BAB III
METODE PENELITIAN

3.1 Metode Penelitian

Penelitian merupakan suatu proses penyelidikan yang bertujuan untuk menemukan kebenaran dan membuktikan suatu fenomena (Mweshi & Sakyi, 2020). Dalam proses ini, terjadi serangkaian kegiatan intelektual yang bertujuan untuk mengungkap pengetahuan baru, memperbaiki pemahaman yang ada, serta mengoreksi atau menghilangkan kesalahpahaman yang dapat terjadi. Proses ini mencakup upaya sistematis untuk menggali informasi, menganalisis data, dan menyusun kesimpulan yang dapat memperkaya pengetahuan yang sudah ada atau mengarahkan pada penemuan-penemuan baru yang signifikan. Setiap penelitian membutuhkan cara yang tepat untuk mengumpulkan data. Metode penelitian adalah pendekatan yang digunakan untuk mengumpulkan dan menganalisis data yang diperoleh. Pemilihan metode ini disesuaikan dengan tujuan penelitian, sehingga hasil yang didapatkan bisa benar-benar mencerminkan apa yang ingin dipelajari. Dengan metode yang tepat, peneliti bisa memastikan bahwa data yang dikumpulkan akurat dan hasil penelitian bisa dipercaya (Waruwu, 2023).

3.2 Lokasi dan Waktu Penelitian

3.2.1. Lokasi Penelitian

Mengacu pada Peraturan Pemerintah Republik Indonesia Nomor 26 Tahun 2008 tentang Rencana Tata Ruang Wilayah Nasional, Pasal 61 ayat 4, disebutkan bahwa kawasan yang berada pada zona sesar aktif ditentukan dengan kriteria lebar paling sedikit 250 (dua ratus lima puluh) meter dari tepi garis sesar aktif, yang dianalisis menggunakan perangkat lunak pemetaan (*buffer zone*).

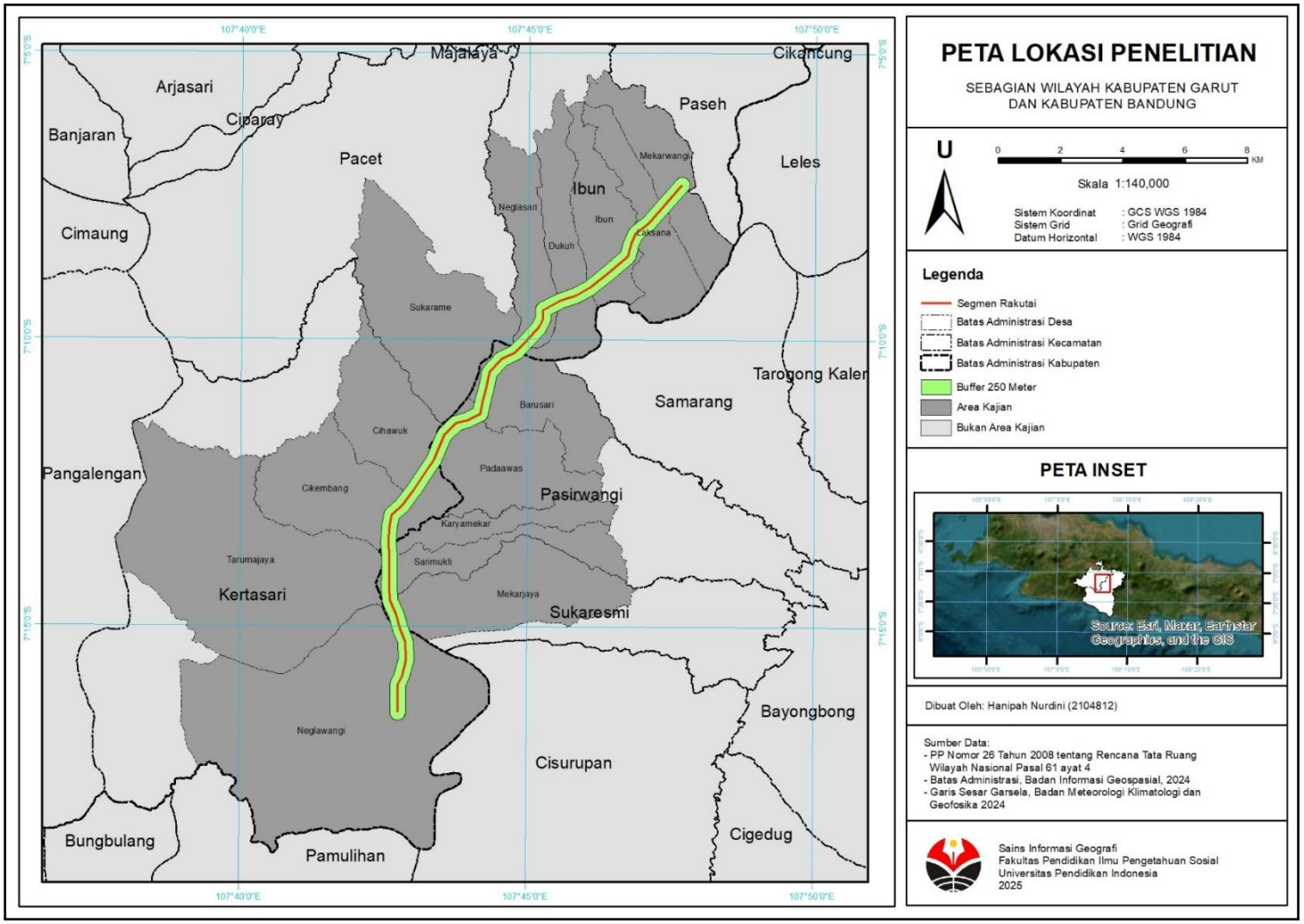
Adapun batas geografis lokasi penelitian ditentukan sebagai berikut:

- Sebelah utara: Kecamatan Ibun yang merupakan bagian dari Kabupaten Bandung berada pada koordinat 107°46'31.67" BT dan 7°7'3.71" LS.
- Sebelah selatan: Kecamatan Kertasari yang merupakan bagian dari Kabupaten Bandung berada pada koordinat 107°41'33.24" BT dan 7°16'33.73" LS.
- Sebelah barat: Kecamatan Kertasari yang merupakan bagian dari Kabupaten Bandung berada pada koordinat yang sama, yaitu 107°41'33.24" BT dan 7°16'33.73" LS.
- Sebelah timur: Kecamatan Pasirwangi yang merupakan bagian dari Kabupaten Bandung Garut berada pada koordinat 107°45'6.40" BT dan 7°12'4.56" LS.

Analisis *buffer* dengan jarak 250 meter dari jalur sesar aktif mencakup wilayah-wilayah berikut:

1. Kabupaten Bandung:
    - Kecamatan Pacet: Desa Sukarame.
    - Kecamatan Ibun: Desa Neglasari, Desa Dukuh, Desa Ibun, Desa Laksana, dan Desa Mekarwangi.
    - Kecamatan Kertasari: Desa Cihawuk, Desa Cikembang, Desa Tarumajaya, dan Desa Neglawangi.
2. Kabupaten Garut:
    - Kecamatan Pasirwangi: Desa Barusari, Desa Baruwaas, Desa Karyamekar, dan Desa Sarimukti.
    - Kecamatan Sukaresmi: Desa Mekarjaya.

    Visualisasi peta lokasi penelitian disajikan pada Gambar 3.1 berikut untuk memberikan gambaran yang lebih jelas mengenai wilayah yang menjadi fokus dalam penelitian ini



Gambar 3. 1 Peta Lokasi Penelitian

3.2.2. Waktu Penelitian

Penelitian dilaksanakan dalam waktu empat bulan terhitung dari bulan Oktober 2024 hingga bulan Maret 2025 dengan rincian sebagai berikut:

Tabel 3. 1 Waktu Penelitian

Kegiatan	Bulan Oktober 2024				Bulan November 2024				Bulan Desember 2024				Bulan Januari 2025				Bulan Februari 2025				Bulan Maret 2025			
	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
Pra Penelitian																								
1. Penetuan permasalahan dan tema penelitian																								
2. Penentuan judul penelitian																								
3. Pencarian sumber literatur																								
4. Pembuatan proposal penelitian																								

Pelaksanaan Penelitian																							
1. Pengumpulan data																							
2. Pengolahan data																							
3. Analisis data																							
Pasca Penelitian																							
1. Penyusunan laporan akhir																							

Sumber : Analisis Peneliti (2025)

3.3 Alat dan Bahan Penelitian

Dalam penelitian ini digunakan alat dan bahan sebagai penunjang berjalannya penelitian. Jika alat dan bahan yang diperlukan kurang lengkap maka akan menjadi hambatan dalam pelaksanaan penelitian ini. Untuk itu kelengkapan alat dan bahan yang diperlukan merupakan hal yang krusial dalam pelaksanaan penelitian ini. Berikut merupakan rincian alat dan bahan yang diperlukan dalam penelitian ini.

3.3.1 Alat

Adapun alat yang digunakan dalam penelitian ini adalah sebagai berikut.

Tabel 3. 2 Alat yang digunakan dalam penelitian

No	Alat	Fungsi
1	Laptop Acer Aspire 5, Prosesor AMD Ryzen 7 5700U with Radeon Graphics, Windows 11 Home Single Language 64-bit, Memory 16384 MB RAM	Penunjang pengoperasian perangkat lunak (*software*), pengolahan data, analisis data dan penyusunan laporan penelitian.

2	Microsoft Office	Digunakan untuk pembuatan laporan dan pembuatan bahan presentasi
3	ESA SNAP (9.0.0)	Mengolah citra radar Sentinel-1 menjadi citra interferogram yang memiliki nilai deformasi permukaan
4	Google Colab	Perangkat lunak (*software*) yang berfungsi sebagai wadah untuk menulis dan mengembangkan kode program (*script coding*)
5	ArcGIS/ArcGIS Pro	Digunakan untuk layout peta
6	QGIS	Digunakan untuk *resampling* dan *labeling*

Sumber : Analisis Peneliti (2025)

3.3.2 Bahan

Adapun bahan yang digunakan dalam penelitian ini adalah sebagai berikut.

Tabel 3. 3 Bahan Yang Digunakan Dalam Penelitian

No	Bahan	Sumber	Tahun	Jenis Data	Fungsi
1	Batas administrasi Kabupaten Garut dan Kabupaten Bandung	BIG	2024	Vektor	Sebagai batas administrasi wilayah kajian
2	Citra Sentinel- 1A	*Alaska Satellite Facility* (ASF)	Juli 2024 – Januari 2025	Raster (.tif)	Untuk mengekstrak nilai deformasi permukaan.

Sumber : Analisis Peneliti (2025)

3.4 Tahapan Penelitian

3.4.1 Pra Penelitian

Tahapan pra penelitian merupakan langkah awal dalam sebuah penelitian. Tahapan ini merupakan tahapan persiapan penelitian yang di dalamnya memuat langkah-langkah pengumpulan data dan bahan penunjang penelitian. Adapun tahapan ini terdiri dari beberapa tahapan, yakni :

- Menentukan Permasalahan dan Judul Penelitian

Tahapan ini membentuk fondasi yang kokoh untuk penelitian yang akan dilakukan. Pada tahap ini, peneliti mengidentifikasi dan menganalisis permasalahan yang relevan dengan bidang kajian, dalam hal ini terkait dengan pengembangan model *machine learning* untuk analisis deformasi. Kelayakan permasalahan yang diangkat perlu dikaji terlebih dahulu agar dapat dieksekusi menjadi sebuah penelitian yang bermanfaat. Setelah menganalisis permasalahan, langkah selanjutnya adalah menentukan judul penelitian yang memberikan gambaran singkat dan informatif mengenai substansi penelitian. Penentuan judul juga berperan dalam membatasi lingkup penelitian serta mengkomunikasikan esensi dan fokus penelitian secara jelas.

- Mengumpulkan Literatur

Tahapan ini merupakan tahapan yang menyediakan dasar pengetahuan yang komprehensif mengenai topik penelitian yang berasal dari jurnal, buku, skripsi, prosiding dan lain sebagainya. Melalui tahapan ini, peneliti dapat mengeksplorasi pendapat-pendapat penelitian terdahulu, teori-teori yang relevan, temuan-temuan penelitian sebelumnya, serta pendekatan-pendekatan metodologi yang telah digunakan oleh peneliti sebelumnya.

- Membuat Proposal Penelitian

Proposal penelitian berisi mengenai usulan penelitian yang akan dilakukan yang di dalamnya menjelaskan hal-hal yang berkaitan dengan penelitian. Proposal penelitian berisi latar belakang, rumusan masalah, tujuan penelitian, manfaat penelitian, definisi operasional, penelitian terdahulu, tinjauan pustaka dan metode penelitian.

3.4.2 Pelaksanaan Penelitian

Tahapan ini merupakan tahapan pelaksanaan berdasarkan proposal yang telah disusun sebelumnya. Tahapan ini berfokus pada pengolahan data serta produk penelitian yang dihasilkan dari setiap tahap. Seluruh kegiatan dilakukan secara sistematis untuk mencapai tujuan penelitian dan menjawab rumusan masalah yang telah dirumuskan sebelumnya.

- Tahapan Pengumpulan Data

Pada tahap ini, peneliti mengumpulkan berbagai data spasial dan citra penginderaan jauh yang diperlukan untuk menganalisis deformasi permukaan tanah di wilayah studi. Data utama yang digunakan adalah citra radar Sentinel-1A sebanyak enam pasang akuisisi dari periode 3 Juli 2024 hingga 11 Januari 2025, yang mencakup rentang waktu sebelum dan sesudah terjadinya perubahan deformasi signifikan. Setiap pasang citra terdiri dari dua tanggal akuisisi yang kemudian digunakan untuk menghitung *displacement* antarwaktu. Selain citra radar, data pendukung lainnya seperti batas administrasi dan tutupan lahan wilayah studi dikumpulkan untuk keperluan *overlay* dan interpretasi spasial lanjutan.

Tahapan ini juga mencakup verifikasi awal terhadap kelengkapan dan kualitas data. Proses ini bertujuan memastikan bahwa seluruh data yang digunakan memiliki cakupan spasial dan resolusi temporal yang konsisten agar proses analisis deformasi dapat dilakukan secara berurutan dan komprehensif. Semua data yang dikumpulkan selanjutnya diolah dalam tahapan lanjutan dengan pendekatan pemodelan *machine learning*.

- Tahapan Pengolahan Data

Pengolahan data dimulai dengan pemrosesan citra Sentinel-1 menggunakan software ESA SNAP 9.0.0 dengan metode *Differential Interferometric Synthetic Aperture Radar* (DInSAR). Setiap pasang citra menghasilkan interferogram yang kemudian dikonversi menjadi citra *displacement* (deformasi) dalam satuan meter, mengacu pada pergeseran sepanjang garis pandang sensor (*Line of Sight*/LOS). Hasil dari keenam periode pemrosesan ini berupa enam peta deformasi yang menggambarkan perubahan permukaan tanah selama periode pengamatan.

Citra deformasi yang dihasilkan kemudian dilakukan proses koreksi geometrik, resampling, serta konversi ke format raster dan .csv. Data .csv berisi nilai deformasi untuk setiap piksel pada enam periode berbeda. Data ini selanjutnya digunakan sebagai input dalam pemodelan *machine learning*. Peneliti menggunakan dua algoritma, yaitu *Random Forest* (RF) dan *Multi-Layer Perceptron* (MLP), yang dibangun menggunakan platform *Google Colab*. Proses *tuning hyperparameter* dilakukan dengan metode *GridSearchCV* untuk memperoleh parameter optimal pada kedua model.

- Tahapan Evaluasi dan Validasi Data

Setelah model terbentuk, evaluasi dilakukan terhadap performa masing-masing algoritma menggunakan metrik-metrik evaluasi seperti akurasi, presisi, *recall*, dan *F1-score* yang dihitung dari *confusion matrix*. Evaluasi ini bertujuan untuk mengidentifikasi keandalan model dalam mengklasifikasikan tingkat deformasi permukaan, sekaligus mengevaluasi apakah model menunjukkan gejala *overfitting* atau *underfitting*. Penilaian performa juga mencakup analisis validasi silang (*cross-validation*) guna memastikan stabilitas prediksi dalam berbagai *subset* data.

Selain evaluasi numerik, validasi lapangan dilakukan untuk memverifikasi kesesuaian antara hasil klasifikasi deformasi dan kondisi aktual di beberapa titik representatif. Kegiatan ini melibatkan kunjungan ke lokasi-lokasi dengan nilai deformasi ekstrem berdasarkan hasil pemodelan, baik *uplift* maupun *subsidence*. Observasi lapangan mencakup dokumentasi visual dan pengukuran tambahan untuk mengkonfirmasi apakah perubahan yang terdeteksi oleh citra satelit benar terjadi di permukaan tanah, sehingga mendukung aspek reliabilitas model secara spasial.

- Tahapan Analisis Data

Analisis data difokuskan pada interpretasi spasial dan statistik dari sebaran deformasi permukaan hasil pemodelan. Hasil klasifikasi dari algoritma RF dan MLP dibandingkan untuk mengidentifikasi pola deformasi dominan, konsistensi antarperiode, serta lokasi-lokasi dengan kecenderungan *uplift* atau *subsidence* yang signifikan. Analisis spasial dilakukan dengan pendekatan visualisasi peta dan grafik batang (*bar chart*) untuk melihat distribusi kelas deformasi dalam satuan hektare.

Perbandingan antara kedua model juga dianalisis secara kuantitatif berdasarkan performa prediksi dan hasil spasialnya. Analisis ini memberikan dasar dalam menyimpulkan keunggulan relatif masing-masing algoritma, baik dari segi ketepatan klasifikasi maupun efisiensi komputasi. Hasil akhir digunakan untuk menyusun simpulan terkait kondisi deformasi permukaan di wilayah studi dan memberikan rekomendasi teknis untuk pemantauan deformasi jangka panjang menggunakan pendekatan berbasis *machine learning*.

3.4.3 Pasca Penelitian

Tahap ini mencakup penyusunan laporan hasil penelitian yang disusun berdasarkan seluruh rangkaian proses pengolahan dan analisis data deformasi permukaan pasca gempa bumi. Hasil pengolahan mencakup informasi mengenai pola spasial deformasi, distribusi nilai deformasi berdasarkan *Line of Sight* (LOS), serta klasifikasi tingkat deformasi menggunakan model machine learning, yaitu algoritma *Random Forest* (RF) dan *Multi-Layer Perceptron* (MLP).

Laporan ini tidak hanya berperan sebagai dokumentasi ilmiah dari proses penelitian, tetapi juga diharapkan dapat menjadi referensi praktis bagi pemangku kepentingan, seperti pemerintah daerah, instansi penanggulangan bencana, maupun lembaga sosial. Informasi yang dihasilkan diharapkan mampu meningkatkan pemahaman terhadap dinamika deformasi permukaan dan potensi dampaknya terhadap masyarakat. Dengan demikian, hasil penelitian ini dapat mendukung perumusan kebijakan mitigasi yang lebih tepat sasaran, berbasis data spasial, dan mempertimbangkan aspek kerentanan sosial serta infrastruktur di wilayah terdampak.

3.5 Sampel Penelitian

Sampel merupakan bagian dari populasi yang memiliki karakteristik tertentu dan dipilih untuk mewakili keseluruhan populasi dalam suatu penelitian. Dengan kata lain, sampel berperan sebagai representasi populasi untuk memperoleh gambaran yang lebih objektif mengenai kondisi populasi tersebut (Sugiyono, 2017). Teknik pengambilan sampel umumnya dibagi menjadi dua kategori utama yaitu *probability sampling* dan *non-probability sampling*. *Probability sampling* atau random sampling digunakan ketika setiap anggota populasi memiliki peluang yang sama untuk terpilih, sedangkan non-probability sampling digunakan dalam kondisi di mana tidak semua anggota populasi memiliki kesempatan yang sama untuk menjadi sampel (Supardi, 2016).

Dalam penelitian ini, teknik *purposive sampling* digunakan sebagai metode pemilihan sampel. Teknik ini termasuk dalam *non-probability sampling*, di mana sampel dipilih secara sengaja berdasarkan kriteria klasifikasi nilai *Line of Sight* (LOS) dan dalam rentang $-\pi - \pi$ dalam satuan cm/tahun yang telah di klasifikasikan dalam 5 kelas. Pengklasifikasian ini didasarkan pada besaran nilai deformasi yang didapatkan setelah pengolahan deformasi permukaan selesai. Kemudian sampel digunakan untuk mengevaluasi hasil klasifikasi spasial deformasi permukaan dari dua model *machine learning* yaitu *Random Forest* (RF) dan *Multi-Layer Perceptron* (MLP) guna menilai kesesuaian prediksi model dengan kondisi spasial aktual sehingga pengambilan data menjadi lebih relevan dengan tujuan penelitian. Dengan menggunakan *purposive sampling*, diharapkan data yang dikumpulkan lebih sesuai dengan kebutuhan penelitian, sehingga dapat menghasilkan analisis yang lebih akurat terkait pola deformasi permukaan di daerah studi.

3.6 Variabel Penelitian

Variabel penelitian dalam penelitian ini berperan sebagai indikator untuk menilai objek yang diteliti. Penulis menetapkan variabel-variabel ini sebagai acuan dalam menganalisis dan mengukur berbagai aspek yang berkaitan dengan penelitian. Variabel yang digunakan mencerminkan faktor-faktor yang dianggap relevan dalam mencapai tujuan penelitian. Informasi lebih lanjut mengenai variabel penelitian yang digunakan dapat dilihat pada tabel berikut.

Tabel 3. 4 Variabel Penelitian

No	Variabel	Indikator
1.	Nilai deformasi permukaan	• *Deformasi Line of Sight* (LOS) dengan rentang -π sampai π (mm/tahun atau cm/tahun)
2.	Ketepatan visualisasi peta deformasi permukaan hasil model algoritma *Random Forest* dan *Multi-Layer Perceptron*	• Luas tiap kelas deformasi dan sebaran spasial kelas deformasi

Sumber: Analisis Peneliti (2025)

3.7 Teknik Pengumpulan Data

3.7.1 Studi Literatur

Studi pustaka merupakan metode pengumpulan data yang melibatkan pemahaman dan pembelajaran teori-teori yang terdapat dalam berbagai literatur yang relevan dengan fokus penelitian. Pengumpulan data dilakukan dengan mencari dan mengonstruksi informasi dari berbagai sumber, seperti buku, jurnal, dan riset yang telah dilakukan sebelumnya. Bahan pustaka yang diperoleh dari beragam referensi tersebut dianalisis secara untuk mendukung proposisi dan gagasan penelitian. Peneliti melakukan studi literatur yang sistematis untuk mengetahui teori dan metode yang tepat untuk digunakan dalam penelitian analisis deformasi permukaan dengan algoritma *random forest* dan *multi layer perceptron*.

3.7.2 Studi Dokumentasi

Studi dokumentasi adalah cara untuk memperoleh data dalam bentuk bahan tertulis yang relevan dengan penelitian (Akhmad, 2015). Studi dokumentasi ini bertujuan untuk mengumpulkan data yang berkaitan dengan variabel-variabel tertentu, yang dapat berupa catatan, transkrip, buku, surat kabar, majalah, prasasti, notulen rapat, legger, agenda, dan lain sebagainya. Untuk melengkapi data yang diperlukan dalam penelitian ini, digunakan studi dokumentasi sebagai alat bantu dan pendukung yakni SHP batas administrasi, citra Sentinel-1 bulan Juli 2024 sampai bulan Januari 2025

3.7.3 Observasi Tidak Langsung

Observasi tidak langsung merupakan cara pengumpulan data melalui perantara kemudian dilakukan pengamatan dan analisis pada objek atau fenomena yang akan diteliti. Observasi tidak langsung dalam penelitian ini adalah memperoleh data yang dibutuhkan dari internet maupun instansi yang berkaitan dalam penelitian ini. Data yang diperoleh merupakan data dasar yang menjadi bahan dasar dalam pengolahan data selanjutnya.

3.7.4 Observasi Langsung

Observasi langsung adalah suatu cara pengumpulan data yang dilakukan dengan terjun langsung ke lapangan. Peneliti mengamati kondisi fisik wilayah kajian yang terindikasi mengalami deformasi berdasarkan hasil analisis spasial guna memastikan kesesuaian antara data hasil pemodelan dengan realitas di lapangan. Observasi ini sekaligus mendukung interpretasi deformasi.

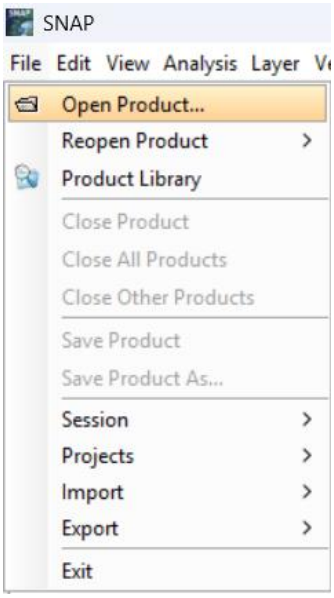
3.8 Teknik Analisis Data

3.8.1 Nilai Deformasi Permukaan Berdasarkan *Line of Sight* (LOS)

Peta deformasi dibuat menggunakan citra *master*/*primary* yakni citra dengan waktu tertua dan citra *secondary*/*slave* atau citra dengan waktu termuda. Tahap *Stacking SAR Image* diawali dengan membuka dua citra Sentinel-1 SLC tertua dan termuda dalam hal ini peneliti akan mengolah citra Sentinel-1 dari bulan Juli 2024 sampai bulan Januari 2025 sehingga akan menghasilkan 6 citra dengan pengolahan berulang kali.

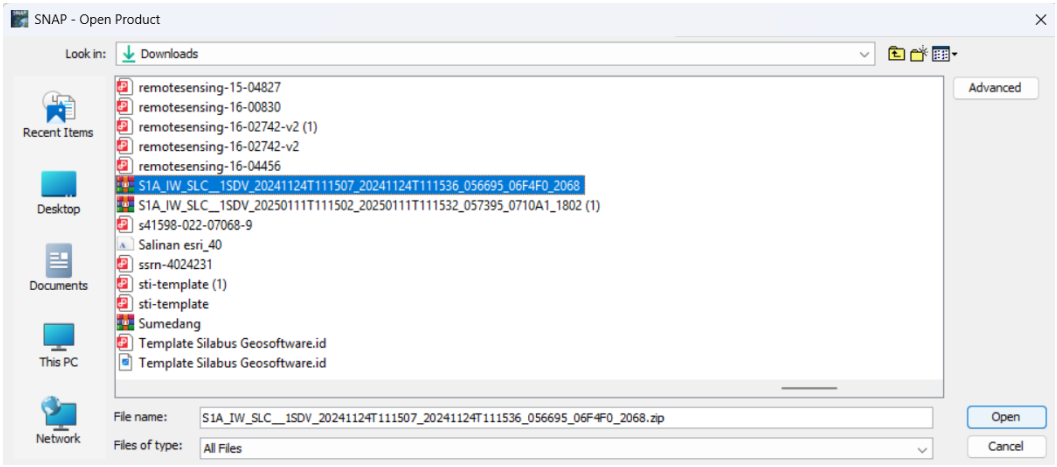
Caranya:

- Buka *software* SNAP, versi yang saya gunakan disini adalah 9.0.0
- Untuk memasukkan file zip citra Sentinel-1 SAR, klik *File* lalu klik *Open Product*



Gambar 3. 2 Tampilan Menu File Pada Software SNAP

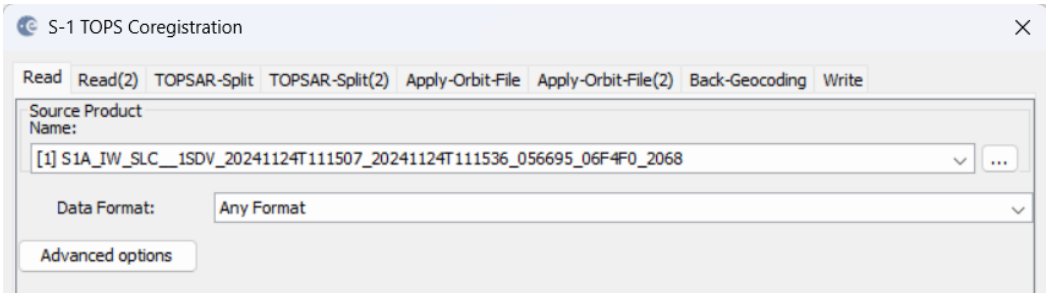
- Masukkan citra Sentinel-1 SAR (*primary* dan *secondary*) yang berbentuk .zip. Klik *Open*



Gambar 3. 3 Tampilan Jendela Open Product

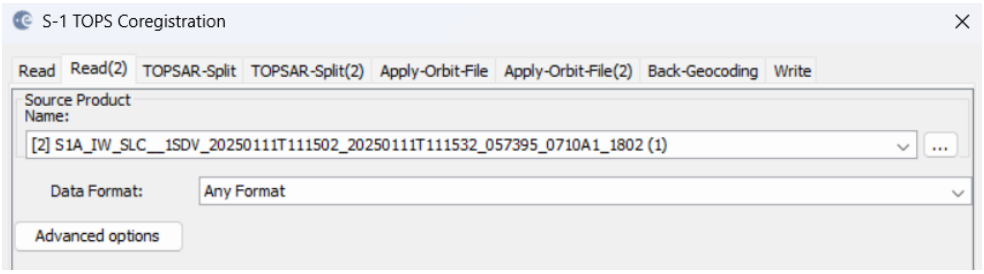
3.8.1.1. *S-1 TOPS Coregistration*

a. *Tab Read* diisi dengan citra *primary* (atau *master*) yang berfungsi sebagai acuan referensi geometris.



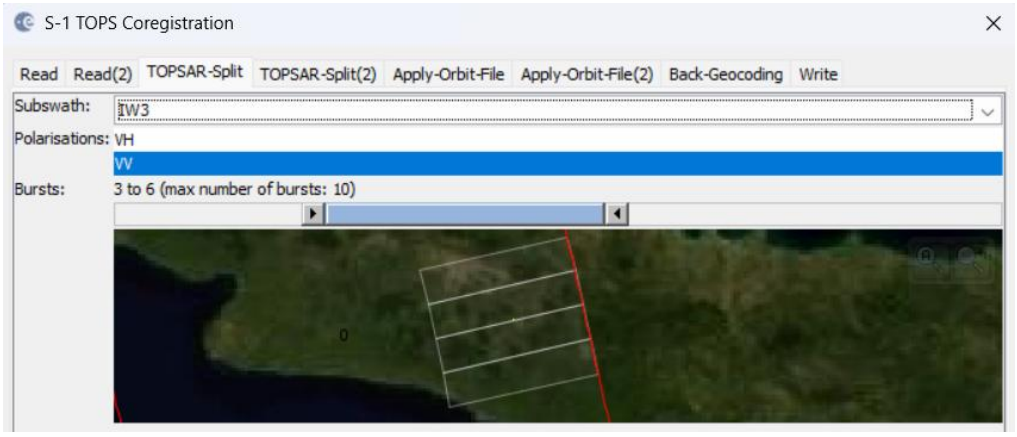
Gambar 3. 4 Tampilan Jendela S-1 TOPS Coregistration Tab Read

b. Sementara itu, tab *Read*(2) diisi dengan citra *secondary* (*slave*) yang akan diselaraskan terhadap citra *master*.



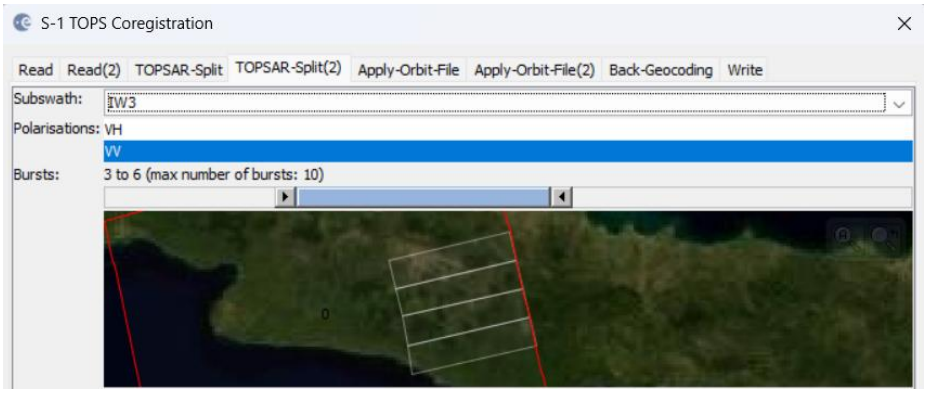
Gambar 3. 5 Tampilan Jendela S-1 TOPS Coregistration Tab Read(2)

c. Pada *tab TOPSAR-Split* dipilih *sub-swath* IW3 dan polaritas VV dari citra *primary* (*master*). Selain itu, ditentukan pula rentang *burst* yaitu *burst* 3 to 6 untuk memastikan area kajian yang diteliti termasuk dalam cakupan *burst* tersebut.



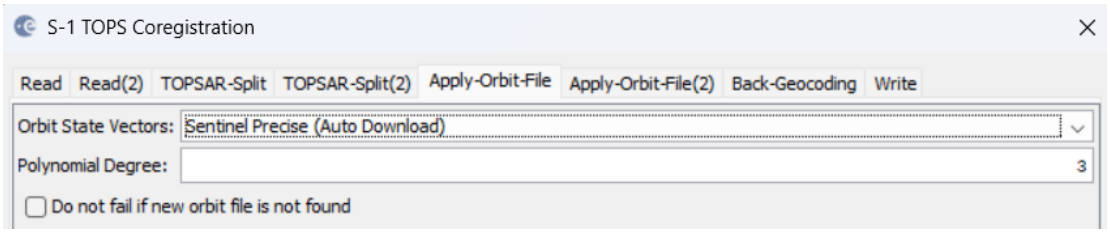
Gambar 3. 6 Tampilan Jendela S-1 TOPS Coregistration Tab TOPSAR Split

d. Hal yang sama dilakukan pada tab *TOPSAR-Split(2)* yaitu untuk citra *secondary* (*slave*) dengan pengaturan *sub-swath* dan *burst* yang sama dengan citra *master* agar keduanya mencakup area spasial yang sama.



Gambar 3. 7 Tampilan Jendela S-1 TOPS Coregistration Tab TOPSAR Split(2)

e. *Tab Apply-Orbit-File* dan *Apply-Orbit-File(2)* diisi dengan pilihan yang sama, yaitu "*Sentinel Precise (Auto Download)*" yang berarti SNAP akan secara otomatis mengunduh dan menerapkan data orbit presisi dari server ESA.

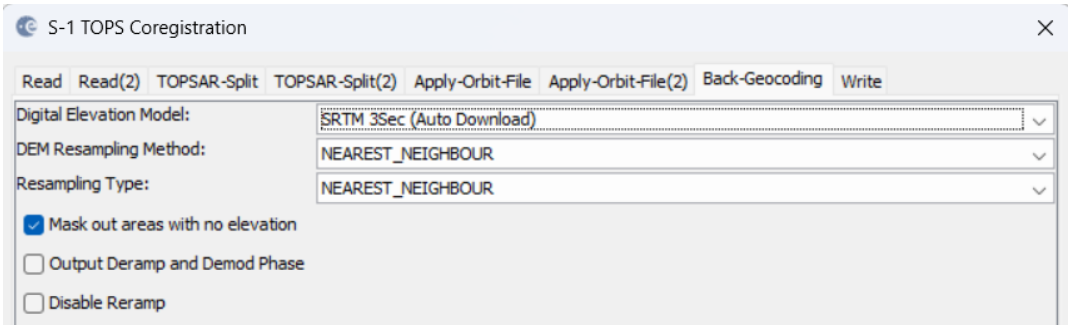


Gambar 3. 8 Tampilan Jendela S-1 TOPS Coregistration Tab Apply-Orbit-File

f. Selanjutnya *tab Back-Geocoding* yang berisi fitur-fitur:

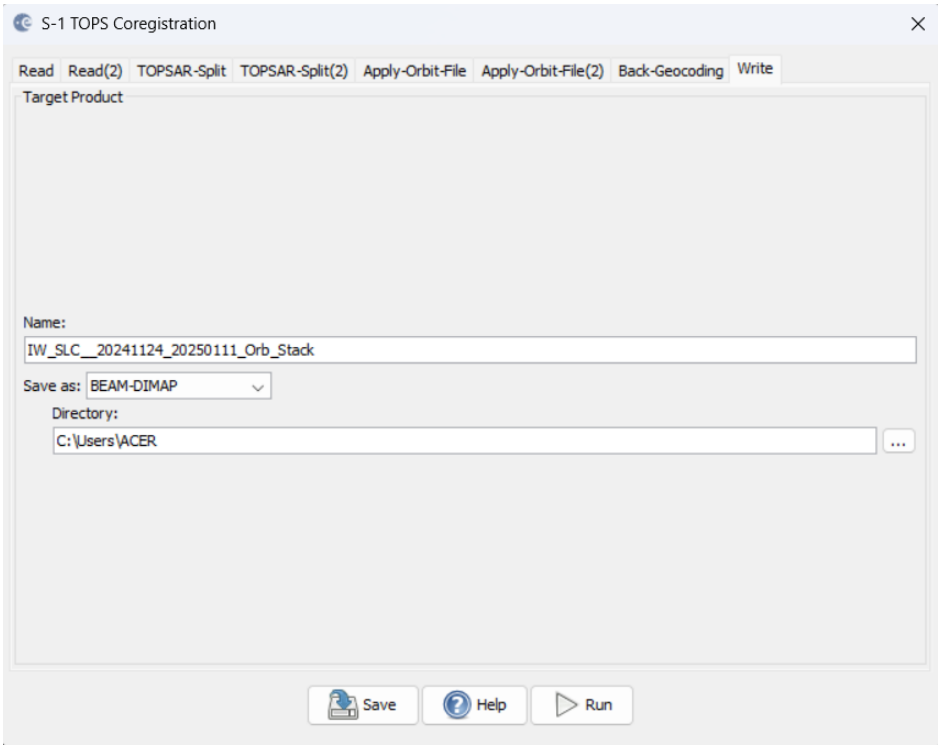
- *Digital Elevation Model* dipilih *SRTM 3Sec (Auto Download)* yang secara otomatis mengunduh data DEM resolusi 90 meter.
- *DEM Resampling Method* maupun *Resampling Type* diatur ke *NEAREST_NEIGHBOUR* untuk menjaga keaslian nilai piksel DEM tanpa interpolasi tambahan.

- Opsi "*Mask out areas with no elevation*" dicentang agar area tanpa data elevasi tidak memengaruhi hasil koregistrasi.



Gambar 3. 9 Tampilan Jendela S-1 TOPS Coregistration Tab Back-Geocoding

g. Pada bagian *Write*, tentukan lokasi penyimpanan hasil proses koregistrasi. Setelah semua parameter dari langkah-langkah sebelumnya telah disesuaikan, klik tombol *Run* di bagian bawah jendela.



Gambar 3. 10 Tampilan Jendela S-1 TOPS Coregistration Tab Write

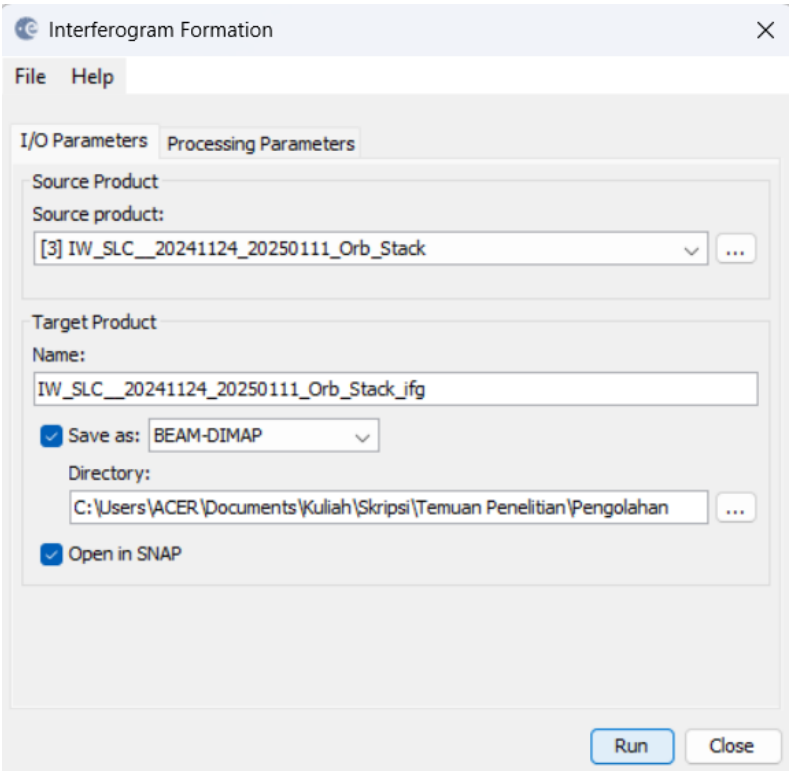
3.8.1.2. *Interferogram*

Langkah ini bertujuan untuk menganalisis perbedaan fase gelombang radar dari dua atau lebih citra SAR untuk menghasilkan informasi tentang perubahan jarak atau deformasi permukaan bumi. Caranya:

a. Klik *Radar* > *Interferometric* > *Products* > *Interferogram Formation.*

b. Pada *tab I/O Parameters* berisi:

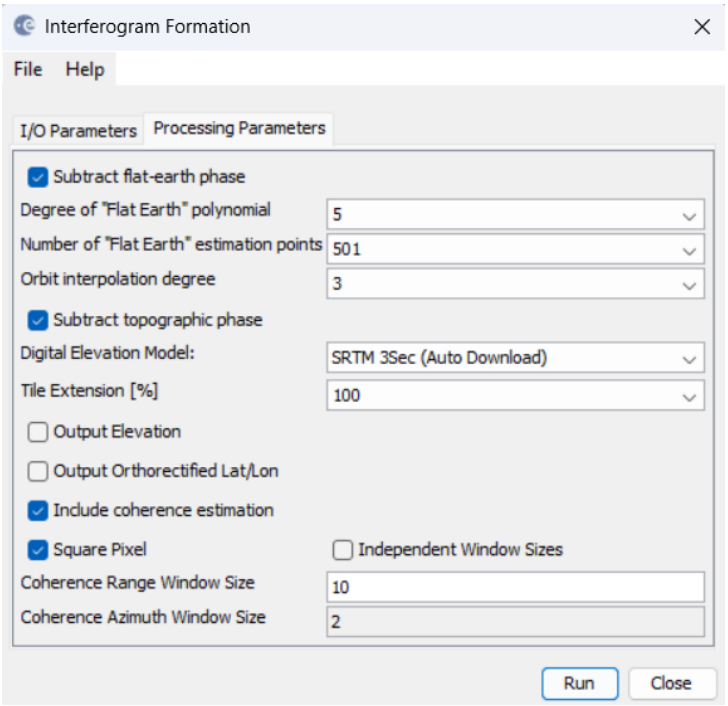
- Bagian "*Source Product*" diisi hasil pengolahan coregistration (…_Orb_Stack).
- Pada kolom "*Target Product*" ditentukan nama output yang akan dihasilkan.
- Format penyimpanan yang dipilih adalah BEAM-DIMAP dan direktori output diarahkan ke folder lokal pengguna.
- Opsi "*Open in SNAP*" dicentang agar hasil interferogram secara otomatis terbuka di jendela kerja SNAP setelah proses selesai.



Gambar 3. 11 Tampilan Jendela Interferogram Formation Tab I/O Parameters

c. Kemudian pada *tab Processing Parameters*:

- Opsi "*Subtract flat-earth phase*" dicentang untuk menghilangkan komponen fase yang disebabkan oleh bentuk lengkung bumi agar fase interferometrik lebih fokus pada perubahan lokal.
- Nilai *degree of polynomial* untuk pengurangan fase *flat-earth* diatur ke 5, dengan jumlah titik estimasi sebanyak 501.
- *Orbit interpolation degree* diisi 3 untuk meningkatkan ketelitian estimasi lintasan satelit.
- Opsi "*Subtract topographic phase*" juga diaktifkan agar fase yang disebabkan oleh variasi topografi dapat dihilangkan.
- Kotak "*Include coherence estimation*" dicentang untuk menghasilkan peta koherensi.
- Koherensi ini dihitung menggunakan jendela analisis 10 piksel dalam arah *range* dan 2 piksel dalam arah azimuth, serta dengan opsi "*Square Pixel*" aktif agar ukuran piksel tetap proporsional.



Gambar 3. 12 Tampilan Jendela Interferogram Formation Tab Processing Parameters

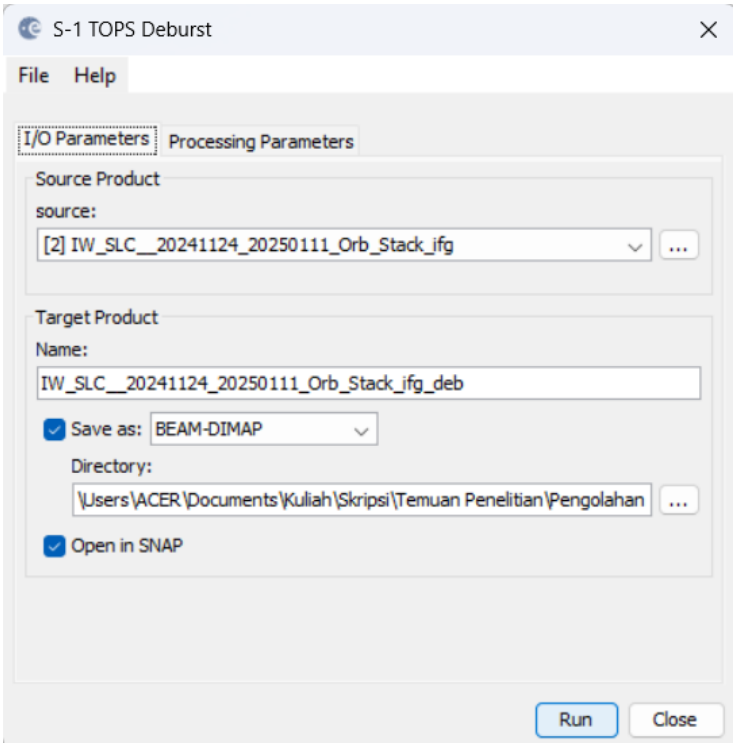
3.8.1.3. Proses *Deburst*

Langkah ini bertujuan untuk menyusun kembali citra yang terpisah-pisah (disebut *burst*) menjadi satu gambar yang utuh dan kontinu.

a. Selanjutnya adalah tahap *S-1 TOPS Deburst*, caranya klik *Radar > Sentinel-1 TOPS > S-1 TOPS Deburst.*

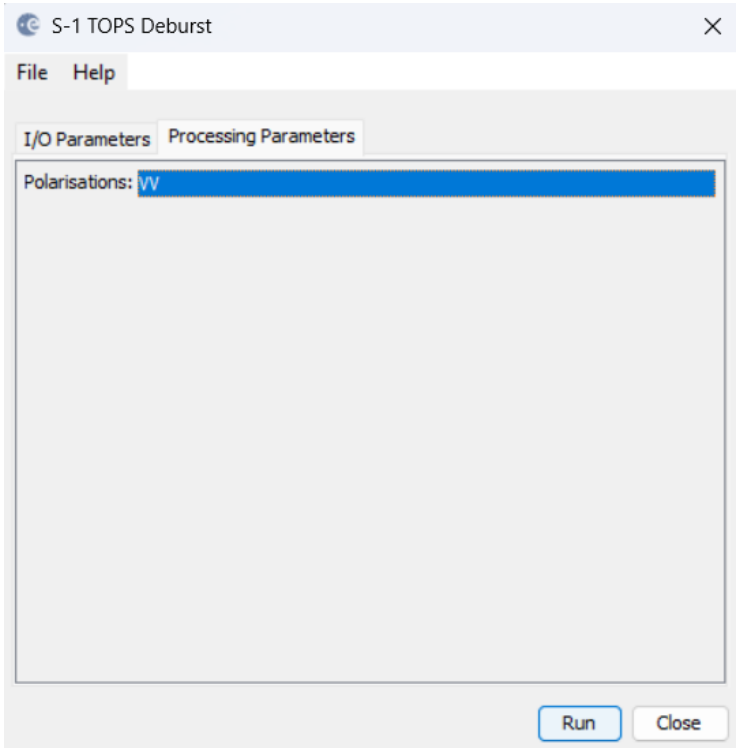
b. Pada *tab I/O Parameters*:

- *Source Product* yang digunakan adalah …_Orb_Stack_ifg,.
- *Target Product* dinamai …_Orb_Stack_ifg_deb.
- Direktori penyimpanan diarahkan ke folder pengguna.



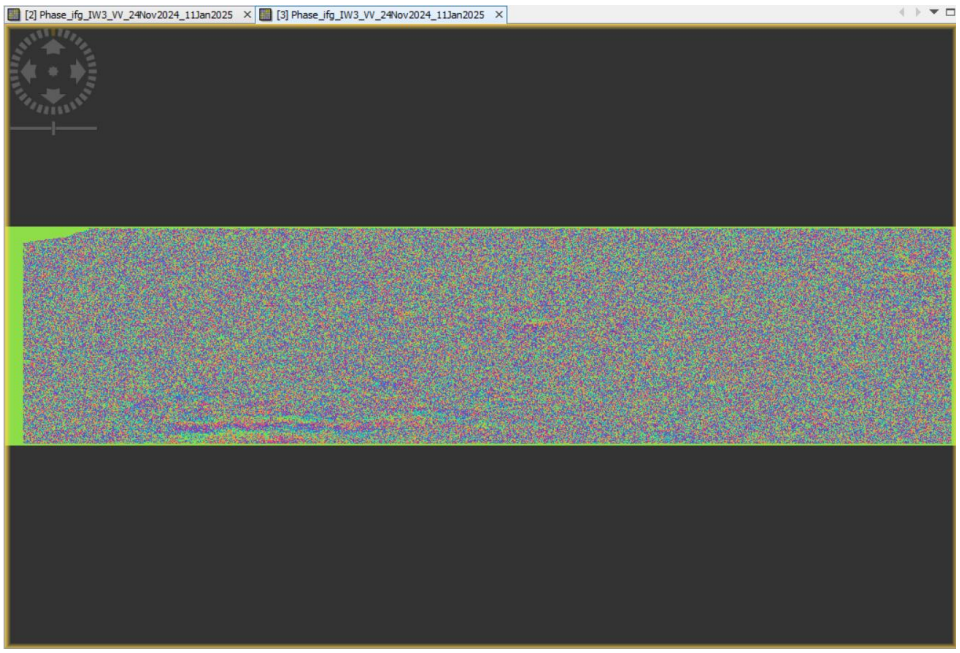
Gambar 3. 13 Tampilan Jendela S-1 TOPS Deburst Tab I/O Parameters

c. Pada *tab Processing Parameters* menampilkan parameter Polarisation diisi dengan VV. Artinya hanya data dengan polarisasi VV (*Vertical transmit, Vertical receive*) yang akan diproses. Kemudian klik tombol "*Run*" untuk memulai proses *Deburst*.



Gambar 3. 14 Tampilan Jendela S-1 TOPS Deburst Tab Processing Parameters

Berikut hasil dari proses *deburst*.



Gambar 3. 15 Tampilan Hasil Proses Deburst

Warna-warni yang muncul menunjukkan distribusi nilai fase dari -π hingga +π, yang merepresentasikan perbedaan jarak antara satelit dan permukaan bumi akibat deformasi. Pola warna pelangi yang muncul, khususnya di bagian bawah, mengindikasikan adanya perubahan elevasi atau pergeseran permukaan (deformasi) yang terjadi antara dua waktu akuisisi. Setiap satu siklus penuh warna (*fringe*) mewakili perubahan jarak. Area dengan tampilan kasar dan acak menandakan tingkat dekorelasi yang tinggi, yang bisa disebabkan oleh vegetasi, perairan, atau permukaan yang berubah drastis. Pada gambar ini juga terlihat tepi berwarna hijau terang yang kemungkinan berasal dari perbedaan ukuran atau ketidaksesuaian spasial citra masukan saat proses *coregistration*.

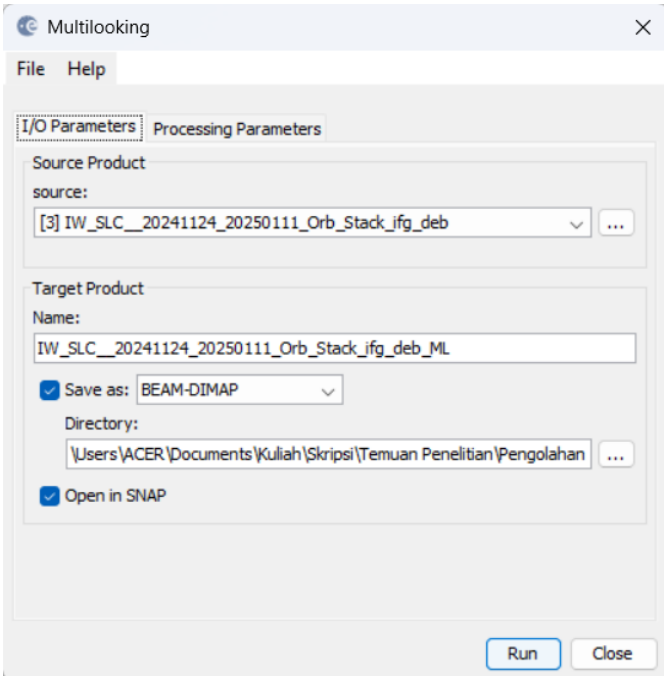
3.8.1.4. *Multilook*

Multilooking bertujuan untuk mengurangi *speckle* dan meningkatkan rasio *signal-to-noise* dengan cara merata-ratakan beberapa piksel dalam arah tertentu, terutama dalam arah range. Proses ini diterapkan pada *band-band* penting seperti fase interferogram, koherensi, dan komponen *real-imaginer*, guna mempertajam pola deformasi tanpa terlalu mengorbankan resolusi spasial.

Caranya:

a. Klik *Radar* > *SAR Utilities* > *Multilooking*

b. Pada bagian *I/O Parameters* terlihat bahwa data sumber yang digunakan adalah …_Orb_Stack_ifg_deb. Output dari proses ini adalah …_Orb_Stack_ifg_deb_ML.

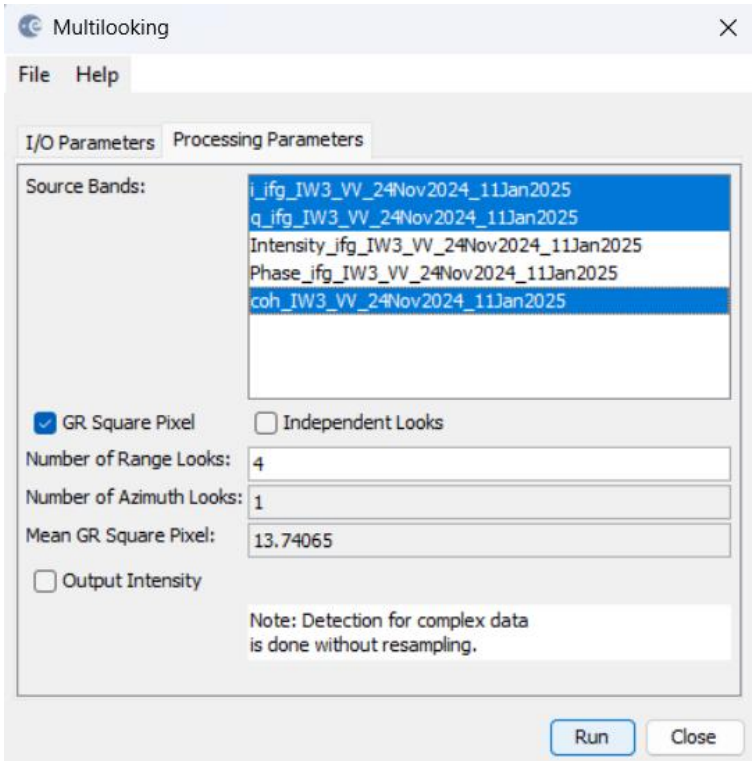


Gambar 3. 16 Tampilan Jendela Multilooking I/O Parameters

c. Selanjutnya pada tab Processing Parameters:

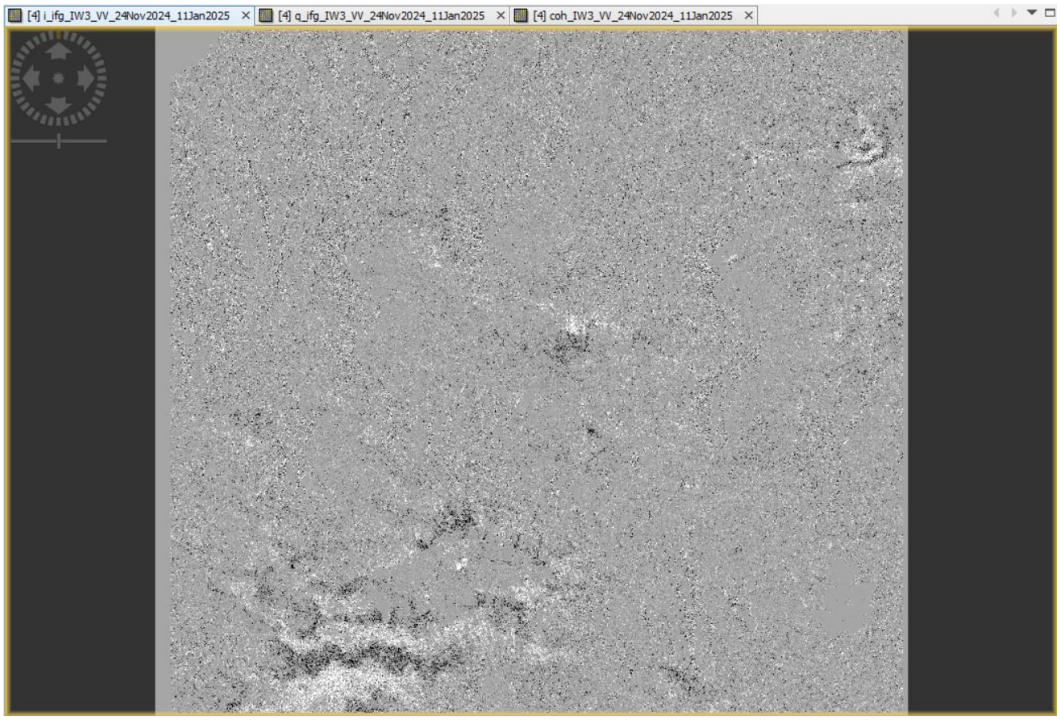
- Pada bagian *Source Bands*, terlihat bahwa lima band kompleks telah dipilih sebagai input, yaitu i_ifg, q_ifg, Intensity_ifg, Phase_ifg, dan coh, yang semuanya berasal dari pasangan citra Sentinel-1. *Band-band* ini mewakili komponen *real-imaginer (i/q)*, intensitas, fase interferogram, dan koherensi.

- Di bagian *GR Square Pixel* yang aktif (dicentang), yang berarti *output multilooking* akan dihasilkan dengan piksel berbentuk bujur sangkar dalam sistem koordinat *ground range*.
- Pengaturan *Number of Range Looks* diatur ke angka 4 dan *Number of Azimuth Looks* ke angka 1, yang artinya 4 piksel akan dirata-rata dalam arah *range* (horizontal terhadap arah sensor) dan 1 piksel dalam arah azimuth (searah gerak satelit). Ini bertujuan untuk menyeimbangkan antara resolusi spasial dan pengurangan noise, terutama pada fase interferogram.
- Nilai *Mean GR Square Pixel* yang ditampilkan secara otomatis oleh SNAP adalah sekitar 13.74, menunjukkan ukuran rata-rata piksel hasil *multilooking* dalam satuan meter.
- Opsi *Output Intensity* tidak dicentang karena fokus utama biasanya adalah pada data fase dan koherensi untuk analisis deformasi, bukan intensitas radar.



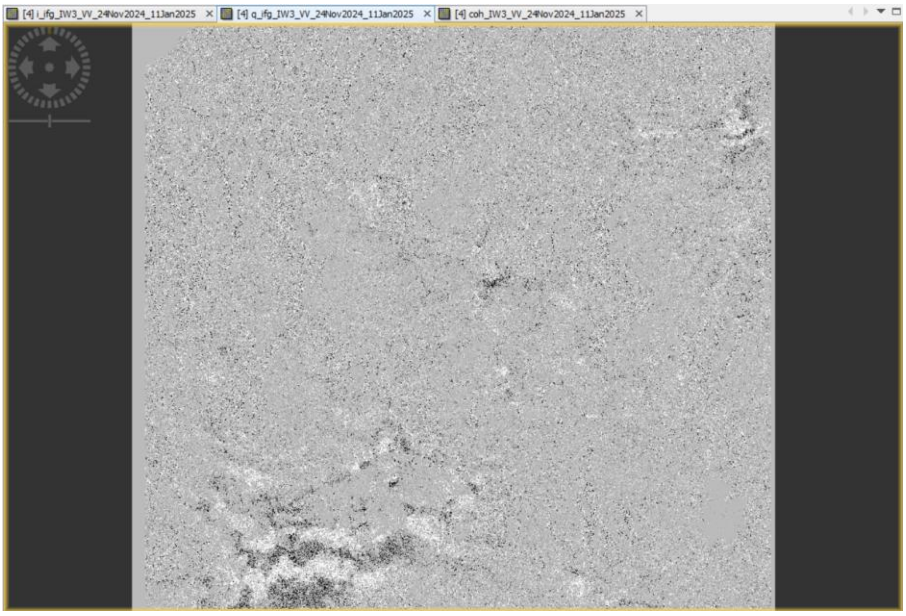
Gambar 3. 17 Tampilan Jendela Multilooking Processing Parameters

Hasilnya adalah Gambar 3.18 yang menampilkan amplitudo *real part* (i_ifg) dari sinyal kompleks interferogram antara citra *primary* dan *secondary* Gambar ini menunjukkan variasi tekstur permukaan yang dihasilkan dari perbedaan sinyal radar, dan membantu mengidentifikasi perubahan yang terjadi di permukaan tanah.



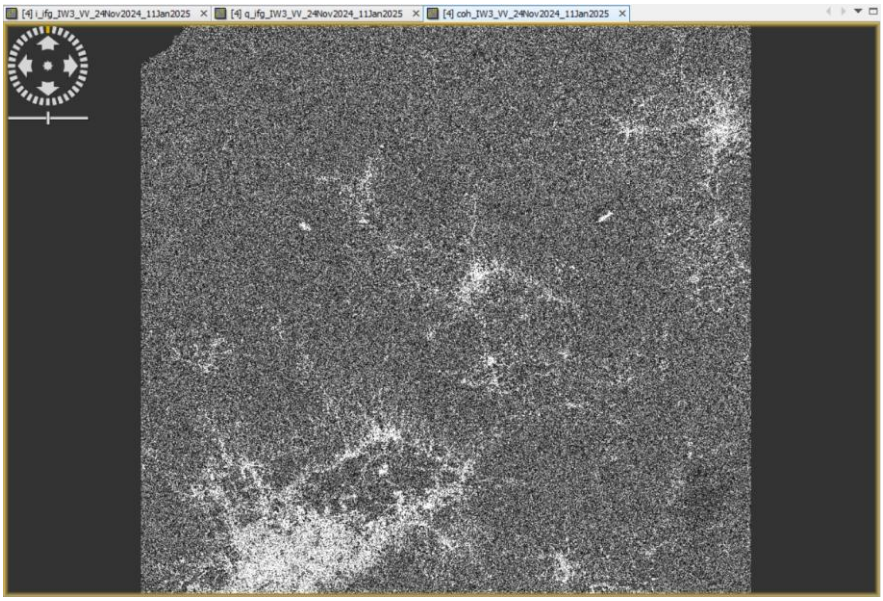
Gambar 3. 18 Tampilan Hasil Proses Multilooking (1)

Gambar 3.19 menampilkan amplitudo *imaginary part* (q_ifg), yang merupakan bagian imajiner dari data kompleks. Gambar ini juga memberikan gambaran tekstur permukaan, namun berasal dari sudut fase yang berbeda, dan keduanya (i dan q) diperlukan dalam pengolahan interferometri untuk memperoleh nilai fase dan koherensi.



Gambar 3. 19 Tampilan Hasil Proses Multilooking (2)

Sementara itu, Gambar 3.20 menampilkan peta koherensi (coh) hasil interferometri. Nilai koherensi ini mencerminkan seberapa konsisten pantulan sinyal radar antara dua waktu perekaman. Semakin tinggi nilai koherensi (warna putih), semakin baik kualitas interferogram-nya, yang menandakan tidak adanya perubahan signifikan atau gangguan (seperti vegetasi, air, atau pergeseran tanah besar). Sebaliknya, daerah dengan koherensi rendah (warna gelap) biasanya menunjukkan perubahan besar, tutupan lahan yang dinamis, atau gangguan *speckle* yang tinggi.



Gambar 3. 20 Tampilan Hasil Proses Multilooking (3)

Secara keseluruhan, ketiga gambar ini saling melengkapi dalam mengevaluasi kualitas dan karakteristik interferogram

3.8.1.5. *Goldstein Phase Filtering*

Goldstein Phase Filtering adalah proses penyaringan fase interferogram yang bertujuan untuk mengurangi derau (*noise*) dan memperkuat pola interferensi agar lebih jelas sebelum memasuki tahap *unwrapping*. Filter ini bekerja dengan menerapkan transformasi *Fourier* untuk memisahkan komponen sinyal dari gangguan, menghasilkan tampilan *fringe* yang lebih halus dan mudah ditafsirkan.

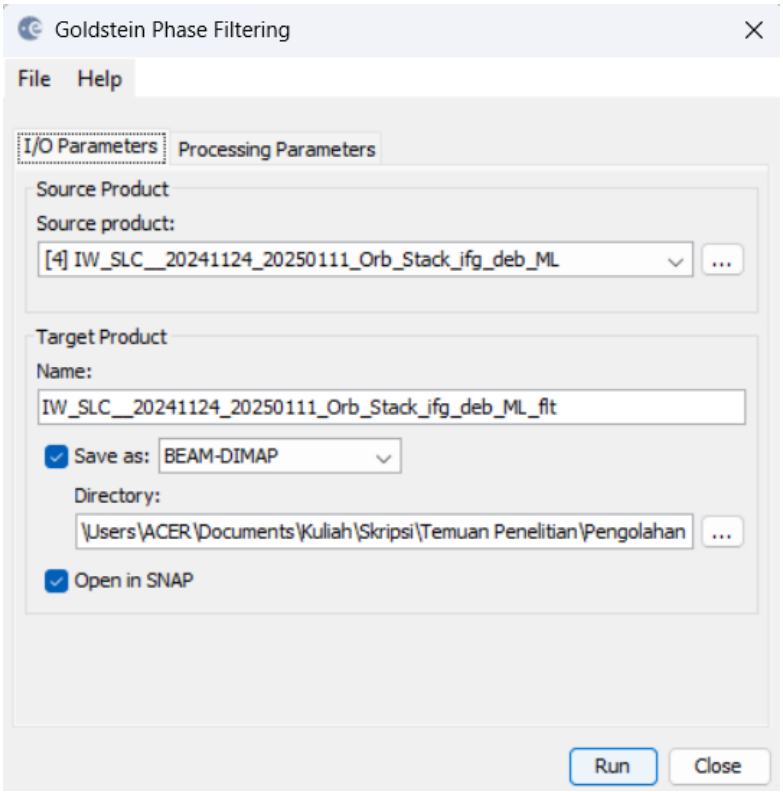
Caranya:

a. Caranya adalah klik *Radar → Interferometric → Filtering → Goldstein Phase Filtering*

b. Pada *tab I/O Parameters*, fitur-fitur yang diisi:

- Bagian *Source Product*, pilih produk interferogram hasil koregistrasi dan *multilooking* (ifg_deb_ML) yang akan difilter.

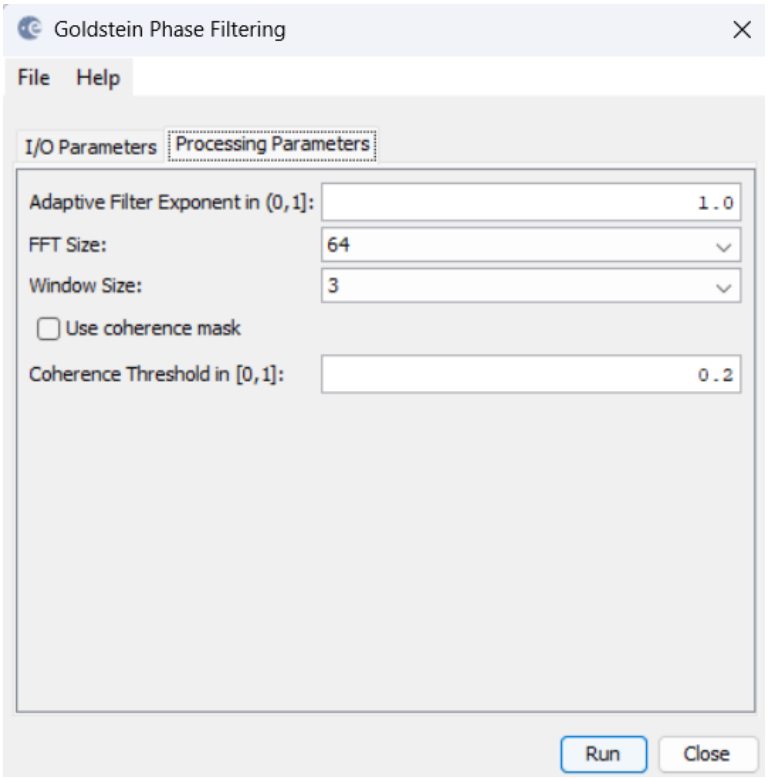
- Bagian *Target Product* ditentukan nama hasil file dengan penambahan akhiran _flt untuk menandakan bahwa *file* tersebut telah melalui tahap *filtering*.



Gambar 3. 21 Tampilan Jendela Goldstein Phase Filtering Tab I/O Parameters

c. *Tab Processing Parameters* memiliki beberapa parameter, antara lain:
   - *Adaptive Filter Exponent* (0–1) di-set ke nilai 1.0 menandakan filtrasi maksimal dan memberikan hasil yang lebih halus namun dapat menghilangkan beberapa detail penting jika digunakan berlebihan.
   - *FFT Size* di-set ke 64 menunjukkan ukuran blok untuk transformasi *Fourier* yang digunakan dalam proses *filtering*.
   - *Window Size* diatur ke 3 menandakan ukuran jendela (*window*) lokal yang digunakan untuk mengaplikasikan filter.
   - Opsi *Use coherence mask* tidak dicentang berarti filter akan diterapkan ke seluruh interferogram tanpa mempertimbangkan nilai koherensi.

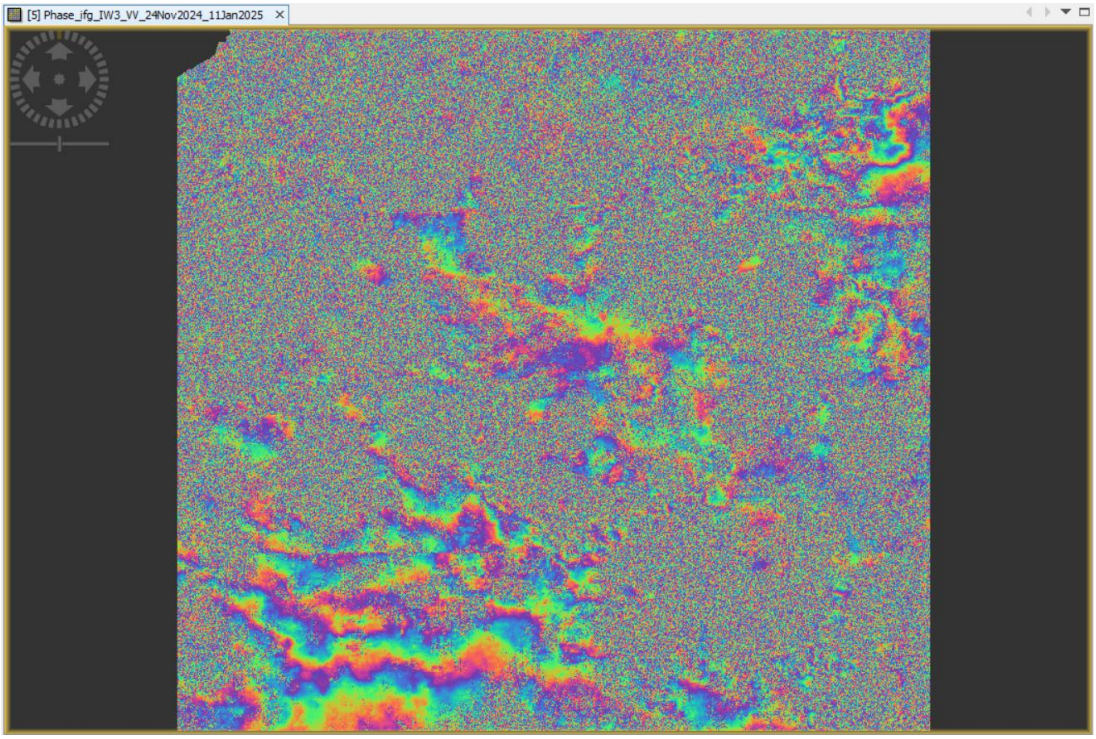
- *Coherence Threshold* in [0,1] di-set ke 0.2, namun tidak aktif karena *checkbox* "*Use coherence mask*" belum dicentang. Lalu klik tombol *Run*.



Gambar 3. 22 Tampilan Jendela Goldstein Phase Filtering Tab

Proses ini menghasilkan citra yang memperlihatkan informasi perbedaan fase antara dua citra radar dari tanggal yang berbeda. Perbedaan fase ini bisa disebabkan oleh deformasi permukaan tanah, perubahan kondisi atmosfer, vegetasi, atau *noise*.

Warna-warni pelangi (seperti ungu, hijau, merah, biru) menggambarkan variasi fase, namun interferogram ini masih terlihat sangat bising (*noisy*) terutama di area-area dengan tekstur kasar dan tidak beraturan. Hal ini umum terjadi sebelum dilakukan *Goldstein Phase Filtering* yang bertujuan untuk menghaluskan pola-pola fase dan mengurangi noise agar lebih mudah dalam proses selanjutnya seperti *phase unwrapping* dan analisis deformasi permukaan.



Gambar 3. 23 Tampilan Hasil Proses Goldstein Phase Filtering

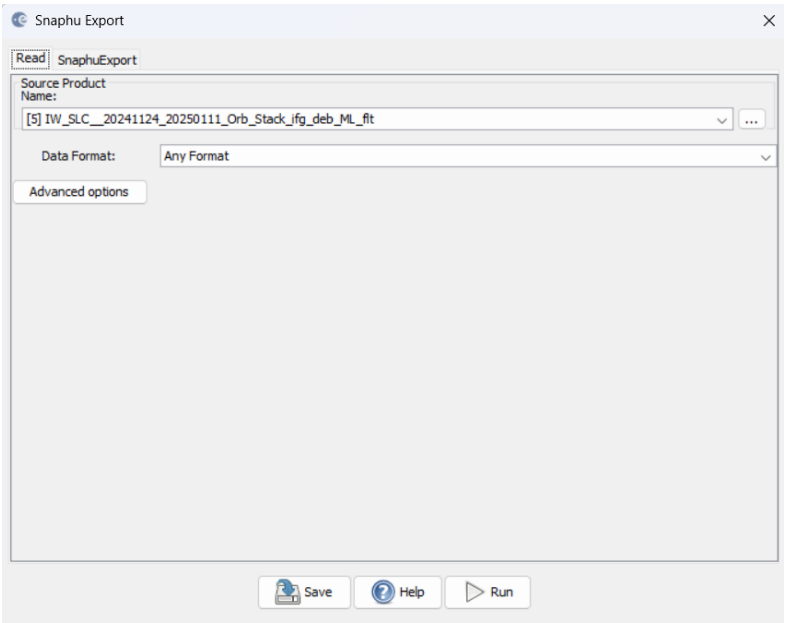
3.8.1.6. *Unwrapping*

a) Snaphu Export

Snaphu Export merupakan tahap awal dalam proses *unwrapping* fase *interferogram* dengan mempersiapkan data agar kompatibel dengan perangkat lunak pihak ketiga bernama SNAPHU. Caranya:

1). Untuk mengekspor data ke format yang dapat dibaca oleh perangkat lunak pihak ketiga bernama SNAPHU. Langkah ini dilakukan melalui menu *Radar > Interferometric > Unwrapping > Snaphu Export.*

2). Pada *tab Read*, opsi *Source Product* diisi dengan produk hasil *Goldstein Phase Filtering*

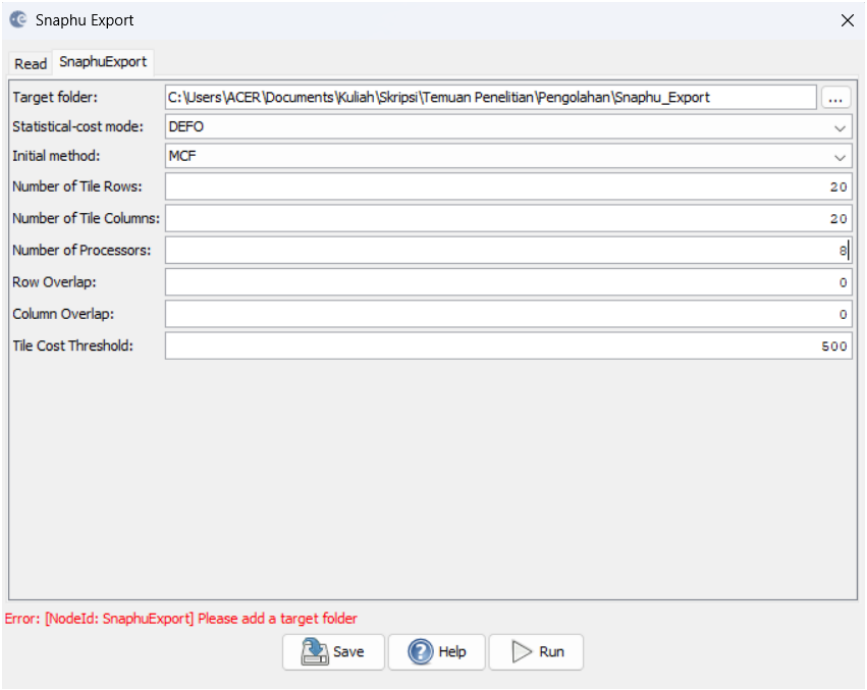


Gambar 3. 24 Tampilan Jendela Snaphu Export Tab Read

3). Pada *tab Snaphu Export*, parameter-parameter yang diisi adalah sebagai berikut.

- *Target Folder* menentukan lokasi penyimpanan output ekspor.
- *Statistical-cost mode* diatur ke DEFO untuk analisis deformasi seperti gempa atau subsidensi.

- *Initial Method* diatur ke MCF (*Minimum Cost Flow*) yang cocok untuk menjaga kestabilan *unwrapping* pada data deformasi.
- *Number of Tile Rows* dan *Columns* menentukan pembagian interferogram menjadi ubin-ubin kecil yakni 10 x 10 untuk mempercepat proses dan mengurangi beban komputasi.
- *Number of Processors* menunjukkan jumlah inti CPU yang akan digunakan selama proses *unwrapping* di SNAPHU diisi 8 tergantung kemampuan perangkat.
- *Row Overlap* dan *Column Overlap* dibiarkan default-nya 0.
- *Tile Cost Threshold* mengatur ambang batas biaya jaringan *tile* yang *default*-nya 500 dan berfungsi untuk mengontrol sensitivitas algoritma terhadap variasi fase dalam tile.
- Klik *Run* untuk menjalankan proses ini.



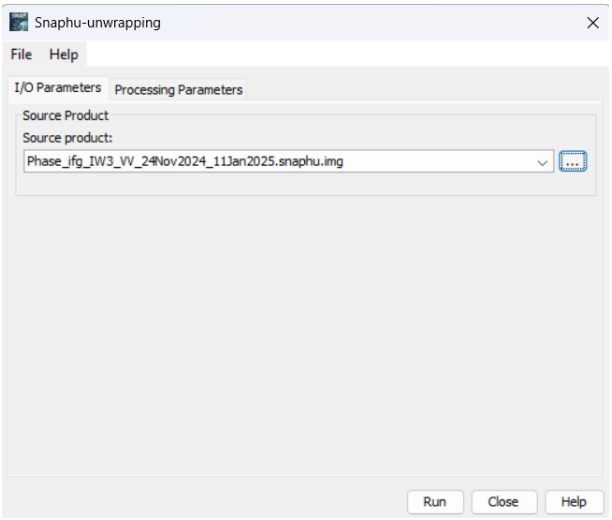
Gambar 3. 25 Tampilan Jendela Snaphu Export Tab SnaphuExport

b) Snaphu-unwrapping

Snaphu Unwrapping merupakan proses utama dalam mereduksi nilai fase yang masih dalam bentuk *wrapped phase* menjadi *unwrapped phase* menggunakan algoritma dari SNAPHU. Proses ini memanfaatkan file hasil ekspor sebelumnya dan dijalankan dengan mengatur lokasi kerja serta parameter pemrosesan di dalam SNAP.

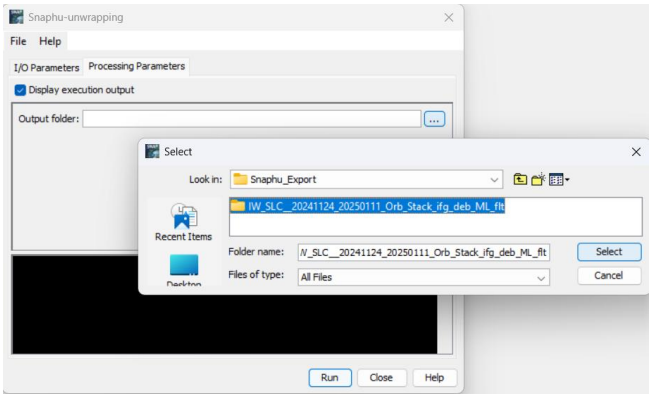
1). Klik *Radar* > *Interferometric* > *Unwrapping* > *Snaphu-unwrapping*

2). Pada *tab I/O Parameters*, bagian *Source Product* diisi dengan file hasil *unwrapping* yang berformat .snaphu.img.



Gambar 3. 26 Tampilan Jendela Snaphu-unwrapping Tab I/O Parameters

3). Kemudian pada *tab Processing Parameters* pilih hasil ekspor SNAPHU sebelumnya sebagai lokasi kerja untuk mengeksekusi proses *unwrapping*.



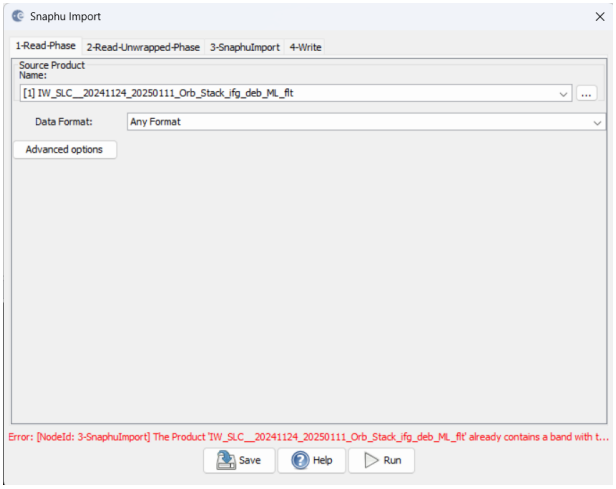
Gambar 3. 27 Tampilan Jendela Select Pada Tab Processing Parameters

c) Snaphu Import

Snaphu Import adalah tahapan untuk memasukkan kembali hasil *unwrapping* dari SNAPHU ke dalam SNAP agar dapat digunakan dalam proses lanjutan analisis deformasi. Caranya:

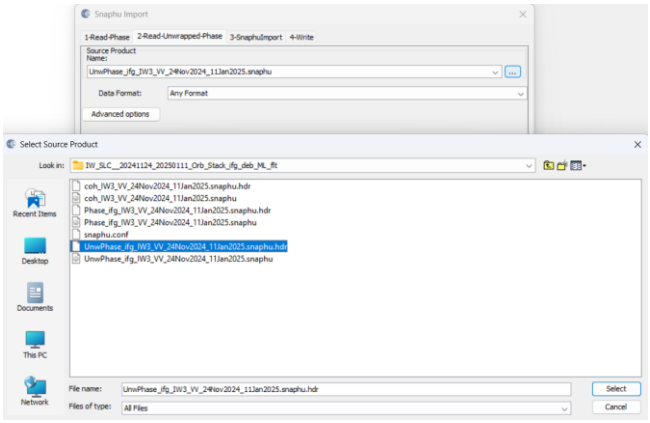
1). Klik *Radar > Interferometric > Unwrapping > Snaphu Import.*

2). Pada *tab 1-Read-Phase*, isi *Source Product* dengan produk hasil *Goldstein Phase Filtering* sebelumnya. Selain itu, terdapat error yang muncul karena memilih produk yang sudah memiliki *band unwrapped phase*.



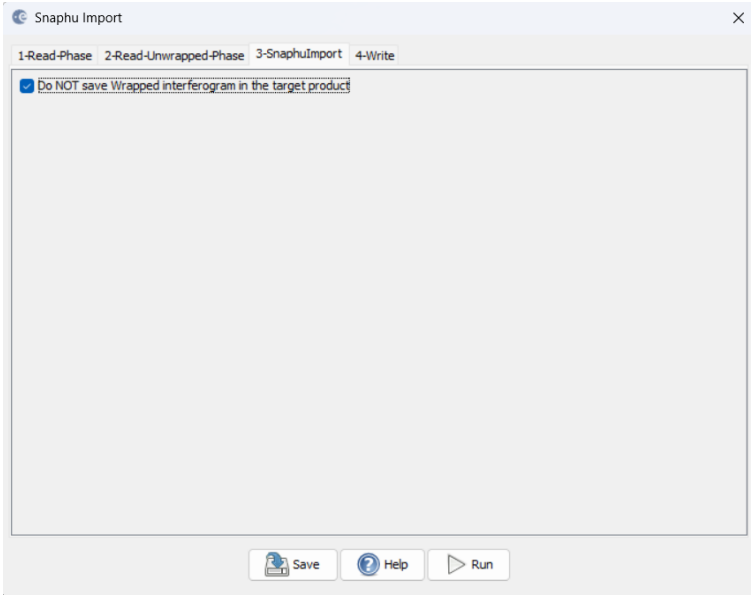
Gambar 3. 28 Tampilan Jendela Snaphu Import Tab 1-Read-Phase

3). Pada bagian *Source Product di tab 2- Read-Unwrapped-Phase*, pilih file hasil *unwrapping* dari output SNAPHU yang berisi informasi metadata hasil *unwrapping* dari folder *Snaphu Export*



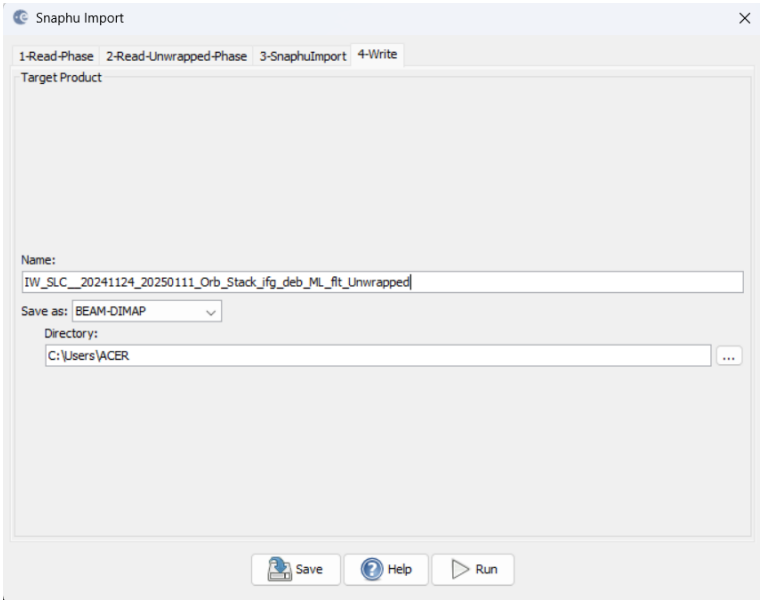
Gambar 3. 29 Tampilan Jendela Select Source Product

4). Kemudian ceklis *Do NOT save Wrapped interferogram in the target product* untuk tidak menyimpan *wrapped interferogram* di produk akhir. Ini adalah opsi opsional yang tidak mempengaruhi hasil *unwrapped*-nya.



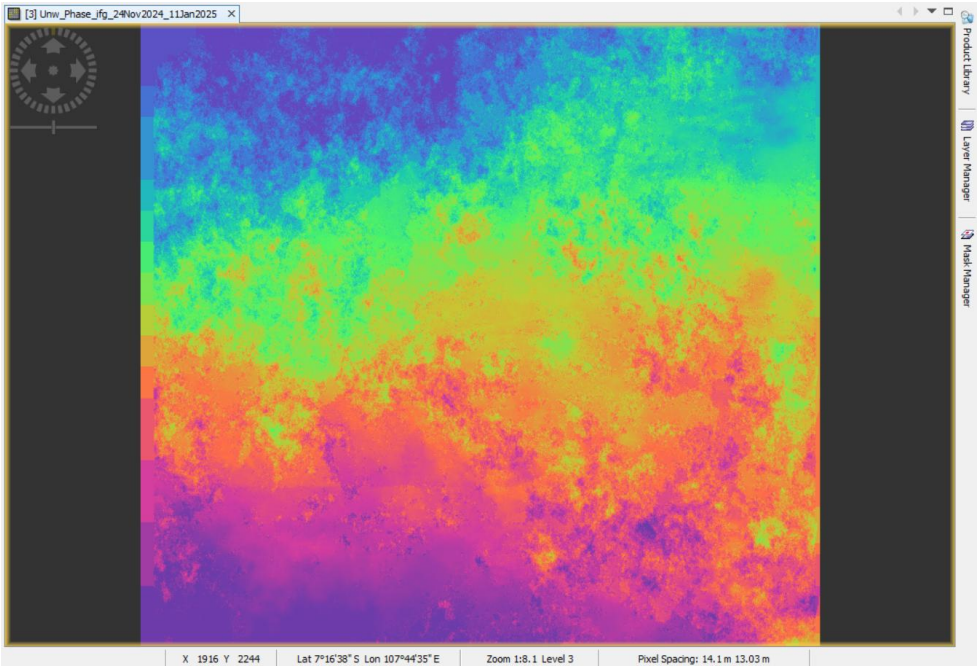
Gambar 3. 30 Tampilan Jendela Snaphu Import Tab 3-SnaphuImport

5). Pada *tab Write*, beri nama produk akhir dengan akhiran *_Unwrapped*, dan memilih format BEAM-DIMAP untuk menyimpan hasilnya di komputer. Klik *Run* untuk menjalankan proses ini.



Gambar 3. 31 Tampilan Jendela Snaphu Import Tab 4-Write

Hasilnya dapat dilihat dari Gambar 3.32 bahwa hasil *unwrapping* cukup halus dengan sedikit gangguan visual, menandakan proses berjalan baik tanpa adanya artefak besar atau kesalahan pengolahan. Pola warna yang muncul menunjukkan adanya variasi spasial yang kemungkinan besar mencerminkan deformasi akibat aktivitas tektonik. Secara visual, pola kontur fase menyiratkan kemungkinan adanya deformasi radial atau linier tergantung orientasi garis pandang radar (*line of sight*) dan lokasi episenter gempa. Lokasi pada peta juga menunjukkan area di sekitar Kabupaten Garut, Jawa Barat, Indonesia, dengan *pixel spacing* sekitar 11 x 11 meter, mengindikasikan bahwa data ini telah di-*multilook* untuk mengurangi *speckle* dan memperhalus tampilan deformasi skala besar.



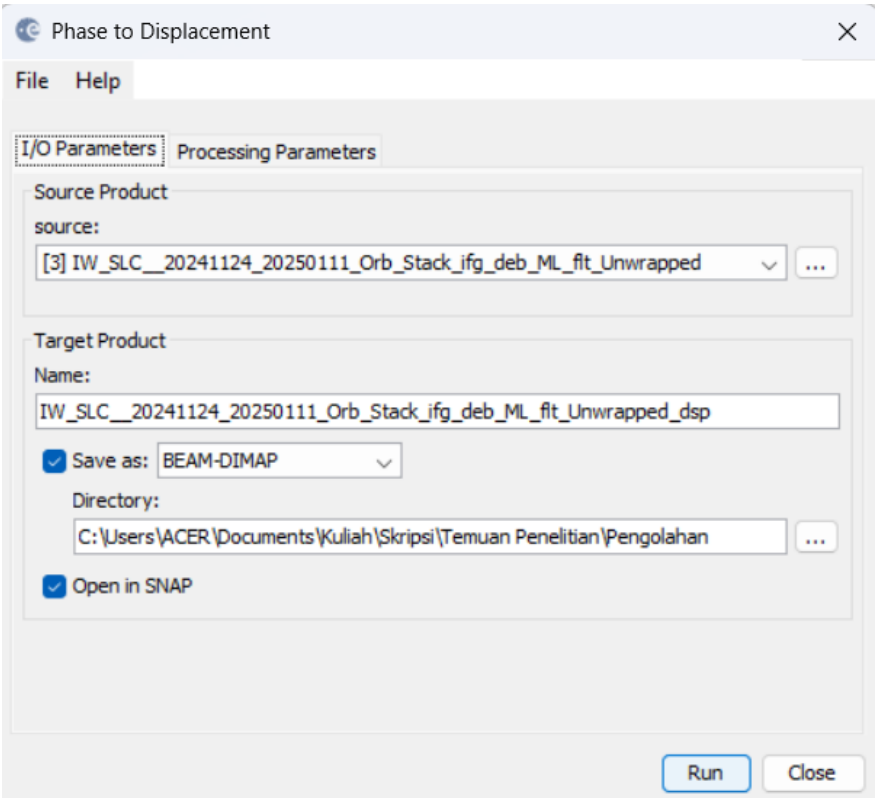
Gambar 3. 32 Tampilan Hasil Proses Snaphu Import

3.8.1.7. *Phase to Displacement*

Phase to Displacement merupakan tahap konversi dari fase interferogram yang telah di-*unwrap* menjadi nilai pergeseran permukaan (*displacement*) dalam satuan milimeter. Proses ini mengkuantifikasi besarnya deformasi tanah, seperti akibat gempa atau penurunan tanah. Caranya:

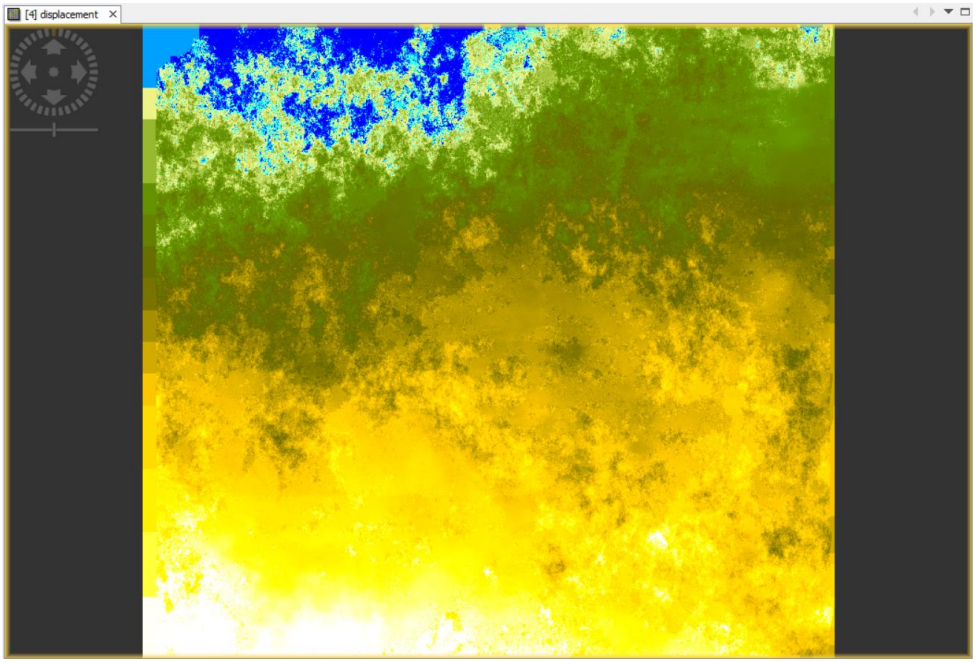
a. Klik *Radar* > *Interferometric* > *Products* > *Phase to Displacement*. Menu ini digunakan untuk mengubah hasil *unwrapping* fase interferometri menjadi nilai *displacement* (pergeseran permukaan tanah) dalam satuan milimeter.

b. Pada *tab I/O Parameters*, produk sumber yang digunakan adalah hasil *unwrapped interferogram*, dan *output*-nya adalah file *displacement* yang disimpan dalam format BEAM-DIMAP. Lokasi penyimpanan file ditentukan di direktori lokal. Opsi "*Open in SNAP*" mencentang agar hasilnya langsung terbuka setelah pemrosesan selesai.



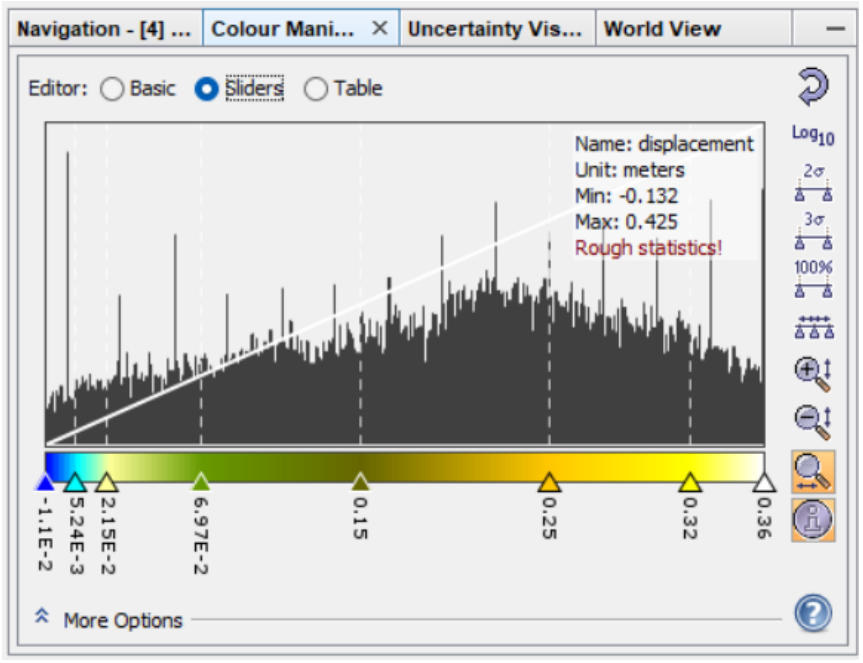
Gambar 3. 33 Tampilan Jendela Phase to Displacement Tab I/O Parameters

c. Gambar 3.34 memperlihatkan hasil visualisasi *displacement* pada *viewport SNAP*. Warna yang berbeda menunjukkan variasi *displacement* (pergeseran permukaan tanah), di mana warna biru mengindikasikan subsiden (penurunan), dan warna kuning hingga putih menunjukkan *uplift* (kenaikan). Peta ini menjadi hasil akhir dari pemrosesan konversi fase ke *displacement* dan penting dalam mendeteksi perubahan ketinggian permukaan bumi akibat aktivitas geologi, vulkanik, atau antropogenik.



Gambar 3. 34 Tampilan Hasil Proses Phase to Displacement

Selain itu terdapat *tab* yang menampilkan histogram dan *color mapping* dari nilai *displacement*. Sumbu X menunjukkan nilai *displacement* dalam milimeter. Skema warna dari biru ke kuning/putih mempermudah interpretasi visual area yang mengalami deformasi permukaan. Histogram ini juga menunjukkan distribusi piksel berdasarkan nilai *displacement*-nya.



Gambar 3. 35 Tampilan Statistik Hasil Proses Phase to Displacement

3.8.1.8. *Terrain Correction*

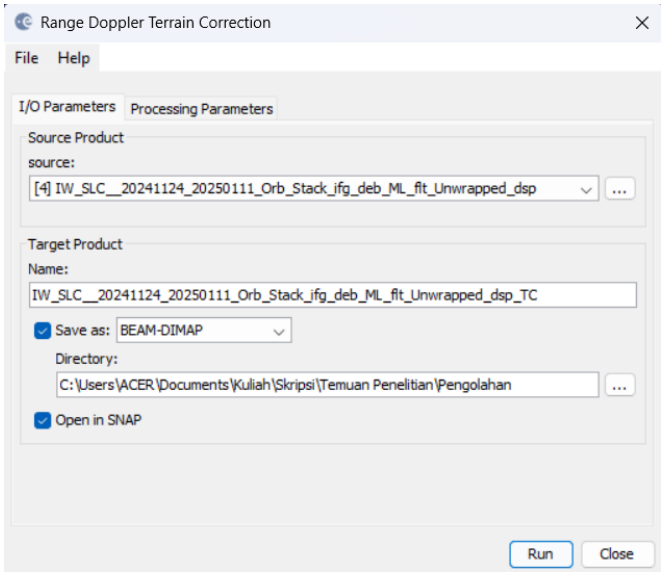
Terrain Correction adalah proses akhir dalam pengolahan deformasi yang bertujuan untuk mereproyeksi data hasil *displacement* ke sistem koordinat geospasial serta mengoreksi distorsi geometrik akibat topografi dan sudut pandang sensor radar. Caranya:

a. Caranya klik *Radar* > *Geometric* > *Terrain Correction* > *Range-Doppler Terrain Correction*

b. Pada *tab I/O Parameters*, berisi:

- Isi bagian *Source Product* dengan *file input* yang digunakan yaitu hasil *unwrapping displacement*.

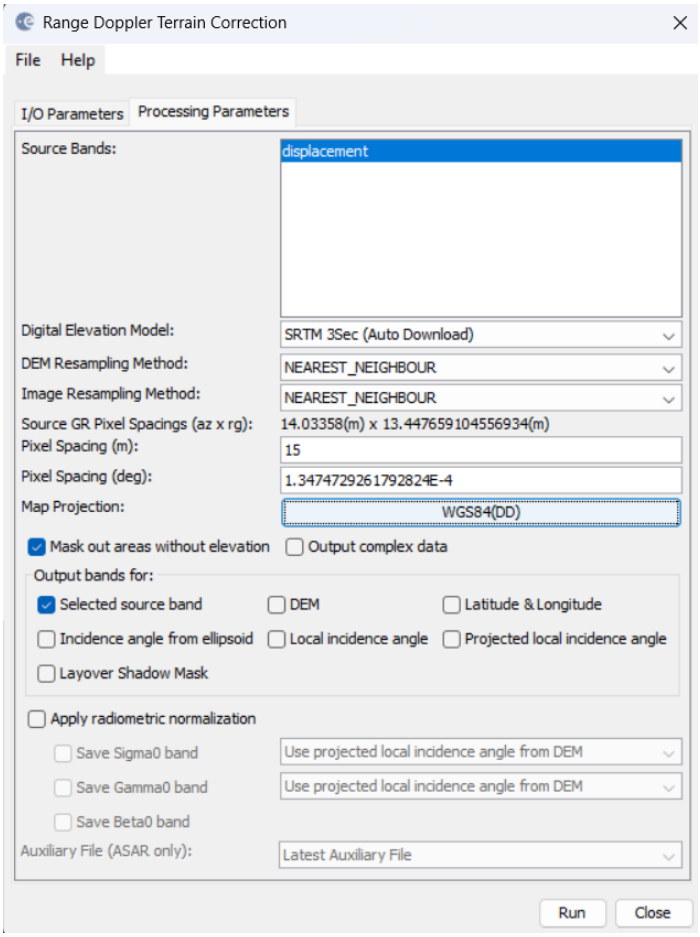
- Di bagian *Target Product*, nama *output file* disesuaikan dengan menambahkan akhiran _TC (singkatan dari *Terrain Correction*), dan format penyimpanan yang dipilih adalah BEAM-DIMAP.
- Jalur direktori output ditentukan ke folder pengguna.
- Opsi *Open in SNAP* dicentang agar hasil langsung dibuka setelah proses selesai.



Gambar 3. 36 Tampilan Jendela Range-Doppler Terrain Correction Tab I/O Parameters

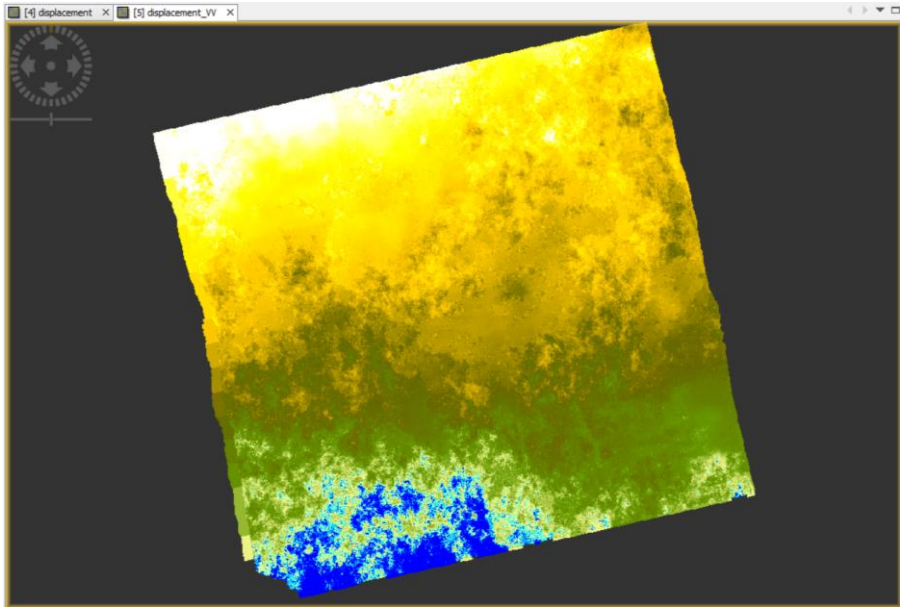
c. Selanjutnya *tab Processing* untuk mengatur detail teknis proses koreksi *geometric*:
   - Pada bagian *Source Bands*, dipilih *band* yaitu data hasil perhitungan deformasi tanah.
   - *Digital Elevation Model* (DEM) dipilih SRTM 3Sec (*Auto Download*)
   - Metode *resampling* untuk DEM dan citra diset ke *NEAREST_NEIGHBOUR* yang mempertahankan nilai piksel asli tanpa interpolasi kompleks, cocok untuk data kuantitatif seperti *displacement*.
   - Bagian *Source GR Pixel Spacings* menunjukkan resolusi asli citra radar dalam arah *azimuth* (14.03358 m) dan *range* (13.44766 m).
   - *Pixel Spacing* (m) disesuaikan menjadi 15 meter agar output memiliki resolusi *grid* yang lebih seragam.

- Sistem proyeksi menggunakan WGS84 (DD)
- Opsi *Mask out areas without elevation* dicentang yang berarti area tanpa data elevasi akan diabaikan dalam hasil koreksi agar tidak menghasilkan nilai yang tidak valid.
- Bagian *Output bands for* hanya *Selected source band* yang dicentang, artinya hanya *band displacement* yang akan diproses dan disimpan sebagai output.
- Koreksi radiometrik tidak dilakukan karena opsi *Apply radiometric normalization* tidak dicentang, dan *Auxiliary File* diset ke "*Latest Auxiliary File*", yang mengarahkan SNAP untuk menggunakan file pendukung terbaru dalam proses koreksi.



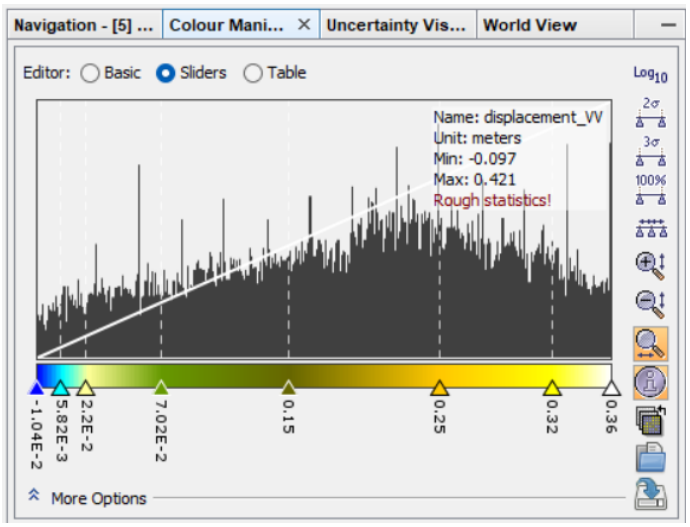
Gambar 3. 37 Tampilan Jendela Range-Doppler Terrain Correction Tab Processing Parameters

Hasilnya adalah gambar warna variasi nilai *displacement* dengan spektrum dari biru ke kuning dimana biru menunjukkan nilai negatif (penurunan permukaan/turun ke arah sensor), hijau netral, dan kuning menunjukkan nilai positif (kenaikan permukaan/menjauh dari sensor).



Gambar 3. 38 Tampilan Hasil Proses Terrain Correction

Kemudian terdapat juga jendela *Colour Manipulation* yang digunakan untuk mengatur palet warna dan interpretasi nilai numerik pada *band displacement_VV*. Histogram di bagian tengah menunjukkan distribusi nilai *displacement* pada seluruh piksel. Nilai minimum *displacement* adalah sekitar -0.097 milimeter, dan maksimum 0.421 milimeter (selisih 0.324 milimeter), menunjukkan adanya variasi deformasi tanah antar dua tanggal pengamatan. Skala warna diatur dari biru untuk nilai negatif, hijau untuk netral, dan kuning-putih untuk nilai positif.



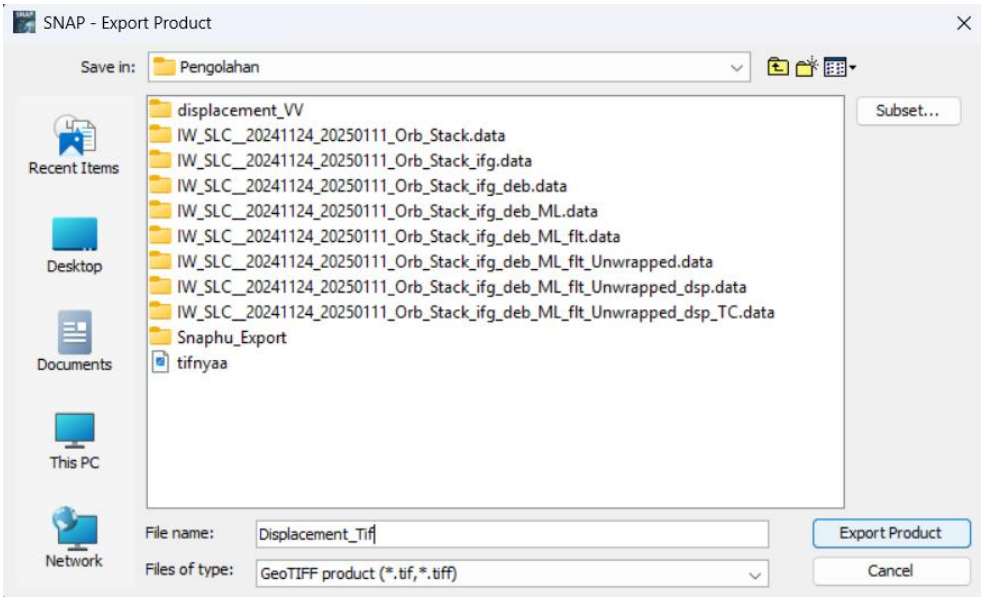
Gambar 3. 39 Tampilan Statistik Hasil Proses Terrain Correction

3.8.1.9. *Export*

Proses *Export* merupakan tahap akhir dalam *workflow* pengolahan data Sentinel-1 untuk deformasi permukaan, di mana produk hasil analisis yakni citra *displacement* yang telah melalui *terrain correction* diekspor ke format GeoTIFF.

a. Selanjutnya export produk ke format GeoTIFF dengan cara klik *File* > *Export* > *GeoTIFF/BigTIFF*

b. Beri nama produk yang akan *export* pada folder yang dipilih, lalu klik *Export Product.*



Gambar 3. 40 Tampilan Jendela Export Product yang Dihasilkan

3.8.2 Pemodelan Deformasi Permukaan Menggunakan Algoritma *Random Forest* dan *Multi-Layer Perceptron*

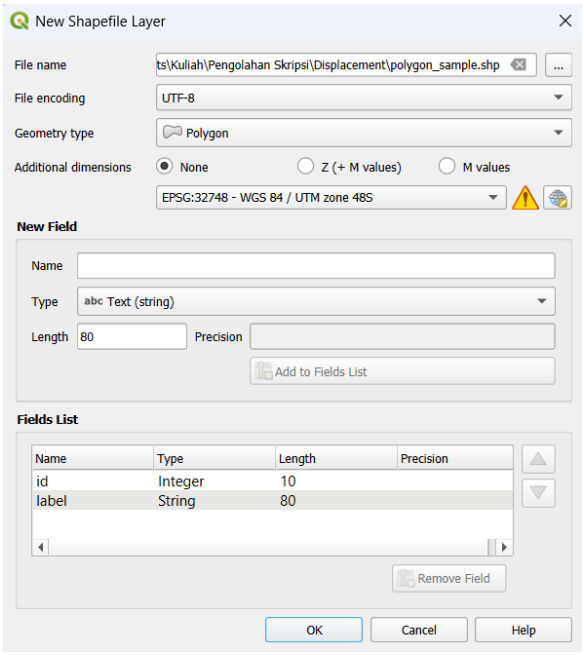
3.8.2.1 Source and Prepare Data

a. *Sampling* menggunakan *software* QGIS

Tahap berikutnya adalah pembuatan sampel pelatihan menggunakan QGIS dengan metode digitasi polygon, caranya adalah:

1). Klik *Layer* → *Create Layer* → *New Shapefile Layer*

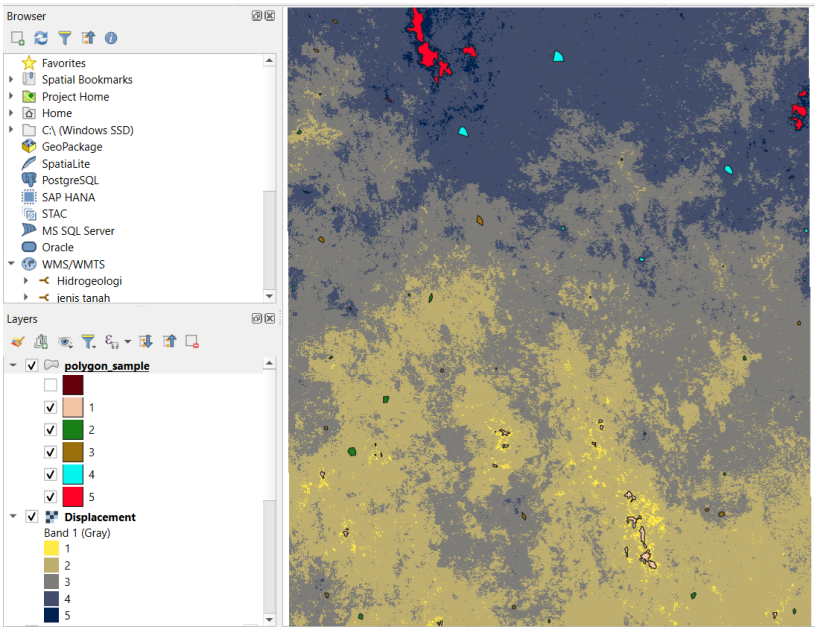
2). *File Name* diisi dengan nama *polygon_sample* dan tentukan lokasinya, *Geometry Type* pilih *Polygon*, CRS isi dengan UTM Zone 48S, pada *Field List* tambahkan kolom *label* bertipe data *string*. Klik OK



Gambar 3. 41 Tampilan Jendela Tools New Shapefile Layer

3). Setelah shp polygon_sample muncul, klik *Toggle Editing* lalu klik *Add Polygon Feature* untuk membuat *polygon sample*

4). Lalu buat *polygon-polygon sample* sesuai label kelas yang ada pada citra *displacement*



Gambar 3. 42 Membuat Polygon Sample Sesuai Jumlah Kelas Displacement

b. Persiapan Data dan Lingkungan Kerja

Pertama *mounting Google Drive* ke *Google Colab* agar dapat membaca dan menyimpan file seperti data *raster* (.tif), *shapefile*, hasil klasifikasi, dan grafik ke dalam *Google Drive*. *Google Drive* akan di-*mount* ke direktori/*content*/*drive* sehingga file seperti pada komputer lokal bisa diakses.

```
from google.colab import drive
drive.mount('/content/drive')
```

Sebelum menjalankan *script*, *install* semua *modul Python* di lingkungan kerja yang dipakai.

```
!pip install rasterio
import geopandas as gpd
from shapely.geometry import Point
import numpy as np
import rasterio
import rasterio.plot
import pandas as pd
import matplotlib.pyplot as plt
import os
from matplotlib.colors import ListedColormap, BoundaryNorm
```

c. Point Sampling

Langkah ini bertujuan untuk menghasilkan titik-titik sampel dalam area berlabel (*polygon GeoJSON*), di mana setiap titik akan mengambil nilai dari seluruh raster yang digunakan dan mewarisi *label* dari *polygon* tempatnya berada. Area berlabel yang direpresentasikan dalam format *polygon GeoJSON* akan diisi dengan titik-titik sampel dengan jarak antar titik sebesar 20 meter (*spacing* = 20).

Untuk mengambil nilai raster pada lokasi titik-titik tersebut, dibuat sebuah fungsi bernama *sample_points*. Fungsi ini menerima dua argumen, yaitu *path* ke file raster (*raster_path)* dan daftar titik (*points*). Di dalam fungsi, raster akan dibuka terlebih dahulu, lalu untuk setiap titik, koordinat geografisnya akan dikonversi menjadi indeks baris dan kolom dalam raster, kemudian dibaca nilai pikselnya. Jika titik berada di luar batas raster, maka nilainya akan diisi dengan *NaN*.

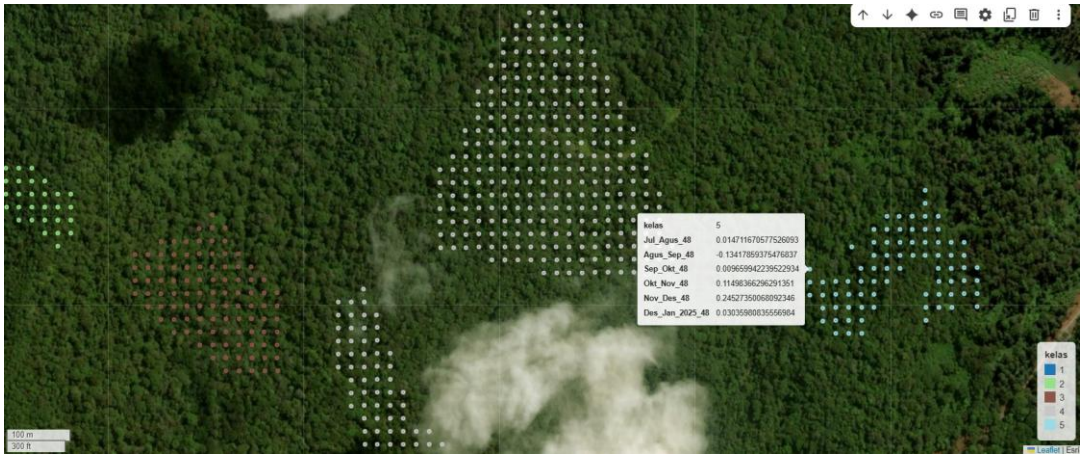
Dalam blok kode utama, geometri titik-titik dan labelnya disimpan dalam sebuah *dictionary* bernama data. Kemudian, fungsi *sample_raster_at_points* dipanggil untuk setiap file raster yang terdaftar dalam list files, dan hasil ekstraksi nilai disimpan ke *dictionary* data dengan nama file raster sebagai *key*. Terakhir, data ini dikonversi menjadi sebuah *GeoDataFrame* bernama *gdf_result*, dengan sistem koordinat yang disamakan dengan data aslinya.

```
import geopandas as gpd
import rasterio
import numpy as np
from shapely.geometry import Point
import os

# 1. Parameter
spacing = 20
polygon_path = '/content/drive/MyDrive/Skripsi
2/polygon_sample/polygon_sample48.geojson'
output_raster_paths = [
    '/content/drive/MyDrive/Skripsi 2/Input/Jul_Agus_48.tif',
    '/content/drive/MyDrive/Skripsi 2/Input/Agus_Sep_48.tif',
    '/content/drive/MyDrive/Skripsi 2/Input/Sep_Okt_48.tif',
    '/content/drive/MyDrive/Skripsi 2/Input/Okt_Nov_48.tif',
    '/content/drive/MyDrive/Skripsi 2/Input/Nov_Des_48.tif',
    '/content/drive/MyDrive/Skripsi 2/Input/Des_Jan_2025_48.tif'
]

# 2. Baca polygon sample
gdf = gpd.read_file(polygon_path)

# 3. Membuat titik sample dari polygon
def create_sample_points(polygon, spacing):
    minx, miny, maxx, maxy = polygon.bounds
    x_coords = np.arange(minx, maxx, spacing)
    y_coords = np.arange(miny, maxy, spacing)
    points = [Point(x, y) for x in x_coords for y in y_coords if
polygon.contains(Point(x, y))]
    return points

sample_points = []
labels = []

for _, row in gdf.iterrows():
    points = create_sample_points(row.geometry, spacing)
    sample_points.extend(points)
    labels.extend([row['kelas']] * len(points))

print(f"Jumlah titik sample: {len(sample_points)}")
```

Visualisasikan titik sampel menggunakan *script* berikut:

```
gdf_result.explore(
    tiles=
'https://server.arcgisonline.com/ArcGIS/rest/services/World_Imagery/MapServer/tile/{z}/{y}/{x}',
    attr='Esri',
    column='kelas'
)
```

Pada Gambar 3.43, terlihat bahwa seluruh titik sampel tersusun rapi di dalam area *polygon* berlabel, sehingga dapat dipastikan bahwa titik-titik tersebut telah merepresentasikan *polygon* berlabel yang telah dibuat sebelumnya.



Gambar 3. 43 Titik-Titik Sampel Dalam Area Berlabel (polygon GeoJSON)

Selanjutnya adalah mengambil nilai piksel dari data raster berdasarkan titik-titik sampel yang sebelumnya telah dibuat. Fungsi *sample_raster_at_points* digunakan untuk membaca nilai raster pada lokasi titik-titik tersebut. Fungsi ini menerima dua argumen, yaitu *raster_path* (lokasi file raster) dan *points* (daftar titik). Di dalam fungsi, raster dibuka menggunakan *rasterio*, kemudian setiap titik dikonversi menjadi indeks baris dan kolom raster untuk membaca nilai piksel yang sesuai. Jika titik berada di luar batas raster, maka nilai *NaN* akan diberikan.

Setelah fungsi dibuat, proses dilanjutkan dengan mendefinisikan daftar raster yang akan diambil nilainya. Untuk setiap file raster dalam daftar, dilakukan proses sampling nilai menggunakan fungsi tersebut, lalu hasilnya disimpan dalam *dictionary* data dengan nama file sebagai kuncinya.

Kemudian, *dictionary* ini diubah menjadi sebuah *GeoDataFrame* bernama gdf_result, yang berisi kolom geometri titik, label dari *polygon*, dan nilai raster yang telah disampling. Proyeksi spasialnya (CRS) disamakan dengan data awal. Terakhir, hasil *GeoDataFrame* ini disimpan ke dalam file *GeoJSON* agar dapat digunakan lebih lanjut, seperti visualisasi atau pemodelan.

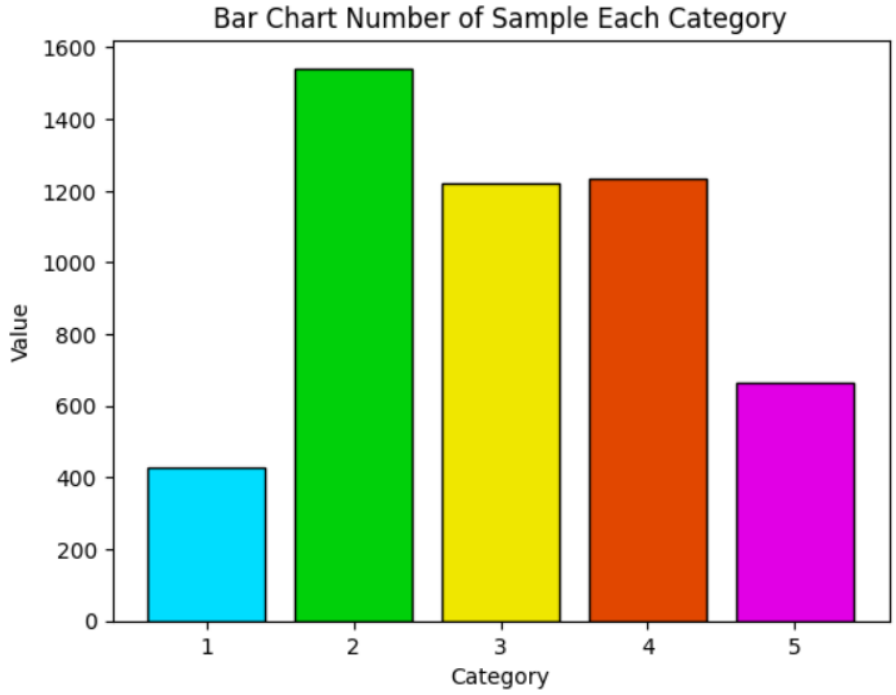
```
# 4. Sampling nilai raster
def sample_raster_at_points(raster_path, points):
    results = []
    with rasterio.open(raster_path) as src:
        for point in points:
            try:
                row, col = src.index(point.x, point.y)
                value = src.read(1)[row, col]
                results.append(value)
            except:
                results.append(np.nan)  # Handle jika titik di
luar raster
    return results

# 5. Simpan ke GeoDataFrame
data = {'geometry': sample_points, 'kelas': labels}

for path in output_raster_paths:
    name = os.path.basename(path).replace('.tif', '')
    print(f"Sampling raster {name}...")
    sampled_value = sample_raster_at_points(path, sample_points)
    data[name] = sampled_value

gdf_result = gpd.GeoDataFrame(data, crs=gdf.crs)

# 6. Simpan hasil ke file GeoJSON
output_path = '/content/drive/MyDrive/Skripsi
2/sample_points.geojson'
gdf_result.to_file(output_path, driver='GeoJSON')
print(f"Sukses disimpan ke {output_path}")
```

Berikutnya adalah memvisualisasikan jumlah sampel per kategori label kelas dalam bentuk diagram batang. Data label diambil dari *GeoDataFrame gdf_result*, lalu dihitung jumlah masing-masing kategori menggunakan *np.unique*. Warna batang ditentukan manual agar tiap kategori terlihat jelas. Diagram batang dibuat menggunakan *matplotlib*, dengan label pada sumbu x dan jumlah sampel pada sumbu y.

```
import matplotlib.pyplot as plt

# Ambil label dari gdf_result
labels = gdf_result['kelas']

# Hitung jumlah kemunculan tiap kategori
categories, values = np.unique(labels, return_counts=True)
unique_counts = dict(zip(categories, values))

# Print untuk cek hasil
print(unique_counts)

# Warna disesuaikan dengan urutan kategorinya (misal 5 kelas)
colors = ['#00ddff', '#00d00a', '#efe700', '#e14700', '#e100e5']

# Buat bar chart
fig, ax = plt.subplots()
bars = ax.bar(categories, values, color=colors[:len(categories)],
edgecolor='black')

# Judul dan label
ax.set_title('Bar Chart Number of Sample Each Category')
ax.set_xlabel('Category')
ax.set_ylabel('Value')

# Simpan gambar
plt.savefig('bar_chart.png', dpi=300)

# Tampilkan plot
plt.show()
```

Berikut ini adalah visualisasi diagram batang banyaknya sampel setiap kelas yang dibuat.

```
{'1': np.int64(426), '2': np.int64(1542), '3': np.int64(1221), '4': np.int64(1232), '5': np.int64(666)}
```



Gambar 3. 44 Diagram Batang Banyaknya Sampel Setiap Kelas Yang Dibuat.

1.8.2.2 *Data Cleaning*

Langkah ini dilakukan untuk mengecek apakah ada data duplikat, data yg hilang, data yg tidak valid, atau data *outlier* yang menyebabkan model menjadi bias. *Data cleaning* dilakukan pada file *sample_points.geojson* yang berisi titik-titik sampel. Langkah-langkah pembersihan yang dilakukan meliputi:

1. Statistik deskriptif ditampilkan untuk mengecek sebaran dan nilai ekstrem.
2. Pengecekan data hilang (*NaN*) untuk tiap kolom.
3. Pengecekan duplikasi baris dalam dataset.
4. Identifikasi nilai negatif, khususnya di kolom *VALUE*, untuk mendeteksi kemungkinan anomali.

```
import geopandas as gpd
import numpy as np

# Load sample points hasil sampling
filename = '/content/drive/MyDrive/Skripsi
2/sample_points.geojson'
gdf_sample = gpd.read_file(filename)

# Kolom raster sesuai nama file (otomatis jadi nama kolom)
columns = [
    'Jul_Agus_48',
    'Agus_Sep_48',
    'Sep_Okt_48',
    'Okt_Nov_48',
    'Nov_Des_48',
    'Des_Jan_2025_48'
]

# Statistik deskriptif
print("===== Statistik Deskriptif GeoDataFrame =====")
print(gdf_sample[columns].describe())

# Cek nilai NaN atau None
print('=========================================')
count_nan = gdf_sample.isna().sum()
print("Jumlah NaN dalam masing-masing kolom:")
print(count_nan)

# Cek data duplikat
print('=========================================')
duplicated_rows = gdf_sample.duplicated()
print(f"Jumlah data yang duplikat = {duplicated_rows.sum()}")

# Cek nilai negatif
print('=========================================')
count_negative = (gdf_sample[columns] < 0).sum()
print("Jumlah nilai negatif dalam data:")
print(count_negative)
```

Hasilnya adalah:

```
===== Statistik Deskriptif GeoDataFrame =====
       Jul_Agus_48  Agus_Sep_48   Sep_Okt_48   Okt_Nov_48   Nov_Des_48  \
count  5087.000000  5087.000000  5087.000000  5087.000000  5087.000000
mean      0.004307    -0.116730    -0.076798     0.205176     0.152266
std       0.021878     0.024117     0.149940     0.214988     0.051211
min      -0.070293    -0.202283    -0.393714    -0.080058     0.046117
25%      -0.010506    -0.132889    -0.240448     0.040923     0.111677
50%       0.006358    -0.112960     0.009176     0.097766     0.143201
75%       0.016281    -0.099756     0.034890     0.454105     0.189338
max       0.079461    -0.054466     0.094536     0.681935     0.276928

       Des_Jan_2025_48
count      5087.000000
mean          0.017815
std           0.024748
min          -0.048303
25%           0.000614
50%           0.015136
75%           0.036478
max           0.102690
========================================
Jumlah NaN dalam masing-masing kolom:
kelas              0
Jul_Agus_48        0
Agus_Sep_48        0
Sep_Okt_48         0
Okt_Nov_48         0
Nov_Des_48         0
Des_Jan_2025_48    0
geometry           0
dtype: int64
========================================
Jumlah data yang duplikat = 0
========================================
Jumlah nilai negatif dalam data:
Jul_Agus_48        1899
Agus_Sep_48        5087
Sep_Okt_48         2288
Okt_Nov_48          547
Nov_Des_48            0
Des_Jan_2025_48    1238
dtype: int64
```

Hasil pembersihan dan pemeriksaan kualitas data pada *GeoDataFrame* menunjukkan bahwa dataset telah siap untuk dianalisis lebih lanjut. Tidak ditemukan nilai kosong (*NaN*) maupun data duplikat, yang berarti integritas data terjaga dengan baik. Selain itu, kolom geometri (*geometry*) dan kategorisasi kelas (kelas) juga lengkap, sehingga data valid secara spasial maupun tematik.

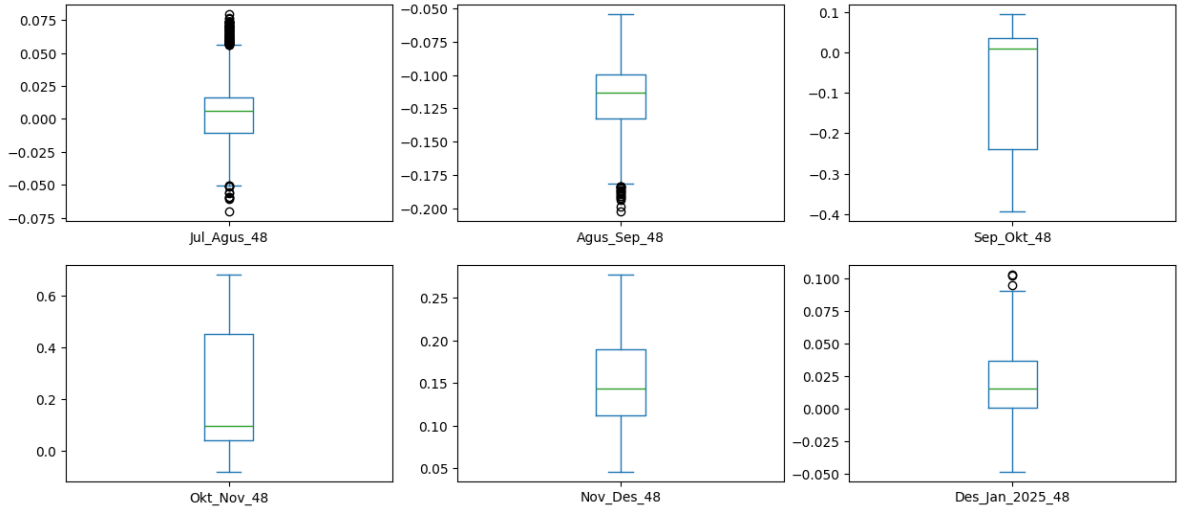
Statistik deskriptif untuk masing-masing periode bulanan (dari Juli 2024 hingga Januari 2025) mengungkapkan variasi nilai deformasi permukaan, diantaranya:

- Rata-rata deformasi tertinggi terjadi pada bulan Oktober–November sebesar 0.205, menunjukkan kemungkinan fase *uplift* yang signifikan. Rata-rata deformasi terendah berada di periode Agustus–September sebesar -0.117, menunjukkan dominasi fase subsiden saat itu.
- Nilai minimum paling ekstrem terpantau di periode September–Oktober (-0.393714), yang bisa menunjukkan area dengan penurunan permukaan tertinggi.
- Sebaliknya, nilai maksimum tertinggi tercatat pada periode Oktober–November (0.681935), mengindikasikan potensi area pengangkatan tanah yang sangat besar.

Selanjutnta membuat *boxplot* dari kolom *VALUE* pada data *gdf_sample* untuk memeriksa distribusi nilai dan mengidentifikasi potensi *outlier* secara visual (melihat kualitas data). *Plot* ditampilkan menggunakan *matplotlib*, dengan satu sumbu (*layout* 1x1) dan ukuran 10 x 6 inci.

```
import matplotlib.pyplot as plt
gdf_sample.plot(kind='box',
                subplots=True,
                layout=(3, 3),
                figsize=(15, 10),
                sharex=False,
                sharey=False)
plt.show()
```

Berikut merupakan visualisasi box plot dari kolom *VALUE*.



Gambar 3. 45 Box Plot untuk Memeriksa Distribusi Nilai

Dari *boxplot* diatas, berikut penjelasan untuk masing-masing data deformasi permukaan dari rentang waktu yang berbeda:

1. Jul_Agus_48
    - Sebaran kecil, nilai-nilai cenderung mendekati nol.
    - Terdapat banyak *outlier* di kedua sisi (atas dan bawah), menunjukkan variabilitas tinggi meskipun mayoritas data tidak terlalu menyimpang.

Median mendekati nol → perubahan relatif stabil pada periode ini.

2. Agus_Sep_48
    - Nilai deformasi umumnya negatif, median juga negatif.
    - Menunjukkan adanya penurunan permukaan (*subsidence*) selama periode ini.
    - Outlier juga lebih banyak ke arah negatif → ada daerah yang mengalami deformasi signifikan ke bawah.
3. Sep_Okt_48
    - Sebaran data cukup luas dari sekitar -0.4 hingga 0.1.
    - Median mendekati batas atas, artinya banyak piksel mengalami kenaikan permukaan (*uplift*), namun ada sebagian cukup besar yang turun jauh.
    - Outlier ke bawah cukup ekstrem → bisa jadi ada anomali lokal atau noise yang tinggi.

4. Okt_Nov_48
    - Distribusi positif secara keseluruhan, median di atas nol.
    - Hampir semua nilai menunjukkan kenaikan permukaan.
    - Rentang interkuartil (IQR) juga luas → variasi deformasi tinggi, tapi positif semua → indikasi ada proses geologi atau pasca-gempa yang mendorong elevasi.
5. Nov_Des_48
    - Hampir seluruh nilai positif, median cukup tinggi.
    - *Outlier* tidak ekstrem → kenaikan permukaan terjadi secara merata.
    - Mungkin menunjukkan periode pemulihan atau *uplift* pasca deformasi sebelumnya.
6. Des_Jan_2025_48
    - Median tetap positif tapi rendah → kenaikan tetap terjadi, namun tidak sekuat periode sebelumnya.
    - Ada beberapa outlier di atas dan bawah → variasi mulai muncul kembali, mungkin sistem mulai menstabilkan diri.

1.8.2.3 *Encoding Categorical Variable*

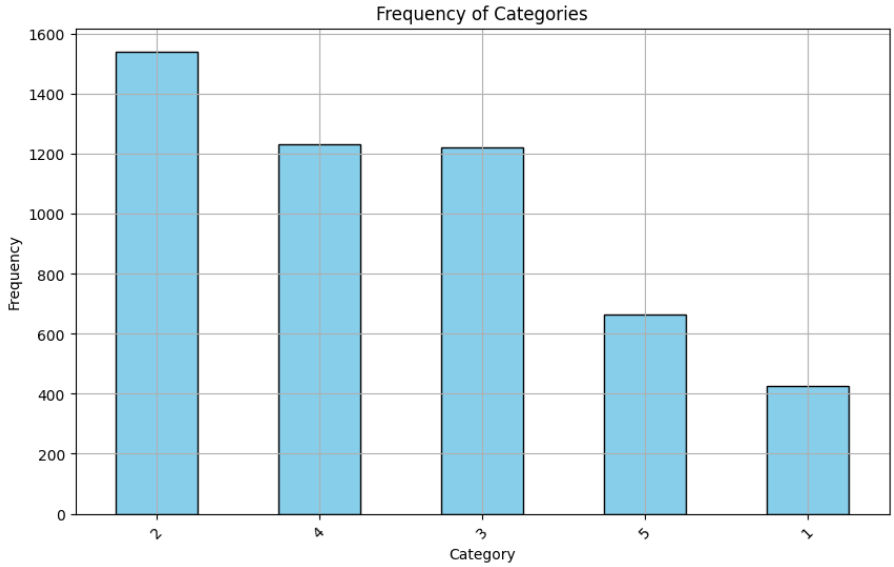
Selanjutnya melakukan *encoding* terhadap label kategorikal pada data geospasial menggunakan `*LabelEncoder*` dari *Scikit-learn*. Awalnya, data label dari kolom `*'label'*` pada `*GeoDataFrame*` diubah menjadi format numerik agar dapat digunakan dalam model *machine learning*. Kemudian, dibuat pemetaan antara label numerik dan label asli dalam bentuk *dictionary* (`*label_mapping*`). Setelah itu, fungsi `*get_unique_count*()` digunakan untuk menghitung jumlah kemunculan setiap label yang telah di-*encoding*, memberikan gambaran distribusi kelas dalam bentuk *dictionary*.

Untuk validasi hasil *encoding*, *script* ini juga membandingkan distribusi label asli menggunakan metode `*value_counts*()` dari pandas. Hasilnya ditampilkan dalam bentuk *bar chart* untuk memvisualisasikan frekuensi masing-masing kategori. Dengan pendekatan ini, pengguna dapat memastikan bahwa proses *encoding* berjalan dengan benar dan distribusi label tetap konsisten sebelum digunakan dalam tahap pelatihan model atau analisis lanjutan.

```
from sklearn.preprocessing import LabelEncoder
X = gdf_sample[columns]
Y = gdf_sample['kelas']

# encoded label
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(Y)

# mapping encoder
label_mapping = {index: label for index, label in
enumerate(label_encoder.classes_)}

# mendapatkan nilai count untuk masing-masing unqiue value
def get_unique_count(labels):
    unique_values, counts = np.unique(labels, return_counts=True)
    unique_counts = dict(zip(unique_values, counts))
    return unique_counts
unique_counts = get_unique_count(y_encoded)

print(f"label mapping: {label_mapping}")
print(unique_counts)

print('============================================================'
)
print('validasi dengan pandas')
value_counts = gdf_sample['kelas'].value_counts()
print(value_counts)
plt.figure(figsize=(10, 6))
value_counts.plot(kind='bar', color='skyblue', edgecolor='black')
plt.title('Frequency of Categories')
plt.xlabel('Category')
plt.ylabel('Frequency')
plt.xticks(rotation=45)
plt.grid(True)
plt.show()
```

Berikut adalah hasilnya

```
label mapping: {0: '1', 1: '2', 2: '3', 3: '4', 4: '5'}
{np.int64(0): np.int64(426), np.int64(1): np.int64(1542), np.int64(2): np.int64(1221), np.int64(3): np.int64(1232), np.int64(4): np.int64(666)}
============================================================
validasi dengan pandas
kelas
2    1542
4    1232
3    1221
5     666
1     426
Name: count, dtype: int64
```



Gambar 3. 46 Diagram Batang Frekuensi dari Tiap Kategori Label

Proses pemetaan label dilakukan dengan mengonversi label numerik hasil prediksi model ke dalam kelas sebenarnya menggunakan label mapping, yaitu:

- Label 0 sebagai kelas 1,
- Label 1 sebagai kelas 2,
- Label 2 sebagai kelas 3,
- Label 3 sebagai kelas 4, dan
- Label 4 sebagai kelas 5.

Berdasarkan hasil prediksi, diperoleh jumlah piksel untuk masing-masing kelas yaitu

- Kelas 1 sebanyak 426 piksel,
- Kelas 2 sebanyak 1542 piksel,
- Kelas 3 sebanyak 1221 piksel,
- Kelas 4 sebanyak 1232 piksel, dan
- Kelas 5 sebanyak 666 piksel.

1.8.2.4 *Splitting Train and Test Data*

Fungsi *train_test_split* dari *Scikit-learn* untuk membagi data fitur (X) dan label yang sudah di-*encoding* (*y_encoded*) menjadi data pelatihan dan pengujian dengan proporsi 80% untuk pelatihan dan 20% untuk pengujian. Parameter *random_state*=42 digunakan agar proses pembagian data bersifat *reproducible* atau konsisten saat dijalankan ulang. Setelah data terbagi, fungsi *get_unique_count*() dipanggil untuk menghitung jumlah data di setiap kategori label dalam *subset* data pelatihan (*y_train*).

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y_encoded,
test_size=0.2, random_state=42)
unique_counts = get_unique_count(y_train)
print(unique_counts)
```

Hasilnya adalah

```
{np.int64(0): np.int64(330), np.int64(1): np.int64(1232), np.int64(2): np.int64(979), np.int64(3): np.int64(1000), np.int64(4): np.int64(528)}
```

Hasil pembagian data pelatihan (*training*) menunjukkan distribusi jumlah piksel untuk setiap kelas setelah dilakukan proses *encoding*. Kelas 0 (yang mewakili kelas 1 setelah pemetaan) memiliki 330 piksel, kelas 1 sebanyak 1232 piksel, kelas 2 sebanyak 979 piksel, kelas 3 sebanyak 1000 piksel, dan kelas 4 sebanyak 528 piksel. Distribusi ini menunjukkan bahwa data pelatihan cukup seimbang di antara kelas-kelas utama, meskipun terdapat sedikit perbedaan jumlah

antar kelas, yang masih dalam batas wajar untuk keperluan pelatihan model klasifikasi.

1.8.2.5 *Normalize*

Normalize dilakukan agar skala data numerik memiliki rentang yang seragam dimana akan digunakan *min-max scaling* atau *standardization*. Data *displacement* (cm) diubah menjadi nilai antara 0 dan 1. Teknik *oversampling* dengan algoritma SMOTE (*Synthetic Minority Over-sampling Technique*) untuk menangani ketidakseimbangan kelas pada data pelatihan. Fungsi SMOTE dari pustaka *imblearn.over_sampling* secara otomatis membuat sampel sintetis dari kelas-kelas minoritas sehingga jumlah data pada tiap kategori menjadi seimbang. Setelah proses resampling dilakukan, data fitur (*X_train_resampled*) dan label (*y_train_resampled*) diacak ulang menggunakan *shuffle* untuk memastikan distribusi data acak secara merata saat dilatih ke model.

```
from imblearn.over_sampling import SMOTE
from sklearn.utils import shuffle

sm = smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled =
smote.fit_resample(X_train,y_train)
X_train_resampled, y_train_resampled = shuffle(X_train_resampled,
y_train_resampled, random_state=42)

unique_counts = get_unique_count(y_train_resampled)
print(unique_counts)
```

Hasilnya adalah

```
{np.int64(0): np.int64(1232), np.int64(1): np.int64(1232), np.int64(2): np.int64(1232), np.int64(3): np.int64(1232), np.int64(4): np.int64(1232)}
```

Setelah dilakukan proses oversampling menggunakan metode SMOTE (*Synthetic Minority Over-sampling Technique*), distribusi jumlah piksel untuk setiap kelas dalam data pelatihan menjadi seimbang, yaitu masing-masing kelas memiliki 1232 piksel. Hal ini menunjukkan bahwa ketimpangan jumlah data antar kelas berhasil diatasi, sehingga model memiliki peluang yang lebih adil dalam mempelajari pola dari setiap kelas secara setara.

Selanjutnya melakukan normalisasi data menggunakan metode *Standardization* dengan *StandardScaler* dari pustaka *sklearn.preprocessing*. Standardisasi ini mengubah skala fitur sehingga memiliki rata-rata (*mean*) 0 dan standar deviasi 1. Tujuannya adalah untuk memastikan bahwa semua fitur memiliki kontribusi yang sebanding terhadap model, terutama penting untuk algoritma yang sensitif terhadap skala data seperti MLP atau KNN.

Pada proses ini, normalisasi hanya dilakukan dengan *.fit_transform*() pada data pelatihan (*X_train_resampled*). Artinya, nilai rata-rata dan standar deviasi dihitung hanya dari data latih. Kemudian, transformasi yang sama diterapkan ke data uji (*X_test*) dengan menggunakan *.transform*() (tanpa *fit*). Hal ini penting dilakukan untuk mencegah data *leakage*, yaitu masuknya informasi dari data uji ke dalam proses pelatihan model, yang bisa menyebabkan hasil evaluasi menjadi tidak valid.

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_train_normalized = scaler.fit_transform(X_train_resampled)

#  NOTE: X_test tidak menggunakan fit_transform tapi menggunakan
scaler dari train
X_test_normalized = scaler.transform(X_test)
```

Simpan objek penting seperti *label_mapping* dan *scaler* ke dalam file .pkl menggunakan modul *pickle* agar dapat dengan mudah memuat kembali hasil pemetaan label dan model normalisasi tanpa harus memproses ulang dari awal. Proses ini sangat berguna untuk efisiensi dalam *pipeline machine learning*, terutama saat *deployment* atau saat melakukan prediksi ulang di waktu berbeda.

Selain itu, untuk menyimpan dan memuat *array NumPy*, dapat digunakan *np.save*() dan *np.load*() yang efisien dalam menangani data numerik berukuran besar. Teknik penyimpanan ini membantu menjaga konsistensi *preprocessing* dan hasil pelatihan saat model digunakan kembali.

```
from pickle import dump

with open("/content/drive/MyDrive/Skripsi
2/train_data/label_mapping.pkl", "wb") as f:
    dump(label_mapping, f, protocol=5)
with open("/content/drive/MyDrive/Skripsi
2/train_data/scaller.pkl", "wb") as f:
    dump(scaler, f, protocol=5)
```

Simpan data fitur dan label hasil *preprocessing* (*oversampling* dan normalisasi) ke dalam format *.npy* menggunakan np.save() dari *NumPy*. Empat file yang disimpan mencakup data latih (*X_train_resampled_normalized.npy* dan *Y_train_resampled_normalized.npy*) serta data uji (*X_test_resampled_normalized.npy* dan *Y_test_resampled_normalized.npy*). Penyimpanan ini bertujuan agar data dapat dimuat kembali dengan cepat tanpa perlu mengulangi proses *preprocessing*, cukup dengan *np.load*() saat dibutuhkan.

```
import numpy as np
np.save("/content/drive/MyDrive/Skripsi
2/train_data/X_train_resampled_normalized.npy",X_train_normalized)
np.save("/content/drive/MyDrive/Skripsi
2/train_data/Y_train_resampled_normalized.npy",y_train_resampled)
np.save("/content/drive/MyDrive/Skripsi
2/train_data/X_test_resampled_normalized.npy",X_test_normalized)
np.save("/content/drive/MyDrive/Skripsi
2/train_data/Y_test_resampled_normalized.npy",y_test)
```

1.8.2.6 Membangun Model *Machine Learning*

Sebelum melatih model menggunakan algoritma *Random Forest* dan MLP, data label hasil *resampling* ditampilkan seluruhnya untuk memastikan urutan label telah teracak sebelum digunakan dalam pelatihan model

```
import numpy as np
from pickle import load

X_train = np.load("/content/drive/MyDrive/Skripsi
2/train_data/X_train_resampled_normalized.npy")
y_train = np.load("/content/drive/MyDrive/Skripsi
2/train_data/Y_train_resampled_normalized.npy")
X_test = np.load("/content/drive/MyDrive/Skripsi
2/train_data/X_test_resampled_normalized.npy")
y_test = np.load("/content/drive/MyDrive/Skripsi
2/train_data/Y_test_resampled_normalized.npy")

with open("/content/drive/MyDrive/Skripsi
2/train_data/label_mapping.pkl", "rb") as f:
    label_mapping = load(f)
    print(label_mapping)
with open("/content/drive/MyDrive/Skripsi
2/train_data/scaller.pkl", "rb") as f:
    scaller = load(f)

# print untuk memastikan label sudah acak
with np.printoptions(threshold=np.inf):
    print(np.array_str(y_train))
```

Berikut ini hasilnya

```
{0: '1', 1: '2', 2: '3', 3: '4', 4: '5'}
[2 4 2 1 3 0 4 1 4 1 2 1 1 3 1 1 3 0 1 2 3 3 1 1 4 1 3 2 3 3 1 4 1 1 1 1 3
 0 1 4 2 4 1 0 1 2 1 1 0 3 1 3 1 4 4 0 2 2 3 3 1 4 4 3 3 2 0 0 3 0 4 0 4 4
 1 0 3 4 2 0 3 3 0 4 3 1 0 1 1 3 4 3 1 3 4 3 3 0 2 4 0 1 1 3 4 0 1 0 3 1 1
 0 3 4 2 1 1 2 1 0 1 1 1 3 2 4 3 2 0 4 4 2 3 3 2 1 0 0 1 4 1 2 4 4 1 1 1 1
 1 0 1 0 3 2 3 4 3 3 2 0 4 2 1 3 0 0 0 1 3 0 2 2 4 2 1 4 2 4 3 0 4 4 0 3 4
 2 2 0 2 3 2 4 2 3 4 4 4 1 2 2 2 2 1 4 0 0 0 2 3 4 3 3 0 3 0 2 2 2 3 4 2 2
```

Label kategori yang awalnya berupa string (‘1’ hingga ‘5’) telah diubah menjadi nilai numerik (0–4) menggunakan *LabelEncoder*, sehingga dapat digunakan dalam pelatihan model. Proses ini juga mencetak seluruh isi *y_train* untuk memastikan distribusi label sudah teracak dan mencerminkan hasil *resampling* yang seimbang.

a. Membuat Model *Random Forest*

Pertama, model *RandomForestClassifier* diimpor dari *sklearn.ensemble*, lalu diinisialisasi dengan *random_state=42* untuk memastikan hasil yang konsisten. Model dilatih menggunakan *X_train* dan *y_train* yang telah di-*resample* dan dinormalisasi. Setelah pelatihan selesai, model digunakan untuk memprediksi data pelatihan dan pengujian melalui metode *.predict*(), kemudian akurasinya dihitung menggunakan *accuracy_score* dari *sklearn.metrics*. Hasil akurasi untuk data pelatihan dan data uji dicetak untuk mengevaluasi performa model.

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

rf = RandomForestClassifier(random_state=42)
rf.fit(X_train, y_train)

# prediksi dari data traning
y_train_predict = rf.predict(X_train)
training_accuracy = accuracy_score(y_train, y_train_predict)


# akurasi terhadap data test
rf_y_test_predict = rf.predict(X_test)
test_accuracy = accuracy_score(y_test, rf_y_test_predict)

print('================================================')
print('random forrest')
print('================================================')
print(f"Training Accuracy: {training_accuracy}")
print(f"Test Accuracy: {test_accuracy}")
```

Hasilnya adalah

```
================================================
random forrest
================================================
Training Accuracy: 1.0
Test Accuracy: 0.9734774066797642
```

Model *Random Forest* yang dilatih pada data tanpa *resampling* menunjukkan akurasi yang sangat tinggi, dengan nilai akurasi sebesar 100% pada data pelatihan (*training*) dan 97,35% pada data pengujian (*test*). Hal ini mengindikasikan bahwa model mampu mengenali pola dalam data dengan sangat baik dan memiliki performa generalisasi yang kuat terhadap data yang belum pernah dilihat sebelumnya.

b. Membuat Model *Multi Layer Perceptron* (MLP)

Pertama, model *MLPClassifier* diinisialisasi dengan *random_state*=42 untuk memastikan hasil yang konsisten setiap kali dijalankan. Model kemudian dilatih menggunakan data pelatihan (*X_train, y_train*) yang telah diproses sebelumnya. Setelah pelatihan, model digunakan untuk melakukan prediksi terhadap data pelatihan dan data pengujian, kemudian akurasi dari masing-masing prediksi dihitung menggunakan fungsi *accuracy_score*.

```
from sklearn.neural_network import MLPClassifier

# Instantiate the MLP classifier
mlp = MLPClassifier(random_state=42)

# Train the model
mlp.fit(X_train, y_train)

# Predictions on the training data
y_train_predict = mlp.predict(X_train)
training_accuracy = accuracy_score(y_train, y_train_predict)

# Predictions on the test data
mlp_y_test_predict = mlp.predict(X_test)
test_accuracy = accuracy_score(y_test, mlp_y_test_predict)
```

```
print('================================================')
print('multi layer perceptron')
print('================================================')
print(f"Training Accuracy: {training_accuracy}")
print(f"Test Accuracy: {test_accuracy}")
```

Hasilnya adalah

```
================================================
multi layer perceptron
================================================
Training Accuracy: 0.9808441558441559
Test Accuracy: 0.9597249508840865
```

Model *Multi-Layer Perceptron* (MLP) yang dilatih tanpa *resampling* menghasilkan akurasi pelatihan sebesar 98,08% dan akurasi pengujian sebesar 95,97%. Hasil ini menunjukkan bahwa model mampu mempelajari pola dalam data dengan cukup baik dan tetap mempertahankan performa yang baik pada data pengujian, meskipun sedikit lebih rendah dibandingkan dengan *Random Forest*.

1.8.2.7 *Hyperparameter Tuning*

a. *Cross Validation*

Performa model klasifikasi dievaluasi menggunakan teknik validasi silang (*cross-validation*) sebanyak 4 lipatan (*4-fold cross-validation*). Dua model yang diuji adalah *Random Forest* (RF) dan *Multi-Layer Perceptron* (MLP). Fungsi *cross_val_score* dari pustaka *sklearn* digunakan untuk menghitung akurasi model pada setiap lipatan data pelatihan dan pengujian yang berbeda. Nilai akurasi dari tiap lipatan disimpan dalam variabel *scores*, lalu ditampilkan bersama dengan rata-rata akurasinya. Pendekatan ini memberikan gambaran umum tentang seberapa baik model mampu melakukan generalisasi terhadap data yang belum pernah dilihat, serta membantu menghindari bias evaluasi dari hanya menggunakan satu pembagian data pelatihan dan pengujian.

```
from sklearn.model_selection import cross_val_score
import warnings

# Abaikan warning dari MLP yang belum konvergen
warnings.filterwarnings('ignore', category=UserWarning)

# Cross-validation untuk Random Forest
scores = cross_val_score(rf, X_train, y_train, cv=4)
print('================================================')
print('random forest')
print('================================================')
print(f"Cross-Validation Scores: {scores}")
print(f"Mean Accuracy: {scores.mean()}")

# Cross-validation untuk MLP
scores = cross_val_score(mlp, X_train, y_train, cv=4)
print('================================================')
print('multi layer perceptron')
print('================================================')
print(f"Cross-Validation Scores: {scores}")
print(f"Mean Accuracy: {scores.mean()}")
```

Hasilnya adalah

```
================================================
random forest
================================================
Cross-Validation Scores: [0.98441558 0.97922078 0.98441558 0.98766234]
Mean Accuracy: 0.9839285714285715
================================================
multi layer perceptron
================================================
Cross-Validation Scores: [0.97077922 0.95974026 0.97012987 0.97532468]
Mean Accuracy: 0.9689935064935066
```

Berdasarkan hasil *cross-validation* menggunakan 4-*fold*, performa model *Random Forest* menunjukkan skor akurasi yang sangat tinggi dan konsisten, dengan nilai *cross-validation* berkisar antara 0.9792 hingga 0.9877, serta rata-rata akurasi (*mean accuracy*) sebesar 0.9839. Hal ini menunjukkan bahwa model *Random Forest* memiliki kestabilan yang baik dan mampu melakukan generalisasi dengan sangat akurat terhadap data pelatihan.

Sementara itu, model *Multi-Layer Perceptron* (MLP) juga menunjukkan performa yang cukup baik, dengan nilai akurasi *cross-validation* berkisar antara 0.9597 hingga 0.9753, dan rata-rata akurasi sebesar 0.9690. Meskipun sedikit lebih rendah dibandingkan *Random Forest*, performa MLP masih tergolong tinggi dan menunjukkan kemampuan klasifikasi yang cukup andal, meskipun terdapat indikasi bahwa proses *training*-nya belum sepenuhnya konvergen pada parameter yang digunakan saat ini.

Secara keseluruhan, kedua model memberikan hasil yang baik, namun *Random Forest* menunjukkan keunggulan dalam hal akurasi dan stabilitas model berdasarkan evaluasi *cross-validation*.

- *Evaluation Metric Random Forest*:

Performa model klasifikasi *Random Forest* dievaluasi dengan menghitung metrik evaluasi seperti akurasi, *precision*, *recall*, dan *F1-score* menggunakan pendekatan rata-rata berbobot (*weighted*). Selain itu, script ini juga menampilkan *classification report* yang memberikan rincian metrik untuk setiap kelas, serta confusion matrix yang divisualisasikan dengan label kelas yang sesuai dari *label_mapping*. Dengan fungsi *calc_metric*, pengguna dapat langsung melihat gambaran menyeluruh terhadap kinerja model terhadap data uji (*y_test*) dan prediksi model (*rf_y_test_predict*).

```
from sklearn.metrics import (
accuracy_score, precision_score, recall_score, f1_score,
confusion_matrix, classification_report, ConfusionMatrixDisplay
)

def calc_metric(y_data,y_predict,label_mapping):
    # Accuracy
    print(f"Accuracy: {accuracy_score(y_data,y_predict)}")

    # Precision, Recall, F1-score
    print(f"Precision: {precision_score(y_data,y_predict,
average='weighted')}")
    print(f"Recall: {recall_score(y_data,y_predict,
average='weighted')}")
    print(f"F1-score: {f1_score(y_data,y_predict,
average='weighted')}")
```

```
    # Classification Report
    print("Classification Report:")
    print(classification_report(y_data,y_predict))

    # Confusion Matrix
    print("Confusion Matrix:")
    # print(confusion_matrix(y_data,y_predict))
    labels = list(label_mapping.keys())
    display_labels = list(label_mapping.values())
    cm = confusion_matrix(y_test, y_predict, labels=labels)
    disp =
ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=display_
labels)
    disp.plot()


print('================================================')
print('random forest')
print('================================================')
calc_metric(y_test, rf_y_test_predict,label_mapping)
```

Hasilnya:

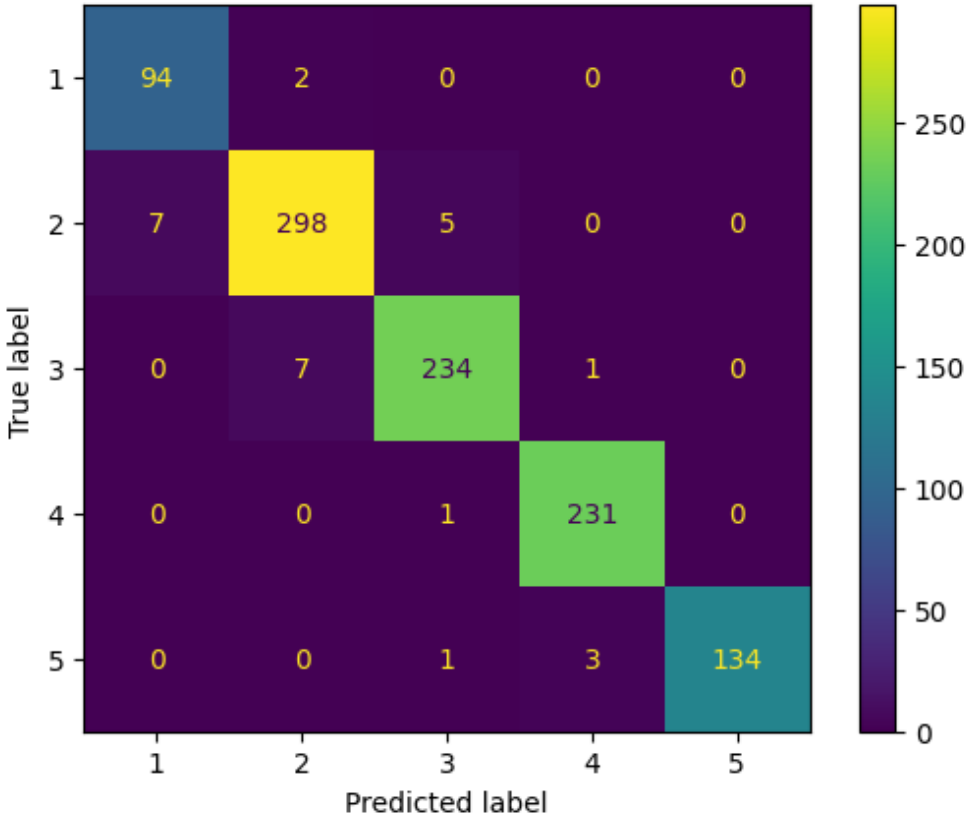
Evaluasi model

```
================================================
random forest
================================================
Accuracy: 0.9734774066797642
Precision: 0.9737530500563486
Recall: 0.9734774066797642
F1-score: 0.9735110805064084
Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.98      0.95        96
           1       0.97      0.96      0.97       310
           2       0.97      0.97      0.97       242
           3       0.98      1.00      0.99       232
           4       1.00      0.97      0.99       138

    accuracy                           0.97      1018
   macro avg       0.97      0.97      0.97      1018
weighted avg       0.97      0.97      0.97      1018
```

Berdasarkan hasil evaluasi model *Random Forest*, akurasi yang diperoleh sebesar 97.35%, yang menunjukkan bahwa model ini mampu mengklasifikasikan data dengan sangat baik. Nilai precision (ketepatan) sebesar 97.37%, recall (daya tangkap) sebesar 97.35%, dan *f1-score* sebesar 97.35%, semuanya berada pada kisaran tinggi, menandakan performa model yang seimbang antara ketepatan dan daya tangkap untuk semua kelas.

Dari *classification report*, diketahui bahwa setiap kelas memiliki nilai evaluasi yang tinggi. Misalnya, kelas 0 memiliki *precision* 0.93 dan *recall* 0.98, yang berarti sebagian besar prediksi kelas 0 benar, dan hampir semua data kelas 0 teridentifikasi dengan baik. Kelas 3 dan 4 bahkan menunjukkan performa yang sangat tinggi, dengan *recall* mencapai 1.00 dan *precision* mencapai 1.00 berturut-turut.

- *Confusion Matrix*



Gambar 3. 47 Confusion Matrix Evaluasi Model Random Forest – Cross Validation

Confusion matrix di atas menggambarkan performa model dalam mengklasifikasikan data uji ke dalam lima kelas yang merepresentasikan kategori deformasi. Secara umum, angka-angka pada diagonal utama dari kiri atas ke kanan bawah menunjukkan jumlah prediksi yang benar untuk masing-masing kelas. Sebagai contoh, model berhasil mengklasifikasikan 94 sampel kelas 1 dengan benar, 298 sampel kelas 2, 234 sampel kelas 3, 231 sampel kelas 4, dan 134 sampel kelas 5. Sementara itu, angka-angka di luar diagonal utama mencerminkan kesalahan prediksi. Dengan mayoritas prediksi berada pada diagonal utama dan kesalahan yang relatif kecil, dapat disimpulkan bahwa model memiliki performa yang sangat baik dalam mengklasifikasikan data. Hal ini juga sejalan dengan hasil evaluasi metrik lainnya, yaitu akurasi sebesar 97,35% dan *f1-score* yang cukup merata di seluruh kelas.

- Evaluasi Model *Multi-Layer Perceptron* (MLP)

Performa model *Multi-Layer Perceptron* (MLP) dievaluasi menggunakan berbagai metrik evaluasi seperti akurasi, *precision*, *recall*, *F1-score*, *classification report*, dan *confusion matrix*. Metrik-metrik ini digunakan untuk menilai seberapa baik model MLP memprediksi label kelas pada data uji (*y_test*) dibandingkan dengan hasil prediksinya (*mlp_y_test_predict*). *Confusion matrix* ditampilkan secara visual untuk memperlihatkan distribusi prediksi terhadap kelas sebenarnya.

```
print('================================================')
print('multi layer perceptron')
print('================================================')
calc_metric(y_test, mlp_y_test_predict,label_mapping)
```

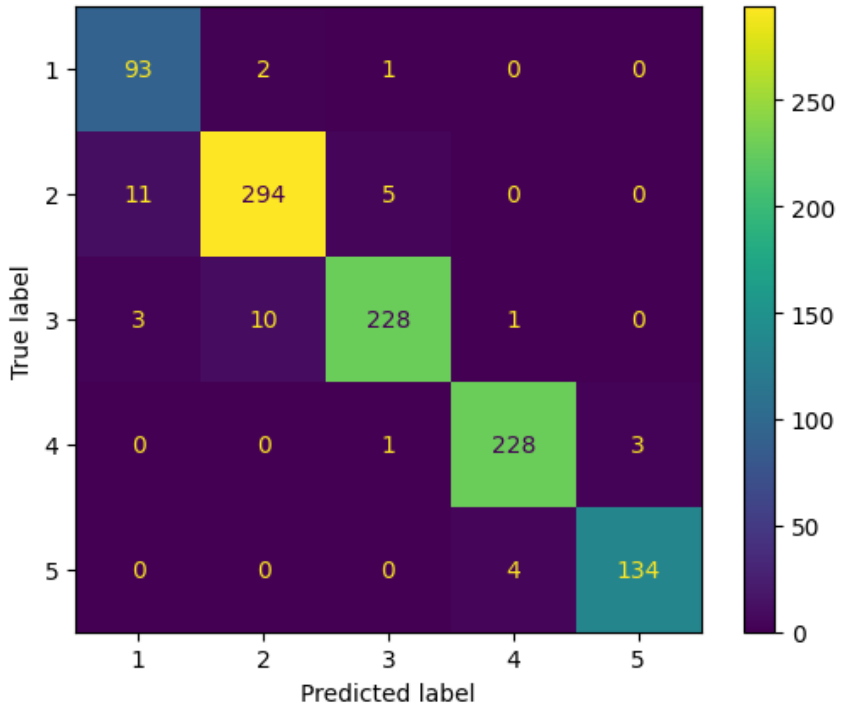
Hasilnya adalah

```
================================================
multi layer perceptron
================================================
Accuracy: 0.9597249508840865
Precision: 0.9607794109727912
Recall: 0.9597249508840865
F1-score: 0.959933736874665
Classification Report:
              precision    recall  f1-score   support

           0       0.87      0.97      0.92        96
           1       0.96      0.95      0.95       310
           2       0.97      0.94      0.96       242
           3       0.98      0.98      0.98       232
           4       0.98      0.97      0.97       138

    accuracy                           0.96      1018
   macro avg       0.95      0.96      0.96      1018
weighted avg       0.96      0.96      0.96      1018
```

Berdasarkan hasil evaluasi model *Multi Layer Perceptron* (MLP), terlihat bahwa model memiliki performa yang sangat baik dalam mengklasifikasikan data uji. Nilai akurasi keseluruhan mencapai 95.97%, yang menunjukkan bahwa sebagian besar prediksi model sesuai dengan label sebenarnya. Selain itu, nilai *precision*, *recall*, dan *f1-score* juga berada pada kisaran tinggi dan seimbang di semua kelas, masing-masing sekitar 95–96%.

Secara rinci, kelas 0 memiliki precision sebesar 0.87 dan recall 0.97, artinya meskipun model cukup akurat dalam mengenali kelas ini, masih ada beberapa prediksi kelas lain yang salah diklasifikasikan sebagai kelas 0. Kelas lainnya menunjukkan kinerja yang lebih stabil dengan nilai *f1-score* mendekati atau bahkan mencapai 0.98, seperti pada kelas 3 dan 4. Hal ini menunjukkan bahwa MLP cukup andal dan konsisten dalam mengenali pola-pola pada seluruh kelas data. Dengan demikian, model ini dapat dikatakan memiliki kinerja klasifikasi yang kuat secara menyeluruh.

- *Confusion matrix*



Gambar 3. 48 Confusion Matrix Evaluasi Model MLP – Cross Validation

Confusion matrix di atas menunjukkan performa model *Multi-Layer Perceptron* (MLP) dalam mengklasifikasikan data uji ke dalam lima kelas deformasi. Nilai-nilai pada diagonal utama (dari kiri atas ke kanan bawah) menunjukkan jumlah prediksi yang benar oleh model. Angka-angka di luar diagonal utama merupakan prediksi yang keliru, seperti misalnya 11 sampel kelas 2 yang diprediksi sebagai kelas 1, atau 10 sampel kelas 3 yang diprediksi sebagai

kelas 2. Meski terdapat beberapa kesalahan prediksi, proporsinya tergolong kecil dibandingkan jumlah prediksi yang benar.

Dengan mayoritas prediksi terkonsentrasi pada diagonal utama dan jumlah kesalahan yang relatif rendah, dapat disimpulkan bahwa model MLP memiliki performa klasifikasi yang sangat baik. Hal ini selaras dengan metrik evaluasi lainnya, yaitu akurasi sebesar 95,9% dan f1-score yang konsisten pada setiap kelas, menunjukkan kestabilan model dalam mengenali pola dari setiap kategori deformasi.

b. Prediksi Raster

Pertama, daftar file raster disiapkan dalam bentuk list, kemudian file raster pertama dibuka menggunakan pustaka rasterio untuk mengambil ukuran (*shape*) sebagai referensi. Selanjutnya, setiap file raster dibaca menggunakan fungsi *read_raster_to_nan*, di mana data raster diambil dari band pertama dan nilai *NoData* diubah menjadi *np.nan*. Jika ukuran raster tidak sesuai dengan referensi, maka akan disesuaikan (dipotong atau dipadukan) agar semua raster memiliki dimensi yang sama. Setelah itu, seluruh raster diratakan (di-*flatten*) menjadi array satu dimensi dan dikumpulkan dalam *list*. Proses stacking dilakukan dengan menyusun array-array tersebut secara kolom menggunakan *np.column_stack*, sehingga membentuk array dua dimensi dengan setiap kolom mewakili satu citra waktu. Hasil akhirnya disimpan dalam variabel *raster_data*.

```
import rasterio
import numpy as np

# Daftar file input raster
files = [
    '/content/drive/MyDrive/Skripsi 2/Input/Jul_Agus_48.tif',
    '/content/drive/MyDrive/Skripsi 2/Input/Agus_Sep_48.tif',
    '/content/drive/MyDrive/Skripsi 2/Input/Sep_Okt_48.tif',
    '/content/drive/MyDrive/Skripsi 2/Input/Okt_Nov_48.tif',
    '/content/drive/MyDrive/Skripsi 2/Input/Nov_Des_48.tif',
    '/content/drive/MyDrive/Skripsi 2/Input/Des_Jan_2025_48.tif'
]
```

```
# Ambil shape referensi dari file pertama
with rasterio.open(files[0]) as ref_src:
    ref_shape = ref_src.read(1).shape

# Fungsi baca raster dan convert no data menjadi np.nan, serta
sesuaikan ke shape referensi
def read_raster_to_nan(file_path):
    with rasterio.open(file_path) as src:
        data = src.read(1)
        no_data_value = src.nodata
        data = np.where(data == no_data_value, np.nan, data)

        # Penyesuaian shape
        padded = np.full(ref_shape, np.nan, dtype=np.float32)
        min_rows = min(ref_shape[0], data.shape[0])
        min_cols = min(ref_shape[1], data.shape[1])
        padded[:min_rows, :min_cols] = data[:min_rows, :min_cols]

        return padded

# Stack semua raster ke samping (kolom per file)
def stack_rasters(file_paths):
    stacked_data = []
    for file_path in file_paths:
        raster_data = read_raster_to_nan(file_path)
        stacked_data.append(raster_data.ravel())
    stacked_array = np.column_stack(stacked_data)
    return stacked_array

# Eksekusi stacking
raster_data = stack_rasters(files)
```

c. Normalize Raster Data

Baris kode *raster_data_normalize = scaller.transform(raster_data)* berfungsi untuk melakukan normalisasi data raster yang telah distack sebelumnya. *Objek scaller* di sini mengacu pada *instance* dari kelas scaler (biasanya *MinMaxScaler* atau *StandardScaler* dari *sklearn.preprocessing*) yang telah dilatih (*fitted*) terlebih dahulu pada data pelatihan. Proses normalisasi ini bertujuan untuk menyelaraskan skala nilai piksel agar berada dalam rentang tertentu, misalnya 0 hingga 1, sehingga model *machine learning* dapat melakukan prediksi secara lebih

stabil dan akurat. Hasil dari transformasi ini adalah *raster_data_normalize*, yaitu array dua dimensi dengan nilai yang telah dinormalisasi dan siap digunakan sebagai input ke dalam model prediktif.

```
raster_data_normalize = scaller.transform(raster_data)
```

- Prediksi Raster

Fungsi *raster_prediction_single_band* digunakan untuk melakukan prediksi pada data raster satu band menggunakan model *machine learning* yang telah dilatih. Fungsi ini membuat *array* kosong berisi *NaN*, lalu mengisi nilai prediksi hanya pada piksel yang valid (tidak *NaN*) menggunakan *model.predict*(). Hasil prediksi kemudian diubah kembali ke bentuk dua dimensi sesuai bentuk raster asli, lalu disimpan ke dalam file *raster output* menggunakan *rasterio*. Fungsi ini dipanggil dua kali untuk menghasilkan peta prediksi deformasi dari dua model berbeda, yaitu *Random Forest* (*rf_pred*) dan *Multi-Layer Perceptron* (*mlp_pred*), yang masing-masing disimpan dalam file .tif terpisah.

```
def raster_prediction(model,output_file,raster_stack,meta_data):
    # prediksi dengan mengabaikan raster yang nan
    mask = np.isnan(raster_stack).any(axis=1)
    raster_prediction = np.full(raster_stack.shape[0], np.nan)
    raster_prediction[~mask] = model.predict(raster_stack[~mask])

    # konversi ke bentuk 2d lagi
    raster_prediction =
raster_prediction.reshape(meta_data['height'],meta_data['width'])

    # save raster hasil prediksi
    with rasterio.open(output_file, 'w', **meta_data) as dst:
        dst.write(raster_prediction.astype('float32'), 1)

    return raster_prediction
```

```
# mendapatkan meta data refrensi
with rasterio.open(files[0]) as src:
    meta_data = src.meta.copy()
    meta_data.update({
    'dtype' : 'float32',
    'driver' : 'GTiff',
    "count": 1})

rf_raster_prediction =
raster_prediction(rf,'prediction_rf_initial.tif',raster_data_norma
lize,meta_data)
mlp_raster_prediction =
raster_prediction(mlp,'prediction_mlp_initial.tif',raster_data_nor
malize,meta_data)
```

Load Data

```
import numpy as np
from pickle import load

X_train = np.load("/content/drive/MyDrive/Skripsi
2/train_data/X_train_resampled_normalized.npy")
y_train = np.load("/content/drive/MyDrive/Skripsi
2/train_data/Y_train_resampled_normalized.npy")
X_test = np.load("/content/drive/MyDrive/Skripsi
2/train_data/X_test_resampled_normalized.npy")
y_test = np.load("/content/drive/MyDrive/Skripsi
2/train_data/Y_test_resampled_normalized.npy")

with open("/content/drive/MyDrive/Skripsi
2/train_data/label_mapping.pkl", "rb") as f:
    label_mapping = load(f)
with open("/content/drive/MyDrive/Skripsi
2/train_data/scaller.pkl", "rb") as f:
    scaller = load(f)
```

- *Evaluation Metric*

Performa model klasifikasi dievaluasi dengan menghitung dan menampilkan berbagai metrik evaluasi. Fungsi *calc_metric* digunakan untuk mengevaluasi hasil prediksi model dengan menampilkan metrik akurasi, presisi, *recall*, dan *F1-score* secara menyeluruh. Selain itu, fungsi ini juga mencetak laporan klasifikasi per kelas dan menampilkan *confusion matrix* menggunakan label yang telah dipetakan, sehingga performa model dapat dianalisis secara detail dan visual.

```
from sklearn.metrics import (
accuracy_score, precision_score, recall_score, f1_score,
confusion_matrix,
classification_report, ConfusionMatrixDisplay
)

def calc_metric(y_data,y_predict,label_mapping):
    # Accuracy
    print(f"Accuracy: {accuracy_score(y_data,y_predict)}")

    # Precision, Recall, F1-score
    print(f"Precision: {precision_score(y_data,y_predict,
average='weighted')}")
    print(f"Recall: {recall_score(y_data,y_predict,
average='weighted')}")
    print(f"F1-score: {f1_score(y_data,y_predict,
average='weighted')}")

    # Classification Report
    print("Classification Report:")
    print(classification_report(y_data,y_predict))

    # Confusion Matrix
    print("Confusion Matrix:")
    # print(confusion_matrix(y_data,y_predict))
    labels = list(label_mapping.keys())
    display_labels = list(label_mapping.values())
    cm = confusion_matrix(y_data, y_predict, labels=labels)
    disp =
ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=disp
lay_labels)
    disp.plot()
```

d. *GridSearchCV*

Selanjutnya adalah menjalankan *Grid Search* untuk mencari kombinasi *hyperparameter* terbaik dari suatu model *machine learning* menggunakan *GridSearchCV*. Proses ini mengevaluasi performa model berdasarkan data pelatihan menggunakan *cross-validation*, lalu memilih model terbaik untuk diuji pada data uji. Setelah itu, fungsi menampilkan metrik evaluasi (akurat, presisi, *recall*, *F1-score*, dan *confusion matrix*) dan, jika terdapat dua parameter, menampilkan visualisasi hasil *grid search* dalam bentuk *scatter plot* untuk membantu interpretasi performa tiap kombinasi *hyperparameter*.

```
from sklearn.model_selection import GridSearchCV
import matplotlib.pyplot as plt

def plot_2d_grid_search(grid_search,param_grid):
    results = grid_search.cv_results_
    print('grid_search.cv_results = ',results)
    keys_list = list(param_grid.keys())
    # Extract the results into arrays
    first_param =
np.array(results[f"param_{keys_list[0]}"].data, dtype=float)
    second_param =
np.array(results[f"param_{keys_list[1]}"].data, dtype=float)
    mean_test_score = results['mean_test_score']

    # Create a scatter plot
    plt.figure(figsize=(10, 6))
    sc = plt.scatter(first_param, second_param,
c=mean_test_score, cmap='nipy_spectral')
    plt.colorbar(sc, label='Mean Test Score')
    plt.xlabel('Max Depth')
    plt.ylabel('Number of Estimators')
    plt.title('Grid Search Scores')
    plt.show()

def grid_search_model(model,param_grid,X_train,
y_train,X_test,y_test,label_mapping,cv=4):
    # Initiate the grid search model
    grid_search = GridSearchCV(estimator=model,
param_grid=param_grid,
                               cv=cv, n_jobs=-1, verbose=2)
```

```
    # Fit the grid search
    grid_search.fit(X_train, y_train)
    print(f"Best Hyperparameters: {grid_search.best_params_}")

    # The best estimator is already trained on the entire
training data, so there's no need to fit it again.
    # check refit parameter: https://scikit-
learn.org/stable/modules/generated/sklearn.model_selection.Gri
dSearchCV.html
    best_model = grid_search.best_estimator_

    # Evaluate the best model on the test data
    y_pred = best_model.predict(X_test)
    calc_metric(y_test,y_pred,label_mapping)

    if len(param_grid) == 2:
        plot_2d_grid_search(grid_search,param_grid)

    return best_model
```

- *GridSearch* untuk *Random Forest*

Untuk memperoleh performa model yang optimal, dilakukan proses *hyperparameter tuning* pada algoritma *Random Forest Classifier* menggunakan metode *Grid Search*. *Hyperparameter* yang diuji meliputi jumlah pohon keputusan (*n_estimators*) sebanyak 1000 pohon, kedalaman maksimum pohon (*max_depth*) sebesar 50, jumlah minimum sampel untuk melakukan pemisahan *node internal* (*min_samples_split*) yaitu 2, 4, dan 8, serta jumlah minimum sampel pada *leaf node* (*min_samples_leaf*) yaitu 1, 2, dan 4. Selain itu, digunakan dua pendekatan dalam pemilihan fitur terbaik pada tiap node, yaitu sqrt dan log2 (*max_features*). Nilai parameter *n_estimators* dan *max_depth* diambil dari hasil tuning awal yang menunjukkan performa terbaik, sedangkan parameter lainnya disesuaikan untuk pengujian lanjutan.

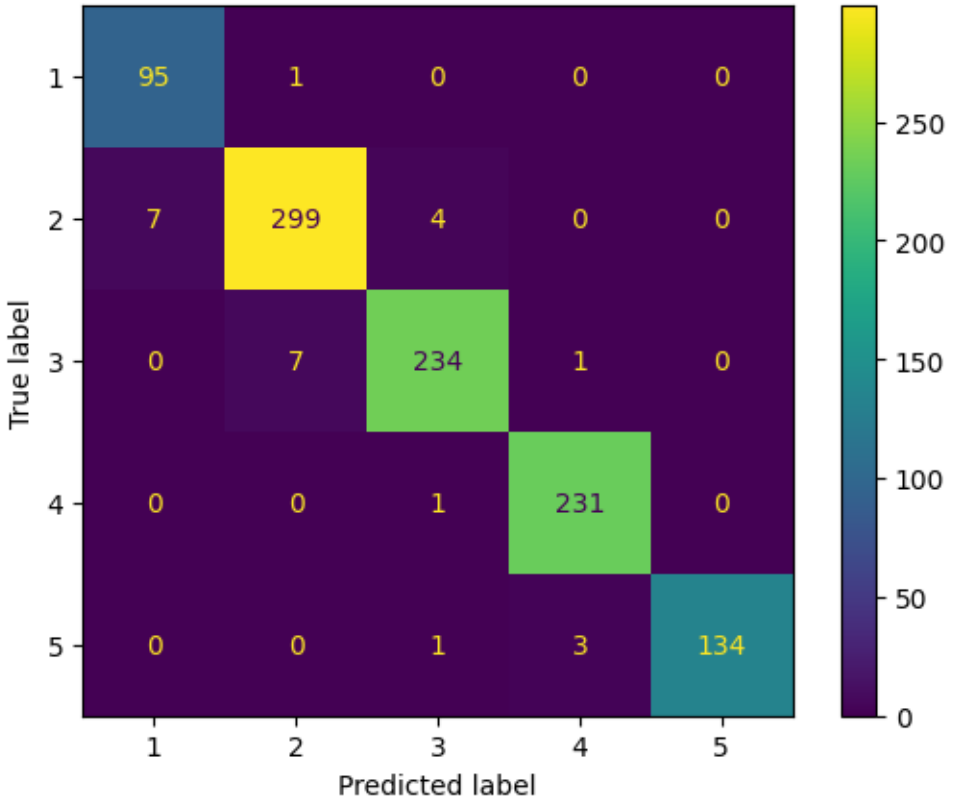
Proses pencarian kombinasi parameter terbaik dilakukan menggunakan fungsi *grid_search_model*, yang mengevaluasi performa model berdasarkan akurasi dan metrik evaluasi lainnya pada data pelatihan dan pengujian. Model dengan kombinasi parameter terbaik selanjutnya digunakan untuk klasifikasi data deformasi permukaan.

```
from sklearn.ensemble import RandomForestClassifier

# Define the parameter grid
# param_grid = {
#     'n_estimators': [100, 500, 1000, 2000],
#     'max_depth': [10, 50, 100, 200],
# }

param_grid = {
    'n_estimators': [1000], # from best param
    'max_depth': [50], # from best param
    'min_samples_split': [2, 4, 8],
    'min_samples_leaf': [1, 2, 4],
    'max_features':['sqrt', 'log2']
}

# Create a base model
rf = RandomForestClassifier(random_state=42)

best_rf_model = grid_search_model(rf,param_grid,X_train,
y_train,X_test,y_test,label_mapping)
```

Hasilnya:

```
Fitting 4 folds for each of 18 candidates, totalling 72 fits
Best Hyperparameters: {'max_depth': 50, 'max_features': 'sqrt', 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 1000}
Accuracy: 0.9754420432220039
Precision: 0.9757707786246603
Recall: 0.9754420432220039
F1-score: 0.9754740898060434
Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.99      0.96        96
           1       0.97      0.96      0.97       310
           2       0.97      0.97      0.97       242
           3       0.98      1.00      0.99       232
           4       1.00      0.97      0.99       138

    accuracy                           0.98      1018
   macro avg       0.97      0.98      0.97      1018
weighted avg       0.98      0.98      0.98      1018
```

Hasil evaluasi performa model *Random Forest* menunjukkan performa klasifikasi yang sangat baik. Model berhasil mencapai akurasi sebesar 97,54%, dengan nilai *precision* sebesar 97,58%, *recall* sebesar 97,54%, dan *f1-score* sebesar 97,55%. Nilai-nilai ini mengindikasikan bahwa model memiliki kemampuan yang seimbang dalam mendeteksi dan mengklasifikasikan masing-masing kelas. Berdasarkan *classification report*, kelas dengan label 3 dan 4 memiliki nilai *f1-score* tertinggi yaitu 0,99, menandakan bahwa model mampu mengidentifikasi kedua kelas ini dengan sangat baik. Seluruh metrik rata-rata seperti *macro average* dan *weighted average* juga bernilai tinggi (0,97–0,98), memperkuat keandalan model dalam menghadapi distribusi data yang tidak seimbang. Parameter terbaik yang diperoleh dari proses *tuning* menggunakan *GridSearchCV* yaitu *max_depth*=50, *max_features*='sqrt', *min_samples_leaf*=1, *min_samples_split*=2, dan *n_estimators*=1000. Hasil ini menegaskan bahwa kombinasi parameter tersebut menghasilkan model dengan generalisasi yang optimal terhadap data latih.



Gambar 3. 49 Confusion Matrix GridSearch untuk Random Forest

Gambar *confusion matrix* di atas memberikan visualisasi yang memperkuat hasil evaluasi performa model *Random Forest*. Mayoritas nilai prediksi berada pada diagonal utama, yang menunjukkan bahwa model mampu memprediksi label kelas dengan sangat akurat dan jarang terjadi kesalahan klasifikasi. Distribusi error yang kecil dan terkonsentrasi di kelas-kelas yang berdekatan secara spasial atau karakteristik menunjukkan bahwa model tidak hanya akurat, tetapi juga mampu menghindari kesalahan klasifikasi besar. Hal ini selaras dengan nilai *F1-Score* dan akurasi tinggi yang telah dijelaskan sebelumnya, menandakan bahwa model *Random Forest* sangat andal untuk digunakan dalam klasifikasi data ini.

- *GridSearch* untuk *Multi Layer Perceptron* (MLP)

Pertama, ditentukan *parameter grid* (*param_grid*) yang berisi kombinasi parameter yang akan diuji, seperti arsitektur jaringan (*hidden_layer_sizes*) dengan satu atau dua lapisan tersembunyi, fungsi aktivasi (*activation*) seperti *tanh* dan *relu*, jenis *learning rate* (*constant* atau *adaptive*), jumlah maksimum iterasi sebanyak 500, serta nilai awal *learning rate* (*learning_rate_init*) yang bervariasi antara 0.0001 hingga 0.01. Kemudian, objek *MLPClassifier* dibuat dengan random_state untuk memastikan hasil yang konsisten. Selanjutnya, fungsi *grid_search_model* dipanggil untuk melakukan pencarian kombinasi parameter terbaik berdasarkan *cross-validation* menggunakan data pelatihan (*X_train*, *y_train*) dan mengujinya pada data pengujian (*X_test, y_test*). Fungsi ini juga menerima label_mapping untuk keperluan visualisasi atau evaluasi model. Output dari proses ini adalah *best_mlp_model*, yaitu model MLP dengan kombinasi parameter terbaik.

```
from sklearn.neural_network import MLPClassifier

# Define the parameter grid

param_grid = {
'hidden_layer_sizes': [(100,), (100, 50)],
'activation': ['tanh', 'relu'],
'learning_rate': ['constant', 'adaptive' ],
'max_iter': [500],
'learning_rate_init': [0.0001, 0.001, 0.01]
}
```

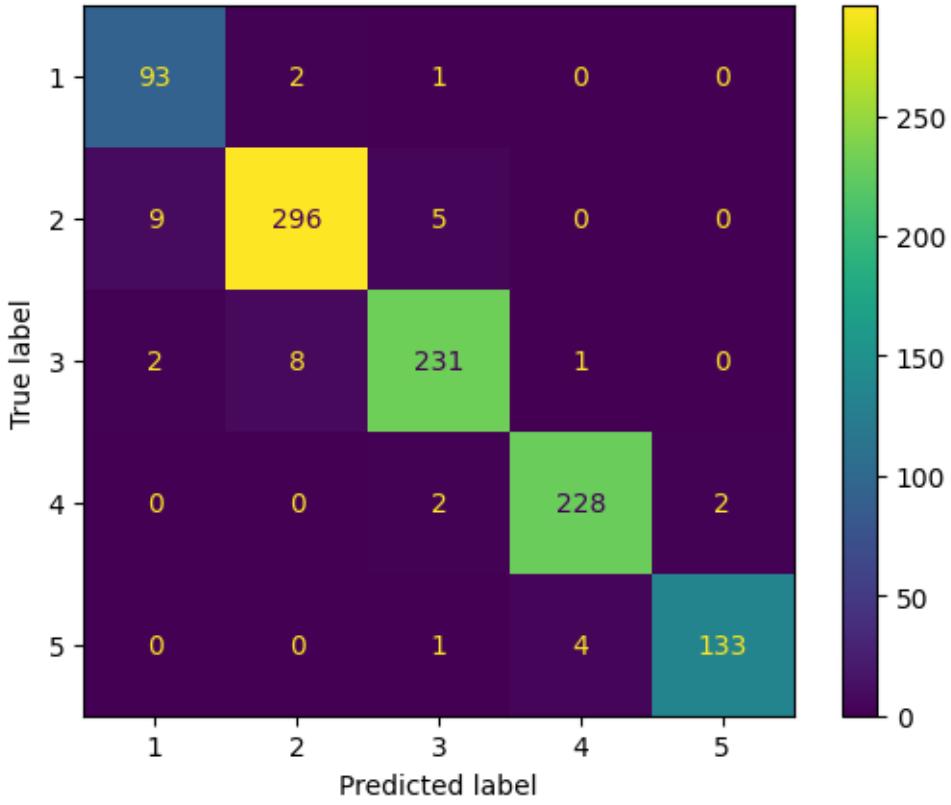
```
mlp = MLPClassifier(random_state=42)

best_mlp_model = grid_search_model(mlp,param_grid,X_train,
y_train,X_test,y_test,label_mapping)
```

Hasilnya:

```
Fitting 4 folds for each of 24 candidates, totalling 96 fits
Best Hyperparameters: {'activation': 'tanh', 'hidden_layer_sizes': (100, 50), 'learning_rate': 'constant', 'learning_rate_init': 0.001, 'max_iter': 500}
Accuracy: 0.9636542239685658
Precision: 0.9642607560657454
Recall: 0.9636542239685658
F1-score: 0.963783693543259
Classification Report:
              precision    recall  f1-score   support

           0       0.89      0.97      0.93        96
           1       0.97      0.95      0.96       310
           2       0.96      0.95      0.96       242
           3       0.98      0.98      0.98       232
           4       0.99      0.96      0.97       138

    accuracy                           0.96      1018
   macro avg       0.96      0.96      0.96      1018
weighted avg       0.96      0.96      0.96      1018
```

Berdasarkan hasil pelatihan model menggunakan *Multi-Layer Perceptron* (MLP), diperoleh parameter terbaik yaitu fungsi aktivasi *'tanh'*, arsitektur jaringan tersembunyi (100, 50), serta nilai *learning rate* sebesar 0.001 dengan metode pembelajaran *constant* dan jumlah iterasi maksimum sebanyak 500. Model ini menghasilkan nilai akurasi sebesar 96.36%, precision 96.43%, *recall* 96.36%, dan *F1-score* 96.38%, yang menunjukkan performa klasifikasi yang sangat baik secara keseluruhan.

Lebih lanjut, performa model terhadap masing-masing kelas menunjukkan hasil yang kuat. Kelas 0 memperoleh *precision* 0.89 dan *recall* 0.97 dengan *F1-score* 0.93, sementara kelas 1 hingga kelas 4 masing-masing memiliki *F1-score* di atas 0.96. Kelas 4 bahkan mencapai *precision* 0.99. Nilai rata-rata keseluruhan, baik makro maupun tertimbang, adalah 0.96 untuk *precision*, *recall*, dan *F1-score*. Hal ini menunjukkan bahwa model mampu mengklasifikasikan data secara seimbang dan akurat di seluruh kategori kelas dengan jumlah total sampel sebanyak 1.018 data.



Gambar 3. 50 Confusion Matrix GridSearch untuk MLP

Berdasarkan *confusion matrix* di atas, dapat disimpulkan bahwa model *Multi-Layer Perceptron* (MLP) memiliki kinerja yang sangat baik dalam mengklasifikasikan setiap kelas. Diagonal *confusion matrix* menunjukkan prediksi yang benar untuk masing-masing kelas, dan nilainya cukup tinggi, mengindikasikan bahwa mayoritas data berhasil diklasifikasikan dengan tepat. Dominasi angka yang tinggi pada diagonal menunjukkan bahwa model mampu mengenali pola pada tiap kelas dengan akurasi yang tinggi. Jumlah kesalahan antar kelas relatif kecil dan tidak menunjukkan bias yang signifikan terhadap kelas tertentu. Hal ini sejalan dengan metrik evaluasi yang sebelumnya telah dijelaskan, di mana nilai *precision*, *recall*, dan *F1-score* berada pada kisaran sangat baik (di atas 0.93 untuk semua kelas). Dengan demikian, *confusion matrix* ini memperkuat bukti bahwa model MLP sangat andal untuk tugas klasifikasi pada dataset ini.

Simpan model *Random Forest* dan MLP terbaik yang telah dilatih ke dalam file .pkl menggunakan modul *pickle*, agar dapat digunakan kembali di lain waktu tanpa perlu dilatih ulang.

```
import pickle
# Save the model
with open('/content/drive/MyDrive/Skripsi
2/Pickle/best_rf_model.pkl', 'wb') as file:
    pickle.dump(best_rf_model, file)

with open('/content/drive/MyDrive/Skripsi
2/Pickle/best_mlp_model.pkl', 'wb') as file:
    pickle.dump(best_mlp_model, file)
```

e. Visualisasi Hasil

Setelah proses *hyperparameter tuning* selesai dilakukan untuk masing-masing model machine learning, langkah selanjutnya adalah menggunakan model tersebut untuk memprediksi seluruh data raster yang tersedia. Untuk itu, diperlukan pembacaan ulang terhadap semua data raster yang sebelumnya telah digunakan dalam proses pelatihan model. Proses ini melibatkan penggunaan modul *rasterio* dan *numpy* untuk membaca file raster GeoTIFF dan menggabungkannya menjadi sebuah *array* dua dimensi.

Pertama-tama, sistem mengambil *shape* referensi dari file raster pertama dalam daftar sebagai acuan. Selanjutnya, fungsi *read_raster_to_nan* digunakan untuk membaca setiap file raster, mengonversi nilai *no data* menjadi *NaN*, serta menyesuaikan ukuran data dengan *shape* referensi. Jika ukuran data raster lebih kecil dari *shape* referensi, maka akan dilakukan padding menggunakan nilai *NaN* agar semua raster memiliki dimensi yang konsisten.

Setelah itu, fungsi stack_rasters akan memanggil *read_raster_to_nan* untuk setiap file dalam daftar. Data raster kemudian diratakan (*flattened*) menjadi satu dimensi dan ditumpuk secara horizontal menggunakan *np.column_stack*. Proses ini menghasilkan sebuah *array* dua dimensi yang merepresentasikan seluruh nilai raster dari semua layer dalam format yang sesuai untuk input model *machine learning*, yaitu satu baris per piksel dan satu kolom per layer raster.

```
# Ambil shape referensi dari file pertama
with rasterio.open(files[0]) as ref_src:
    ref_shape = ref_src.read(1).shape

# Fungsi baca raster dan convert no data menjadi np.nan, serta
sesuaikan ke shape referensi
def read_raster_to_nan(file_path):
    with rasterio.open(file_path) as src:
        data = src.read(1)
        no_data_value = src.nodata
        data = np.where(data == no_data_value, np.nan, data)

        # Penyesuaian shape
        padded = np.full(ref_shape, np.nan, dtype=np.float32)
        min_rows = min(ref_shape[0], data.shape[0])
        min_cols = min(ref_shape[1], data.shape[1])
        padded[:min_rows, :min_cols] = data[:min_rows,
:min_cols]

        return padded

# Stack semua raster ke samping (kolom per file)
def stack_rasters(file_paths):
    stacked_data = []
    for file_path in file_paths:
        raster_data = read_raster_to_nan(file_path)
        stacked_data.append(raster_data.ravel())
    stacked_array = np.column_stack(stacked_data)
    return stacked_array

# Eksekusi stacking
raster_data = stack_rasters(files)

# Normalize Raster Data
raster_data_normalize = scaller.transform(raster_data)
```

Data raster yang telah di-*stacking* kemudian dinormalisasi menggunakan *scaller.transform*, di mana nilai *scaller* harus sama persis dengan nilai yang digunakan pada saat pelatihan model (*training*). Selanjutnya, dibuat sebuah fungsi bernama *raster_prediction* yang bertugas untuk melakukan prediksi terhadap data raster. Fungsi ini menerima empat parameter utama: model yang digunakan (misalnya *best_rf_model* atau *best_mlp_model*), nama file output hasil prediksi, data raster yang sudah di-stack, dan metadata raster.

Di dalam fungsi ini, data raster yang memiliki nilai *NaN* akan diabaikan dengan menggunakan masking. Kemudian, prediksi dilakukan hanya pada piksel yang tidak memiliki nilai *NaN*. Hasil prediksi yang masih dalam bentuk array 1 dimensi kemudian dikonversi kembali ke dalam bentuk 2 dimensi sesuai dengan ukuran tinggi dan lebar dari metadata raster. Metadata ini diperoleh dari file raster referensi (misalnya *files*[0]) dan diperbarui agar sesuai, termasuk memastikan tipe data (*dtype*) dalam format *float32*, jumlah band (*count*) adalah 1, dan format penyimpanan adalah GTiff.

Akhirnya, hasil prediksi disimpan dalam file raster baru melalui proses penulisan menggunakan rasterio. Fungsi raster_prediction ini dijalankan secara terpisah untuk masing-masing model terbaik, yaitu *best_rf_model* dan *best_mlp_model*, sehingga dihasilkan dua file raster hasil prediksi yang terpisah.

```
def
raster_prediction(model,output_file,raster_stack,meta_data):
    # prediksi dengan mengabaikan raster yang nan
    mask = np.isnan(raster_stack).any(axis=1)
    raster_prediction = np.full(raster_stack.shape[0], np.nan)
    raster_prediction[~mask] =
model.predict(raster_stack[~mask])

    # konversi ke bentuk 2d lagi
    raster_prediction =
raster_prediction.reshape(meta_data['height'],meta_data['width
'])
```

```
# save raster hasil prediksi
    with rasterio.open(output_file, 'w', **meta_data) as dst:
        dst.write(raster_prediction.astype('float32'), 1)

    return raster_prediction

# mendapatkan meta data refrensi
with rasterio.open(files[0]) as src:
    meta_data = src.meta.copy()
    meta_data.update({
    'dtype' : 'float32',
    'driver' : 'GTiff',
    "count": 1})

rf_raster_prediction =
raster_prediction(best_rf_model,'prediction_rf_best.tif',raste
r_data_normalize,meta_data)
mlp_raster_prediction =
raster_prediction(best_mlp_model,'prediction_mlp_best.tif',ras
ter_data_normalize,meta_data)
```

Selanjutnya membuat diagram batang (*bar chart*) yang menggambarkan perbandingan luas kategori deformasi permukaan hasil klasifikasi dari dua model *machine learning*, yaitu *Random Forest* (RF) dan *Multi Layer Perceptron* (MLP). Luas deformasi diukur dalam satuan hektar (ha) dan diklasifikasikan ke dalam lima kategori numerik (1 hingga 5), yang masing-masing memiliki nilai luasan tertentu untuk tiap model.

Pertama, data luasan dari masing-masing kategori untuk model *Random Forest* dan MLP disimpan dalam bentuk *dictionary* (*area_rf* dan *area_mlp*). Kemudian, dibuat *label_mapping* untuk menghubungkan nomor kategori dengan label pada sumbu x. Fungsi *plot_bar_chart()* didefinisikan untuk membuat diagram batang. Fungsi ini menerima lima parameter: data luas, label kategori, nama model, nama file hasil ekspor, dan warna batang.

```
# Mapping label kategori ke keterangan
label_mapping = {
    1: '1',
    2: '2',
    3: '3',
    4: '4',
    5: '5'
}

def plot_bar_chart(data, label_mapping, model_name, filename,
color):
    labels = [label_mapping[k] for k in data.keys()]
    values = list(data.values())

    plt.figure(figsize=(10, 6))
    bars = plt.bar(labels, values, color=color,
edgecolor='black')

    plt.title(f'Luas Kategori Deformasi - {model_name}')
    plt.xlabel('Kategori Deformasi')
    plt.ylabel('Luas (ha)')
    plt.xticks(rotation=45)
    plt.tight_layout()

    for bar in bars:
        yval = bar.get_height()
        plt.text(bar.get_x() + bar.get_width()/2.0, yval +
100, f'{yval:.0f}', ha='center', va='bottom')

    plt.savefig(filename, dpi=300)
    plt.show()

# Warna soft: merah muda dan biru muda
soft_red = '#F1948A'   # soft red
soft_blue = '#85C1E9'  # soft blue

# Plot untuk Random Forest
plot_bar_chart(area_rf, label_mapping, "Random Forest",
"bar_chart_rf.png", soft_red)
```

```
# Plot untuk MLP
plot_bar_chart(area_mlp, label_mapping, "Multi Layer
Perceptron", "bar_chart_mlp.png", soft_blue)
```

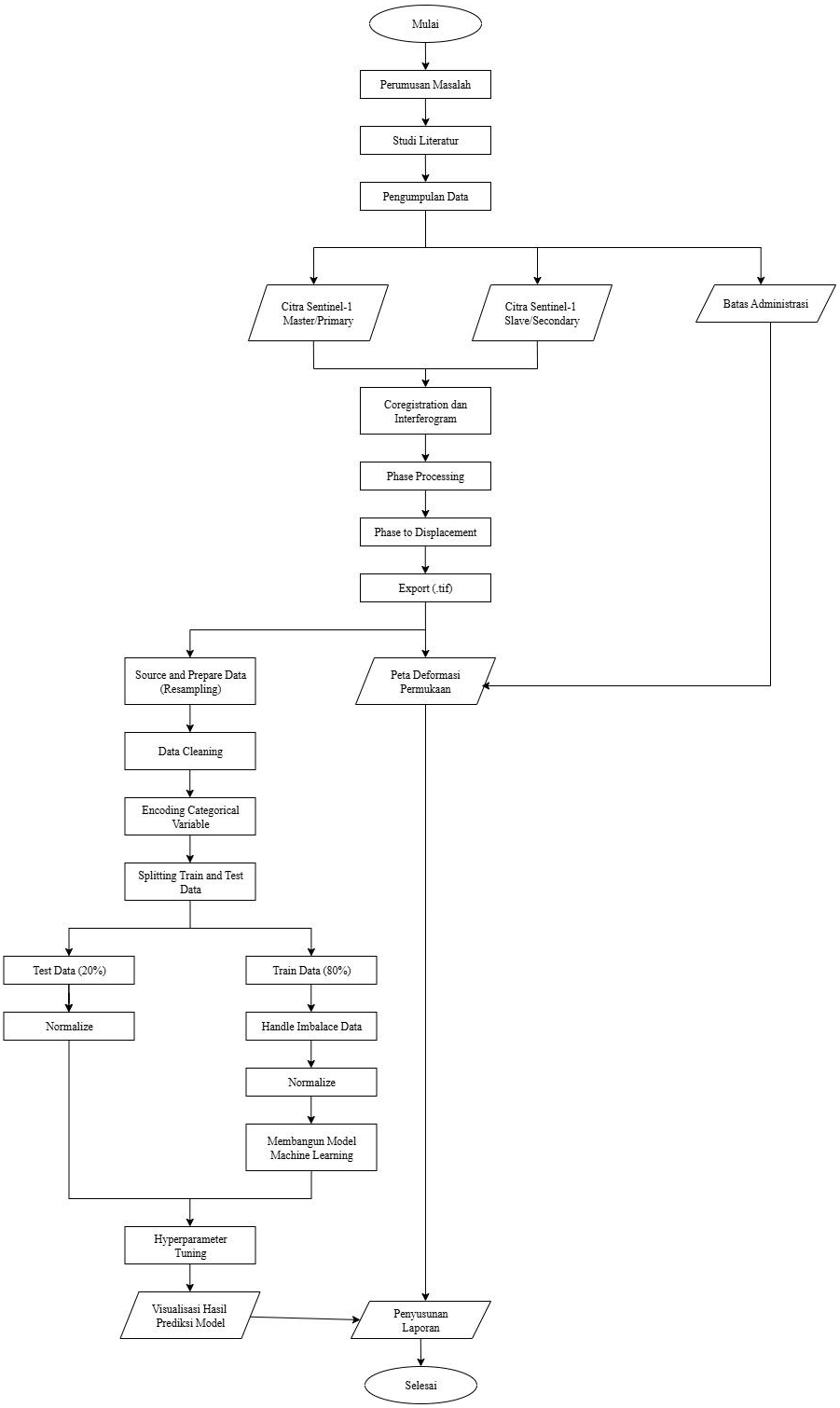
Selanjutnya adalah membuat perbandingan hasil prediksi model *Random Forest* (RF) dan *Multi-Layer Perceptron* (MLP) dalam bentuk visualisasi peta kelas deformasi permukaan. Proses dimulai dengan mendefinisikan fungsi *read_raster* untuk membaca file raster (.tif) dan mengubah nilai *NoData* menjadi *NaN* agar tidak ikut divisualisasikan. Kemudian, dua file raster hasil prediksi, yaitu *prediction_rf_best.tif* dan *prediction_mlp_best.tif,* dibaca menggunakan fungsi tersebut. Visualisasi dilakukan dalam format dua panel berdampingan (*side-by-side*), di mana peta sebelah kiri menunjukkan hasil dari *Random Forest* dan sebelah kanan dari MLP. Keduanya menggunakan skema warna viridis dengan rentang nilai kelas dari 1 hingga 5, serta dilengkapi *colorbar* yang memberikan label "Kelas Deformasi".

```
import rasterio
from rasterio.plot import plotting_extent
import matplotlib.pyplot as plt
import numpy as np

# Fungsi untuk baca raster
def read_raster(path):
    with rasterio.open(path) as src:
        data = src.read(1)
        nodata = src.nodata
        if nodata is not None:
            data = np.where(data == nodata, np.nan, data)
        extent = plotting_extent(src)
    return data, extent

# Baca raster
rf_data, rf_extent = read_raster('prediction_rf_best.tif')
mlp_data, mlp_extent = read_raster('prediction_mlp_best.tif')

# Plot side-by-side
fig, axes = plt.subplots(1, 2, figsize=(14, 7))
```

```
# Plot RF
im1 = axes[0].imshow(rf_data, cmap='viridis', vmin=1, vmax=5,
extent=rf_extent)
axes[0].set_title("Random Forest Prediction")
axes[0].axis('off')
cbar1 = fig.colorbar(im1, ax=axes[0], ticks=[1, 2, 3, 4, 5],
shrink=0.7)
cbar1.set_label('Kelas Deformasi')

# Plot MLP
im2 = axes[1].imshow(mlp_data, cmap='viridis', vmin=1, vmax=5,
extent=mlp_extent)
axes[1].set_title("MLP Prediction")
axes[1].axis('off')
cbar2 = fig.colorbar(im2, ax=axes[1], ticks=[1, 2, 3, 4, 5],
shrink=0.7)
cbar2.set_label('Kelas Deformasi')

plt.tight_layout()
plt.show()
```

3.9 Diagram Alir

Untuk lebih memahami rangkaian alur penelitian ini, maka dibuat diagram alir penelitian sebagai berikut.



Gambar 3. 51 Diagram Alir