor
 Dosen Pembimbing I : I Kadek Dendy S., S.T., M.Eng.

| | |
|---|---|
| <div style="display: flex; justify-content: space-between;"> <div>Tanggal: 14 - 2 - 2025</div> <div>Paraf: </div> </div> <div style="margin-top: 10px;">mendiskusikan pembagian tugas implementasi</div> | <div style="display: flex; justify-content: space-between;"> <div>Tanggal: 18 - 2 - 2025</div> <div>Paraf: </div> </div> <div style="margin-top: 10px;">konsul implementasi Gaussian Blur</div> |
| <div style="display: flex; justify-content: space-between;"> <div>Tanggal: 21 - 3 - 2025</div> <div>Paraf: </div> </div> <div style="margin-top: 10px;">mendiskusikan cara pengujian sistem</div> | <div style="display: flex; justify-content: space-between;"> <div>Tanggal: 29 - 4 - 2025</div> <div>Paraf: </div> </div> <div style="margin-top: 10px;">- mendiskusikan pengujian stress menggunakan PSNR dan light exposure.</div> |
| <div style="display: flex; justify-content: space-between;"> <div>Tanggal: 14 - 5 - 2025</div> <div>Paraf: </div> </div> <div style="margin-top: 10px;">Revisi Laporan 1</div> | <div style="display: flex; justify-content: space-between;"> <div>Tanggal: 23 - 5 - 2025</div> <div>Paraf: </div> </div> <div style="margin-top: 10px;">Revisi Laporan 2.</div> |
| <div style="display: flex; justify-content: space-between;"> <div>Tanggal:</div> <div>Paraf:</div> </div> | <div style="display: flex; justify-content: space-between;"> <div>Tanggal:</div> <div>Paraf:</div> </div> |

1 of 2

2/24/2025, 10:49 AM

 Scanned with CamScanner

LAMPIRAN C

KARTU KONSULTASI DOSEN 2



Kartu Konsultasi Tugas Akhir

Program Studi Informatika Fakultas Teknologi Informasi
Universitas Kristen Duta Wacana Yogyakarta
Dr. Wahidin Sudirahusada 5-25 Yogyakarta, 55224 Telp. (0274)563929

NIM : 71210689/Gian Pradipta Gunawan
Judul : Penerapan Gaussian Blur Untuk Peningkatan Akurasi Sistem Presensi Multifaktor
Dosen Pembimbing II : Budi Susanto, SKom, M.T.

| | | | | | |
|--|-----------------------|------------|--|-------------------------|------------|
| 1 | Tanggal: 24-4-2025 | Paraf: | 2 | Tanggal: 1-5-2025 | Paraf: |
| <p>- masalah tidak jelas</p> <p>- metodologi tidak tepat.</p> | | | <p>- Bab 1 - 3 harus dengan di bangun,</p> <p>- Metodologi penelitian tidak jelas.</p> | | |
| 3 | Tanggal: 6-5-2025 | Paraf: | 4 | Tanggal: 7-5-2025 | Paraf: |
| <p>Bab 1 & 2 tidak sederhana, ganti metode pakai PSNR dan SSIM</p> | | | <p>- Tambah metode PSIM.</p> | | |
| 5 | Tanggal: | Paraf: | 6 | Tanggal: 20 Mei 2025 | Paraf: |
| <p>- Bab 4 & 5 OK.</p> | | | <p>OK sudah siap untuk persaman!</p> | | |
| 7 | Tanggal: | Paraf: | 8 | Tanggal: | Paraf: |
| | | | | | |

Dibuat pada 24 Februari 2025 10:49