

Segmentasi Semantik Menggunakan Model DeepLabv3+ untuk Citra Satelit pada Area Tutupan Lahan dan Objek Gunung Api



Disusun Oleh:

N a m a : Rasyad Alief Mursalin
NIM : 20523022

**PROGRAM STUDI INFORMATIKA – PROGRAM SARJANA
FAKULTAS TEKNOLOGI INDUSTRI
UNIVERSITAS ISLAM INDONESIA
2025**

HALAMAN PENGESAHAN DOSEN PEMBIMBING

**Segmentasi Semantik Menggunakan Model DeepLabv3+ untuk
Citra Satelit pada Area Tutupan Lahan dan Objek Gunung Api**

TUGAS AKHIR



HALAMAN PENGESAHAN DOSEN PENGUJI

Segmentasi Semantik Menggunakan Model DeepLabv3+ untuk Citra Satelit pada Area Tutupan Lahan dan Objek Gunung Api

TUGAS AKHIR

Telah dipertahankan di depan sidang penguji sebagai salah satu syarat untuk memperoleh gelar Sarjana Komputer dari Program Studi Informatika – Program Sarjana di Fakultas Teknologi Industri Universitas Islam Indonesia
Yogyakarta, 13 Januari 2025

Tim Penguji

Arrie Kurniawardhani, S.Si., M.Kom

Anggota 1

Chandra Kusuma Dewa, S.Kom., M.Cs.,
Ph.D.

Anggota 2

Mukhammad Andri Setiawan, S.T., M.Sc.,
Ph.D.

Mengetahui,

Ketua Program Studi Informatika – Program Sarjana

Fakultas Teknologi Industri

Universitas Islam Indonesia



(Dhomas Hatta Fudholi, S.T., M.Eng., Ph.D.)

HALAMAN PERNYATAAN KEASLIAN TUGAS AKHIR

Yang bertanda tangan di bawah ini:

Nama : Rasyad Alief Mursalin

NIM : 20523022

Tugas akhir dengan judul:

**Segmentasi Semantik Menggunakan Model DeepLabv3+ untuk
Citra Satelit pada Area Tutupan Lahan dan Objek Gunung Api**

Menyatakan bahwa seluruh komponen dan isi dalam tugas akhir ini adalah hasil karya saya sendiri. Apabila di kemudian hari terbukti ada beberapa bagian dari karya ini adalah bukan hasil karya sendiri, tugas akhir yang diajukan sebagai hasil karya sendiri ini siap ditarik kembali dan siap menanggung risiko dan konsekuensi apapun.

Demikian surat pernyataan ini dibuat, semoga dapat dipergunakan sebagaimana mestinya.

Yogyakarta, 13 Januari 2025



(Rasyad Alief Mursalin)

HALAMAN PERSEMBAHAN

Penelitian ini saya dedikasikan untuk kedua orang tua saya yang telah memberikan kesempatan untuk bisa selalu meraih cita dan harapan setinggi langit. Khususnya ibu saya yang senantiasa mendukung segala proses dari setiap tahapan perjalanan hidup.

HALAMAN MOTO

"When everything seems to be going against you, remember that the airplane takes off against the wind, not with it."

(Henry Ford)

KATA PENGANTAR

Puji dan syukur penulis ucapkan kehadiran Allah SWT yang mana atas segala nikmat dan karuniaNya penulis diberikan kesehatan dan kelancaran dalam proses penyelesaian tugas akhir ini. Shalawat serta salam semoga selalu tercurah limpahkan kehadiran Nabi besar Muhammad SAW.

Pada kesempatan ini penulis telah menyelesaikan Tugas Akhir yang mana menjadi salah satu syarat untuk mendapatkan gelar Sarjana Komputer di Universitas Islam Indonesia. Tugas akhir yang berjudul “Segmentasi Semantik Menggunakan Model DeepLabv3+ untuk Citra Satelit pada Area Tutupan Lahan dan Objek Gunung Api” dapat terselesaikan melalui beberapa proses yang panjang, juga berkat doa dan dukungan dari beberapa pihak yang ikut serta membantu dalam penyelesaian tugas akhir ini. Oleh karena itu penulis ingin mengucapkan banyak terimakasih khususnya kepada :

1. Orang tua penulis, Bapak Ir. Emud dan Ibu Elin Roslina yang selalu memberikan dukungan materi dan mendoakan keberhasilan penulis, semoga selalu diberikan rahmat dan keberkahan oleh Allah SWT.
2. Keluarga besar Bani Ali dan Bapak Effendi yang senantiasa memberikan dukungan dan doa terhadap penulis. Semoga selalu diberikan Rahmat dari Allah SWT.
3. Ibu Arrie Kurniawardhani, S.Si., M.Kom. selaku dosen pembimbing yang senantiasa meluangkan waktunya untuk membimbing penulis dalam pengerjaan tugas akhir dari awal sampai selesai.
4. Kedua adik penulis Rahmi Ahlus Syaiunilhaq dan Rayinura Khoirotun Hisan semoga selalu diberikan keberkahan dalam meraih cita-cita kalian.
5. Sahabat penulis Danu Muhammad Afarhan, Tisar Ibnu Mansyar, Deni Jaenudin, Encep Anwar Fauzi, Akmal Zulham Nurkamal, Rizky Tazkyatun Nufus. Terimakasih selalu memberikan support dan rela membantu dalam rangkaian tugas akhir ini, semoga dikemudian hari kita berjumpa lagi dengan pencapaian masing-masing yang luar biasa.
6. Rekan-rekan KKN yang ikut andil dalam memberikan dukungan pada proses penulisan tugas akhir ini dan tidak bisa penulis sebutkan satu persatu, terimakasih semoga Allah SWT membalas kebaikan kalian semuanya.
7. Diri saya sendiri yang tidak pantang menyerah dalam menjalani tantangan dari setiap proses kehidupan.

Penulis akui banyak kekurangan dalam penulisan tugas akhir ini, namun tugas akhir ini dibuat dengan kesungguhan dan perjuangan yang tidak dapat dinilai oleh apapun. Besar harapan penulis pada Tugas Akhir ini dapat bermanfaat bagi orang yang membacanya.

Yogyakarta, 13 Januari 2025

A handwritten signature in black ink, consisting of stylized, flowing letters that appear to be 'R' and 'A' followed by a horizontal line and a small flourish.

(Rasyad Alief Mursalin)

SARI

Penelitian ini berfokus pada penerapan model *DeepLabV3+* untuk segmentasi citra satelit pada area tutupan lahan dan objek sekitar gunung api menggunakan metode segmentasi semantik. Latar belakang penelitian ini didasari oleh kebutuhan pemantauan lingkungan dan mitigasi bencana di Indonesia, khususnya pada area vulkanik yang rentan terhadap aktivitas erupsi. Dengan luasnya wilayah Indonesia dan kompleksitas lanskap, teknologi penginderaan jauh dikombinasikan dengan Deep Learning menjadi solusi efektif untuk memetakan area tutupan lahan dan memantau fenomena vulkanik.

Metodologi penelitian melibatkan tahapan pengumpulan data citra satelit dari Google Earth Pro dan *dataset* penelitian Del Rosso yang berisis citra satelit area gunung api, *preprocessing* data untuk menghilangkan *noise* seperti label peta dan *watermark*, serta anotasi data untuk mensegmentasi area yang terbagi menjadi 14 kelas. Model *DeepLabV3+* dengan *ResNet34* sebagai *encoder* dilatih menggunakan *framework* PyTorch. Best model dipilih berdasarkan hasil evaluasi pada data *validation* menggunakan metrik *loss*, *Intersection over Union* (IoU), *accuracy*, *dice*, dan *mean Average Precision* (mAP). Evaluasi kinerja akhir dilakukan pada *test set* menggunakan metrik *mean Intersection over Union* (*mIoU*) dan *mAP*.

Hasil *best model* pada tahap pelatihan pertama menunjukkan performa yang cukup baik dengan nilai mAP sebesar 67%, *IoU* sebesar 52%, dan *dice coefficient* sebesar 63%. Namun, evaluasi pada *test set* menunjukkan bahwa kinerja model masih belum optimal dengan nilai *mIoU* hanya mencapai 39% dan mAP sebesar 34%. Hal ini menunjukkan bahwa model mengalami kesulitan dalam melakukan generalisasi pada data uji, terutama pada kelas-kelas tertentu yang memiliki kemiripan karakteristik visual seperti endapan material vulkanik dan jalur lahar.

Penelitian ini menyimpulkan bahwa meskipun model *DeepLabV3+* memiliki potensi dalam segmentasi citra satelit tetapi performanya masih perlu ditingkatkan. Potensi ini membuka peluang yang luas bagi penelitian selanjutnya. Peningkatan dapat dilakukan melalui eksplorasi lebih mendalam terhadap arsitektur model atau penerapan strategi pelatihan yang lebih optimal.

Kata kunci: Segmentasi Semantik, *DeepLabV3+*, Citra Satelit, Tutupan Lahan, Objek Vulkanik.

GLOSARIUM

Artificial Intelligence	Cabang ilmu komputer yang bertujuan untuk menciptakan sistem yang dapat melakukan tugas-tugas yang biasanya memerlukan kecerdasan manusia, seperti pengenalan gambar dan pengambilan keputusan.
Citra Satelit	Gambar permukaan bumi yang diambil oleh satelit di orbit. Digunakan untuk berbagai aplikasi seperti pemetaan lahan dan pemantauan lingkungan.
Deep Learning	Subbidang pembelajaran mesin yang menggunakan jaringan saraf tiruan untuk memproses data dan mengenali pola yang kompleks.
DeepLabV3+	Model Deep Learning berbasis <i>convolutional neural network</i> (CNN) yang dirancang untuk segmentasi citra semantik dengan memanfaatkan teknik <i>atrous spatial pyramid pooling</i> (ASPP) dan <i>encoder-decoder</i> .
<i>Hyperparameter</i>	Parameter yang digunakan untuk mengontrol proses pelatihan model Deep Learning, seperti <i>learning rate</i> , <i>batch size</i> , dan jumlah <i>epoch</i> .
<i>Semantic Segmentation</i>	Proses pemberian label pada setiap piksel dalam gambar untuk mengidentifikasi berbagai objek atau wilayah.

DAFTAR ISI

HALAMAN JUDUL	i
HALAMAN PENGESAHAN DOSEN PEMBIMBING	ii
HALAMAN PENGESAHAN DOSEN PENGUJI	iii
HALAMAN PERNYATAAN KEASLIAN TUGAS AKHIR	iv
HALAMAN PERSEMBAHAN	v
HALAMAN MOTO	vi
KATA PENGANTAR	vii
SARI.....	ix
GLOSARIUM	x
DAFTAR ISI	xi
DAFTAR TABEL	xiii
DAFTAR GAMBAR.....	xiv
BAB I PENDAHULUAN	1
1.1 Latar Belakang	1
1.2 Rumusan Masalah	2
1.3 Batasan Masalah	2
1.4 Tujuan Penelitian	3
1.5 Manfaat Penelitian	3
1.6 Metodologi Penelitian	3
1.7 Sistematika Penulisan	4
BAB II LANDASAN TEORI	5
2.1 Citra Satelit	5
2.2 Artificial Intelligence (AI).....	5
2.3 Machine Learning	8
2.4 Deep Learning.....	8
2.5 Artificial Neural Network	9
2.6 Computer Vision	10
2.7 Image Segmentation.....	12
2.8 Semantic Segmentation.....	13
2.9 Pytorch	14
2.10 Segmentation Models PyTorch	15
2.11 Convolutional Neural Network (CNN).....	15
2.11.1 <i>Convolutional Layer</i>	16
2.11.2 <i>Pooling Layer</i>	18
2.11.3 <i>Fully Connected Layer</i>	19
2.12 Model DeepLabV3	21
2.13 Model DeepLabV3+	22
2.14 Perbedaan DeepLabV3 dan DeepLabV3+	26
2.15 <i>Transfer Learning</i>	27
2.16 Residual Network (ResNet).....	27
2.17 <i>Hyperparameter</i>	29
2.18 Cross-entropy Loss.....	30
2.19 Dice Loss	31
2.20 <i>Data Augmentation</i>	32
2.21 <i>Performance Measurement</i>	32
2.22 Studi Literatur.....	36
BAB III METODOLOGI PENELITIAN	42
3.1 Pengumpulan Citra.....	42

3.2	Pra-Pemrosesan Citra	48
3.3	Anotasi Citra	50
3.4	<i>Dataset Preparation</i>	52
3.5	Konfigurasi Parameter	52
3.6	Pelatihan Model	53
	3.6.1 <i>Patching Citra</i>	53
	3.6.2 <i>Split Patch Data</i>	54
	3.6.3 <i>Data Augmentation</i>	54
	3.6.4 <i>Train Model</i>	54
3.7	Evaluasi Model	55
3.8	Tes Model	55
3.9	Analisis Hasil	56
	BAB IV HASIL DAN PEMBAHSAN	57
4.1	Pengumpulan Citra	57
4.2	Pra-Pemrosesan Citra	60
4.3	Anotasi Citra	61
4.4	<i>Dataset Preparation</i>	62
4.5	Konfigurasi Parameter	64
4.6	Pelatihan Model	77
	4.6.1 <i>Patching Citra</i>	80
	4.6.2 <i>Split Patch Data</i>	86
	4.6.3 <i>Data Augmentation</i>	95
	4.6.4 <i>Train Model</i>	99
4.7	Evaluasi Model	112
4.8	Tes Model	126
4.9	Analisis Hasil	161
	BAB V KESIMPULAN DAN SARAN	166
5.1	Kesimpulan	166
5.2	Saran	166
	DAFTAR PUSTAKA	168
	LAMPIRAN	175

DAFTAR TABEL

Tabel 3.1 Penjelasan kelas tutupan lahan	43
Tabel 3.2 Penjelasan kelas citra gunung api	45
Tabel 3.3 <i>Color dictionary</i>	50
Tabel 4.1 Deskripsi folder.....	58
Tabel 4.2 Spesifikasi komputer tahap pelatihan	65
Tabel 4.3 Spesifikasi komputer tahap pengujian model	65
Tabel 4.4 Eksperimen tahap pertama.....	67
Tabel 4.5 Eksperimen tahap kedua	69
Tabel 4.6 Eksperimen tahap ketiga.....	71
Tabel 4.7 Eksperimen tahap keempat	73
Tabel 4.8 Eksperimen tahap kelima.....	76
Tabel 4.9 Eksperimen tahap keenam	76
Tabel 4.10 Konfigurasi <i>hyperparameter</i> terbaik	77
Tabel 4.11 Skenario pelatihan.....	77
Tabel 4.12 Indeks kelas.....	94
Tabel 4.13 Skenario <i>testing model</i>	126
Tabel 4.14 Prediksi model skenario pelatihan pertama	135
Tabel 4.15 Prediksi model skenario pelatihan kedua.....	148
Tabel 4.16 Prediksi model skenario pelatihan ketiga	153

DAFTAR GAMBAR

Gambar 2.1 Citra satelit kota Yogyakarta.....	5
Gambar 2.2 Bagian kecerdasan buatan	6
Gambar 2.3 Hirarki kecerdasan buatan.....	7
Gambar 2.4 Struktur neuron manusia	10
Gambar 2.5 Struktur neuron buatan.....	10
Gambar 2.6 <i>Semenatic Segmentation</i>	12
Gambar 2.7 <i>Instance Segmentation</i>	13
Gambar 2.8 <i>CNN Structure</i>	16
Gambar 2.9 <i>Convolutional Layer</i>	18
Gambar 2.10 <i>Pooling Layer</i>	19
Gambar 2.11 <i>Fully Conected Layer</i>	21
Gambar 2.12 <i>Spatial Pyramid Pooling</i>	24
Gambar 2.13 <i>Encoder-Decoder</i>	24
Gambar 2.14 <i>Encoder-Decoder with Spatial Pyramid Pooling</i>	25
Gambar 2.15 Arsitektur model DeepLabV3+ dengan proses menghasilkan prediksi.....	25
Gambar 2.16 ResNet <i>architecure</i>	28
Gambar 2.17 <i>Confusion matrix</i>	36
Gambar 3.1 Diagram proses penelitian.....	42
Gambar 3.2 Kelas-kelas pada citra tutupan lahan.....	44
Gambar 3.3 Kelas-kelas citra gunung api	46
Gambar 3.4 Kelas-kelas citra gunung api	47
Gambar 3.5 Citra gunung berapi memiliki label	49
Gambar 3.6 Proses penghapusan label.....	49
Gambar 3.7 Citra asli	51
Gambar 3.8 Anotasi dengan SketchBook Pro.....	52
Gambar 4.1 Tambahan citra, (a) Citra dari Google Earth Pro, (b) Citra dari <i>paper</i>	57
Gambar 4.2 Citra tambahan pada tahap ketiga, (a) Citra gunung api dari Sentinel Hub, (b) Citra dari <i>paper</i>	58
Gambar 4.3 Citra tutupan lahan pada folder Tile 1	59
Gambar 4.4 Citra gunung api pada Tile 2, (a) Citra Gunung Merapi, (b) Citra dari <i>paper</i>	60
Gambar 4.5 Citra pada folder tes, (a) Citra tutupan lahan, (b) Citra gunung api.	60

Gambar 4.6 Hasil proses penghapusan label dan <i>watermark</i>	61
Gambar 4.7 Citra beserta pasangan <i>mask</i> , (a) Citra tutupan lahan, (b) <i>Mask</i> tutupan lahan. .62	
Gambar 4.8 Citra beserta pasangan <i>mask</i> , (a) Citra gunung api, (b) <i>Mask</i> gunung api.....	62
Gambar 4.9 Struktur <i>dataset</i>	63
Gambar 4.10 Isi Subfolder <i>Tile1</i>	63
Gambar 4.11 Isi Subfolder <i>Images</i>	64
Gambar 4.12 Isi Subfolder <i>Masks</i>	64
Gambar 4.13 Sampel citra tes <i>lava dome</i> , (a) Citra original, (b) <i>Ground truth mask</i> , (c) <i>Predicted mask</i>	70
Gambar 4.14 Sampel citra tes <i>lava dome</i> , (a) Citra original, (b) <i>Ground truth mask</i> , (c) <i>Predicted mask</i>	75
Gambar 4.15 Kode pengaturan <i>device</i>	78
Gambar 4.16 Kode instalasi <i>library</i>	79
Gambar 4.17 Impor <i>library</i>	80
Gambar 4.18 Kode kelas <i>dataset</i>	82
Gambar 4.19 Kode <i>patching</i> gambar	83
Gambar 4.20 Hasil proses <i>patching</i> 62 citra.....	84
Gambar 4.21 Hasil proses <i>patching</i> 72 citra.....	84
Gambar 4.22 Citra beresolusi besar, (a) Citra tutupan lahan, (b) Pasangan <i>mask</i>	84
Gambar 4.23 Hasil proses <i>cropping</i> , (a) Citra tutupan lahan, (b) Pasangan <i>mask</i>	85
Gambar 4.24 <i>Patch image</i> beserta labelnya.....	85
Gambar 4.25 Hasil <i>patching</i> citra	86
Gambar 4.26 Hasil <i>patching mask</i>	86
Gambar 4.27 Kode membagi <i>dataset</i>	87
Gambar 4.28 Kode untuk melihat pasangan citra dan <i>mask</i>	88
Gambar 4.29 Hasil pembagian data 62 citra	88
Gambar 4.30 Hasil pembagian data 72 citra	89
Gambar 4.31 Kode menampilkan jumlah objek	90
Gambar 4.32 Jumlah objek dalam <i>dataset</i> untuk 62 citra.....	91
Gambar 4.33 Jumlah objek dalam <i>dataset</i> untuk 72 citra.....	91
Gambar 4.34 Distribusi kelas 62 citra.....	92
Gambar 4.35 Distribusi kelas 72 citra.....	92
Gambar 4.36 Kode distribusi kelas	93

Gambar 4.37 Jumlah piksel pada 62 citra	94
Gambar 4.38 Jumlah piksel pada 72 citra	94
Gambar 4.39 Kode proses augmentasi	96
Gambar 4.40 Hasil proses augmentasi pada 62 citra	97
Gambar 4.41 Hasil proses augmentasi pada 72 citra	97
Gambar 4.42 <i>Patch</i> utupan lahan, (a) <i>Patch</i> asli, (b) <i>Mask</i> asli	98
Gambar 4.43 <i>Patch</i> utupan lahan, (a) <i>Patch</i> hasil augmentasi, (b) <i>mask</i> hasil augmentasi ...	98
Gambar 4.44 <i>Patch</i> gunung api, (a) <i>Patch</i> asli, (b) <i>Mask</i> asli	98
Gambar 4.45 Gambar 4.46 <i>Patch</i> utupan lahan, (a) <i>Patch</i> hasil augmentasi, (b) <i>mask</i> hasil augmentasi	99
Gambar 4.47 Kode menginisialisasi <i>dataloader</i>	100
Gambar 4.48 Kode untuk pembobotan kelas dan penggabungan fungsi <i>loss</i>	101
Gambar 4.49 Hasil pembobotan kelas	102
Gambar 4.50 Kode pendefinisian model dan <i>hyperparameter</i>	103
Gambar 4.51 Perubahan nilai pada variabel <i>criterion</i>	103
Gambar 4.52 Arsitektur DeepLabV3+	107
Gambar 4.53 Proses pelatihan model	108
Gambar 4.54 Kode pelatihan model	111
Gambar 4.55 Hasil pelatihan model	112
Gambar 4.56 Kode mengekstraksi metrik pelatihan	113
Gambar 4.57 Kode untuk membuat grafik	114
Gambar 4.58 Grafik nilai <i>loss</i> skenario pelatihan pertama	115
Gambar 4.59 Grafik nilai <i>loss</i> skenario pelatihan kedua	116
Gambar 4.60 Grafik nilai <i>loss</i> skenario pelatihan ketiga	117
Gambar 4.61 Grafik <i>Intersection over union</i> skenario pelatihan pertama	117
Gambar 4.62 Grafik <i>Intersection over union</i> skenario pelatihan kedua	118
Gambar 4.63 Grafik <i>Intersection over union</i> skenario pelatihan ketiga	119
Gambar 4.64 Grafik akurasi skenario pelatihan pertama	120
Gambar 4.65 Grafik akurasi skenario pelatihan kedua	121
Gambar 4.66 Grafik akurasi skenario pelatihan ketiga	121
Gambar 4.67 Grafik <i>f1-score (dice)</i> skenario pelatihan pertama	122
Gambar 4.68 Grafik <i>f1-score (dice)</i> skenario pelatihan kedua	123
Gambar 4.69 Grafik <i>f1-score (dice)</i> skenario pelatihan ketiga	123

Gambar 4.70 Grafik mAP skenario pelatihan pertama	124
Gambar 4.71 Grafik mAP skenario pelatihan kedua	125
Gambar 4.72 Grafik mAP skenario pelatihan ketiga	125
Gambar 4.73 Citra data tes.....	127
Gambar 4.74 <i>Ground truth</i>	127
Gambar 4.75 <i>Predicted mask</i> , (a) Ukuran 320x320, (b) Ukuran 512x512, (c) Ukuran 1024x1024.....	127
Gambar 4.76 Kode untuk tes model	131
Gambar 4.77 Citra asli sebelum proses <i>resizing</i>	132
Gambar 4.78 Hasil <i>resizing</i> citra	132
Gambar 4.79 Hasil <i>patching</i> citra	133
Gambar 4.80 Hasil prediksi tiap <i>patch</i>	133
Gambar 4.81 Hasil prediksi model, (a) Ground truth, (b) Predicted mask	133
Gambar 4.82 Tampak visual, (a) Endapan vulkanik lama, (b) Endapan vulkanik baru	139
Gambar 4.83 <i>Confusion matrix</i> data tes skenario pelatihan pertama.....	139
Gambar 4.84 Tampak visual atap bangunan.....	140
Gambar 4.85 Tampak visual, (a) <i>Herbaceous vegetation</i> , (b) Area terbuka	140
Gambar 4.86 Citra tes nomor lima.....	141
Gambar 4.87 Citra tes nomor enam	142
Gambar 4.88 Tampak visual, (a) <i>Herbaceous vegetation</i> , (b) Vegetasi	142
Gambar 4.89 Tampak visual, (a) Kelas air, (b) Kelas <i>lava dome</i>	143
Gambar 4.90 Citra tes nomor satu	144
Gambar 4.91 Area jalur lahar.....	144
Gambar 4.92 Tampak visual kelas jalur lahar diantara kelas endapan vulkanik lama	145
Gambar 4.93 Tampak visual, (a) Kelas <i>cropland</i> , (b) Kelas endapan vulkanik lama	146
Gambar 4.94 Tampak visual, (a) Vegetasi pada data <i>training</i> , (b) Endapan baru.....	146
Gambar 4.95 Metrik per kelas	147
Gambar 4.96 <i>Confusion matrix</i> data tes skenario pelatihan kedua.....	152
Gambar 4.97 <i>Confusion matrix</i> data tes skenario pelatihan ketiga ukuran <i>patch</i> 320x320...157	
Gambar 4.98 <i>Confusion matrix</i> data tes skenario pelatihan ketiga ukuran <i>patch</i> 512x512...158	
Gambar 4.99 <i>Confusion matrix</i> data tes skenario pelatihan ketiga ukuran <i>patch</i> 1024x1024	159
Gambar 4.100 Citra tes nomor delapan	160

Gambar 4.101 <i>Ground truth</i> citra nomor delapan	160
Gambar 4.102 Hasil prediksi <i>lava dome</i> , (a) Ukuran patch 320x320, (b) Ukuran patch 512x512, (c) Ukuran patch 1024x1024	160
Gambar 4.103 Citra Tes.....	162
Gambar 4.104 <i>Ground truth</i>	162
Gambar 4.105 <i>Predicted mask</i>	162
Gambar 4.106 Citra tes	163
Gambar 4.107 <i>Ground truth</i>	163
Gambar 4.108 <i>Predicted mask</i>	163
Gambar 4.109 Citra tes	164
Gambar 4.110 <i>Ground truth</i>	164
Gambar 4.111 <i>predicted label</i>	164

BAB I

PENDAHULUAN

1.1 Latar Belakang

Indonesia adalah negara kepulauan terbesar di dunia, terdiri dari 13.466 pulau dengan luas daratan mencapai 1.922.570 km² dan wilayah perairan seluas 3.257.483 km². Negara ini mencakup berbagai jenis lanskap seperti daratan, lautan, gunung, pantai, serta beragam tutupan lahan lainnya (Badan Informasi Geospasial, n.d.). Berdasarkan informasi dari BMKG, Indonesia terletak di kawasan Cincin Api Pasifik yang menjadikannya rentan terhadap aktivitas vulkanik dan tektonik (Badan Meteorologi, 2018). Mengingat luasnya wilayah Indonesia, pemantauan tutupan lahan serta mitigasi bencana menjadi tantangan besar yang memerlukan solusi strategis. Dalam hal ini, penginderaan jauh dapat berperan sebagai salah satu alat penting untuk memantau wilayah dan melakukan pengawasan aktivitas gunung api.

Menurut Mertikas et al. (2020) penginderaan jauh merupakan ilmu dan teknologi yang memungkinkan identifikasi serta penentuan karakteristik dan sifat objek di bumi dari jarak jauh. Melalui penginderaan jauh, pemantau dapat mendeteksi, mengukur, memetakan dan memantau berbagai objek untuk keperluan manajemen lingkungan. Salah satu data dari hasil penginderaan jauh adalah data citra satelit. Citra satelit adalah hasil yang vital dalam pemetaan dan pemantauan lingkungan. Berbagai metode statistik dan komputasi telah banyak dikembangkan untuk mendapatkan data tutupan lahan dan penggunaan lahan berbasis citra satelit (Mertikas et al., 2020).

Deep Learning semakin diminati untuk klasifikasi otomatis pada citra satelit. Deep Learning mampu menangani masalah data besar dan kompleksitas fitur pada kasus identifikasi penggunaan lahan. Algoritma ini telah banyak digunakan untuk klasifikasi gambar dan segmentasi objek. Metode Deep Learning terbukti sangat efektif dalam berbagai aplikasi penginderaan jauh termasuk segmentasi citra satelit (Thapa et al., 2023).

Selain untuk kebutuhan segmentasi lahan, Deep Learning juga dapat digunakan untuk pemantauan aktivitas vulkanik pada gunung api. Pemantauan terkait pergerakan awan panas, *lava flow*, material vulkanik dan hal-hal terkait kebencanaan gunung api. Penelitian (Del Rosso et al., 2021) telah membuktikan bahwa Deep Learning mampu diterapkan dengan baik untuk mengidentifikasi erupsi dari sebuah gunung api. Tentunya dengan berbagai modifikasi dan penggunaan model Deep Learning yang tepat akan membuka jalan untuk penelitian terbarukan berikutnya.

Salah satu algoritma terbaru yang sangat efektif dalam segmentasi citra adalah DeepLabV3+ (L.-C. Chen et al., 2018). Model ini menggunakan arsitektur *convolution neural network* (CNN) yang dilengkapi dengan teknik *dilated convolutions* dan *atrous spatial pyramid pooling* (ASPP), sehingga model ini memungkinkan untuk mengambil informasi dari berbagai skala. Penggunaan DeepLabV3+ memungkinkan peneliti untuk memetakan dan mengklasifikasikan tutupan lahan dengan lebih baik, memberikan kontribusi besar dalam analisis perubahan lingkungan dan perencanaan tata ruang.

Oleh karena itu, penelitian ini bertujuan untuk mengusulkan penggunaan model segmentasi DeepLabV3+ pada citra satelit area tutupan lahan dan objek sekitar gunung api. Model yang diusulkan diharapkan mampu mendukung proses segmentasi area tutupan lahan dan identifikasi objek-objek sekitar gunung api. Fokus utama penelitian ini adalah pada implementasi serta evaluasi kinerja model segmentasi tersebut.

1.2 Rumusan Masalah

Berdasarkan uraian latar belakang yang telah disampaikan, dapat dirumuskan beberapa pertanyaan penelitian sebagai berikut:

1. Apakah model DeepLabV3+ dapat digunakan untuk segmentasi citra satelit pada area tutupan lahan dan objek di sekitar gunung api?
2. Bagaimana kinerja model DeepLabV3+ dalam melakukan segmentasi citra satelit pada area tutupan lahan dan objek di sekitar gunung api?

1.3 Batasan Masalah

Agar masalah yang diteliti tidak terlalu luas dan menyimpang, dilakukan pembatasan masalah sebagai berikut:

- a. Citra satelit yang diambil adalah area tutupan lahan di Indonesia menggunakan citra Google Earth Pro.
- b. Citra untuk objek gunung api diambil dari Google Earth Pro, Sentinel Hub, Google Images, dan *dataset* gunung api pada penelitian (Del Rosso et al., 2021).
- c. Jumlah kelas utama yang digunakan adalah 14 kelas.
- d. *Best model* ditentukan berdasarkan hasil nilai *evaluation loss* terendah pada saat proses pelatihan.
- e. Performa model ditentukan pada data tes dengan metrik *mean Intersection over union* (mIoU), dan *mean average precision* (mAP).

1.4 Tujuan Penelitian

Tujuan penelitian ini adalah sebagai berikut:

- a. Menerapkan model DeepLabV3+ dalam segmentasi citra satelit pada citra satelit tutupan lahan dan gunung api.
- b. Mengevaluasi efektivitas dan performa model DeepLabV3+ dalam segmentasi citra satelit pada area tutupan lahan dan objek sekitar gunung api.

1.5 Manfaat Penelitian

Penelitian ini diharapkan dapat memberikan manfaat sebagai berikut:

- a. Menambah literatur ilmiah mengenai penggunaan model DeepLabV3+ dalam segmentasi citra satelit.
- b. Menggali potensi teknologi Deep Learning dalam penginderaan jauh, khususnya dalam segmentasi citra satelit tutupan lahan dan gunung api.

1.6 Metodologi Penelitian

Metodologi yang diterapkan dalam penelitian ini terdiri dari beberapa tahap, yaitu:

- a. Pengumpulan Data: Mengumpulkan citra satelit tutupan lahan dan citra satelit gunung berapi yang akan digunakan dalam penelitian.
- b. *Image Preprocessing*: Melakukan data *preprocessing* citra satelit, seperti pemilihan citra satelit yang sesuai dengan kebutuhan penelitian.
- c. *Image Annotation* : Citra dikumpulkan dan dilabeli sesuai kelas yang ditentukan.
- d. Konfigurasi Parameter : Mencari konfigurasi parameter terbaik untuk melatih model.
- e. Pelatihan Model: Melatih model DeepLabV3+ menggunakan data yang telah diproses untuk menghasilkan model segmentasi optimal.
- f. Evaluasi Model: Mengevaluasi kinerja model DeepLabV3+ dengan menggunakan metrik evaluasi seperti *loss*, IoU, *accuracy*, *F1-score* dan mAP pada citra satelit area tutupan lahan dan gunung berapi.
- g. Pengujian Model: Menguji model DeepLabV3+ yang telah dilatih pada *dataset* uji untuk mengevaluasi kinerja model dalam situasi nyata.
- h. Analisis Hasil: Menganalisis hasil segmentasi yang diperoleh dari model, dan mengidentifikasi kekuatan dan kelemahan model.

1.7 Sistematika Penulisan

Struktur penulisan dalam laporan tugas akhir ini terbagi menjadi beberapa bab yang mencakup rangkuman masalah serta metode penyelesaiannya. Berikut adalah sistematika penulisan yang terbagi menjadi 5 bab:

a. **BAB I PENDAHULUAN**

Bab ini membahas latar belakang masalah, rumusan masalah, batasan masalah, tujuan penelitian, manfaat penelitian, metodologi penelitian, dan sistematika penulisan.

b. **BAB II LANDASAN TEORI**

Bab ini mengulas teori-teori dasar, berkaitan dengan segmentasi citra satelit menggunakan model DeepLabV3+, serta meninjau penelitian sebelumnya yang relevan dengan topik ini.

c. **BAB III METODOLOGI PENELITIAN**

Bab ini menjelaskan langkah-langkah penelitian, meliputi analisis masalah, kajian literatur, pengumpulan data, pra-pemrosesan data, pelatihan model, pengujian model, evaluasi model, dan analisis hasil.

d. **BAB IV HASIL DAN PEMBAHASAN**

Bab ini memaparkan hasil implementasi model DeepLabV3+ untuk segmentasi citra satelit, serta mengevaluasi dan menganalisis hasil pengujian model pada citra tutupan lahan dan objek sekitar gunung api.

e. **BAB V KESIMPULAN DAN SARAN**

Bab ini menyajikan kesimpulan dari seluruh proses penelitian yang telah dilakukan dan memberikan saran untuk penelitian lebih lanjut di masa depan.

BAB II

LANDASAN TEORI

2.1 Citra Satelit

Citra satelit adalah gambar tampak permukaan bumi, diambil oleh satelit yang mengorbit di luar angkasa. Citra ini digunakan untuk berbagai macam tujuan seperti pemetaan lahan, pemantauan lingkungan, manajemen sumber daya alam, dan analisis perubahan permukaan bumi. Citra satelit dapat mencakup berbagai spektrum cahaya. Contohnya gelombang elektromagnetik, cahaya tampak, inframerah, maupun radar.

Menurut Kamus Besar Bahasa Indonesia (KBBI), citra adalah "gambaran yang dimiliki orang banyak mengenai pribadi, perusahaan, organisasi, atau produk". Dalam konteks citra satelit ini merujuk pada gambaran permukaan bumi yang diperoleh melalui teknologi satelit. Selain itu, menurut Earth Resources Observation and Science citra satelit adalah "gambar digital planet bumi yang diambil oleh sensor satelit di orbit planet bumi (Earth Resources Observation and Science (EROS) Center, n.d.). Sensor ini digunakan untuk berbagai aplikasi ilmiah, militer, atau tujuan komersial". Penggunaan citra satelit sangat penting dalam penginderaan jauh karena kemampuan mereka untuk menangkap data dari area yang luas dan bahkan sulit dijangkau secara terestrial.



Gambar 2.1 Citra satelit kota Yogyakarta

Sumber: Google Earth Pro

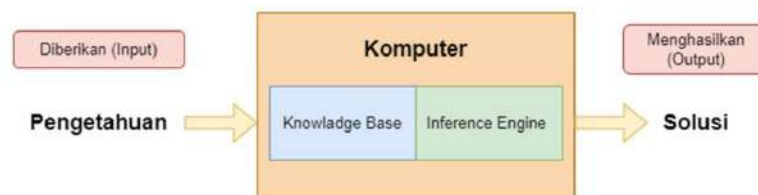
2.2 Artificial Intelligence (AI)

Artificial Intelligence (AI) adalah cabang dari ilmu komputer, bertujuan untuk menciptakan sistem yang mampu melakukan tugas-tugas di mana biasanya memerlukan kecerdasan manusia. Tugas-tugas tersebut mencakup pemahaman seperti bahasa alami,

pengenalan suara atau gambar, pengambilan keputusan, serta pemecahan masalah kompleks lainnya (Galdran et al., 2022)(Alexander et al., 2020).

Menurut Haenlein & Kaplan, (2019). AI dapat didefinisikan sebagai "sistem yang menunjukkan perilaku cerdas dengan menganalisis lingkungan mereka dan mengambil tindakan, dengan beberapa derajat otonomi, untuk mencapai tujuan tertentu". AI mencakup berbagai subbidang seperti Machine Learning, *neural networks*, *natural language processing*, dan *robotics*.

Pengembangan kecerdasan buatan melibatkan dua komponen utama, yaitu basis pengetahuan (*Knowledge Base*) dan mesin inferensi (*Inference Engine*). Basis pengetahuan mencakup kumpulan berbagai fakta, pengetahuan, serta teori. Selain itu, mesin inferensi memiliki kemampuan untuk menyimpulkan informasi berdasarkan basis p(Putri, 2020). Proses ini diilustrasikan dalam Gambar 2.2 yang menunjukkan komponen dari kecerdasan buatan. Di samping itu, kecerdasan buatan juga memiliki berbagai turunan seperti diperlihatkan dalam Gambar 2.3.



Gambar 2.2 Bagian kecerdasan buatan

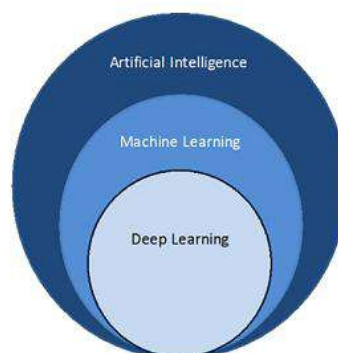
Sumber: Putri (2020)

Pernyataan Alsheibani et al. (2018) mengenai tujuan pengembangan AI adalah untuk memberikan komputer kemampuan seperti manusia dalam melakukan tugas-tugas kognitif yang biasanya memerlukan kecerdasan manusia, seperti pemahaman, penalaran, dan pemecahan masalah. Tujuan ini dapat dibagi menjadi beberapa aspek utama:

- Tujuan ilmiah untuk memahami lebih dalam tentang proses kognitif dan kecerdasan manusia melalui simulasi dengan mesin.
- Tujuan praktis untuk membuat mesin mampu menyelesaikan tugas-tugas kompleks secara mandiri.
- Tujuan bisnis untuk meningkatkan efisiensi dan produktivitas dalam berbagai bidang industri dengan menggunakan teknologi AI untuk otomatisasi dan pengambilan keputusan berbasis data.

Penerapan kecerdasan buatan mencakup berbagai bidang terapan. Menurut penelitian yang dilakukan oleh Grover et al. (2022) AI dapat diterapkan meliputi:

- a. Sistem Pakar (*Expert System*) yang dapat meniru kemampuan seorang pakar dalam menyelesaikan masalah tertentu, misalnya dalam diagnosis medis.
- b. Pengenalan Ucapan (*Speech Recognition*) untuk memungkinkan mesin mengenali dan memproses ucapan manusia, seperti asisten virtual.
- c. Jaringan Saraf Tiruan (*Neural Networks*) yaitu sistem yang meniru cara kerja otak manusia dalam memproses informasi dan belajar dari data.
- d. Pengolahan Bahasa Alami (*Natural Language Processing*) yang memungkinkan mesin memahami dan merespons bahasa manusia secara alami seperti dalam *chatbot*.
- e. Pengolahan Citra (*Image Processing*) yang memungkinkan mesin untuk mengenali dan memproses gambar contohnya sistem keamanan dan diagnosis medis.
- f. *Probabilistic Reasoning* yaitu sistem yang dapat menangani ketidakpastian dalam pengambilan keputusan. Sistem ini sangat penting dalam prediksi dan analisis risiko.
- g. *Game Playing* dapat diterapkan AI dalam mengembangkan strategi permainan yang sangat kompleks, seperti dalam permainan catur dan go.
- h. Optimasi (*Evolutionary Computing*) yaitu metode komputasi yang menggunakan prinsip evolusi untuk menemukan solusi optimal dalam masalah kompleks.
- i. *Robotics* dan *Sensory System* misalnya aplikasi AI dalam pengembangan robot dan sistem sensorik yang dapat berinteraksi dengan lingkungan sekitarnya secara cerdas seperti manusia.



Gambar 2.3 Hirarki kecerdasan buatan

Sumber: Amazon Web Service

Berdasarkan Gambar 2.3 dapat digambarkan bahwa *artificial intelligence* (AI) merupakan bidang lebih luas yang mencakup machine learning (ML) dan Deep Learning (DL). Deep Learning adalah bagian lebih canggih dari Machine Learning. Hal ini dikarenakan Deep Learning menggunakan jaringan saraf tiruan untuk memproses dan memahami data dengan lebih canggih.

2.3 Machine Learning

Pembelajaran mesin (Machine Learning) adalah cabang dari kecerdasan buatan (*artificial intelligence*) di mana komputer dapat belajar secara mandiri tanpa perlu diprogram ulang oleh manusia. Proses pembelajaran mesin dimulai dengan membangun model berdasarkan input yang diberikan untuk menghasilkan prediksi atau pengambilan keputusan berdasarkan data, fakta, atau pengalaman (Sarker, 2021).

Terdapat empat kategori utama dalam pembelajaran mesin, yaitu:

- a. Pembelajaran yang diawasi (*Supervised Learning*) menggunakan data input yang telah diberi label atau kelas yang menunjukkan kelompok dari data. Proses ini menghasilkan model dari data berlabel.
- b. Pembelajaran tanpa pengawasan (*Unsupervised Learning*) menggunakan data yang tidak diberi label sehingga perlu mencari pola dari data yang ada dan melakukan pengelompokan berdasarkan karakteristik yang ditemukan.
- c. Pembelajaran semi diawasi (*Semi supervised Learning*) menggunakan metode pembelajaran tanpa pengawasan untuk menemukan informasi tentang variabel input. Selanjutnya, data yang tidak berlabel ini digunakan dalam pembelajaran diawasi untuk membangun model prediksi baru dari data masukan.
- d. Pembelajaran penguatan (*Reinforcement learning*) merupakan metode pembelajaran yang menggabungkan fase pembelajaran dengan fase pengujian. Proses ini bertujuan untuk memetakan situasi sebenarnya ke dalam tindakan mesin guna memperoleh hasil yang maksimal.

2.4 Deep Learning

Deep Learning adalah jenis pembelajaran mesin yang mencapai fleksibilitas dan kekuatan paling tinggi dalam konsep hierarki kecerdasan buatan. Metode Deep Learning memungkinkan komputer untuk mempelajari konsep-konsep kompleks dari kombinasi konsep-

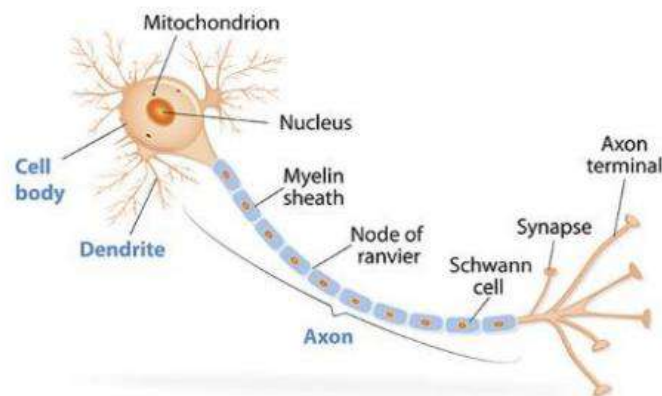
konsep sederhana (M. Chen et al., 2020). Deep Learning dapat diterapkan dalam berbagai bidang seperti klasifikasi teks, suara, gambar, dan video. Dalam pengolahan citra, Deep Learning menunjukkan performa sangat baik (Q. Li et al., 2021). Pada beberapa kasus, penerapan Deep Learning dapat mencapai akurasi sangat tinggi, bahkan melebihi kemampuan manusia. Hal ini dapat diraih karena model dilatih dengan banyak lapisan jaringan syaraf (C. Li & Huang, 2023). Pengklasifikasian data dalam Deep Learning dilakukan melalui dua pendekatan utama, yaitu pelatihan (*training*) dan pengujian (*testing*). Selama proses pelatihan, model Deep Learning mempelajari fitur dari data untuk membedakan antara label yang berbeda. Pada fase pengujian, data diuji dan dianalisis berdasarkan hasil pelatihan sebelumnya (Reader et al., 2020).

2.5 Artificial Neural Network

Dikutip dari Hamdan, (2022). Jaringan saraf tiruan adalah teknologi komputasi yang dirancang untuk meniru otak manusia dalam menyelesaikan masalah. Artificial Neural Network (ANN) belajar dengan membentuk *probability weighted association* antara input dan hasil. Tahap selanjutnya dengan menyimpan nilai bobot tersebut di seluruh jaringan. Proses belajar ini membantu jaringan untuk mengenali pola dan membuat prediksi lebih akurat.

Seperti ditunjukkan pada Gambar 2.4 otak manusia terdiri dari neuron saling terhubung melalui sinapsis. Demikian pula, jaringan saraf buatan terdiri dari neuron buatan yang terhubung melalui koneksi berbobot. Neuron dalam jaringan saraf buatan berfungsi dengan menjumlahkan input yang diterima dan menghasilkan output berdasarkan fungsi aktivasi. Struktur ini memungkinkan jaringan saraf untuk memproses informasi secara paralel, mirip dengan cara kerja otak manusia.

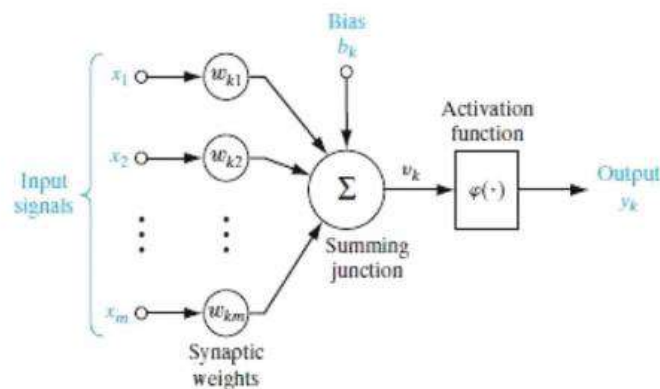
Ada beberapa model jaringan saraf yang umum digunakan. Model paling dasar adalah *single-layer feedforward neural network*, di mana node input terhubung langsung ke node output. Model lebih kompleks adalah *Multilayer feedforward neural network* yang memiliki beberapa lapisan tersembunyi. Lapisan-lapisan tambahan ini memungkinkan jaringan untuk mengekstraksi informasi statistik, lebih tinggi dari input yang diberikan, sehingga meningkatkan kemampuan analisis dan prediksinya.



Gambar 2.4 Struktur neuron manusia

Sumber: Hamdan (2022)

Gambar 2.5 menunjukkan jaringan saraf tiruan terdiri dari unit-unit kecil dan sederhana disebut neuron buatan. Setiap unit ini dihubungkan satu sama lain dengan *weight*. Selanjutnya, unit-unit tersebut menggunakan fungsi aktivasi untuk menghitung jumlah bobot dari input yang diterima dan menentukan outputnya.



Gambar 2.5 Struktur neuron buatan

Sumber: Hamdan (2022)

2.6 Computer Vision

Computer vision atau dalam bahasa Indonesia adalah visi computer merupakan bagian dari kecerdasan buatan, bertujuan untuk membuat komputer mampu memahami dan menafsirkan informasi visual dari gambar atau video. Proses ini mencakup berbagai tahap, seperti akuisisi citra, pemrosesan citra, pengenalan objek, dan pengambilan Keputusan. Cara kerja ini meniru proses sistem penglihatan manusia (Deshmukh, 2021). Dalam melakukan

tugas-tugas tertentu, visi komputer menggunakan berbagai format data, termasuk urutan video, sudut pandang dari beberapa kamera, dan data multidimensi (Singh et al., 2023). Secara keseluruhan, visi komputer adalah kombinasi antara pengolahan citra (*image processing*) dan pengenalan pola (*pattern recognition*) (Shreya M. Shelke et al., 2023).

Kemampuan utama visi komputer meliputi deteksi objek, pengenalan objek, deskripsi objek, inferensi 3D dari objek 2D, dan penafsiran gerakan (Indhurani et al., 2022). Penerapan visi komputer yang luas dalam berbagai bidang seperti ekologi, analisis gambar klinis, dan sektor konstruksi menunjukkan potensinya dalam meningkatkan efisiensi dan akurasi pengolahan informasi visual (Xu et al., 2021). Dengan perkembangan teknologi ini, komputer dapat dilatih untuk mengenali, mengklasifikasikan, dan bereaksi terhadap visual dengan akurasi semakin tinggi (Pandey, 2023). Ada beberapa jenis atau subbidang dari visi komputer, masing-masing menangani aspek berbeda dari pemahaman dan analisis visual. Berikut adalah beberapa jenis tugas visi komputer yang umum:

- a. **Klasifikasi Gambar** : Tugas ini melibatkan pengkategorian gambar ke dalam kelas atau kategori yang telah ditentukan. Contohnya adalah mengklasifikasikan gambar hewan ke dalam spesies yang berbeda atau membedakan antara berbagai objek.
- b. **Deteksi Objek** : Deteksi objek berfokus pada menemukan dan mengidentifikasi objek tertentu dalam sebuah gambar atau video. Biasanya, ini melibatkan menggambar kotak batas di sekitar objek yang dideteksi.
- c. **Segmentasi Semantik (*Semantic Segmentation*)** : Segmentasi semantik melibatkan pemberian label pada setiap piksel dalam gambar untuk membedakan objek atau wilayah yang berbeda. Ini memberikan pemahaman mendetail tentang gambar pada tingkat piksel.
- d. **Segmentasi Objek (*Instance Segmentation*)** : Segmentasi objek selangkah lebih jauh dari segmentasi semantik dengan tidak hanya memberikan label semantik tetapi juga membedakan antara objek individu dari gambar.
- e. **Pose Estimation** : *Pose Estimation* adalah tentang menentukan posisi dan orientasi objek atau bagian tubuh tertentu dalam gambar atau video. Ini biasanya digunakan dalam aplikasi yang melibatkan analisis pose manusia atau robotika.
- f. **Generative Image** : *Generative image* melibatkan pembuatan gambar baru yang memiliki karakteristik tertentu atau memiliki kesamaan dengan set gambar pelatihan tertentu.

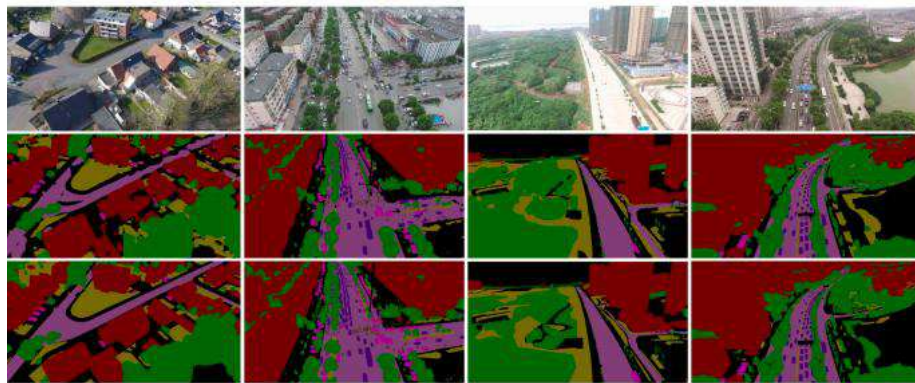
- g. *Visual Recognition* : *Visual Recognition* mencakup berbagai tugas, termasuk klasifikasi gambar, dan deteksi objek. Ini melibatkan pemahaman dan interpretasi keseluruhan konten visual.

2.7 Image Segmentation

Image segmentation adalah proses membagi sebuah citra menjadi beberapa bagian atau segmen agar lebih mudah dianalisis. Setiap segmen terdiri dari piksel-piksel yang memiliki karakteristik serupa berdasarkan kriteria tertentu seperti warna, intensitas, atau tekstur. Tujuan utama dari *image segmentation* adalah untuk menyederhanakan representasi citra menjadi sesuatu lebih bermakna dan lebih mudah dianalisis (Yang et al., 2021). Image Segmentation terbagi menjadi dua jenis yaitu :

a. *Semantic Segmentation*

Semantic segmentation adalah jenis segmentasi di mana setiap piksel dalam gambar diberi label sesuai dengan kelas tertentu. Misalnya, dalam citra jalan, setiap piksel dapat diklasifikasikan sebagai jalan, kendaraan, atau bangunan. Metode ini berfokus pada mengidentifikasi objek pada tingkat piksel tanpa membedakan antara label berbeda dari objek yang sama (L.-C. Chen et al., 2018).



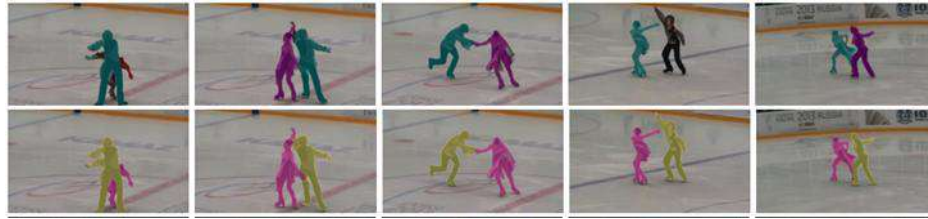
Gambar 2.6 *Semenatic Segmentation*

Sumber: Chen L (2018)

b. *Instance Segmentation*

Pendekatan segementasi ini tidak hanya mengklasifikasikan setiap piksel ke dalam kelas yang sesuai. Tetapi, juga membedakan antara objek berbeda dalam kelas yang sama. Misalnya, dalam sebuah gambar berisi *humans*, *instance segmentation* akan

mengidentifikasi setiap *humans* sebagai entitas terpisah meskipun semuanya berada dalam kelas "*humans*" (Cheng & Zhao, 2022).



Gambar 2.7 *Instance Segmentation*

Sumber: Cheng L (2022)

2.8 Semantic Segmentation

Semantic segmentation adalah proses dalam visi komputer yang bertujuan untuk memberikan label pada setiap piksel dalam sebuah gambar (Guo et al., 2018). Berbeda dengan klasifikasi gambar yang hanya memberikan satu label untuk keseluruhan gambar, atau deteksi objek yang memberikan label dan kotak pembatas untuk objek tertentu, *semantic segmentation* menghasilkan keterangan label yang lebih detail (Guo et al., 2018). Setiap piksel dalam gambar akan diklasifikasikan ke dalam kategori yang sesuai, sehingga memberikan representasi yang lebih mendalam mengenai struktur visual gambar tersebut.

Teknik ini biasanya dilakukan dengan menggunakan Convolutional Neural Networks (CNN) yang mampu mengekstrak fitur dari gambar. Salah satu pendekatan yang paling umum adalah Fully Convolutional Networks (FCN) yang dirancang untuk mengubah model CNN standar menjadi sistem yang dapat memproses gambar dengan ukuran berapa pun dan menghasilkan prediksi label piksel secara langsung. Selain itu, model-model seperti DeepLab dan U-Net juga telah banyak digunakan karena kemampuannya dalam menghasilkan segmentasi yang lebih akurat, terutama pada batas-batas objek yang kompleks (Guo et al., 2018).

Semantic segmentation banyak diterapkan dalam berbagai bidang. Dalam kendaraan otonom. Penerapan ini misalnya digunakan untuk mengenali jalan, rambu, kendaraan, dan pejalan kaki dalam waktu nyata. Dalam dunia medis, *semantic segmentation* membantu dalam analisis citra seperti MRI atau CT scan untuk mendeteksi organ, jaringan, atau tumor. Pada citra satelit, teknik ini mempermudah proses pemetaan geografis dengan membedakan elemen-elemen seperti air, hutan, atau area terbangun (Guo et al., 2018).

Namun, *semantic segmentation* memiliki tantangan tersendiri. Proses ini membutuhkan *dataset* yang besar dengan anotasi hingga *pixel level*. Dengan kemajuan teknologi Deep Learning, *semantic segmentation* terus berkembang dan mencapai hasil yang semakin mendekati kemampuan segmentasi manusia. Teknik ini telah membuka berbagai peluang baru dalam analisis visual yang lebih kompleks dan presisi (Guo et al., 2018).

2.9 Pytorch

PyTorch adalah sebuah *framework* yang beroperasi dalam skala besar dan berjalan di lingkungan heterogen. PyTorch dikembangkan oleh riset Facebook's AI Research (FAIR). Tujuan utama dari *framework* ini adalah untuk pembelajaran mesin dan penelitian Deep Learning (Mishra, 2019). PyTorch menggunakan tensor mirip dengan NumPy dengan tambahan kemampuan menggunakan GPU untuk mempercepat komputasi. *Framework* ini sangat fleksibel dan modular, memungkinkan pengembang untuk bereksperimen dengan berbagai optimisasi dan algoritma pelatihan yang diharapkan. PyTorch mendukung berbagai aplikasi yang berfokus pada pelatihan dan inferensi pada *neural network*.

Menurut (Mishra, 2019). PyTorch memiliki fitur utama sebagai berikut:

- a. PyTorch memiliki skalabilitas yang besar untuk mengolah data yang sangat besar dan memungkinkan komputasi yang mudah.
- b. PyTorch mendukung berbagai teknik Machine Learning, Deep Learning, *reinforcement learning*, dan *convolutional networks*.
- c. Dalam PyTorch memungkinkan definisi, perhitungan, serta pengoptimalan melalui ekspresi matematis yang melibatkan array multidimensi (tensor) yang dapat dimodifikasi selama *runtime*.
- d. Pemrosesan yang dilakukan *GPU* bersifat transparan dengan manajemen otomatis serta optimalisasi memori yang sama dengan data yang digunakan, dan penggunaan sumber daya yang efisien.

PyTorch juga menyediakan berbagai modul penting, antara lain:

- a. Autograd: Memberikan fungsionalitas untuk diferensiasi otomatis tensor yang sangat membantu dalam implementasi model jaringan saraf tiruan.

- b. Optim: Menyediakan teknik optimisasi untuk meminimalkan fungsi error pada model, dan mendukung berbagai metode optimisasi canggih seperti Adam dan *stochastic gradient descent* (SGD).
- c. NN: Modul jaringan saraf tiruan yang memudahkan pendefinisian *layer*, fungsi aktivasi, fungsi loss, dan fungsi optimisasi secara otomatis.

2.10 Segmentation Models PyTorch

Segmentation Models PyTorch (SMP) adalah pustaka Python berbasis PyTorch yang dirancang untuk mempermudah pembangunan model Deep Learning dalam tugas segmentasi gambar. Dengan API yang sederhana, SMP memungkinkan pembuatan model hanya dalam beberapa baris kode. Pustaka ini mendukung berbagai arsitektur populer, seperti U-Net, DeepLabV3+, dan FPN, serta dilengkapi dengan 104 encoder yang memiliki bobot pre-trained, termasuk ResNet, EfficientNet, dan DenseNet. Fitur ini memungkinkan model dalam mencapai hasil yang lebih cepat dan akurat. SMP juga menyediakan fungsi preprocessing yang dapat menyesuaikan data dengan bobot pre-trained, sehingga mempercepat proses pelatihan dan meningkatkan metrik performa. Selain itu, pustaka ini dilengkapi dengan dokumentasi yang lengkap dan integrasi dengan framework seperti Catalyst dan PyTorch Lightning, membuatnya fleksibel untuk berbagai kebutuhan segmentasi. SMP didistribusikan dengan lisensi MIT, sehingga dapat digunakan secara bebas untuk kebutuhan akademik maupun komersial (Segmentation Models, n.d.).

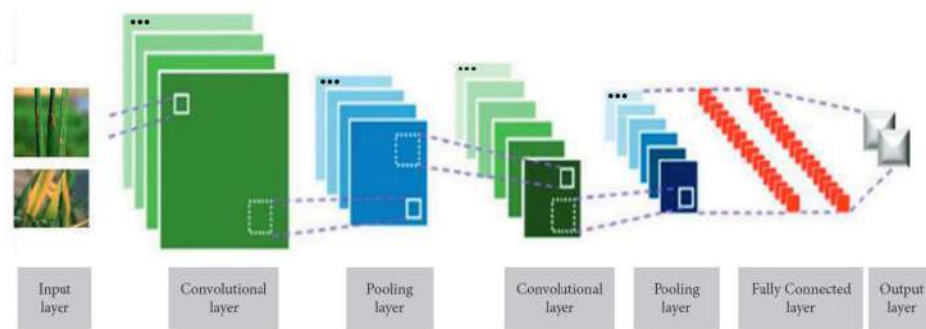
2.11 Convolutional Neural Network (CNN)

Convolutional Neural Network atau disingkat dengan CNN adalah salah satu algoritma Deep Learning yang populer. Algoritma ini dapat digunakan untuk melakukan klasifikasi langsung pada berbagai bidang dua dimensi seperti teks, suara, gambar, dan video (Z. Li et al., 2022). Namun, CNN biasanya digunakan untuk pemrosesan gambar. CNN termasuk dalam jenis *neural network* karena memiliki tingkat kedalaman jaringan yang tinggi (Saleem et al., 2022).

Struktur CNN terdiri dari beberapa lapisan fungsional utama yang bekerja secara berurutan (Z. Li et al., 2022). Diantaranya:

- a. *Convolutional Layer*: Lapisan ini menggunakan filter untuk melakukan konvolusi pada gambar input, menghasilkan *feature map* yang dapat mengekstraksi fitur-fitur dasar dari gambar.

- b. *Pooling Layer*: Lapisan ini berfungsi untuk mengurangi dimensi *feature map* dan memperkuat fitur yang telah diekstraksi dengan mengambil nilai maksimum (*max pooling*) atau rata-rata (*average pooling*) dari area tertentu pada *feature map*.
- c. *Fully Connected Layer*: Lapisan ini menghubungkan semua *neuron* dari lapisan sebelumnya ke setiap *neuron* di lapisan ini, yang biasanya digunakan untuk menghasilkan output klasifikasi akhir.



Gambar 2.8 *CNN Structure*

Sumber: Saleem M (2022)

Gambar 2.8 menunjukkan struktur dari *Convolutional Neural Network* (CNN). Dimulai dari lapisan input menerima gambar asli, kemudian gambar tersebut melewati beberapa lapisan konvolusi (*convolutional layers*). Lapisan ini bertujuan untuk mengekstraksi fitur dasar seperti tepi dan tekstur. Setelah itu, *feature map* yang dihasilkan dari lapisan konvolusi akan diproses oleh lapisan *pooling*. Lalu, mengurangi dimensi *feature map* dan memperkuat fitur paling penting dengan menggunakan teknik seperti *max pooling*. Proses ini diulang beberapa kali, dengan lapisan konvolusi dan *pooling* berturut-turut. Pada akhirnya, lapisan *fully connected* menghubungkan semua *neuron* dari lapisan sebelumnya dan menghasilkan output klasifikasi akhir berdasarkan fitur-fitur yang telah diekstraksi dan diproses oleh lapisan-lapisan sebelumnya.

2.11.1 *Convolutional Layer*

Lapisan konvolusi adalah bagian awal dari arsitektur CNN setelah lapisan input yang terdiri dari kombinasi kernel konvolusi (*neuron*). Setiap kernel (*neuron*) terkait dengan bagian kecil. Bagian kecil ini disebut bidang reseptif (*receptive field*). Proses ini bekerja dengan

membagi gambar input menjadi potongan-potongan gambar lebih kecil (bidang reseptif) dan mengonvolusikan dengan satu set bobot tertentu (Saleem et al., 2022).

Proses pada konvolusi ke-i, dapat dinyatakan sebagai berikut:

- Input: a^{l-1} dengan ukuran $(n_H^{l-1}, n_W^{l-1}, n_C^{l-1})$, a^0 sebagai input gambar.
- Padding: $p^{[l]}$, Stride: $S^{[l]}$
- Beberapa filter: n_C^l di mana setiap $K^{(n)}$ memiliki dimensi: $f^{[l]}, f^{[l]}, n_C^{[l-1]}$
- Bias dari konvolusi ke-n: $b_n^{[l]}$
- Fungsi aktivasi: $g^{[l]}$
- Output: $a^{[l]}$ dengan ukuran $(n_H^{[l]}, n_W^{[l]}, n_C^{[l]})$

Proses konvolusi ini dijelaskan secara matematis dengan rumus-rumus:

$$\forall n \in [1, 2, \dots, n_C^{[l]}] \quad (2.1)$$

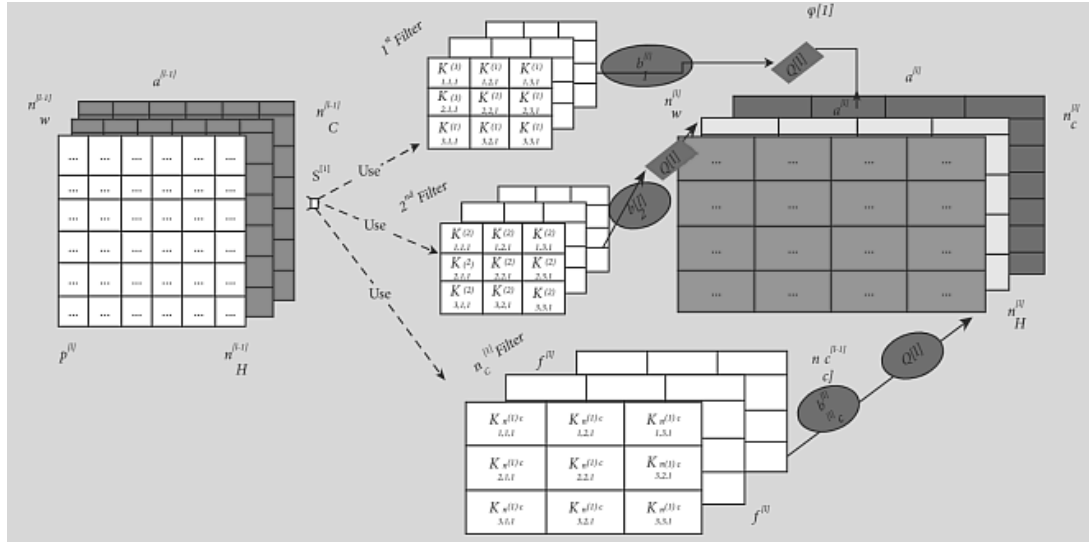
$$\text{Conv}(a^{l-1}, K^{(n)})_{x,y} = g^{[l]} \left(\sum_{i=1}^{n_H^{l-1}} \sum_{j=1}^{n_W^{l-1}} \sum_{k=1}^{n_C^{l-1}} K_{i,j,k}^{(n)} a_{x+i-1, y+j-1, k}^{l-1} + b_n^{[l]} \right) \quad (2.2)$$

$$a^{[l]} = \left[g^{[l]} \left(\text{Conv}(a^{l-1}, K^{(1)}) \right), g^{[l]} \left(\text{Conv}(a^{l-1}, K^{(2)}) \right), \dots, g^{[l]} \left(\text{Conv}(a^{l-1}, K^{(n_C^{[l]})}) \right) \right] \quad (2.3)$$

Pada persamaan 2.1 menyatakan bahwa dalam proses konvolusi memiliki $n_C^{[l]}$ filter untuk setiap konvolusi ke-i. Persamaan 2.2 mendefinisikan bagaimana konvolusi dilakukan pada input dengan filter tertentu dan ditambahkan bias, lalu diterapkan fungsi aktivasi. Filter $K^{(n)}$ diterapkan pada input a^{l-1} dengan penjumlahan hasil perkalian elemen, ditambah bias $b_n^{[l]}$ dan kemudian diterapkan fungsi aktivasi $g^{[l]}$. Persamaan 2.3 menyatakan bahwa output dari layer ke-l ($a^{[l]}$) adalah hasil dari fungsi aktivasi $g^{[l]}$ yang diterapkan pada setiap hasil konvolusi dari input a^{l-1} dengan masing-masing filter $K^{(n)}$.

Dengan demikian, setiap langkah dalam proses konvolusi dijelaskan oleh rumus-rumus yang telah dijabarkan. Secara sederhana persamaan 2.1, 2.2, dan 2.3 menunjukkan bagaimana input diproses melalui beberapa filter, ditambahkan bias, dan diterapkan fungsi aktivasi untuk menghasilkan output pada *layer* berikutnya. Operasi ini adalah dasar dari cara kerja jaringan

saraf konvolusi (CNN) dalam memproses dan mengekstrak fitur dari gambar input. Ilustrasi proses pada convolutional layer divisualisasikan pada Gambar 2.9.



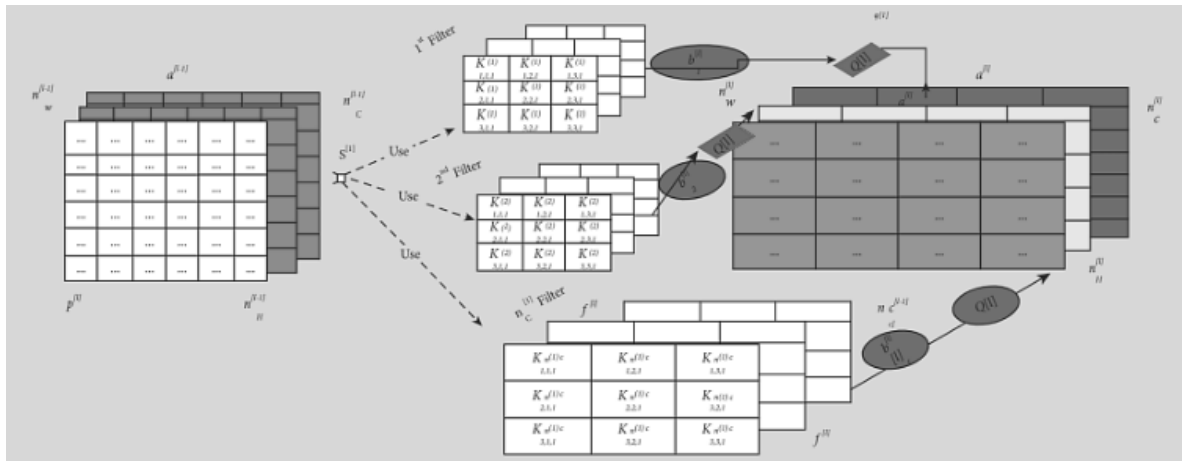
Gambar 2.9 Convolutional Layer

Sumber: Saleem M (2022)

2.11.2 Pooling Layer

Setelah operasi konvolusi, lapisan berikutnya dalam arsitektur CNN adalah *pooling*. Proses ini ditampilkan pada Gambar 2.10. Lapisan ini melakukan *downsampling*. Tugasnya adalah untuk memperkecil informasi yang dikumpulkan dari lapisan konvolusi dari setiap fitur dan menjaga informasi penting. Pada saat yang sama, *input* dimasukkan ke lapisan *pooling*, notasi pada proses *pooling layer* dijelaskan dengan proses dibawah ini

- Input: $a^{[l-1]}$ dengan ukuran $(n_H^{[l-1]}, n_W^{[l-1]}, n_C^{[l-1]})$, di mana a^0 adalah input gambar
- Padding: $p^{[l]}$, Stride: $S^{[l]}$
- Ukuran *filter pooling*: $f^{[l]}$
- Fungsi *pooling*: $\phi^{[l]}$
- Output: $a^{[l]}$ dengan ukuran $(n_H^{[l]}, n_W^{[l]}, n_C^{[l]}) = n_C^{[l-1]}$

Gambar 2.10 *Pooling Layer*

Sumber: Saleem M (2022)

2.11.3 Fully Connected Layer

Fully connected layer adalah komponen penting dari jaringan saraf konvolusi (CNN). Proses ini diilustrasikan pada Gambar 2.11. *Layer* ini sangat efektif dalam tugas klasifikasi gambar. Lapisan ini menghubungkan setiap *neuron* di satu lapisan ke setiap *neuron* di lapisan berikutnya, mirip dengan jaringan saraf tiruan tradisional. *Fully connected layer* berfungsi untuk menggabungkan fitur yang diekstraksi oleh lapisan konvolusi dan *pooling*, lalu memprosesnya untuk menghasilkan output akhir biasanya berupa klasifikasi. Proses matematis dalam lapisan *fully connected layer* dijabarkan berikut ini:

a. Menghitung *Node* ke-*j*

Setiap *node* di lapisan *fully connected* dihitung dengan kombinasi linear dari output lapisan sebelumnya:

$$z_j^{[l]} = \sum_{i=0}^{n_l-1} w_{ij}^{[l]} a_i^{[l-1]} + b_j^{[l]} \quad (2.4)$$

Di mana:

- $z_j^{[l]}$ adalah hasil kombinasi linear untuk *node* ke-*j*.
- $w_{ij}^{[l]}$ adalah bobot antara *neuron* ke-*i* pada lapisan sebelumnya dan *neuron* ke-*j*.
- $a_i^{[l-1]}$ adalah output dari *neuron* ke-*i* pada lapisan sebelumnya.

- $b_j^{[l]}$ adalah bias untuk *neuron* ke-j.

b. Hasil kombinasi linear kemudian diterapkan fungsi aktivasi untuk menghasilkan output akhir dari *node* tersebut:

$$a_j^{[l]} = \varphi^{[l]}(z_j^{[l]}) \quad (2.5)$$

Di mana:

- $a_j^{[l]}$ adalah output setelah fungsi aktivasi diterapkan.
- $\varphi^{[l]}$ adalah fungsi aktivasi yang digunakan.

c. Dimensi Input

Input ke lapisan *fully connected* merupakan hasil dari operasi konvolusi atau *pooling* dengan dimensi:

$$(n_H^{[l-1]}, n_W^{[l-1]}, n_C^{[l-1]}), \quad (2.6)$$

Seperti dinyatakan dalam persamaan 2.5:

$$\text{Dim}(\text{Conv}(a^{l-1}, K^{(n)})) = (n_H^{[l]}, n_W^{[l]}) \quad (2.7)$$

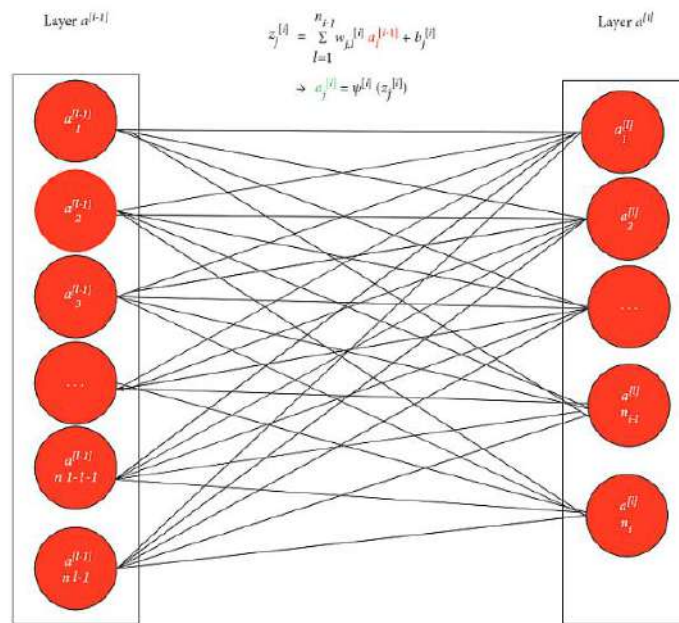
d. *Flatten*

Untuk *input* ke dalam *fully connected layer*, tensor dari lapisan sebelumnya perlu diratakan menjadi *vector* satu dimensi:

$$n_H^{[l-1]} \cdot n_W^{[l-1]} \cdot n_C^{[l-1]} \quad (2.8)$$

Pada lapisan ini ada dua jenis parameter yaitu:

- Bobot (*Weights*) : w_{ij} dengan $n_{l-1} \times n_l$ parameter.
- Bias : dengan n_l parameter.

Gambar 2.11 *Fully Connected Layer*

Sumber: Saleem M (2022)

2.12 Model DeepLabV3

DeepLabv3 adalah model yang dirancang untuk *semantic segmentation* dengan pendekatan utama *atrous convolution* atau dikenal juga sebagai *dilated convolution* (L.-C. Chen et al., 2017). Pendekatan ini memungkinkan model untuk menangkap konteks multi-skala melalui penggunaan berbagai tingkat dilatasi dalam filter konvolusi (L.-C. Chen et al., 2017). Model ini mengatasi dua tantangan utama dalam segmentasi semantik: resolusi fitur yang menurun akibat pooling atau convolution striding, dan kebutuhan untuk mengenali objek pada berbagai skala.

DeepLabv3 menggunakan Atrous Spatial Pyramid Pooling (ASPP) sebagai inti pengolahannya. Modul ASPP memanfaatkan konvolusi *atrous* dengan berbagai tingkat dilatasi untuk menangkap informasi pada skala yang berbeda, dilengkapi dengan fitur tingkat global melalui pooling rata-rata global. Dengan pendekatan ini, ASPP mampu menyajikan field-of-view yang efektif untuk mendeteksi objek di berbagai skala tanpa memerlukan peningkatan jumlah parameter atau sumber daya komputasi yang signifikan (L.-C. Chen et al., 2017). Selain itu, model ini mengadopsi ResNet yang telah dilatih pada *dataset* ImageNet sebagai *backbone*, dengan mengganti beberapa operasi *downsampling* dengan konvolusi *atrous*. Ini memungkinkan model menghasilkan peta fitur yang lebih padat tanpa kehilangan informasi spasial yang penting. Konsep *output stride* juga diterapkan untuk mengontrol resolusi peta

fitur, dengan nilai yang lebih kecil menghasilkan prediksi yang lebih halus namun membutuhkan sumber daya komputasi lebih besar (L.-C. Chen et al., 2017).

DeepLabv3 telah dievaluasi pada beberapa *dataset benchmark* seperti PASCAL VOC 2012 dan Cityscapes. Pada *dataset* PASCAL VOC 2012, model ini mencapai mIoU sebesar 85.7% tanpa menggunakan post-processing DenseCRF. Hasil ini menunjukkan peningkatan yang signifikan dibandingkan versi sebelumnya, DeepLabv2, terutama karena perbaikan dalam penyesuaian parameter *batch normalization* dan pengayaan informasi kontekstual melalui ASPP (L.-C. Chen et al., 2017).

2.13 Model DeepLabV3+

DeepLabv3+ merupakan model yang dirancang untuk segmentasi gambar semantik dengan memadukan dua pendekatan utama dalam Deep Learning yaitu *spatial pyramid pooling* seperti Gambar 2.12 dan struktur *encoder-decoder*. Model ini dikembangkan untuk memperbaiki hasil segmentasi terutama pada batas-batas objek. Pendekatan ini menambahkan modul *decoder* sederhana namun efektif pada DeepLabv3 yang sudah ada sebelumnya (L.-C. Chen et al., 2018). Segmentasi gambar semantik bertujuan untuk memberikan label semantik pada setiap piksel dalam gambar. Pendekatan ini adalah salah satu tugas utama dalam visi komputer yang telah mengalami peningkatan signifikan berkat penggunaan *convolution neural network* (CNN).

DeepLabv3+ adalah hasil pembaharuan DeepLabv3 dengan menambahkan modul *decoder* yang bertujuan untuk memulihkan detail batas objek. Model ini menggabungkan informasi semantik yang dikodekan dalam modul *encoder* dengan kemampuan modul *decoder* untuk menghasilkan segmentasi lebih halus dan detail. *Atrous convolution* memungkinkan kontrol resolusi fitur yang dihasilkan. Resolusi ini dapat disesuaikan dengan sumber daya komputasi yang tersedia (L.-C. Chen et al., 2018). Dengan demikian, model ini dapat berfungsi secara efektif pada berbagai skala dalam penerapannya dibandingkan model lain.

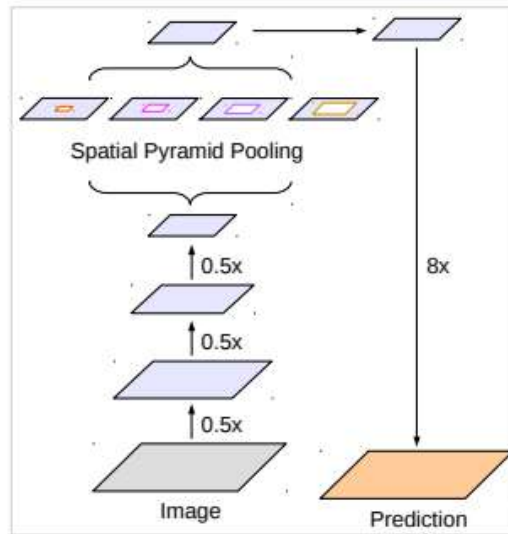
Untuk meningkatkan kecepatan dan efisiensi komputasi, DeepLabv3+ menerapkan *depthwise separable convolution* yang mengurangi kompleksitas komputasi dan jumlah parameter jaringan tanpa mengorbankan akurasi. Pada modul *Atrous Spatial Pyramid Pooling* (ASPP) dan modul *decoder*, teknik ini diterapkan untuk meningkatkan performa jaringan (L.-C. Chen et al., 2018). Model ini menggunakan DeepLabv3 sebagai modul *encoder* yang menghasilkan *feature map* dengan resolusi lebih rendah tetapi kaya akan informasi semantik. Modul *decoder* kemudian meningkatkan resolusi *feature map* ini melalui *upsampling* bilinear

dan menggabungkannya dengan fitur level rendah dari *backbone* jaringan. Hasil akhir adalah segmentasi lebih detail dan akurat (L.-C. Chen et al., 2018).

Cara kerja yang lebih spesifik untuk DeepLabv3+ dalam penerapan Atrous Convolution terbagi dua bagian utama, yaitu pada *encoder* dan *decoder*. Pada bagian *encoder*, DeepLabv3+ menggunakan Atrous Convolution dengan berbagai *dilation rates* dalam modul Atrous Spatial Pyramid Pooling (ASPP) untuk menangkap informasi kontekstual dari berbagai skala seperti yang ditunjukkan pada Gambar 2.15. Dengan metode ini, model dapat mempertahankan resolusi tinggi tanpa harus menggunakan operasi pooling yang dapat menyebabkan hilangnya detail spasial (L.-C. Chen et al., 2018). *Output stride* dalam *encoder* dapat dikontrol, biasanya dengan nilai 8 atau 16, sehingga fitur yang dihasilkan tetap padat dan informatif tanpa meningkatkan jumlah parameter atau beban komputasi secara signifikan (L.-C. Chen et al., 2018).

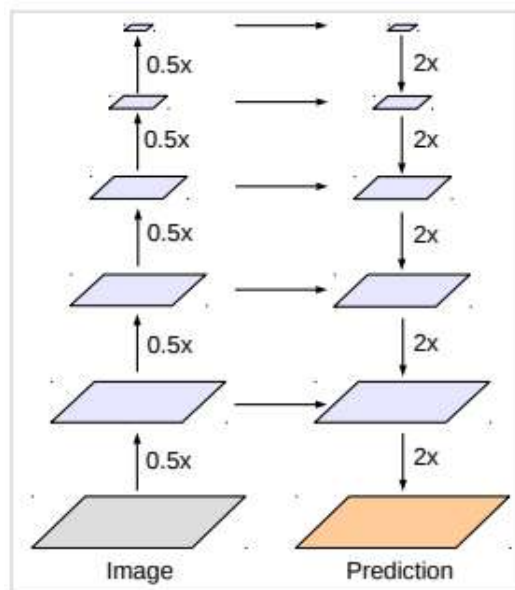
Setelah fitur diproses oleh *encoder*, *decoder* dalam DeepLabv3+ digunakan untuk mengembalikan detail batas objek yang sering hilang akibat konvolusi dengan *stride*. Proses ini dapat dilihat pada Gambar 2.15. *Decoder* ini bekerja dengan mengambil kembali *low-level feature* dari jaringan *backbone* seperti ResNet-101. Setelah itu dilanjutkan membantu meningkatkan akurasi prediksi di sekitar batas objek (L.-C. Chen et al., 2018). Dengan demikian, kombinasi Atrous Convolution dalam *encoder* serta mekanisme *decoding* yang lebih efektif di DeepLabv3+ memungkinkan segmentasi semantik yang lebih akurat dengan mempertahankan detail objek dalam gambar (L.-C. Chen et al., 2018).

DeepLabv3+ telah dievaluasi pada beberapa *dataset benchmark* seperti PASCAL VOC 2012 dan Cityscapes, DeepLabv3+ mampu menunjukkan peningkatan kinerja yang signifikan. Model ini mencapai akurasi 89.0% pada *dataset* PASCAL VOC 2012 dan 82.1% pada *dataset* Cityscapes. Selain itu, modifikasi yang dilakukan pada model Xception, seperti penggantian *max pooling* dengan *depthwise separable convolution* turut berkontribusi pada peningkatan kinerja model tersebut (L.-C. Chen et al., 2018). Gambar 2.15 menampilkan arsitektur model DeepLabv3+ dan juga proses untuk menghasilkan label prediksi.



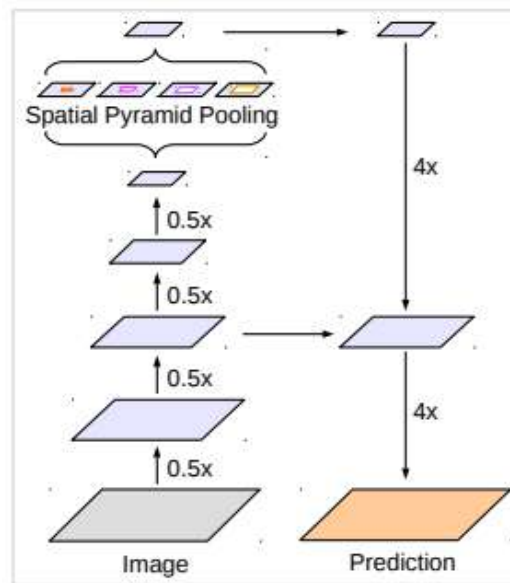
Gambar 2.12 *Spatial Pyramid Pooling*

Sumber: Chen L (2018)



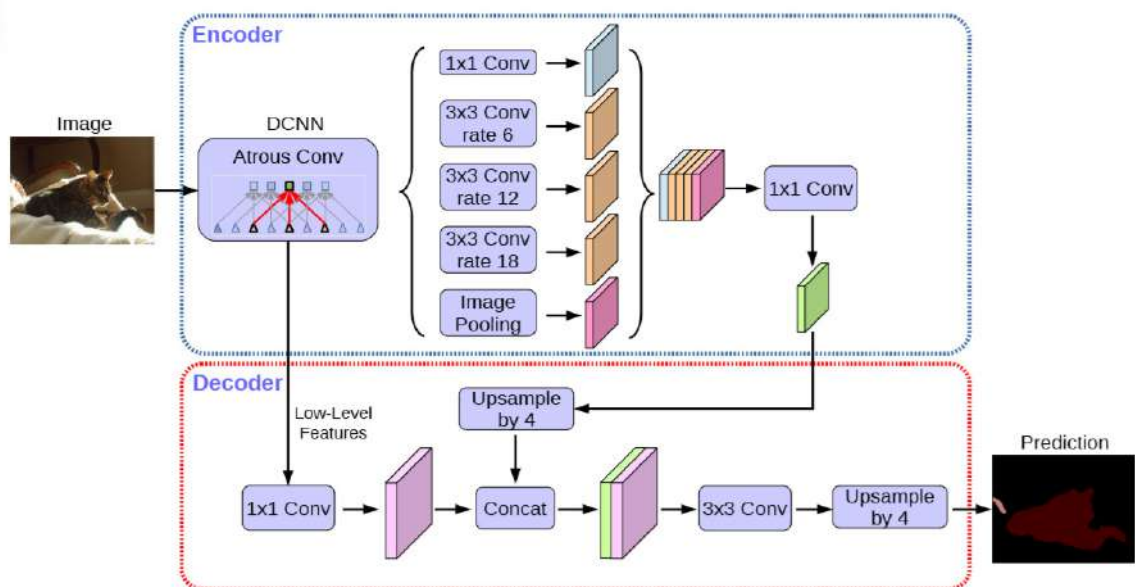
Gambar 2.13 *Encoder-Decoder*

Sumber: Chen L (2018)



Gambar 2.14 *Encoder-Decoder with Spatial Pyramid Pooling*

Sumber: Chen L (2018)



Gambar 2.15 Arsitektur model DeepLabV3+ dengan proses menghasilkan prediksi

Sumber : Chen L (2018)

Gambar 2.15 menunjukkan arsitektur DeepLabv3+, yang merupakan pengembangan dari DeepLabv3 dengan menambahkan modul decoder untuk meningkatkan segmentasi, terutama pada batas-batas objek. Konsep utama dari *encoder* dan *decoder* adalah mengekstraksi fitur-fitur dari gambar menggunakan *encoder*, kemudian *decoder* merekonstruksi peta segmentasi dengan ukuran yang sama seperti gambar asli.

Encoder pada DeepLabv3+ menggunakan Atrous Spatial Pyramid Pooling (ASPP) memanfaatkan *atrous convolution* dengan tingkat dilasi berbeda (rate 6, 12, dan 18) untuk menangkap informasi kontekstual dari berbagai skala, ditambah *pooling* gambar untuk menyediakan informasi global. Convolution 1x1 digunakan untuk menggabungkan informasi dari berbagai tingkat dilasi dan *pooling*.

Selain itu, DeepLabv3+ memanfaatkan *low level feature* dari *backbone*, seperti lapisan awal pada ResNet yang diambil sebelum dilakukan *downsampling*. Hal ini dapat mempertahankan detail spasial yang tinggi. Modul *decoder* kemudian menerima keluaran dari *encoder* (ASPP) dan *low level feature*, di mana *low level feature* ini diproses melalui *convolution* 1x1 untuk mengurangi dimensi saluran sebelum digabungkan (*concatenate*) dengan fitur dari ASPP. Setelah penggabungan, dilakukan beberapa *convolution* 3x3 untuk menyempurnakan hasil segmentasi, diikuti oleh *upsampling* bilinear dua kali untuk mengembalikan resolusi gambar ke ukuran aslinya.

Pendekatan ini memungkinkan DeepLabv3+ menghasilkan segmentasi semantik yang lebih baik terutama batas tepian pada sekitar objek. Dengan penggunaan metode penggabungan fitur dari *spatial pyramid pooling* untuk menangkap informasi multi-skala dan struktur *encoder-decoder* untuk memulihkan detail batas objek, menjadikannya lebih unggul dibandingkan DeepLabv3 dalam berbagai tugas segmentasi.

2.14 Perbedaan DeepLabV3 dan DeepLabV3+

DeepLabv3+ adalah pengembangan dari DeepLabv3 yang menggabungkan keunggulan pendekatan *spatial pyramid pooling* dan struktur *encoder-decoder* untuk segmentasi semantik. DeepLabv3 menggunakan modul Atrous Spatial Pyramid Pooling (ASPP) untuk menangkap informasi kontekstual multi-skala, tetapi tidak memiliki modul *decoder* sehingga hasil segmentasi pada batas objek kurang optimal. Sebaliknya, DeepLabv3+ menambahkan modul *decoder* sederhana yang dirancang untuk memulihkan detail pada batas objek, meningkatkan kualitas segmentasi secara keseluruhan (L.-C. Chen et al., 2018).

Perbedaan utama lainnya adalah penggunaan *depthwise separable convolution* pada DeepLabv3+. Pendekatan ini diterapkan pada modul ASPP dan *decoder* untuk mengurangi kompleksitas komputasi tanpa mengorbankan akurasi, sehingga membuat model lebih efisien dibandingkan DeepLabv3. Selain itu, DeepLabv3+ memungkinkan pengaturan resolusi fitur yang dihasilkan oleh *encoder* melalui *atrous convolution*, memberikan fleksibilitas untuk menyeimbangkan antara presisi dan kebutuhan komputasi (L.-C. Chen et al., 2018).

Evaluasi menunjukkan bahwa DeepLabv3+ memberikan peningkatan akurasi yang signifikan dibandingkan DeepLabv3, terutama pada dataset seperti PASCAL VOC 2012 dan Cityscapes. Hal ini membuktikan bahwa penambahan modul *decoder* dan penggunaan *depthwise separable convolution* memberikan kontribusi besar dalam meningkatkan performa segmentasi, terutama di area batas objek yang lebih tajam dan detail (L.-C. Chen et al., 2018).

2.15 Transfer Learning

Dalam penelitian yang dilakukan oleh Liu et al. (2022), metode *transfer learning* digunakan untuk melatih jaringan segmentasi daun tanaman menggunakan MobileNetV2-DeepLabV3+. *Transfer learning* adalah teknik memanfaatkan model yang sudah dilatih pada tugas tertentu dan kemudian menyesuaikannya untuk tugas berbeda namun terkait. Dalam studi ini, gambar RGB berukuran 512x512 piksel dimasukkan ke dalam model untuk pelatihan menggunakan metode *transfer learning*. Hasil segmentasi menunjukkan bahwa jaringan MobileNetV2-DeepLabV3+ dapat melakukan segmentasi vena daun dengan akurat (Liu et al., 2022).

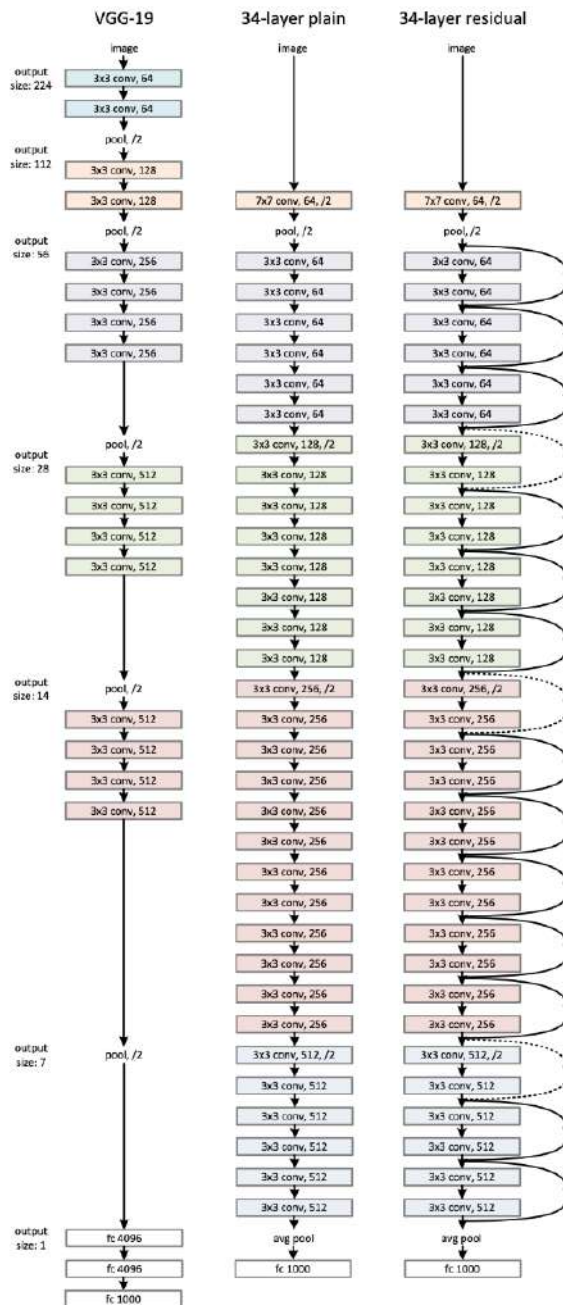
Transfer learning membantu mengurangi waktu pelatihan dan jumlah data yang diperlukan untuk melatih model baru, karena model memulai pelatihan dari titik yang sudah dioptimalkan. Dalam konteks ini misalnya MobileNetV2 sudah dilatih sebelumnya digunakan sebagai jaringan ekstraksi fitur, yang kemudian diadaptasi untuk tugas segmentasi vena daun dengan menggunakan struktur ASPP untuk memperoleh informasi fitur semantik lebih dalam (Liu et al., 2022).

Transfer learning berikutnya ada penggunaan *residual networks*. ResNet (Residual Network) adalah arsitektur jaringan saraf tiruan yang dirancang untuk mengatasi masalah degradasi akurasi (He et al., 2015). ResNet hadir dalam berbagai varian seperti ResNet-34, 50, 101, dan 152, dan versi lebih *advance* seperti penggunaan desain *bottleneck* untuk efisiensi parameter. Penamaan angka pada ResNet mewakili jumlah *layer* pada arsitektur tersebut. ResNet mampu menghasilkan peningkatan akurasi signifikan pada tugas klasifikasi besar seperti ImageNet (He et al., 2015).

2.16 Residual Network (ResNet)

ResNet, atau Residual Network, adalah arsitektur jaringan yang dirancang untuk mengatasi masalah degradasi performa pada Neural Network (He et al., 2015). Inti dari ResNet adalah *residual block*, di mana setiap blok menggunakan *skip connection* yang langsung

menghubungkan *input* awal ke output blok tersebut. Pendekatan ini memungkinkan jaringan untuk mempelajari fungsi residual, yaitu perbedaan antara input dan keluaran yang diinginkan, sehingga membuat optimasi jaringan menjadi lebih mudah (He et al., 2015).



Gambar 2.16 ResNet *architecture*

Sumber : He (2015)

Pada Gambar 2.16, *skip connection* ditunjukkan dengan panah hitam melengkung yang menghubungkan beberapa blok konvolusi. Mekanisme ini memungkinkan output dari satu

lapisan sebelumnya langsung diteruskan dan dijumlahkan dengan *output* dari lapisan yang lebih dalam, sehingga metode ini dapat mempertahankan informasi penting dan mempercepat proses konvergensi.

ResNet juga dirancang untuk mendukung kedalaman yang jauh lebih besar dibandingkan model lain seperti VGG atau GoogLeNet, tanpa mengalami penurunan akurasi yang sering terjadi pada *backbone* lain. Dengan desain ini, ResNet dapat memanfaatkan *skip connection* untuk mengurangi kompleksitas parameter sekaligus meningkatkan efisiensi komputasi. Selain itu, ResNet memiliki kemampuan generalisasi yang sangat baik, yang membuatnya tidak hanya unggul dalam tugas klasifikasi seperti ImageNet, tetapi juga pada tugas lain seperti deteksi objek dan segmentasi semantik. Desainnya yang modular dan fleksibel menjadikan ResNet sebagai salah satu *backbone* arsitektur yang paling sering digunakan dalam berbagai aplikasi Deep Learning (He et al., 2015).

2.17 Hyperparameter

Hyperparameter adalah elemen penting dalam Deep Learning yang mempengaruhi kinerja dan efisiensi model. Fungsi utama *hyperparameter* adalah untuk mengontrol proses pelatihan model dan membantu mengoptimalkan kinerja model. Pengaturan *hyperparameter* yang tepat dapat meningkatkan akurasi model, mempercepat konvergensi pelatihan, dan mencegah *overfitting* atau *underfitting* (Xiao et al., 2020). *Hyperparameter* mencakup berbagai elemen dalam arsitektur dan proses pelatihan model Deep Learning, seperti *learning rate*, *batch size*, jumlah *epochs*, *optimizer*, *weight decay*, dan momentum (Bartz-Beielstein et al., 2021).

Learning rate menentukan seberapa besar perubahan yang dilakukan pada bobot model di setiap iterasi pembaruan, sementara *batch size* adalah jumlah sampel data yang diproses sebelum model diperbarui. *Epochs* merujuk pada satu siklus penuh melalui seluruh *dataset* pelatihan. *Optimizer* adalah algoritma yang digunakan untuk mengubah atribut model, seperti bobot dan bias, untuk mengurangi kesalahan (Amboage et al., 2023). Contoh *optimizer* termasuk Stochastic Gradient Descent (SGD), Adam, RMSProp, dan AdaGrad. Momentum membantu percepatan konvergensi jaringan menuju minima dengan menambahkan fraksi dari pembaruan sebelumnya ke pembaruan terbaru (Zaim Awang Pon et al., 2021).

Dalam model DeepLabv3+, beberapa *hyperparameter* diatur termasuk *learning rate*, *batch size*, jumlah *epochs*, *optimizer*, dan *weight decay*. Sebagai contoh, model DeepLabv3+ dapat menggunakan *learning rate* awal sebesar 0.001 dengan *scheduler* untuk menyesuaikan

learning rate selama pelatihan. Scheduler bekerja dengan menurunkan *learning rate* berdasarkan aturan tertentu, misalnya pada StepLR, *learning rate* diturunkan secara bertahap setiap beberapa epoch dengan mengalikan nilai awal *learning rate* dengan faktor tertentu (γ). *Batch size* diatur berdasarkan kapasitas memori GPU yang tersedia dan nilainya tergantung seberapa kuat hardware pada komputer, serta Adam *optimizer* dengan parameter tertentu untuk mengatur momentum (Kiran & Ozyildirim, 2022).

Pengaturan *hyperparameter* yang tepat sangat penting untuk mencapai kinerja optimal dari model jaringan saraf tiruan. Metode seperti *grid search*, *random search*, *bayesian optimization*, dan algoritma genetika sering digunakan untuk menemukan kombinasi *hyperparameter* terbaik. *Hyperparameter* yang dioptimalkan dengan baik dapat menghasilkan model lebih akurat, lebih efisien, dan lebih andal dalam aplikasi dunia nyata (Kadra et al., 2023).

Dropout dan *weight decay* adalah dua pendekatan umum diterapkan untuk mengatasi *overfitting* dalam pelatihan jaringan saraf. *Dropout* berfungsi dengan menonaktifkan secara acak sejumlah node pada jaringan selama proses pelatihan, mendorong jaringan untuk lebih adaptif dan tidak terlalu mengandalkan node tertentu. Sementara itu, *weight decay* bekerja dengan memberikan penalti pada bobot bernilai besar guna mencegah model menjadi terlalu kompleks. (Helmbold & Long, 2016).

2.18 Cross-entropy Loss

Cross-Entropy (CE) Loss adalah salah satu fungsi kerugian yang paling umum digunakan dalam tugas klasifikasi, termasuk segmentasi citra. Fungsi ini bekerja dengan mengukur perbedaan antara probabilitas yang diprediksi oleh model dan label *ground-truth*. Dalam segmentasi citra, setiap piksel diklasifikasikan ke dalam satu kelas tertentu, dan CE Loss menghitung log-probabilitas dari kelas yang benar untuk setiap piksel. Dengan cara ini, model dilatih untuk meningkatkan probabilitas prediksi terhadap kelas yang benar dan mengurangi probabilitas terhadap kelas yang salah. CE Loss sering digunakan karena sifatnya yang probabilistik, sehingga dapat memberikan keyakinan terhadap prediksi yang dihasilkan oleh model (Galdran et al., 2022).

Keunggulan utama dari CE Loss adalah kemampuannya dalam menangani segmentasi dengan distribusi kelas yang seimbang. Fungsi ini juga lebih umum digunakan dalam segmentasi citra alami karena kemampuannya dalam memberikan pembelajaran yang stabil selama proses optimasi model. Selain itu, penelitian menunjukkan bahwa CE Loss memiliki

performa yang lebih baik dibandingkan fungsi kerugian lainnya dalam menghadapi data yang berbeda dari distribusi pelatihan, atau yang dikenal sebagai Out-of-Distribution (OoD) data. Hal ini menjadikannya pilihan yang lebih andal dalam skenario di mana model harus bekerja dengan gambar yang memiliki variasi besar atau kondisi yang tidak selalu sesuai dengan data pelatihan (Galdran et al., 2022).

Dalam implementasi teknisnya, CE Loss bekerja dengan membandingkan probabilitas yang dihasilkan oleh model dengan label ground-truth yang dikodekan dalam bentuk one-hot encoding. Pada segmentasi biner, CE Loss dihitung dengan mengambil log dari probabilitas kelas foreground dan background untuk setiap piksel, lalu mengoptimalkan model agar nilai probabilitasnya semakin mendekati label yang benar. Dengan pendekatan ini, CE Loss tidak hanya memberikan hasil yang lebih stabil, tetapi juga membantu model untuk lebih mudah dikalibrasi dalam berbagai kondisi segmentasi citra. Oleh karena itu, pemilihan CE Loss sebagai fungsi kerugian utama dalam segmentasi citra dapat menjadi pilihan yang efektif, terutama dalam kasus di mana distribusi kelas relatif seimbang dan model harus mampu beradaptasi dengan variasi data yang luas (Galdran et al., 2022).

2.19 Dice Loss

Dice Loss adalah salah satu fungsi kerugian yang digunakan dalam segmentasi citra untuk mengoptimalkan kesamaan antara hasil prediksi dan label *ground-truth*. Fungsi ini didasarkan pada Dice Similarity Coefficient (DSC), yang mengukur tingkat *overlap* antara area yang diprediksi sebagai objek dengan area yang sebenarnya dalam citra. Berbeda dengan metode lain yang mengevaluasi piksel secara individual, Dice Loss mempertimbangkan hubungan antara keseluruhan area yang diprediksi dan area ground-truth, sehingga lebih efektif dalam menangani dataset yang memiliki ketidakseimbangan kelas, di mana jumlah piksel latar belakang jauh lebih dominan dibandingkan objek yang disegmentasi (Galdran et al., 2022).

Salah satu keunggulan utama Dice Loss adalah kemampuannya dalam meningkatkan akurasi segmentasi pada kelas dengan jumlah piksel yang lebih sedikit. Karena Dice Loss lebih menekankan pada *overlap* antara prediksi dan label, metode ini mampu mengurangi dampak dari dominasi kelas mayoritas dalam perhitungan kesalahan model. Hal ini sangat berguna dalam segmentasi citra medis dan kasus lain di mana objek yang ingin dideteksi cenderung berukuran kecil dibandingkan keseluruhan gambar. Namun, meskipun memiliki keunggulan dalam menangani ketidakseimbangan kelas, Dice Loss memiliki beberapa kelemahan yang perlu diperhatikan (Galdran et al., 2022).

Salah satu keterbatasan Dice Loss adalah instabilitas numerik yang dapat terjadi ketika tidak ada piksel foreground dalam label ground-truth. Dalam kondisi ini, fungsi kerugian dapat mengalami kesulitan dalam memberikan umpan balik yang stabil kepada model selama proses pelatihan. Selain itu, Dice Loss tidak selalu menghasilkan probabilitas prediksi yang terkalibrasi dengan baik, sehingga model dapat menjadi terlalu percaya diri terhadap hasil segmentasi yang dihasilkan. Oleh karena itu, meskipun Dice Loss banyak digunakan dalam segmentasi citra, dalam beberapa kasus metode ini sering dikombinasikan dengan fungsi kerugian lain untuk mengatasi keterbatasannya dan meningkatkan stabilitas pelatihan model (Galdran et al., 2022).

2.20 Data Augmentation

Data augmentation telah menjadi komponen penting dalam pelatihan model Deep Learning untuk tugas-tugas *computer vision* seperti segmentasi semantik, klasifikasi citra, dan deteksi objek. Augmentasi bertujuan untuk menambah *dataset size* yang kurang banyak atau tidak memadai. Teknik ini berpengaruh untuk mencegah model mengalami *overfitting* dan meningkatkan generalisasi dari model. Menurut Yang et al. (2023) metode augmentasi data dapat dikelompokkan menjadi metode dasar dan lanjutan. Metode dasar mencakup manipulasi citra seperti rotasi, *flipping*, *scale*, dan translasi, serta metode penghapusan citra seperti *cutout* dan *random erasing*. Selain itu, metode pencampuran citra, seperti *mixup* dan *cutmix*, menggabungkan beberapa citra untuk meningkatkan variasi data.

Untuk metode lanjutan, *AutoAugment* menggunakan pembelajaran otomatis untuk menemukan teknik augmentasi terbaik, sedangkan model generatif digunakan untuk membuat data sintetis menyerupai data asli. Hasil penelitian menunjukkan bahwa augmentasi data secara konsisten meningkatkan performa model dalam berbagai tugas *computer vision* dengan meningkatkan metrik seperti *Intersection over Union* (IoU) dan akurasi.

2.21 Performance Measurement

Pengukuran kinerja dalam Deep Learning adalah aspek krusial untuk menentukan seberapa baik model berfungsi dan seberapa efisien model tersebut dalam berbagai tugas. *Loss*, *confusion matrix* seperti pada Gambar 2.17, *precision*, *recall*, dan *F1 score* sering digunakan untuk mengukur kinerja model dalam tugas klasifikasi multikelas. *Confusion matrix*

memberikan gambaran tentang klasifikasi benar dan salah yang dilakukan oleh model, sementara presisi (*precision*) mengukur ketepatan prediksi positif model. *Recall* mengukur seberapa baik model dalam menemukan semua instance positif, dan *F1 Score* adalah rata-rata harmonik dari presisi dan *recall*, *sscore* ini memberikan gambaran keseimbangan antara keduanya.

Dalam semantic segmentation pengukuran untuk IoU juga sangat penting. IoU (*Intersection over Union*), atau dikenal juga sebagai *Jaccard Index*, adalah metrik yang sering digunakan untuk mengukur tingkat kesamaan antara dua bentuk. Metrik ini memperhitungkan karakteristik seperti lebar, tinggi, dan posisi objek, lalu menghitung perbandingan area yang tumpang tindih dengan keseluruhan area. Karena sifatnya yang tidak bergantung pada skala, IoU menjadi alat evaluasi yang populer untuk tugas seperti segmentasi, dan deteksi objek (Rezatofighi et al., 2019). Persamaan IoU dijelaskan pada persamaan 2.9. Kemudian, untuk menghitung mIoU caranya dengan mengambil nilai rata-rata IoU untuk semua kelas dalam *dataset* (Wang et al., 2023). Rumus mIoU dapat dilihat pada persamaan 2.10.

$$IoU = \frac{|A \cap B|}{|A \cup B|} \quad (2.9)$$

$$mIoU = \frac{1}{C} \sum_{c=1}^C IoU_c \quad (2.10)$$

Di mana :

- C adalah jumlah kelas dalam *datases*
- $mIoU_c$ adalah nilai IoU untuk kelas ke- c

Menurut D. Chen et al. (2022), tingkat kerugian pelatihan atau *training loss rate* adalah salah satu dari lima indikator kinerja yang digunakan untuk menilai arsitektur Deep Learning. *Training loss* menunjukkan seberapa baik model memprediksi sampel individu dengan benar selama proses pelatihan. Jika model memprediksi dengan akurat, nilai kerugiannya akan mendekati nol, dan jika tidak, nilai kerugiannya akan lebih besar. Fungsi kerugian umum digunakan dalam klasifikasi multikelas adalah *cross-entropy loss*, diukur sebagai tingkat ketidakpastian. *Cross-entropy loss* dihitung dengan persamaan 2.11.

$$Loss = - \sum_{c=1}^M y_{o,c} \log(p_{o,c}) \quad (2.11)$$

Dalam persamaan 2.11 c adalah jumlah kelas, y adalah indikator biner (0 atau 1) apakah kelas label c adalah klasifikasi yang benar untuk pengamatan o, dan p adalah probabilitas yang diprediksi bahwa pengamatan o termasuk dalam kelas c.

D. Chen et al. (2022) menekankan pentingnya indikator kinerja seperti tingkat kerugian pelatihan (*training loss*), robustitas, waktu pelatihan, jumlah parameter model, dan kompleksitas komputasi dalam mengevaluasi arsitektur Deep Learning. Tingkat kerugian pelatihan adalah ukuran kesalahan model pada data pelatihan, hal ini membantu dalam memahami seberapa baik model belajar dari data tersebut. Robustitas mengacu pada kemampuan model untuk mempertahankan kinerjanya di berbagai kondisi, termasuk data yang berubah-ubah. Waktu pelatihan adalah durasi yang diperlukan untuk melatih model hingga konvergen. Metrik ini berkaitan dengan efisiensi komputasi. Jumlah parameter model menentukan ukuran dan kapasitas model, sementara kompleksitas komputasi mengukur sumber daya komputasi yang dibutuhkan untuk melatih dan menjalankan model.

Persamaan 2.12 adalah formula untuk menghitung akurasi dalam konteks klasifikasi biner. Akurasi adalah metrik yang mengukur seberapa sering model prediksi benar dibandingkan dengan semua prediksi secara keseluruhan. Penelitian D. Chen et al. (2022) menyatakan TP (*True Positives*) adalah jumlah prediksi positif yang benar, yaitu jumlah kasus di mana model memprediksi kelas positif dan benar-benar positif. TN (*True Negatives*) adalah jumlah prediksi negatif yang benar, yaitu jumlah kasus di mana model memprediksi kelas negatif dan benar-benar negatif. FP (*False Positives*) adalah jumlah prediksi positif yang salah, yaitu jumlah kasus di mana model memprediksi kelas positif tetapi sebenarnya negatif. FN (*False Negatives*) adalah jumlah prediksi negatif yang salah, yaitu jumlah kasus di mana model memprediksi kelas negatif tetapi sebenarnya positif.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (2.12)$$

Recall pada persamaan 2.13 juga dikenal sebagai sensitivitas atau *true positive rate*, persamaan ini mengukur seberapa baik model dalam menemukan semua *instance* positif yang ada (Ekong et al., 2022). Di mana TP (*True Positives*) adalah jumlah prediksi positif yang

benar, dan FN (*False Negatives*) adalah jumlah prediksi negatif yang salah. *Recall* menunjukkan proporsi kasus positif yang benar-benar terdeteksi oleh model.

$$Recall = \frac{TP}{TP + FN} \quad (2.13)$$

Merujuk persamaan 2.14 presisi atau *precision*, adalah metrik kinerja dalam klasifikasi yang mengukur ketepatan prediksi model (Ekong et al., 2022). Presisi menghitung proporsi prediksi positif yang benar-benar positif (True Positives atau TP) dari semua prediksi positif yang dibuat oleh model, baik yang benar (TP) maupun yang salah (*False Positives* atau FP).

$$Precision = \frac{TP}{TP + FP} \quad (2.14)$$

F1-score pada persamaan 2.15 adalah metrik kinerja yang menggabungkan presisi dan *recall* menjadi satu nilai Tunggal. Metrik ini memberikan gambaran menyeluruh tentang kinerja model (Ekong et al., 2022). *F1-score* dihitung sebagai rata-rata harmonis dari presisi dan *recall*. F1-score digunakan untuk memberikan keseimbangan antara presisi dan *recall*, khususnya ketika ada ketidakseimbangan kelas dalam *dataset*. Nilai F1-score tinggi menunjukkan bahwa model memiliki keseimbangan yang baik antara ketepatan (presisi) dan kemampuan menangkap semua instance positif (*recall*).

$$F1\ Score = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \quad (2.15)$$

Mean Average Precision (mAP) adalah metrik evaluasi yang sering digunakan untuk menilai kinerja model (Wood & Chollet, 2022). Metrik ini menghitung rata-rata presisi pada berbagai tingkat *recall* dengan mempertimbangkan semua kelas yang ada dalam *dataset*. Proses perhitungannya diawali dengan menghitung Intersection over Union (IoU). Selanjutnya, presisi dan *recall* dihitung untuk setiap kelas dengan berbagai nilai ambang (*recall thresholds*) yang kemudian digunakan untuk membuat kurva presisi-recall. Luas di bawah kurva tersebut dihitung sebagai *Average Precision* (AP) untuk masing-masing kelas, dan rata-rata dari nilai AP seluruh kelas menghasilkan mAP. Metrik ini memberikan evaluasi yang lebih komprehensif dibandingkan presisi atau recall saja, karena mempertimbangkan tingkat akurasi

prediksi, keseimbangan antara presisi dan recall, serta tingkat overlap (IoU) antara prediksi dan ground truth. Rumus AP ditampilkan pada persamaan 2.16 dan rumus mAP ditampilkan pada persamaan 2.17.

$$AP = \sum_{i=1}^n (R_i + R_{i-1}) \cdot P_i \quad (2.16)$$

Di mana :

- R_i : *Recall* pada titik i
- P_i : *Precision* pada titik i

$$mAP = \frac{1}{N} \sum_{k=1}^N AP_k \quad (2.17)$$

Di mana :

- N : Jumlah kelas
- AP_k : Average Precision untuk kelas k

	Actually Abnormal(1)	Actually Normal(0)
Predicted Abnormal(1)	TP	FN
Predicted Normal(0)	FP	TN

Gambar 2.17 *Confusion matrix*

Sumber: Chen D (2018)

2.22 Studi Literatur

Studi literatur adalah salah satu cara agar memperoleh berbagai pengetahuan mengenai tema yang dibahas. Hal yang dikaji, berkaitan dengan masalah serta solusi yang dihadapi penelitian terdahulu, penggunaan *dataset* pada saat melakukan penelitian, model arsitektur

yang digunakan, serta performa dari arsitektur tersebut. Kajian literatur ini menggunakan 10 literatur nasional dan internasional. Berikut beberapa ulasan literatur mengenai penerapan DeepLabv3+ untuk segmentasi citra satelit menggunakan teknik Deep Learning.

Penelitian oleh (Ovi et al., 2023) mengembangkan arsitektur DeepTriNet berbasis DeepLabv3+ untuk segmentasi semantik citra satelit. Arsitektur ini menggabungkan jaringan *squeeze-and-excitation* (SENet) dan unit *attention* tiga tingkat (TAUs) dengan DeepLabv3+ standar. Hasil penelitian menunjukkan bahwa DeepTriNet mampu mencapai akurasi sebesar 98% dan 77% pada *dataset* Land-Cover.ai dan GID-2, dengan IoU sebesar 80% dan 58%, serta presisi sebesar 88% dan 68%.

Penelitian yang dilakukan Khryashchev et al. (2022) menggunakan arsitektur DeepLabv3+ dengan *encoder* ResNet50 untuk segmentasi citra SAR C-band. Model ini menunjukkan kinerja terbaik dengan nilai *dice* sebesar 0.762 dan *F1-score* sebesar 0.852, mengungguli arsitektur lainnya seperti U-Net dan MPResNet. Sertel et al., (2022) mengevaluasi berbagai arsitektur segmentasi untuk pemetaan penggunaan lahan/area tutupan lahan (LULC) menggunakan citra Worldview-3 resolusi tinggi. Hasil penelitian menunjukkan bahwa arsitektur DeepLabv3+ dengan *encoder* ResNeXt50 mencapai kinerja terbaik dengan IoU sebesar 89.46%, *F1-score* sebesar 94.35%, presisi sebesar 94.25%, dan *recall* sebesar 94.49%.

Penelitian oleh (Fu et al., 2022) menggunakan arsitektur DeepLabv3+ yang telah ditingkatkan dengan MobileNetV2 sebagai *backbone*, modifikasi ini meningkatkan akurasi segmentasi objek tepi dan mengurangi *training loss*. Hasil eksperimen menunjukkan bahwa metode ini mencapai mIoU sebesar 64.76% pada *dataset* WHDL D dan 64.58% pada *dataset* CCF BDCI. (Du et al., 2021) mengintegrasikan DeepLabv3+ dengan analisis citra berbasis objek (OBIA) untuk segmentasi citra VHR, menunjukkan peningkatan akurasi keseluruhan sebesar 90.6% untuk *dataset* Vaihingen dan 85.0% untuk *dataset* Potsdam.

Penelitian oleh Yu et al. (2022) mengembangkan model *Lightweight Complex-Valued* DeepLabv3+ untuk segmentasi citra *dataset* PolSAR. Hasil penelitian menunjukkan bahwa model ini memberikan performa lebih baik pada metrik mIoU dengan nilai 95.52 pada *dataset* Flevoland 1, 96.31 pada *dataset* Flevoland 2, dan 97.01 pada *dataset* San Francisco. Nilai ini lebih tinggi dibandingkan dengan DeepLabv3+ standar pada *dataset* Flevoland dan San Francisco. Penelitian oleh (Akçay et al., 2022) menggunakan model *Dual-Stream* DeepLabv3+ untuk segmentasi citra udara dengan *dataset* Potsdam dan Vaihingen. Hasil penelitian menunjukkan akurasi keseluruhan sebesar 88.87% dan 87.39%.

Penelitian oleh (Zheng, 2020) memodifikasi DeepLabv3+ dengan kernel konvolusi *fractional-order* untuk segmentasi citra pada *dataset* PASCAL VOC2012 menunjukkan mIoU lebih dari 89%. Pada penelitian (W. Li & Zhao, 2022a) mengevaluasi DeepLabv3+ dengan *attention module* untuk segmentasi citra satelit, menunjukkan akurasi segmentasi yang tinggi dengan nilai IoU 75.24 meskipun *dataset* tidak disebutkan secara spesifik. Penelitian oleh Sunandini et al. (2023) mengevaluasi signifikansi ASPP dalam DeepLabv3+ untuk segmentasi badan air pada citra Sentinel-2, menunjukkan peningkatan akurasi segmentasi dengan ASPP dibandingkan tanpa ASPP dengan nilai precision 0.90. Perbandingan dari studi literatur dari penelitian terdahulu disajikan dalam Tabel 2.1.

Tabel 2.1 Perbandingan penelitian terdahulu

Penulis	Dataset	Model	Objek yang Disegmentasi	Jenis Band Citra	Hasil Tes
Ovi et al., (2023)	Land-Cover.ai dan GID-2	DeepTriNet (berbasis DeepLabv3+)	<i>building, woods, water, dan road.</i>	Red, Green, Blue	Akurasi 98% (Land-Cover.ai) dan 77% (GID-2), IoU 80% (Landcover.ai) dan 58% (GID-2), Presisi 88% (Landcover.ai) dan 68%(GID-2)
Khryashchev et al. (2022)	Citra SAR C-band satelit	DeepLabv3+ (ResNet50)	Tidak Diketahui	C-band	Dice 0.762, F1-score 0.852
Sertel et al., 2022	Citra Worldview-3	DeepLabv3+ (ResNeXt50)	<i>Discontinuous urban fabric. Road and rail networks and associated land. Mine, dump, and construction sites, Artificial, non-agricultural vegetated areas, Arable land, Permanent crops, Heterogeneous agricultural areas, Forest, and Inland waters.</i>	Red, Green, Blue, Near-Infrared	IoU 89.46%, F1-score 94.35%, Presisi 94.25%, Recall 94.49%
Fu et al. (2022)	Dataset WHDLD dan CCF BDCI	DeepLabv3+ (MobileNetV2)	<i>building, road, vegetation, water, bare soil, sidewalk, cultivated</i>	Red, Gree, Blue	mIoU 64.76% dan 64.58%

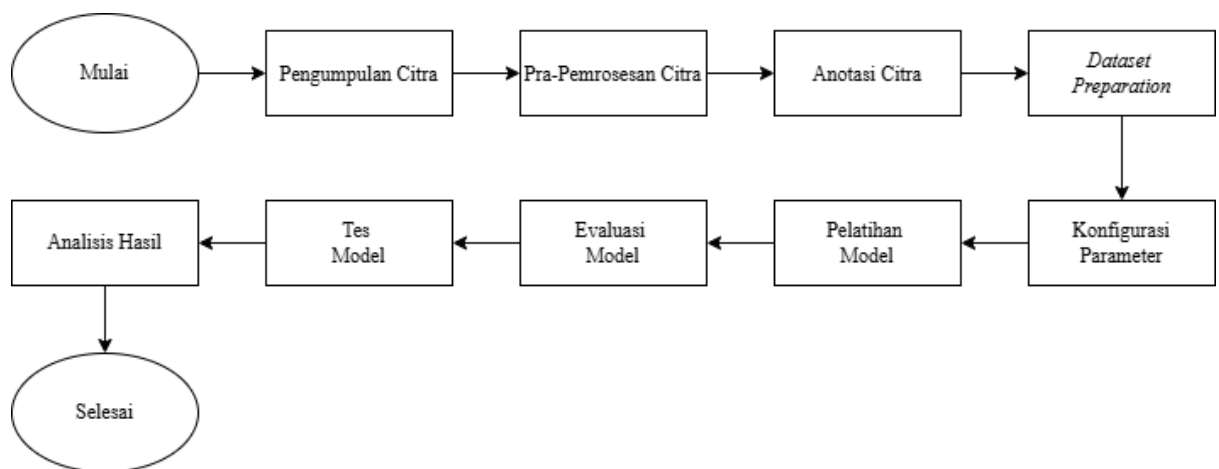
			<i>land, forest, grassland.</i>		
Du et al. (2020)	ISPRS 2D Semantic Labeling	DeepLabv3+ dengan OBIA	<i>Building, Tree, Grass, Car, Road</i>	Red, Green, Near-Infrared	Akurasi keseluruhan 90.6% (Vaihingen) dan 85.0% (Postdam)
Yu et al. (2022)	PolSAR datasets	Lightweight Complex-Valued DeepLabv3+	<i>Background, Stem beans, Peas, Forest, Lucerne, Wheat 1, Beet, Potatoes, Bare soil, Grass, Rapeseed, Barley, Wheat 2, Wheat 3, Water, Building</i>	Red, Green, Blue	mIoU dengan nilai 95.52 pada <i>dataset</i> Flevoland 1, 96.31 pada <i>dataset</i> Flevoland 2, dan 97.01 pada <i>dataset</i> san Francisco. Nilai ini lebih tinggi dibandingkan dengan DeepLabv3+ asli
Akcay et al. (2021)	Potsdam dan Vaihingen	Dual-Stream DeepLabv3+	<i>Impervious surfaces, building, low vegetation, tree, dan car</i>	Red, Green, Blue	Akurasi keseluruhan 88.87% (Postdam) dan 87.39% (Vaihingen)
Zheng (2020)	PASCAL VOC2012	DeepLabv3+ dengan Kernel Konvolusi Fractional-Order	<i>Background, Aeroplane, Bicycle, Bird, Boat, Bottle, Bus, Car, Cat, Chair, Cow, Dining-Table, Dog, Horse, Motorbike, Person, Potted-Plant, Sheep, Sofa, Train, TV/Monitor</i>	Red, Green, Blue	mIoU lebih dari 89%

(W. Li & Zhao, 2022)	Tidak disebutkan	DeepLabv3+ dengan modul <i>attention</i>	<i>Building</i>	Red, Green, Blue	IoU 75.24
Sunandini et al. (2023)	Sentinel-2	DeepLabv3+ dengan ASPP	<i>Water Bodies</i>	Red, Green, Blue	Peningkatan akurasi dengan penggunaan ASPP. Berhasil meraih nilai precision 0.90

BAB III

METODOLOGI PENELITIAN

Penelitian ini mencakup 9 langkah yang diambil oleh peneliti. Langkah-langkah tersebut termasuk pengumpulan citra, *image preprocessing*, *image annotation*, *dataset preparation*, konfigurasi parameter, pelatihan model, evaluasi model, tes model, dan analisis hasil seperti terlihat pada Gambar 3.1.



Gambar 3.1 Diagram proses penelitian

3.1 Pengumpulan Citra

Dalam penelitian ini, peneliti memanfaatkan tiga sumber utama untuk pembuatan *dataset* segmentasi citra satelit yang berfokus pada tutupan lahan dan objek di sekitar area gunung api. Salah satu sumbernya adalah citra dari Google Earth Pro untuk area tutupan lahan (*Google Earth Pro*, n.d.). Sumber ini dipilih karena memiliki keunggulan berupa resolusi yang sangat tinggi dan fleksibilitas dalam memilih resolusi, mulai dari 720p hingga 8K. Citra yang diambil mencakup berbagai wilayah, seperti area sekitar Gunung Merapi Yogyakarta, Klaten, Kota Yogyakarta, Kebumen, Purworejo, Semarang, Cilacap, Ciamis, Surakarta, dan Boyolali. Pemilihan area ini bertujuan untuk mendapatkan data yang beragam, mencerminkan kondisi medan yang berbeda-beda. Sumber lain yang digunakan adalah web sentinel Hub

Peneliti secara khusus menggunakan citra beresolusi 4K untuk citra tutupan lahan dan diambil dari Google Earth Pro dengan skala/*eye altitude* beragam. Resolusi ini dipilih karena keterbatasan sumber daya yang tersedia, namun tetap mampu mempertahankan detail penting

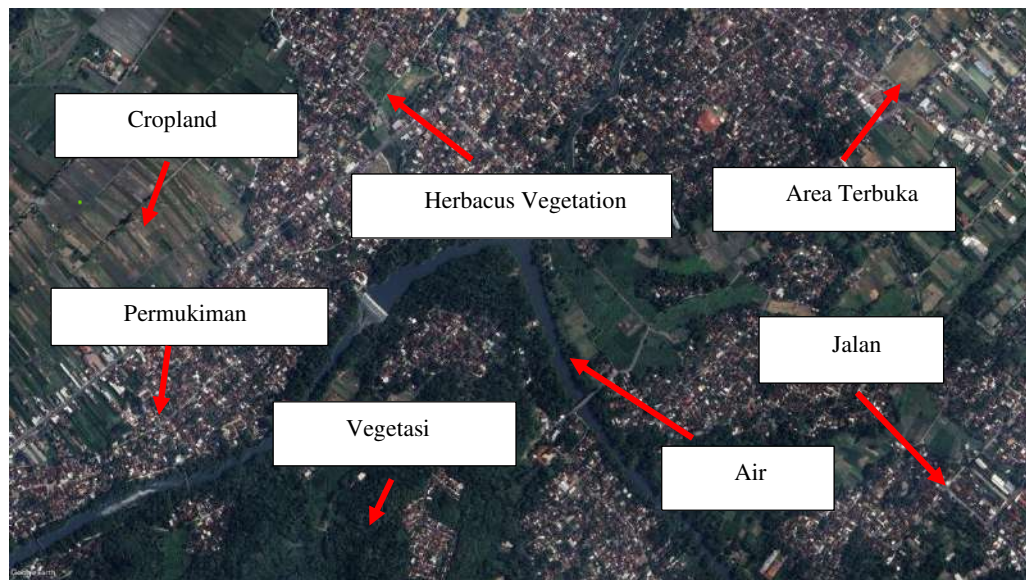
tanpa kehilangan informasi signifikan. Selain itu, variasi skala dan lokasi membantu meningkatkan keragaman *dataset*, bertujuan untuk meningkatkan performa model saat diujikan pada data baru. Skala yang digunakan di setiap lokasi bervariasi, namun nilainya tidak didokumentasikan secara eksplisit. Rentang yang digunakan adalah 500 meter di atas permukaan laut (Mdpl) hingga 1.500 meter di atas permukaan laut (Mdpl).

Penelitian kali membagi kelas sebanyak tujuh kelas untuk tutupan lahan dan tujuh kelas untuk objek gunung api. Citra tutupan lahan telah diklasifikasikan ke dalam 7 kelas yaitu permukiman, *cropland* (lahan pertanian), vegetasi, *herbaceous vegetation*, jalan, air, dan area terbuka. Kelas-kelas ini dipilih berdasarkan pengelompokan yang dibuat oleh Copernicus dalam *dataset* Google Earth Engine (*Copernicus Global Land Cover Layers: CGLS-LC100 Collection 3*, n.d.). Meskipun dokumentasi aslinya mencakup lebih banyak kategori, peneliti memutuskan untuk fokus terhadap 7 kelas guna menyesuaikan dengan keterbatasan sumber daya. Contoh citra tutupan lahan yang memiliki keseluruhan kelas dapat dilihat pada Gambar 3.2. Pilihan kelas ini tetap relevan dan mampu menggambarkan kondisi sebenarnya dari area yang dikaji. Penjelasan setiap kelas dijabarkan dalam Tabel 3.1.

Tabel 3.1 Penjelasan kelas tutupan lahan

Kategori	Penjelasan Kelas
Permukiman	Kawasan permukiman/binaan. Lahan yang tertutupi oleh bangunan dan struktur buatan manusia lainnya.
<i>Cropland</i>	Lahan pertanian mencakup area yang digunakan untuk menanam tanaman musiman dan setelah dipanen biasanya dibiarkan kosong untuk sementara waktu.
Jalan	Area yang mencakup berbagai infrastruktur transportasi, seperti jalan raya, jalan tol, jalan lokal, jalur kendaraan, hingga rel kereta api. Wilayah ini biasanya memiliki permukaan buatan, seperti aspal atau beton.
Vegetasi	Area yang didominasi oleh tumbuhan alami seperti pepohonan dan semak-semak dan lebih besar dari tanaman herba. Wilayah ini mencakup hutan dan area hijau lainnya.
Air	Area yang mencakup danau, waduk, selokan, atau sungai, dapat berupa air tawar maupun air asin.

<i>Herbaceous Vegetation</i>	Area yang ditutupi tanaman dan tidak memiliki batang atau tunas permanen. Lalu, tumbuhan tersebut tidak memiliki struktur kokoh. Tutupan pohon dan semak pada area ini kurang dari 10%.
Area Terbuka	Area didominasi oleh tanah terbuka, pasir, atau batu, dengan tutupan vegetasi tidak pernah melebihi 10% sepanjang tahun.



Gambar 3.2 Kelas-kelas pada citra tutupan lahan

Gambar 3.2 menunjukkan citra tutupan lahan yang mencakup berbagai kelas. Namun, tidak semua citra selalu menampilkan semua kelas secara utuh. Kelas permukiman ditandai dengan keberadaan bangunan, kelas air dikenali melalui keberadaan sungai, kelas *cropland* (lahan pertanian) terlihat dari pola kotak-kotak sawah, kelas vegetasi ditandai dengan keberadaan pohon-pohon yang rindang, kelas area terbuka seperti lapangan, dan *herbacous vegetation* yang terlihat dari rerumputan.

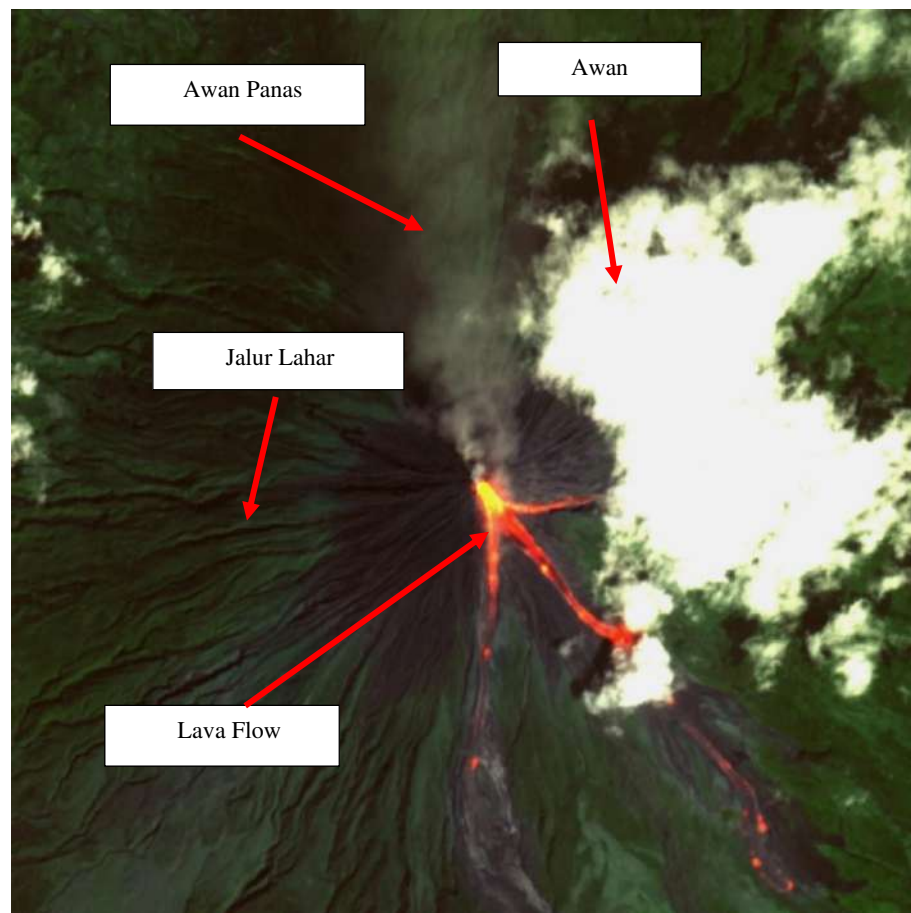
Pengumpulan data untuk segmentasi citra satelit objek sekitar gunung api dilakukan dengan menggabungkan berbagai sumber data yaitu dari Google Earth Pro (*Google Earth Pro*, n.d.), Google Images (*Google Image*, n.d.), dan *dataset* yang digunakan dari paper (Del Rosso et al., 2021). Lokasi pengambilan citra berbeda-beda berdasarkan sumbernya. Citra yang diambil melalui Google Earth Pro mencakup Gunung Merapi di Yogyakarta dan gunung api lain. Sementara itu, untuk citra yang diperoleh dari Google Images, lokasi pengambilannya tidak diketahui. Dalam penelitian Del Rosso et al., 2021 disebutkan bahwa *dataset* yang digunakan juga mencakup citra dari beberapa gunung api lain, namun lokasi spesifiknya tidak

dijelaskan secara rinci. Untuk citra satelit gunung api terbagi menjadi 7 kelas. Penentuan kelas ini dipilih berdasarkan diskusi dengan Dr. Nur Naning Aisyah, S.Si., M.Sc. ahli citra satelit khususnya pada gunung api dari BPPTKG Yogyakarta. Penjelasan setiap kelas dijabarkan dalam Tabel 3.2. Contoh kelas-kelas pada citra gunung api ditampilkan pada Gambar 3.3 dan Gambar 3.4.

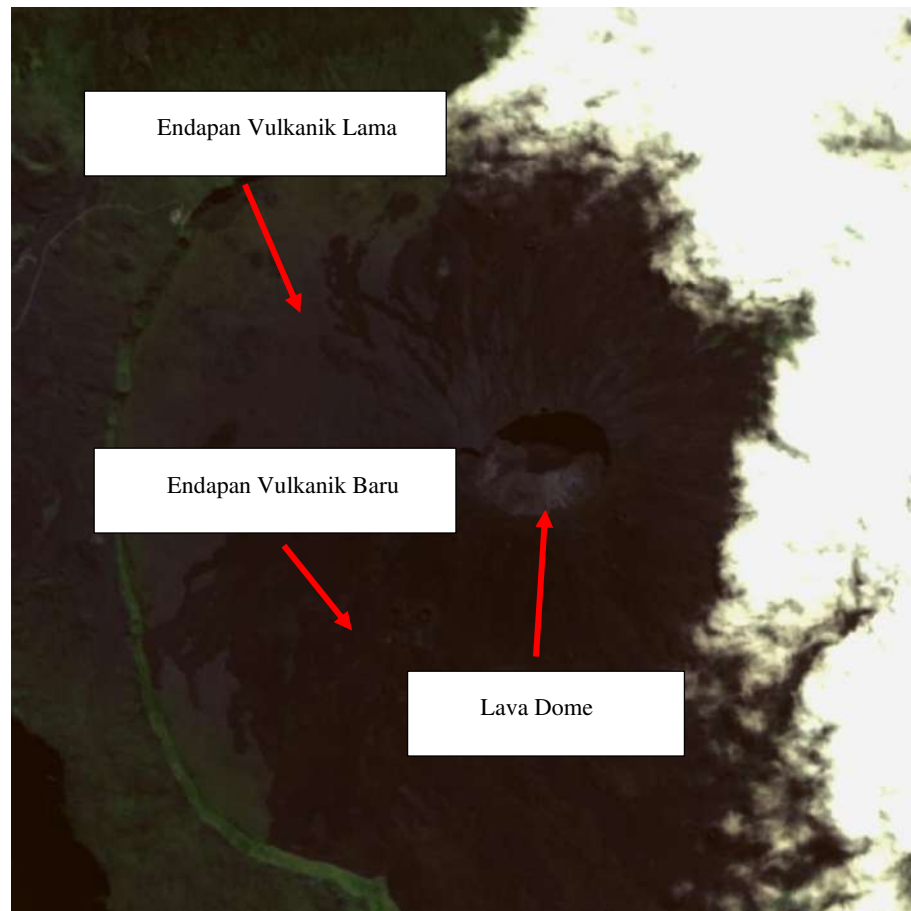
Tabel 3.2 Penjelasan kelas citra gunung api

Kategori Kelas	Penjelasan Kelas
Endapan Vulkanik Lama	Material hasil erupsi gunung berapi masa lampau yang telah mengalami pelapukan, pemadatan, atau sedimentasi ulang. Pada citra satelit, endapan ini memiliki karakteristik visual berupa warna abu-abu kusam hingga coklat tua di area terbuka, atau hijau jika telah ditumbuhi vegetasi. Teksturnya dapat terlihat halus atau kasar tergantung tingkat pelapukan.
Endapan Vulkanik Baru	Material hasil erupsi terkini, seperti abu vulkanik, aliran lava, atau lahar, yang masih segar dan belum mengalami pelapukan. Pada citra satelit, endapan ini terlihat gelap (abu-abu tua hingga hitam) dengan tekstur kasar dan refleksi rendah pada spektrum NIR.
<i>Lava Dome</i>	Struktur berbentuk melingkar atau memanjang. Terbentuk dari magma kental yang keluar ke permukaan tetapi tidak mengalir jauh. Biasanya terletak di kawah utama atau lereng gunung berapi, dengan tekstur kasar dan warna abu-abu tua pada citra satelit.
Jalur Lahar	Alur yang dilalui aliran material vulkanik bercampur air, dan mengikuti topografi lembah atau sungai di lereng gunung berapi. Pada citra satelit, jalur ini tampak sebagai pola linier gelap atau abu memanjang dari kawah hingga dataran rendah.
Awan Panas	Aliran piroklastik yang terdiri dari gas, abu, dan material vulkanik panas hasil erupsi gunung berapi. Pada citra satelit, awan panas terlihat berwarna abu-abu atau coklat dengan pola menyebar menyerupai bentuk bulu domba.
Awan	Kumpulan tetesan air atau kristal es yang terbentuk di atmosfer. Pada citra satelit, awan tampak berwarna putih terang dengan tekstur

	bervariasi, mulai dari halus untuk awan tipis hingga gumpalan padat menyerupai kapas untuk awan tebal.
<i>Lava Flow</i>	Aliran magma yang keluar dari gunung berapi dan menyebar di permukaan bumi. Pada citra satelit, lava flow umumnya terlihat berwarna gelap (abu-abu tua hingga hitam) akibat proses pendinginan aliran lava yang cepat. Namun, dalam beberapa kasus, lava flow juga dapat tampak berwarna merah jika sensor satelit menangkap aliran lava yang masih aktif, dicirikan dengan pola linier atau bercabang yang mengikuti kontur topografi.



Gambar 3.3 Kelas-kelas citra gunung api



Gambar 3.4 Kelas-kelas citra gunung api

Pemilihan resolusi citra dilakukan dengan pertimbangan cermat, khususnya untuk data dari Google Earth Pro. Resolusi Full HD dipilih karena dianggap cukup untuk menggambarkan area objek gunung api tanpa mengambil resolusi 4K. Perbedaan utama antara Full HD dan 4K terletak pada resolusi atau jumlah piksel yang digunakan untuk membentuk gambar pada layar. Pemilihan resolusi Full HD bertujuan untuk menghindari penggunaan skala yang terlalu besar, karena nantinya citra akan dipotong menjadi banyak *patch*. Penggunaan skala yang terlalu besar dapat menyebabkan objek seperti lahan pertanian dan permukiman menjadi sulit terlihat pada citra objek gunung api.

Walaupun resolusinya lebih rendah untuk citra gunung api yang diambil pada Google Earth Pro, detail objek tetap terlihat dengan jelas sehingga tetap mendukung kebutuhan analisis. Selain itu, resolusi sedang ini juga dipilih untuk menghindari dominasi satu kelas dalam satu *patch* akibat ukuran objek gunung yang besar. Pendekatan ini dirancang untuk mengurangi penggunaan *resource* dan waktu pelatihan yang signifikan, sehingga proses

pengolahan data menjadi lebih efisien. Untuk citra yang diambil dari Google Image dan penelitian (Del Rosso et al., 2021) berukuran 750x750 piksel dan di-*resize* menjadi 1024x1024.

Semua citra yang dikumpulkan baik untuk tutupan lahan dan gunung api merupakan citra RGB. Namun, ada perlakuan khusus untuk citra yang diambil dari penelitian (Del Rosso et al., 2021). Penelitian tersebut menjelaskan bahwa *lava flow* dapat terdeteksi oleh sensor satelit. Namun dari hampir keseluruhan kasus, *lava flow* ini sulit dideteksi oleh sensor tersebut. Masalah ini muncul karena keterbatasan satelit yang hanya dapat merekam citra pada waktu tertentu. Jika pengambilan citra tidak dilakukan pada waktu yang sesuai, aliran lava pada citra RGB akan terlihat berwarna hitam dan menyatu dengan sekitarnya akibat proses pendinginan yang berlangsung cepat. Masalah lainnya dapat timbul akibat keberadaan awan yang menutupi aliran lava selama proses pengambilan citra.

Untuk itu, diperlukan prapemrosesan pada citra RGB guna meningkatkan visibilitas aliran lava dengan menambahkan *band* tambahan yaitu *Near Infrared* (NIR). Band NIR ini dimanfaatkan untuk menangkap informasi termal dari aliran lava, sehingga citra RGB pada penelitian (Del Rosso et al., 2021) dapat menampilkan aliran lava dengan warna merah, dengan hasil akhir berupa citra yang menggabungkan band RGB dan NIR. Dari poroses pengumpulan data ini diperoleh 14 kelas utama dengan gabungan dari citra tutupan lahan dan gunung api.

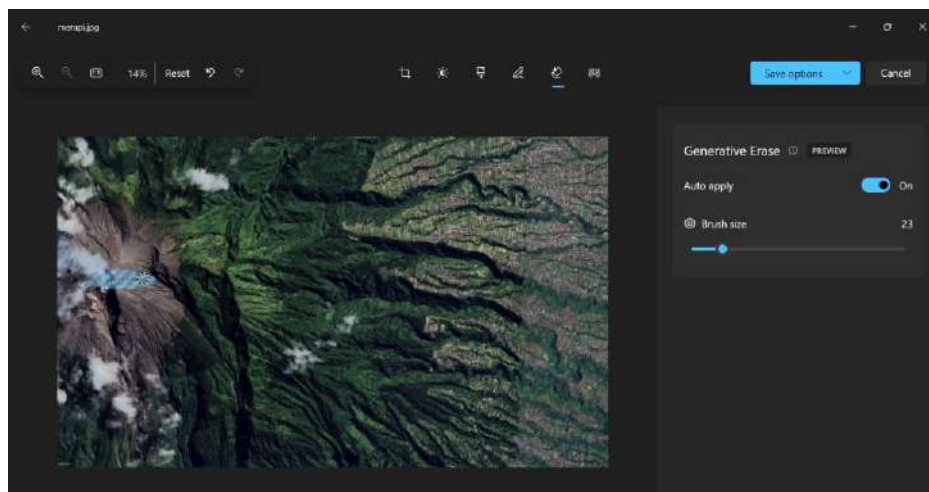
3.2 Pra-Pemrosesan Citra

Pada tahap ini, peneliti menggunakan data citra yang telah dikumpulkan dan melakukan proses pengolahan untuk meminimalkan keberadaan *noise* pada data. *Noise* yang dimaksud meliputi label *pin point* dan *watermark*. Proses pengurangan *noise* dilakukan melalui penghapusan label pada citra satelit gunung berapi. Pada citra satelit yang diperoleh dari Google Earth Pro, sering ditemukan keberadaan label peta yang berpotensi mengganggu proses pelatihan model. Oleh karena itu, penghapusan label-label tersebut dilakukan agar citra yang digunakan bebas dari *noise* dan lebih siap untuk proses pelatihan. Citra yang diperoleh dari Google Earth Pro ditampilkan pada Gambar 3.5.



Gambar 3.5 Citra gunung berapi memiliki label

Gambar 3.5 menunjukkan keberadaan label *pin point* "Mount Merapi" dan *watermark* yang bertuliskan "Google Earth". Kedua *noise* ini perlu dihapus untuk menghindari interferensi dalam proses pelatihan model. Untuk menghilangkan label ini, peneliti menggunakan fitur *generative erase* pada aplikasi pengolah gambar bawaan dari windows. Fitur ini memungkinkan penghapusan label secara spesifik tanpa memengaruhi area lain pada citra. Tahapan penghapusan label dan watermark ini dapat dilihat pada Gambar 3.6.



Gambar 3.6 Proses penghapusan label

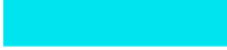
Dari Gambar 3.6, peneliti hanya perlu menyoroti objek yang ingin dihapus. Setelah itu, alat ini akan secara otomatis mengisi piksel yang telah dihapus. Namun, penting untuk memastikan bahwa proses penghapusan dilakukan dengan hati-hati, karena sering kali hasilnya

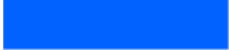
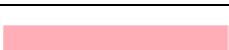
kurang sempurna akibat *software* masih dalam tahap pratinjau yang digunakan pada komputer peneliti.

3.3 Anotasi Citra

Proses *data annotation* dilakukan terhadap keseluruhan citra. Pada tahap ini, data citra yang telah dikumpulkan sebelumnya akan melalui tahap pelabelan. Langkah ini bertujuan untuk memberikan keterangan kategori atau kelas objek sebelum dilakukan pelatihan model. Tahapan pelabelan ini dikenal sebagai proses *masking*, proses pelabelan citra dilakukan secara manual menggunakan aplikasi Sketchbook Pro pada setiap citra dengan bantuan pen tablet. Dalam proses pelabelan seluruh citra, peneliti dibantu dengan deskripsi dari USGS, fitur Street View, serta model permukaan 3D yang tersedia di Google Earth Pro untuk meminimalkan potensi kesalahan dalam pelabelan. Selain menggunakan metode yang sudah digunakan, proses pelabelan citra gunung berapi dilakukan dengan pendampingan dari Dr. Nur Naning Aisyah, S.Si., M.Sc., untuk memastikan keakuratan *mask* pada citra gunung api. Setelah proses pelabelan selesai, dibuat tabel *color dictionary* yang berfungsi untuk mendefinisikan warna masing-masing kelas, seperti ditampilkan pada Tabel 3.3.

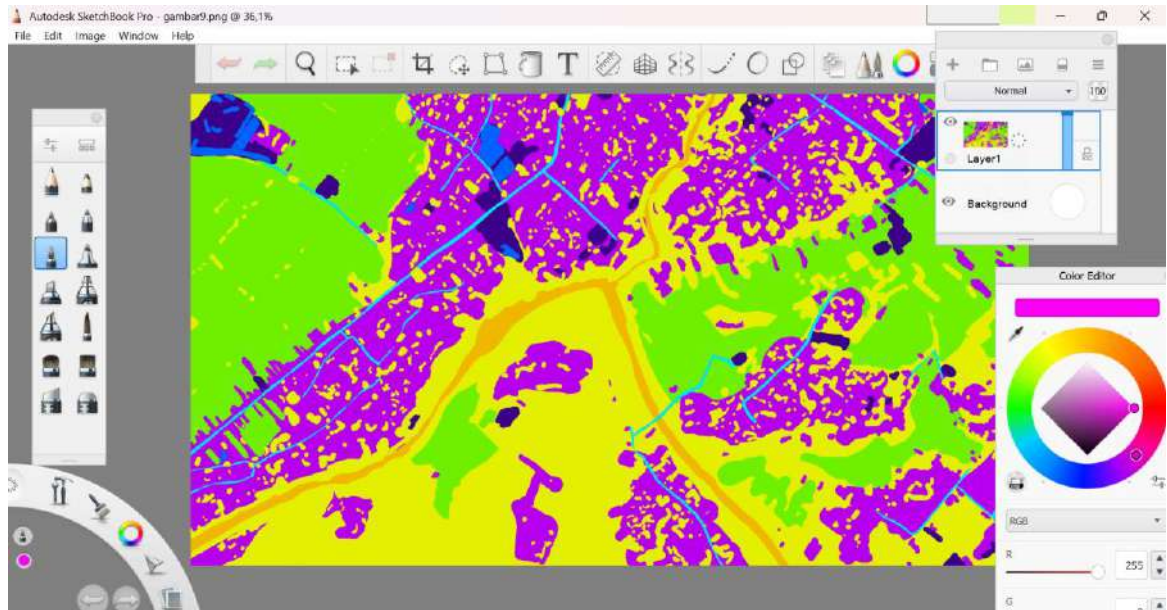
Tabel 3.3 *Color dictionary*

Kategori	RGB	Hex Code	Representasi Warna
<i>Cropland</i>	(112, 240, 0)	#70F000	
Permukiman	(184, 0, 240)	#B800F0	
Air	(240, 184, 0)	#F0B800	
Area Terbuka	(60, 0, 138)	#3C008A	
Jalan	(0, 228, 240)	#00E4F0	
Vegetasi	(228, 240, 0)	#E4F000	

<i>Herbaceous Vegetation</i>	(0, 98, 255)	#0062FF	
<i>Lava Dome</i>	(255, 0, 4)	#FF0004	
Jalur Lahar	(65, 0, 204)	#4100CC	
Awan Panas	(204, 122, 0)	#CC7A00	
Endapan Material Vulkanik Baru	(255, 0, 247)	#FF00F7	
Endapan Material Vulkanik Lama	(120, 120, 120)	#787878	
Awan	(255, 173, 183)	#FFADB7	
<i>Lava Flow</i>	(122,0, 14)	#7A000E	



Gambar 3.7 Citra asli



Gambar 3.8 Anotasi dengan SketchBook Pro

Gambar 3.8 menunjukkan penggunaan SketchBook Pro untuk melakukan anotasi pada citra asli pada Gambar 4.8. Penggunaan warna-warna dalam anotasi disesuaikan dengan kamus warna (*color dictionary*) yang telah ditentukan pada Tabel 3.3. Fitur *layering* dapat dimanfaatkan untuk menganotasi setiap kelas secara terpisah, sehingga memudahkan aplikasi dan pengelolaan anotasi. Proses anotasi dilakukan dengan teknik *tracing* pada citra satelit asli, dan dibantu dengan penggunaan pen tablet untuk mempermudah pengerjaan *masking*.

3.4 Dataset Preparation

Citra satelit yang telah dianotasi mencakup seluruh kelas lalu disusun dalam folder-folder terstruktur untuk memudahkan pengolahan data. Folder data tutupan lahan terdiri dari pasangan folder gambar dan mask, demikian pula dengan folder citra gunung api. Selain itu, terdapat folder khusus untuk citra test yang merupakan kombinasi citra tutupan lahan dan citra gunung api. Penataan ini dirancang agar setiap folder memiliki data yang lengkap dan sistematis, sehingga mendukung efisiensi proses pelatihan model. Pada folder Tes, struktur serupa diterapkan dengan folder gambar digunakan untuk menguji model dan folder mask sebagai ground truth yang akan dibandingkan dengan hasil prediksi model.

3.5 Konfigurasi Parameter

Pada tahap konfigurasi parameter, dilakukan serangkaian eksperimen untuk menyesuaikan berbagai parameter dalam model Deep Learning. Proses ini dikenal sebagai *fine-*

tuning hyperparameter. Langkah ini sangat penting untuk menemukan konfigurasi optimal guna memaksimalkan performa model. Proses ini melibatkan pengujian berbagai kombinasi pengaturan untuk memperoleh hasil terbaik. Beberapa parameter yang disesuaikan meliputi *patch size*, *input image* model DeepLabV3+, *epoch*, *sheduler*, *backbone*, *batch size*, *weight decay*, dan *dropout*. Parameter optimal diperoleh dari model terbaik yang telah disimpan sebelumnya dan hasilnya dimanfaatkan sebagai acuan utama dalam konfigurasi pelatihan model selanjutnya.

3.6 Pelatihan Model

Pada tahap ini, setelah data citra melalui proses pelabelan atau anotasi akan dimanfaatkan untuk melatih model dengan arsitektur DeepLabV3+ menggunakan konfigurasi *hyperparameter* yang telah dirancang sebelumnya dalam proses konfigurasi parameter. Pelatihan ini bertujuan untuk membantu model mempelajari pola serta karakteristik yang terdapat pada data citra. Dengan proses ini, diharapkan model mampu melakukan segmentasi semantik pada citra satelit dengan hasil baik, sehingga dapat mendukung berbagai keperluan analisis berbasis citra satelit. Penting untuk mengatur lingkungan terlebih dahulu sebelum memulai pelatihan model. Pengaturan ini disesuaikan dengan platform yang digunakan. Platform yang digunakan dalam penelitian ini adalah Hanabi Server milik Universitas Islam Indonesia, dan Google Colab.

3.6.1 Patching Citra

Untuk mengatasi ukuran citra satelit yang sangat besar, teknik *patching* digunakan dengan cara membagi citra tersebut menjadi potongan-potongan kecil berukuran tertentu. Teknik ini diterapkan juga pada banyak penelitian untuk segmentasi citra satelit. Misalnya pada penelitian (Sertel et al., 2022) yang membagi citra *very high resolution* (VHR) menjadi *patch* berukuran 512x512. Berbeda dengan teknik *cropping*, *patching* dirancang untuk memecah citra beresolusi tinggi menjadi segmen-segmen kecil yang lebih mudah dikelola oleh model. Teknik ini tidak hanya memperbesar dataset dari citra berukuran besar dengan jumlah yang terbatas, tetapi juga mampu meningkatkan keragaman data yang dapat dimanfaatkan dalam proses pelatihan. Selain itu, *patching* menjadi solusi efektif untuk mengatasi keterbatasan memori GPU dalam memproses citra beresolusi tinggi secara langsung. Oleh sebab itu, memastikan setiap *patch* memiliki pasangan *mask* yang sesuai sangat penting untuk menjaga konsistensi

data selama proses pelatihan. *Patch-patch* ini kemudian digunakan sebagai *input image* bagi model selama pelatihan.

3.6.2 *Split Patch Data*

Sebelum *patch Data* digunakan untuk melatih model, data tersebut terlebih dahulu harus dipisahkan menjadi dua subset yaitu *data train* dan *data validation*. *Data train* digunakan untuk melatih model, sedangkan *data validation* digunakan untuk menguji performa model selama pelatihan dan berfungsi untuk *tuning hyperparameter*. Pemisahan ini dilakukan dengan rasio perbandingan 80% untuk *data train* dan 20% untuk *data validation*.

3.6.3 *Data Augmentation*

Setelah tahap pembagian *dataset* dilakukan, langkah selanjutnya adalah melakukan *data augmentation* untuk menambah volume *data train* atau *train patches*. Teknik augmentasi yang digunakan meliputi *flip*, *blur*, dan *rotate*. Augmentasi data hanya diterapkan pada data train dengan tujuan agar model dapat mengidentifikasi pola dan fitur pada citra dengan lebih baik. Pada tahapan ini juga pastikan agar *images* dan *mask* diaugmentasi berpasangan. Dengan melakukan data augmentasi diharapkan model dapat mempelajari berbagai variasi pola dalam data sehingga berpengaruh dalam meningkatkan akurasi dan kemampuan generalisasi pada data baru.

3.6.4 *Train Model*

Proses pelatihan ini menggunakan model DeepLabV3+. Model ini dipilih dikarenakan DeepLabV3+ dapat menghasilkan prediksi yang baik sesuai dengan hasil penelitian L.-C. Chen et al. (2018). Model ini juga dapat menghasilkan tepian objek yang halus berkat arsitektur modelnya yang telah dijelaskan L.-C. Chen et al. (2018). Setelah proses augmentasi selesai, langkah selanjutnya adalah pelatihan model. Peneliti memulai dengan membuat train loader dan validation loader menggunakan PyTorch untuk gambar dan mask. Train loader digunakan untuk data train, sedangkan val loader digunakan untuk data validation. Kedua loader ini memfasilitasi pembacaan data secara efisien dalam batch selama pelatihan dan validasi.

Peneliti merancang model DeepLabV3+ dengan memanfaatkan teknik transfer learning (He et al., 2015).. Untuk mengurangi risiko overfitting, peneliti menerapkan checkpoint yang dapat menyimpan best model setiap kali nilai validation loss mengalami penurunan. Pemilihan metrik validation loss merujuk pada penerapan teknik early stopping yang dirancang untuk

menghentikan proses pelatihan apabila validation loss tidak lagi menunjukkan peningkatan, sebagaimana dijelaskan pada artikel (Heckel & Yilmaz, 2020). Namun pada penelitian ini, pelatihan tidak dihentikan untuk memungkinkan pengamatan menyeluruh terhadap seluruh proses pelatihan. Sebagai langkah tambahan, model juga dilengkapi dengan weight decay dan dropout untuk meminimalkan overfitting.

Selain itu, peneliti menambahkan variabel untuk menyimpan hasil evaluasi performa model, termasuk metrik seperti loss, IoU (Intersection over Union), accuracy, dice coefficient dan mean average precision (mAP). Metrik-metrik ini digunakan untuk mengukur seberapa baik model melakukan segmentasi semantik pada citra satelit. Dengan pendekatan ini, diharapkan model dapat mempelajari pola dan karakteristik dari data citra dengan lebih baik, sehingga mampu menghasilkan segmentasi yang akurat dan dapat diandalkan.

3.7 Evaluasi Model

Tahap evaluasi model merupakan bagian sangat penting dalam penelitian karena memungkinkan peneliti untuk memahami seberapa baik model yang telah dilatih dapat diaplikasikan terhadap *dataset*. Evaluasi model dilakukan dengan melihat metrik-metrik yang disimpan selama proses pelatihan, seperti *loss*, *IoU (Intersection over Union)*, *accuracy*, *dice coefficient* dan *mean average precision (mAP)*. Tahapan evaluasi juga membantu memastikan bahwa model tidak hanya bekerja dengan baik pada data *train* tetapi juga pada data yang belum pernah dilihat sebelumnya. Tahapan ini memungkinkan peneliti untuk mengidentifikasi kekurangan yang perlu diperbaiki berdasarkan hasil metrik selama proses pelatihan.

3.8 Tes Model

Setelah model selesai dilatih dan dievaluasi menggunakan data latih dan validasi, langkah berikutnya adalah mengujinya pada data uji untuk keperluan inferensi. Pengujian ini bertujuan untuk menilai kemampuan model dalam melakukan segmentasi semantik pada gambar-gambar baru yang belum pernah digunakan selama proses pelatihan. Model yang telah dilatih diterapkan untuk memprediksi hasil segmentasi pada *patch* citra dari data uji. Hasil prediksi ini kemudian dibandingkan dengan *mask ground truth* untuk menghitung metrik evaluasi. Evaluasi metrik pada data test dilakukan menggunakan metrik seperti, *Mean Intersection over Union (mIoU)*, dan *mean average precision (mAP)*. Berbagai metrik ini memberikan wawasan mengenai performa model dalam menangani segmentasi semantik pada data yang menggambarkan *real world case*.

Teknik serupa diterapkan untuk mendeteksi citra pada data tes. Citra data uji diterapkan teknik patching agar mudah disegmentasi oleh model. Setelah prediksi dilakukan pada patch citra uji, patch-patch tersebut digabungkan kembali untuk membentuk citra utuh yang besar. Untuk analisis lebih lanjut, peneliti menggunakan confusion matrix guna mengevaluasi kinerja model dalam mengklasifikasikan setiap kelas pada data uji. Confusion matrix ini memberikan wawasan mendalam terkait kesalahan klasifikasi yang sering muncul. Selain itu, precision, recall, dan f1-score untuk masing-masing kelas juga dianalisis pada tahap ini. Kode keseluruhan pada proses pengujian ditampilkan pada bagian lampiran.

3.9 Analisis Hasil

Untuk tahap akhir adalah peneliti melakukan analisis terhadap hasil penelitian secara menyeluruh untuk mengevaluasi kelebihan dan kekurangan model. Jika hasil pengujian menunjukkan performa memuaskan, ini menandakan kemampuan model dalam melakukan generalisasi yang baik. Sebaliknya, jika performanya belum optimal, peneliti dapat mengidentifikasi aspek yang memerlukan perbaikan. Langkah ini memastikan bahwa pengujian model berfungsi sebagai tahapan verifikasi akhir untuk menilai sejauh mana model mampu melakukan segmentasi semantik pada citra satelit dalam konteks aplikasi nyata di masa depan.

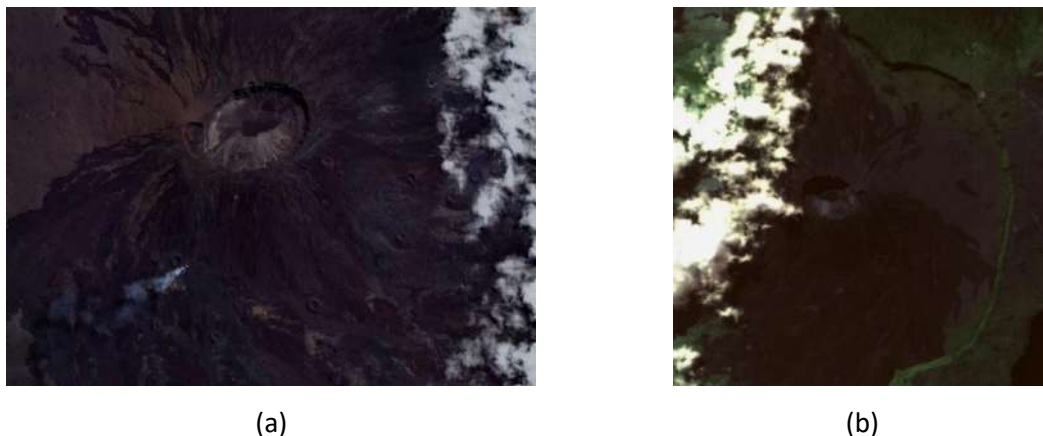
BAB IV

HASIL DAN PEMBAHASAN

4.1 Pengumpulan Citra

Proses pengumpulan citra dari sumber yang disebutkan pada Subbab 3.1, berhasil memperoleh 70 citra. Proses pengumpulan citra ini dilakukan dengan dua kali tahapan. Pada tahapan pertama, keseluruhan citra hanya terdiri dari 68 untuk proses penelitian ini. Pembagiannya terdiri dari 30 citra tutupan lahan dan 30 citra gunung api. Sisanya untuk keperluan tes yang terdiri dari 6 citra tutupan lahan dan 2 untuk citra gunung api. Namun setelah melalui empat kali percobaan, model tidak dapat mengenali satu kelas yaitu kelas “Lava Dome”. Untuk itu pada pengumpulan citra tahap kedua, peneliti menambahkan 2 citra gunung api dengan rincian 1 citra dari penelitian Del Rosso et al. (2021) dan 1 citra dari Google Earth Pro. Kedua citra tambahan ini ditampilkan pada Gambar 4.1.

Citra ini kemudian dibagi menjadi 62 untuk keperluan pelatihan dan validasi, serta 8 gambar untuk pengujian di tahap akhir. Gambar-gambar ini dikelompokkan ke dalam tiga folder yaitu Tile1, Tile2, dan Test. Folder Tile1 khusus digunakan untuk citra tutupan lahan, sedangkan Tile2 berisi citra gunung api. Folder Test disiapkan untuk pengujian setelah proses pelatihan dan validasi selesai. Folder test berisi citra gabungan antara citra area tutupan lahan dan juga gunung api. Pemisahan file ke dalam folder ini murni untuk mengelompokkan citra berdasarkan objeknya saja, pada proses pelatihan semua citra ini dikumpulkan dalam sebuah *array*.

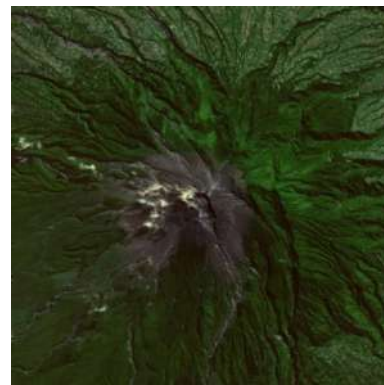


Gambar 4.1 Tambahan citra, (a) Citra dari Google Earth Pro, (b) Citra dari *paper*

Namun untuk menggali lebih mendalam tentang tidak terdeteksinya kelas *lava dome*, pada akhirnya peneliti menambahkan 10 citra gunung api yang memiliki *lava dome* untuk keperluan pelatihan dan 1 citra pada data tes. Total keseluruhan pengumpulan citra pada tahap ketiga yaitu 72 citra untuk proses pelatihan dan validasi lalu 9 citra untuk keperluan test pada akhir tahapan. Sumber diambil dari Google Earth Pro sebanyak 2 citra, Sentinel Hub (<https://www.sentinel-hub.com>) sebanyak 1 citra, dan juga *paper* (Del Rosso et al., 2021) sebanyak 7 citra. Karakteristik dari citra yang diambil dari Sentinel Hub yaitu memiliki resolusi spasial hingga 10 meter. Citra tambahan untuk data tes diambil dari *paper* (Del Rosso et al., 2021). Sample citra tambahan ini diperlihatkan pada Gambar 4.2. Formasi mengenai folder, resolusi citra, dan jumlah citra akan disajikan dalam Tabel 4.1.



(a)



(b)

Gambar 4.2 Citra tambahan pada tahap ketiga, (a) Citra gunung api dari Sentinel Hub, (b) Citra dari *paper*

Tabel 4.1 Deskripsi folder

Folder	Deskripsi	Resolusi	Resize	Jumlah Citra	Sumber
Tile 1	Citra tutupan lahan	3840 x 2160	-	30	Google Earth Pro
Tile 2	Citra Gunung Api	750x750	1024x1024	32	(Del Rosso et al., 2021)
		750x750	1024x1024	1	Google Image
		1080x1920	-	8	Google Earth Pro

		1080x1920	-	1	Sentinel Hub
Test	Pengujian Akhir	750x750	1024x1024	2	(Del Rosso et al., 2021)
		1080x1920	-	7	Google Earth Pro

Dapat dilihat dari Tabel 4.1 terdapat perbedaan ukuran resolusi. Untuk menjaga konsistensi, seluruh citra dari Google Images dan penelitian (Del Rosso et al., 2021) diubah ukurannya menjadi 1024x1024 piksel. Resolusi ini dipilih agar seragam dengan citra Gunung Merapi dari Google Earth Pro yang memiliki resolusi di atas 1000 piksel dan 1024x1024 masih bisa habis dibagi 16. Untuk citra dari Google Earth Pro tidak dilakukan proses *resizing* dikarenakan rasio citra tersebut yaitu 16:9. Jika dilakukan proses *resizing* menjadi resolusi 1024x1024 akan merubah rasio citra menjadi 1:1 yang di mana akan menghilangkan banyak detail pada citra.

Penyesuaian ini dilakukan untuk memastikan bahwa seluruh citra memenuhi persyaratan input model DeepLabV3+, yakni ukuran gambar harus dapat dibagi habis oleh 16 karena merupakan ketentuan penting dalam arsitektur model tersebut (PyTorch, n.d). Resolusi 1024x1024 juga dipilih karena mampu menjaga detail citra tetap jelas dan tidak pecah. Pendekatan ini memastikan *dataset* yang digunakan tidak hanya bervariasi tetapi juga memenuhi persyaratan teknis untuk segmentasi citra. Sampel citra dari setiap folder dapat dilihat pada Gambar 4.3, Gambar 4.4, dan Gambar 4.5.



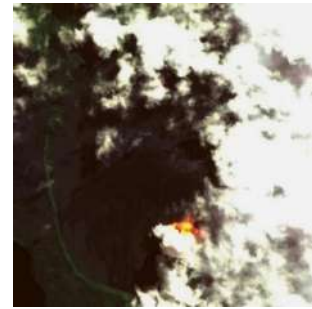
Gambar 4.3 Citra tutupan lahan pada folder Tile 1

Sumber : Google Earth Pro



(a)

Sumber : Google Earth Pro



(b)

Sumber : (Del Rosso et al., 2021)

Gambar 4.4 Citra gunung api pada Tile 2, (a) Citra Gunung Merapi, (b) Citra dari *paper*.



(a)

Sumber : Google Earth Pro



(b)

Sumber : (Del Rosso et al., 2021)

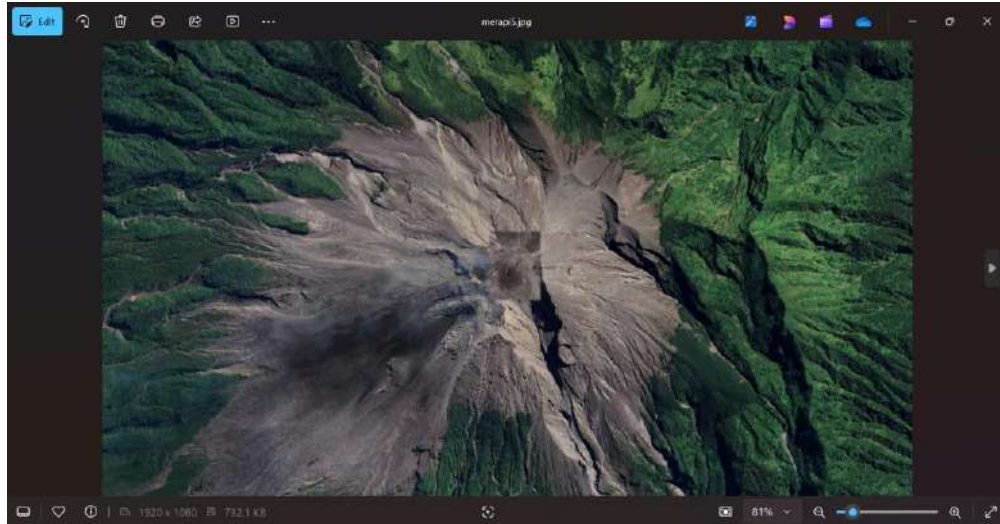
Gambar 4.5 Citra pada folder tes, (a) Citra tutupan lahan, (b) Citra gunung api.

Pada Gambar 4.4 terlihat bahwa Tile1 menampilkan citra satelit area tutupan lahan. Kelas-kelas telah dijelaskan pada Subbab 3.1, seperti permukiman dan lahan pertanian dapat terlihat pada sampel citra. Sementara itu, Tile2 yang ditampilkan pada Gambar 4.4, berisi citra tampak permukaan gunung api. Gambar 4.5 menampilkan kombinasi citra area tutupan lahan dan gunung berapi yang sedang mengalami erupsi pada folder Tes.

4.2 Pra-Pemrosesan Citra

Proses ini menghasilkan citra yang telah sepenuhnya bebas dari label dan *watermark*. Dengan memanfaatkan fitur pada aplikasi pengolah gambar, proses penghapusan label dan *watermark* menjadi lebih efisien. Fitur tersebut memungkinkan penghapusan label dan secara otomatis mengisi piksel yang kosong, sehingga tidak meninggalkan *pixel area* berwarna putih.

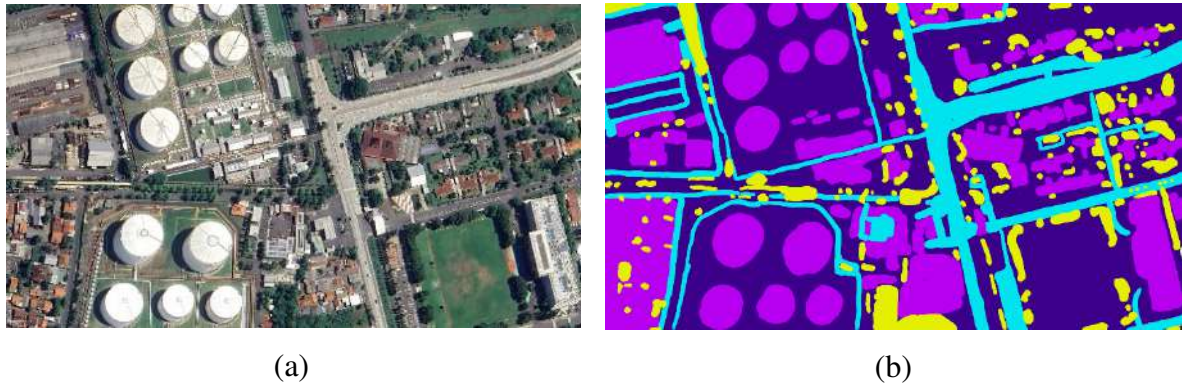
Hasil akhir dari proses dari proses penghapusan label dari contoh citra di Gambar 3.6 dapat dilihat pada Gambar 4.6.



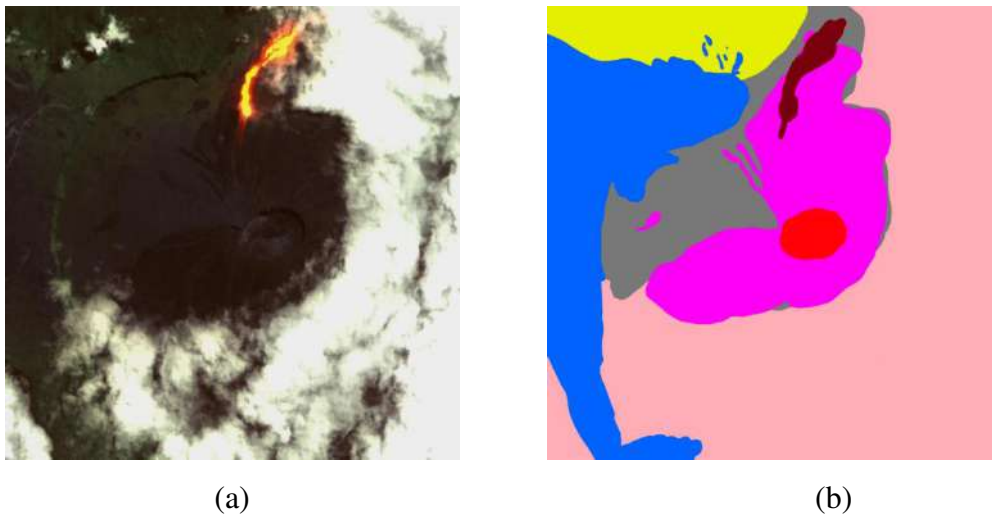
Gambar 4.6 Hasil proses penghapusan label dan *watermark*

4.3 Anotasi Citra

Setelah menyelesaikan proses anotasi yang melibatkan kolaborasi dengan ahli dan referensi dokumen pendukung, hasil anotasi berupa *mask* dari citra satelit disimpan dalam format PNG. Format ini dipilih untuk memastikan kualitas *mask* tetap optimal. Setiap *mask* kemudian dipasangkan dengan citra satelit yang sesuai untuk digunakan sebagai data pelatihan. Sebanyak 81 citra berhasil dianotasi, dengan setiap pasangan citra dan *mask* diberi nama identik untuk mempermudah pengelolaan selama proses pelatihan, misalnya gambar1.jpg untuk citra satelit dan gambar1.png untuk *mask* pasangannya. Contoh pasangan citra dan *mask* disajikan pada Gambar 4.7 dan Gambar 4.8.



Gambar 4.7 Citra beserta pasangan *mask*, (a) Citra tutupan lahan, (b) *Mask* tutupan lahan.



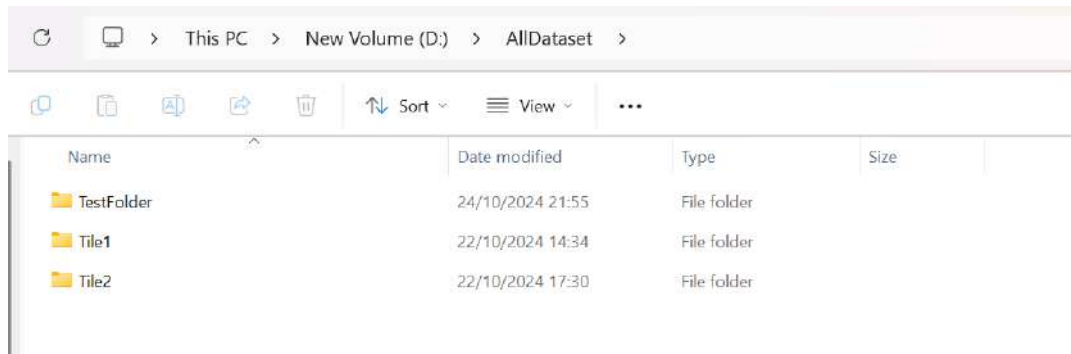
Gambar 4.8 Citra beserta pasangan *mask*, (a) Citra gunung api, (b) *Mask* gunung api.

Gambar 4.7 dan Gambar 4.8 menampilkan citra beserta *mask* yang digunakan sebagai label untuk melatih model, mencakup citra tutupan lahan dan gunung api. Proses anotasi ini memerlukan waktu 60 hari karena menghadapi beberapa kendala seperti proses perizinan, waktu yang dibutuhkan untuk mencari ahli, jadwal ahli yang padat, serta tingkat ketelitian yang tinggi dalam menganotasi citra. Contohnya untuk menganotasi pada kelas vegetasi yang berukuran kecil dan jumlahnya sangat banyak.

4.4 Dataset Preparation

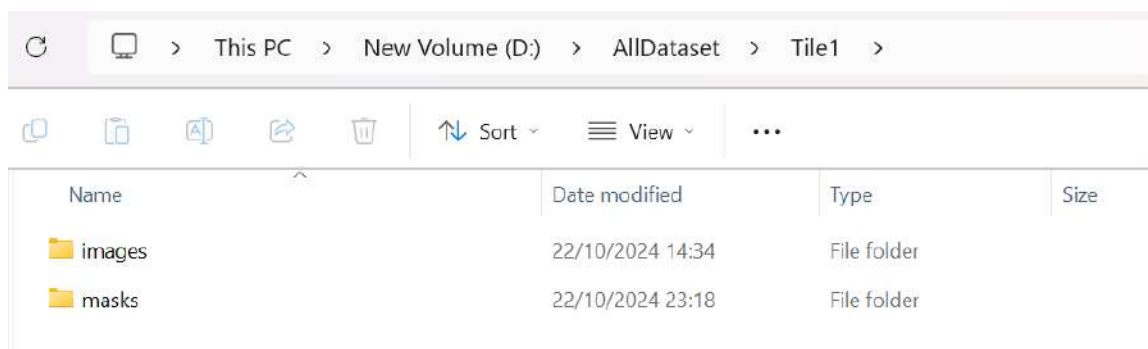
Setelah proses anotasi selesai, struktur folder yang sebelumnya digunakan pada tahap pengumpulan data mengalami sedikit perubahan agar file lebih terorganisasi dengan baik.

Folder Tile1, Tile2, dan Test masing-masing terdiri atas Subfolder *images* dan *masks*. Contoh Struktur *dataset* tersebut dapat dilihat pada Gambar 4.9.



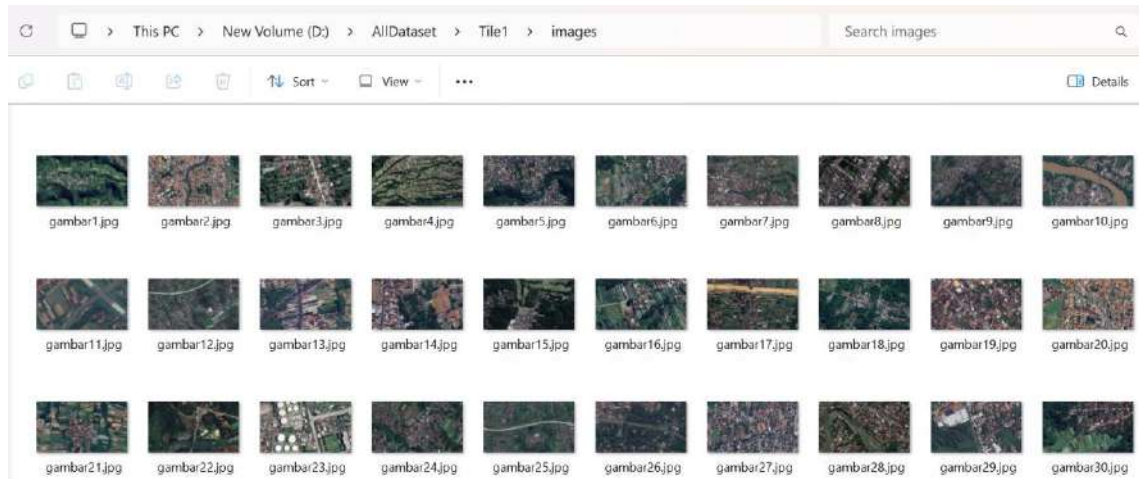
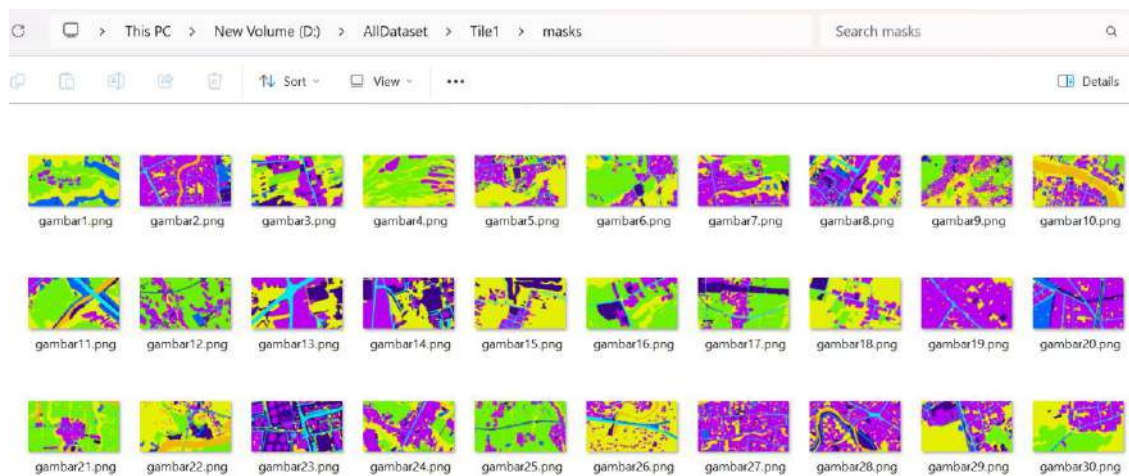
Gambar 4.9 Struktur *dataset*

Gambar 4.9 menunjukkan isi dari *dataset* yang terdiri dari beberapa folder. Di dalam folder tersebut terdapat Subfolder *images* dan *mask*. Contoh Subfolder pada Tile1 ditampilkan pada Gambar 4.10.



Gambar 4.10 Isi Subfolder Tile1

Pada Gambar 4.10 dalam folder *images* terdapat kumpulan citra area tutupan lahan, sedangkan folder *mask* berisi *mask* yang sesuai untuk setiap citra di folder *images*. Contoh sampel dari isi folder *images* pada Gambar 4.11 dan folder *mask* ini ditampilkan pada Gambar 4.12.

Gambar 4.11 Isi Subfolder *Images*Gambar 4.12 Isi Subfolder *Masks*

4.5 Konfigurasi Parameter

Hasil dari proses ini adalah kombinasi parameter yang akan diterapkan pada tahap pelatihan model. Untuk pelatihan model dilakukan pada dua platform yang berbeda yaitu Google Colab dan juga Hanabi Server milik Universitas Islam Indoensia. Sedangkan untuk pengujian model dilakukan pada Google colab. Spesifikasi *hardware* untuk pelatihan ini diperlihatkan pada Tabel 4.2 dan Tabel 4.3.

Tabel 4.2 Spesifikasi komputer tahap pelatihan

Spesifikasi Hardware Proses Pelatihan dan Validasi		
Platform	GPU	VRAM
Colab	Nvidia T4	15 GB
Hanabi Server UII	Tesla V100-SXM2	32 GB

Tabel 4.3 Spesifikasi komputer tahap pengujian model

Spesifikasi Hardware Proses Tes Model		
Platform	GPU	VRAM
Colab	L4	24 GB

Tahapan konfigurasi ini meliputi penyesuaian parameter, seperti *patch size*, *input image* model DeepLabV3+, *epoch*, *sheduler*, *backbone*, *batch size*, *weight decay*, dan *dropout*. Eksplorasi parameter adalah langkah krusial untuk menjamin kemampuan model dalam melakukan segmentasi multi kelas dengan hasil terbaik. Pemilihan parameter optimal ditentukan berdasarkan model terbaik, durasi pelatihan, waktu inferensi selama proses tes, atau pengujian dan nilai pada data *testing* yaitu test mAP dan test mIoU. Model terbaik selama proses pelatihan dipilih berdasarkan nilai *loss* terendah yang dicapai pada data validasi dan hasilnya disimpan pada kolom *saved best model*. Waktu inferensi ditentukan berdasarkan waktu yang paling dominan atau sering muncul pada setiap citra .

Tabel 4.4 merangkum konfigurasi parameter dalam lima percobaan. Pada eksperimen tahap awal peneliti hanya melibatkan 30 citra tutupan lahan dan 30 citra gunung api untuk pelatihan seperti yang telah dijelaskan pada Subbab 4.1. Pemilihan ini bertujuan menjaga keseimbangan citra antara kedua jenis area tersebut.

Patch size ditentukan sebagai ukuran spesifik yang digunakan untuk memecah citra besar menjadi bagian-bagian kecil selama tahap pelatihan model. Teknik ini diterapkan bertujuan untuk mengoptimalkan performa model secara keseluruhan. Pemilihan teknik *patching* juga bertujuan untuk tidak menghilangkan detail pada citra satelit yang beresolusi besar, tanpa mengurangi beban komputasi *hardware*. Jika menggunakan teknik lain untuk mengurangi beban komputasi misalnya dengan *resizing* atau *cropping*, tentunya citra besar tersebut tidak dapat diolah secara sempurna karena banyak detail pada citra yang akan hilang. Teknik ini umum digunakan pada penelitian-penelitian untuk segmentasi citra satelit dan dapat ditemui misalnya pada penelitian (Akçay et al., 2022).

Pada eksperimen tahap pertama peneliti mencoba *patch size* berukuran 256x256 piksel menyesuaikan ukuran *patch* pada penelitian (Akçay et al., 2022). Namun peneliti menyadari bahwa untuk nilai *patch* tersebut terlalu kecil untuk menangkap banyak detail pada *patch*. Maka dari itu peneliti menaikkan nilai *patch size* menjadi 320x320 piksel untuk menangkap banyak detail lebih banyak dalam satu *frame patch*.

Selama tahap pelatihan model. *Input image* dapat diperoleh langsung dari hasil pemotongan berdasarkan *patch size* yang ditetapkan. Dalam Tabel 4.4, berbagai parameter dieksplorasi dalam rentang tertentu. Sepanjang proses percobaan tahap pertama, nilai *epoch* tidak secara langsung menentukan penyimpanan model terbaik, karena umumnya model terbaik sudah tersimpan sebelum mencapai *epoch* ke-50 dalam setiap percobaan.

Scheduler dalam penelitian ini didefinisikan dengan nilai 25 untuk keseluruhan percobaan. Angka 25 dipilih karena peneliti melatih model sebanyak 80 *epoch* pada eksperimen tahap pertama, dan nilai ini tetap dipertahankan untuk eksperimen tahap berikutnya. Nilai 25 ini juga bertujuan untuk menyesuaikan perilaku *scheduler* dalam dokumentasi PyTorch (PyTorch, n.d). Dalam dokumentasi tersebut *scheduler* didefinisikan untuk menurunkan *learning rate* di *epoch* kelipatan 30 pada pelatihan 100 *epoch*, sehingga memungkinkan perubahan *learning rate* sebanyak tiga kali. Maka dari itu perlakuan *scheduler* ini diselaraskan dengan dokumentasi PyTorch untuk mengurangi *learning rate* sebanyak tiga kali, dengan penurunan terjadi pada *epoch* yang merupakan kelipatan dari 25

Tabel 4.4 Eksperimen tahap pertama

Nomor	Patch Size (px)	Input Image Size (px)	Epoch	Scheduler	Backbone	Batch size	Learning Rate	Hasil						
								Train Loss	Val Loss	Saved Best Model	Waktu Pelatihan	Test mIoU	Test mAP	Waktu Inference (s)
1	320x320	320x320	50	25	ResNet34	32	0.001	0.40	0.52	Epoch 34	30 Menit/Epoch	0.38	0.33	0.33
2	320x320	320x320	65	25	ResNet50	32	0.001	0.39	0.52	Epoch 33	13 Menit/Epoch	0.35	0.30	0.33
3	320x320	320x320	50	25	ResNet101	32	0.001	0.43	0.55	Epoch 30	10 Menit/Epoch	0.35	0.30	0.33
4	320x320	320x320	65	25	ResNet34	64	0.001	0.37	0.53	Epoch 33	10 Menit/Epoch	0.39	0.33	0.25
5	250x250	250x250	80	25	ResNet50	64	0.001	0.36	0.50	Epoch 29	11 Menit/Epoch	0.38	0.32	0.33

Pemilihan *backbone* dan *batch size* yang tepat memiliki dampak signifikan terhadap performa model. Hasil percobaan menunjukkan bahwa penggunaan *backbone* yang lebih mutakhir tidak selalu memberikan hasil optimal pada data tes, seperti terlihat pada percobaan nomor dua dan tiga di Tabel 4.4. Pada percobaan tersebut, *backbone* ResNet50 dan ResNet101 memerlukan penyesuaian ukuran *batch* menjadi lebih kecil untuk menyeimbangkan penggunaan memori. Selain itu, penggunaan *backbone* dengan arsitektur yang lebih kompleks tidak menghasilkan hasil segmentasi dengan nilai tinggi pada data tes. Hal ini dapat terjadi karena peningkatan kompleksitas model membuatnya sulit untuk mempelajari data secara optimal (Hu et al., 2021). Sebaliknya, pada percobaan nomor empat. Penggunaan *backbone* dengan arsitektur lebih sederhana, namun diimbangi dengan peningkatan *batch size* mampu mengolah data dengan baik.

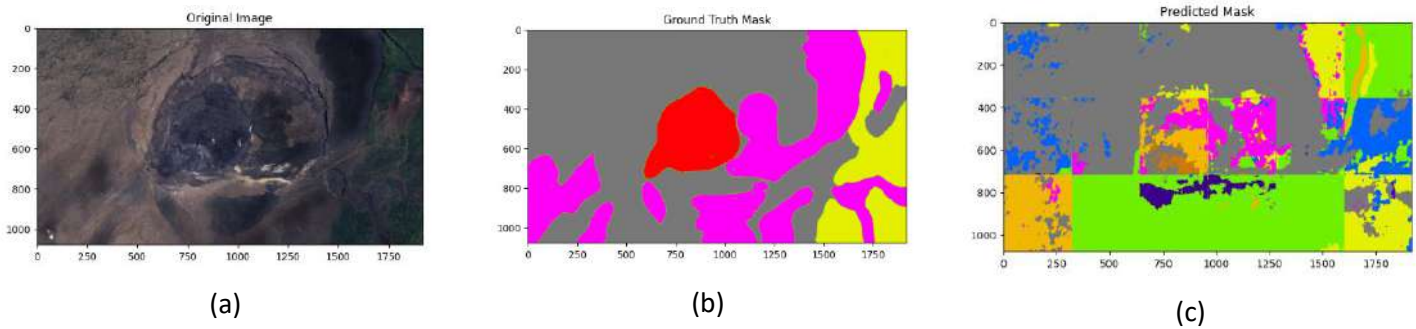
Berdasarkan hasil dari eksperimen tahap pertama, peneliti memutuskan untuk menjadikan kombinasi *hyperparameter* dari percobaan nomor empat dengan *patch size* 320x320 piksel, *input image size* 320x320 piksel, *epoch* bernilai 65, *scheduler* bernilai 25, *backbone* ResNet 34, *batch size* bernilai 64, dan *learning rate* 0.001. *Hyperparameter* ini ditetapkan sebagai konfigurasi dasar untuk uji tahap berikutnya. Hal ini dikarenakan percobaan tersebut mampu menghasilkan nilai tertinggi pada data tes dari keseluruhan percobaan dan menghasilkan waktu *inference* tercepat.

Alasan tidak dipilihnya hasil dari percobaan nomor lima karena tidak mampu menghasilkan nilai mIoU dan mAP yang baik dibandingkan dengan percobaan empat dan percobaan lima memerlukan waktu *training* yang lebih lama. Lalu, untuk percobaan menggunakan *batch* bernilai 128 sudah diuji coba. Namun, GPU tidak mampu mengakomodasi nilai *batch* tersebut. Pada akhirnya peneliti menetapkan nilai 64 untuk dijadikan parameter terbaik dikarenakan semakin besar nilai *batch* semakin efisien dan cepat pada proses pelatihan (Devarakonda et al., 2017).

Hasil pengujian konfigurasi parameter pada eksperimen tahap kedua dirangkum dalam Tabel 4.5. Pada eksperimen ini, peneliti menambahkan dua citra gunung api ke dalam *dataset* sebagaimana yang telah dijelaskan pada Subbab 4.1. Penambahan ini dilakukan setelah pengujian pada data uji dari seluruh percobaan di eksperimen tahap pertama menunjukkan bahwa model gagal mengenali salah satu kelas yaitu "Lava Dome". Sampel untuk hasil *testing* pada kelas "Lava Dome" pada eksperimen tahap pertama diperlihatkan pada Gambar 4.13.

Tabel 4.5 Eksperimen tahap kedua

Nomor	Patch Size (px)	Input Image Size (px)	Epoch	Scheduler	Backbone	Batch Size	Learning Rate	Hasil						
								Train Loss	Val Loss	Saved Best Model	Waktu Pelatihan	Test mIoU	Test mAP	Waktu Inference (s)
1	320x320	320x320	80	25	ResNet34	64	0.01	0.67	0.72	Epoch 66	10 Menit/Epoch	0.26	0.23	0.09
2	320x320	320x320	80	25	ResNet34	64	0.001	0.38	0.51	Epoch 30	10 Menit/Epoch	0.38	0.32	0.08
3	320x320	320x320	80	25	ResNet34	64	0.0001	0.29	0.52	Epoch 30	10 Menit/Epoch	0.35	0.30	0.25



Gambar 4.13 Sampel citra tes *lava dome*, (a) Citra original, (b) *Ground truth mask*, (c) *Predicted mask*

Pada Gambar 4.13 kelas “Lava Dome” yang diwakili oleh label berwarna merah tidak dapat terdeteksi oleh model. Untuk itu 30 citra tutupan lahan dan 32 citra gunung api digunakan pada eksperimen tahap kedua ini guna mengeksplorasi model lebih lanjut. Namun, setelah adanya penambahan citra gunung api, kelas “Lava Dome” masih belum bisa terdeteksi untuk keseluruhan eksperimen. Dengan adanya masalah ini, tentunya pada penelitian berikutnya dapat mengulik lebih jauh tentang tidak terdeteksi kelas “Lava Dome” dan dapat menganalisis penyebabnya.

Pada eksperimen tahap kedua peneliti mengeksplorasi nilai parameter *learning rate*, sementara nilai parameter lainnya tetap dipertahankan. Jumlah *epoch* juga ditingkatkan menjadi 80 untuk menganalisis pola pelatihan lebih lama pada setiap percobaan. Parameter terbaik pada eksperimen tahap pertama kemudian dilatih ulang bersamaan dengan penambahan citra. Hal ini dilakukan untuk memastikan model menggambarkan performa yang komprehensif dan representatif.

Hasil percobaan menunjukkan bahwa konfigurasi parameter dari eksperimen tahap pertama dengan *learning rate* 0.001 masih memberikan hasil optimal, ditandai dengan nilai *val loss* paling rendah. Lalu pada percobaan ini berhasil meraih nilai yang paling tinggi pada *data testing* dengan mIoU bernilai 0.37 dan mAP bernilai 0.32. Seiring bertambahnya data, nilai *val loss* mengalami penurunan sebesar 0.02 dibandingkan dengan hasil eksperimen pada tahap pertama. Hasil *learning rate* terbaik yang diambil pada eksperimen tahap kedua ini bernilai 0.001. Eksperimen tahap ketiga, peneliti mengeksplorasi ukuran *patch size* dan ukuran *input image*, dengan hasil yang dirangkum dalam Tabel 4.6. Pada eksperimen tahap ini juga peneliti menerapkan penyesuaian ukuran (*resizing*) untuk *input image* dengan resolusi 512x512 pixel dari *patch size* 1024x1024 pixel.

Tabel 4.6 Eksperimen tahap ketiga

Nomor	Patch Size (px)	Input Image Size (px)	Epoch	Scheduler	Backbone	Batch Size	Learning Rate	Hasil						
								Train Loss	Val Loss	Saved Best Model	Waktu Pelatihan	Test mIoU	Test mAP	Waktu Inference (s)
1	320x320	320x320	80	25	ResNet34	64	0.001	0.38	0.51	Epoch 30	10 Menit/Epoch	0.38	0.32	0.08
2	512x512	512x512	80	25	ResNet34	64	0.001	0.35	0.50	Epoch 42	40 Menit/Epoch	0.37	0.32	0.09
3	1024x1024	512x512	80	25	ResNet34	64	0.001	0.47	0.66	Epoch 42	3 Menit/Epoch	0.31	0.27	0.09
4	1024x1024	1024x1024	80	25	Rsenet34	16	0.001	0.50	0.67	Epoch 74	27 Menit/Epoch	0.33	0.29	0.09

Pengurangan ukuran *batch* pada percobaan empat dilakukan untuk mengatasi keterbatasan memori GPU yang tidak dapat mendukung proses pelatihan. Pengurangan *batch* diturunkan dari nilai 64 menjadi 16. Selain itu, hasil menunjukkan bahwa *patch size* dengan resolusi 1024x1024 piksel memiliki nilai *val loss* lebih tinggi dibandingkan percobaan lain dalam tahap ini. Temuan ini juga menunjukkan indikasi bahwa model telah kehilangan kemampuannya dalam memahami data secara efektif.

Hasil eksperimen ini menunjukkan bahwa parameter pada eksperimen tahap kedua tetap unggul dengan *patch size* 320x320 piksel, *input image size* 320x320 piksel, *epoch* bernilai 80, *scheduler* bernilai 25, *backbone* ResNet34, dan *batch size* bernilai 64. Parameter ini unggul ditunjukkan dengan selisih nilai *train loss* dan *val loss* paling kecil dibandingkan dengan tiga percobaan pada eksperimen tahap ketiga. Selain itu, waktu pelatihan dan hasil pada *test image* juga menunjukkan kinerja lebih baik dibandingkan percobaan lainnya. Pada eksperimen tahap ketiga, meskipun hasil percobaan satu lebih unggul dibandingkan percobaan lainnya, selisih hasil antara percobaan satu dan percobaan dua sangat kecil. Selain itu, kedua percobaan tersebut dilatih pada platform yang berbeda sehingga menimbulkan hasil yang bias. Untuk itu, pada eksperimen tahap keempat, peneliti memutuskan untuk mengkomparasikan kedua percobaan tersebut dengan menerapkan *dropout* dan *weight decay*.

Dalam eksperimen tahap keempat yang dirangkum pada Tabel 4.7. Peneliti memilih *dropout* bernilai kecil mengikuti rekomendasi dari makalah “On Dropout, Overfitting, and Interaction Effects in Deep Neural Networks” (Lengerich et al., 2020). Makalah tersebut menyarankan penggunaan nilai awal yang kecil untuk *dropout*, sehingga peneliti menetapkan nilai sebesar 0.1. Nilai ini lebih rendah dibandingkan nilai *default* dalam dokumentasi PyTorch, yaitu 0.5. Selain itu, makalah tersebut juga menyoroti dampak *weight decay* terhadap kemampuan model untuk melakukan generalisasi.

Tabel 4.7 Eksperimen tahap keempat

Nomor	Patch Size (px)	Input Image Size (px)	Epoch	Scheduler	Backbone	Batch Size	Learning Rate	Weight Decay	Dropout	Hasil						
										Train Loss	Val Loss	Saved Best Model	Waktu pelatihan	Test mIoU	Test mAP	Waktu Inference (s)
1	320x320	320x320	80	25	ResNet34	64	0.001	0.0001	0.1	0.55	0.53	Epoch 31	35 Menit/Epoch	0.38	0.32	0.08
2	512x512	512x512	80	25	ResNet34	64	0.001	0.0001	0.1	0.46	0.53	Epoch 59	38 Menit/Epoch	0.37	0.32	0.09

Nilai *weight decay* yang terlalu tinggi dapat membuat model sulit belajar, sedangkan nilai yang terlalu rendah dapat mengurangi efektivitas regularisasi. Oleh karena itu, peneliti memilih nilai *weight decay* moderat yaitu 0.0001. Nilai ini dipilih untuk mencapai keseimbangan antara efektivitas regularisasi dan kemampuan pembelajaran model. Nilai ini juga diambil didasari pada buku (Kuhn & Johnson, 2013) yang menjelaskan nilai *weight decay* biasanya berada pada skala logaritmik antara 0 dan 0.1, seperti 0.1, 0.001, 0.0001, dan seterusnya. Untuk hasil *inference time* disini tidak mengalami perbedaan yang signifikan dikarenakan *patch* yang diolah pada citra 512-1024 tidak sebanyak *patch* 320. Lalu penggunaan GPU yang mutakhir yaitu GPU L4 untuk keseluruhan *testing* juga menyebabkan selisih waktu *inference* yang kecil sesuai yang dijelaskan pada artikel (Huang et al., 2016). Lalu cara kerja dari GPU yang bersifat paralel untuk mengolah data sehingga menyebabkan *waktu inference* yang tidak memiliki selisih yang signifikan.

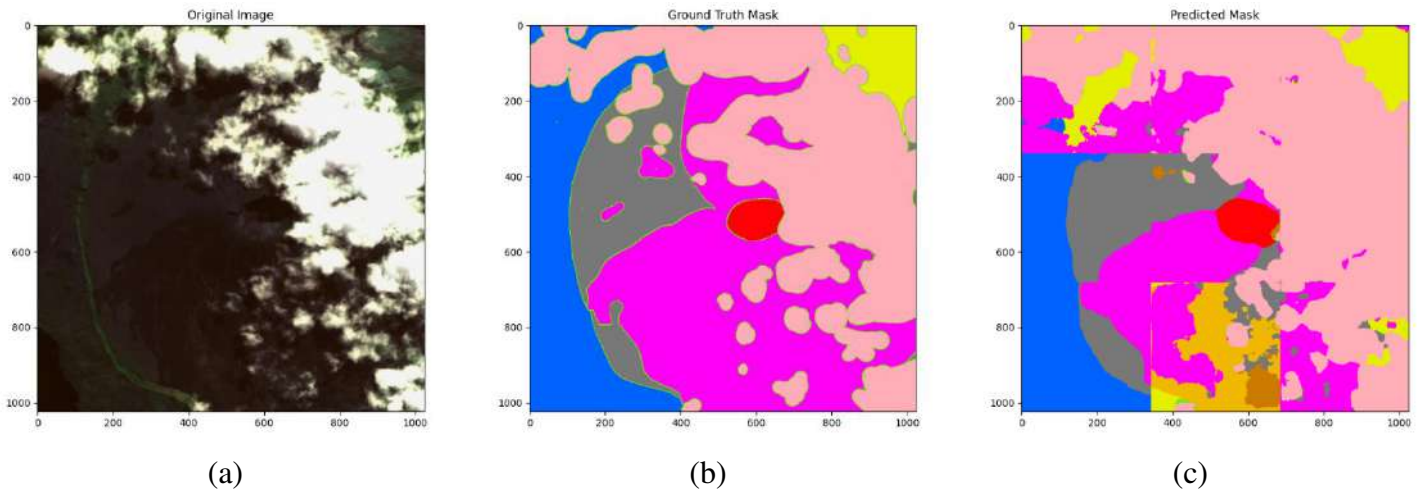
Dari hasil eksperimen tahap keempat, didapatkan konfigurasi *hyperparameter* terbaik pada percobaan satu dengan selisih *loss*, waktu pelatihan, dan hasil tes yang unggul. Penggunaan *weight decay* dan *dropout* terbukti sangat efektif dalam mengurangi *overfitting* pada model. Selisih antara *train loss* dan *validation loss* berhasil diturunkan hingga 0.02 melalui penerapan strategi ini. Sementara itu nilai mAP dan mIoU tidak mengalami perubahan karena kegagalan deteksi kelas “Lava Dome” untuk keseluruhan eksperimen.

Eksplorasi nilai parameter untuk *dropout* dan *weight decay* tidak dilanjutkan oleh peneliti dikarenakan keterbatasan waktu penelitian. Konfigurasi *hyperparameter* akhir yang digunakan untuk pelatihan model dirangkum dalam Tabel 4.10. Konfigurasi ini diterapkan pada pelatihan model pada tahap ke lima dimana peneliti menambahkan 10 citra seperti yang dijelaskan pada Subbab 4.1. Hasil percobaan ini dirangkum pada Tabel 4.8.

Di Tabel 4.8, terlihat bahwa penambahan citra gunung api berhasil menurunkan nilai *validation loss* hingga mencapai 0.50. *Best model* kali ini disimpan lebih lama, yaitu pada *epoch* ke-50, berbeda dengan eksperimen tahap keempat yang menyimpannya di *epoch* ke-31. Penambahan citra pada proses pelatihan model ini juga membuat waktu pelatihan meningkat sekitar 5 menit/*epoch*.

Pada data tes, nilai mIoU meningkat menjadi 0.41, sementara mAP berada di angka 0.36. Peningkatan ini disebabkan oleh kemampuan model yang berhasil mendeteksi kelas *lava dome*. Selain itu, penambahan citra pada data tes turut berkontribusi terhadap peningkatan tersebut dalam mendeteksi kelas *lava dome*. Contoh citra ini ditampilkan pada Gambar 4.14 yang

merupakan data tes tambahan. Citra tersebut dipilih karena memiliki kelas *lava dome* yang terlihat jelas.



Gambar 4.14 Sampel citra tes *lava dome*, (a) Citra original, (b) *Ground truth mask*, (c) *Predicted mask*

Pada Gambar 4.14, terlihat bahwa model berhasil mendeteksi kelas *lava dome* yang ditandai dengan warna merah pada *predicted mask*. Penambahan gambar terbukti mampu meningkatkan kemampuan generalisasi model. Namun, eksperimen ini masih berlanjut ke tahap keenam, yaitu fokus pada menangani ketidakseimbangan *dataset* dengan menerapkan pembobotan kelas. Pembobotan kelas ini peneliti menggunakan library sickit learn dan pembobotan tersebut dilakukan secara otomatis menggunakan fungsi `compute_class_weight`.

Hasil eksperimen tahap keenam dirangkum dalam Tabel 4.9. Penambahan pembobotan kelas dan modifikasi fungsi *loss* pada proses pelatihan tidak menghasilkan peningkatan nilai mIoU dan mAP yang signifikan pada data tes. Hal ini disebabkan oleh nilai *loss* pada eksperimen tahap keenam yang tidak lebih rendah dibandingkan *best model* dari eksperimen tahap kelima. Meskipun demikian, pada tahap ini kelas *lava dome* masih dapat terdeteksi oleh model.

Tabel 4.8 Eksperimen tahap kelima

Nomor	Patch Size (px)	Input Image Size (px)	Epoch	Scheduler	Backbone	Batch Size	Learning Rate	Weight Decay	Dropout	Hasil						
										Train Loss	Val Loss	Saved Best Model	Waktu pelatihan	Test mIoU	Test mAP	Waktu Inference (s)
1	320x320	320x320	80	25	ResNet34	64	0.001	0.0001	0.1	0.45	0.50	Epoch 58	40.5 Menit/Epoch	0.41	0.36	0.08

Tabel 4.9 Eksperimen tahap keenam

Nomor	Patch Size (px)	Input Image Size (px)	Epoch	Scheduler	Backbone	Batch Size	Learning Rate	Weight Decay	Dropout	Hasil						
										Train Loss	Val Loss	Saved Best Model	Waktu pelatihan	Test mIoU	Test mAP	Waktu Inference (s)
1	320x320	320x320	80	25	ResNet34	64	0.001	0.0001	0.1	0.51	0.55	Epoch 77	40.5 Menit/Epoch	0.39	0.34	0.08

Tabel 4.10 Konfigurasi *hyperparameter* terbaik

Parameter	Nilai Terbaik
Patch Size	320
Input Image Size	320
Epoch	80
Scheduler	25
Backbone	ResNet34
Batch Size	64
Learning Rate	0.001
Weight Decay	0.0001
Dropout	0.1

4.6 Pelatihan Model

Proses pelatihan model DeepLabV3+ dilakukan menggunakan dua platform utama yaitu Google Colab dan Hanabi server milik Universitas Islam Indonesia. Keduanya berjalan di Jupyter Notebook sehingga kompatibel dan tidak memerlukan banyak penyesuaian. Kedua platform ini juga dilengkapi dengan perangkat keras mumpuni untuk mendukung kebutuhan pelatihan. Pelatihan ini dibagi dalam tiga skenario citra dengan konfigurasi *hyperparameter* akhir pada Tabel 4.10. Perbedaan dari ketiga skenario ini dijabarkan dalam Tabel 4.11. Pada skenario ketiga, pelatihan dilakukan dengan menambahkan pembobotan kelas ke dalam fungsi cross-entropy loss. Kemudian, fungsi *loss* ini digabungkan dengan *dice loss* didasari penelitian (Galdran et al., 2022). Penelitian tersebut mengungkapkan bahwa kombinasi kedua fungsi *loss* ini mampu meningkatkan robusitas model terhadap ketidakseimbangan kelas dalam *dataset* pada tugas segmentasi gambar.

Tabel 4.11 Skenario pelatihan

Skenario	Jumlah Citra	Pembobotan Kelas	Fungsi Loss
1	62	-	cross-entropy loss
2	72	-	cross-entropy loss

3	72	Dengan Pembobotan Kelas	Gabungan cross-entropy loss dan Dice Loss
---	----	-------------------------	---

Pada Google Colab, proses pelatihan menggunakan GPU NVIDIA T4 dengan VRAM sebesar 15 GB. Sementara pada Hanabi server, pelatihan dilakukan menggunakan GPU Tesla V100-SXM2 yang memiliki VRAM hingga 32 GB. Untuk proses *testing* keseluruhan percobaan menggunakan Colab GPU L4 dengan VRAM 24 GB. Keunggulan lainnya adalah ketersediaan RAM besar pada kedua platform, tentunya sangat diperlukan untuk melatih model menggunakan *framework* PyTorch. Hal ini membuat proses pelatihan lebih efisien dan mampu menangani model dengan kompleksitas tinggi. Sebelum memulai pelatihan model, dilakukan terlebih dahulu langkah-langkah persiapan lingkungan kerja. Untuk pelatihan model DeepLabV3+, terdapat beberapa tahapan penting yang harus dilalui seperti pengaturan perangkat (*device*) yang akan digunakan serta instalasi *library* yang dibutuhkan. Pengaturan device dapat dilihat pada Gambar 4.15. Untuk keseluruhan file dan hasil metrik pada tahap pelatihan dapat dilihat pada lampiran.

```
# Set device
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(f'Using device: {device}')
torch.backends.cudnn.deterministic = True
torch.backends.cudnn.benchmark = True
torch.backends.cudnn.enabled = True
```

Gambar 4.15 Kode pengaturan *device*

Pada Gambar 4.15 menjelaskan bahwa proses pelatihan model didefinisikan untuk dijalankan menggunakan perangkat keras berbasis *Graphics Processing Unit* (GPU) apabila tersedia. Penggunaan GPU ini berfungsi untuk mempercepat proses komputasi. Penentuan perangkat dilakukan secara otomatis menggunakan fungsi `torch.device` yang memilih GPU jika tersedia atau CPU jika GPU tidak tersedia. Selain itu, konfigurasi cuDNN diaktifkan untuk memanfaatkan kemampuan *library* NVIDIA CUDA Deep Neural Network (cuDNN) yang memberikan optimisasi algoritma berdasarkan perangkat keras yang digunakan. Pengaturan deterministik juga diterapkan untuk memastikan hasil pelatihan dapat direproduksi secara konsisten. Hal ini penting untuk menjaga validitas hasil eksperimen yang dilakukan.

Pengaturan selanjutnya yaitu instalasi *library* yang dibutuhkan dalam penelitian, ditampilkan dalam Gambar 4.16.

```
!pip install numpy==1.21.4
!pip install matplotlib==3.4.3
!pip install seaborn==0.11.2
!pip install scikit-learn==0.24.2
!pip install torch==1.10.0
!pip install torchvision==0.11.1
!pip install torchsummary==1.5.1
!pip install torchviz==0.0.2
!pip install opencv-python-headless==4.5.5.64
!pip install pillow==8.4.0
!pip install segmentation-models-pytorch==0.2.1
!pip install albumentations==1.1.0
!pip install patchify==0.2.3
!pip install tqdm
!pip install ipywidgets
!pip install tensorflow==2.4.0
!pip install numpy==1.19.2
!pip install typing-extensions==3.7.4
```

Gambar 4.16 Kode instalasi *library*

Gambar 4.16 menampilkan kode untuk menginstal pustaka-pustaka yang mendukung berbagai aspek dalam proyek ini termasuk *preprocessing* data, augmentasi citra, pelatihan model Deep Learning, evaluasi performa, dan visualisasi data. Kombinasi pustaka torch, segmentation-models-pytorch, dan albumentations berfungsi sebagai komponen utama dalam implementasi model segmentasi citra. Perlu diperhatikan bahwa versi dari setiap pustaka ini dapat berubah sewaktu-waktu, dan kebutuhan instalasi dapat bervariasi tergantung pada konfigurasi komputer yang digunakan. Setelah *library* berhasil diinstal, dilanjutkan mengimpor *library* agar bisa digunakan. Divisualisasikan pada Gambar 4.17.

```
import torch
import segmentation_models_pytorch as smp
import torch.nn as nn
import torch.optim as optim
import numpy as np
import matplotlib.pyplot as plt
from sklearn.utils.class_weight import compute_class_weight
from collections import Counter
import os
from torch.utils.data import Dataset, DataLoader, ConcatDataset
import torchvision.transforms as transforms
import cv2
from PIL import Image
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from tqdm import tqdm
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay,
jaccard_score, precision_score, recall_score, f1_score
import seaborn as sns
import albumentations as A
```

```

from albumentations.pytorch import ToTensorV2
from torch.optim.lr_scheduler import ReduceLROnPlateau
from patchify import patchify, unpatchify
from torchsummary import summary
from torchviz import make_dot
import matplotlib.ticker as mticker
from IPython.display import display, HTML
import random

```

Gambar 4.17 Impor *library*

Gambar 4.17 memperlihatkan kode yang digunakan untuk memanfaatkan berbagai *library*. Dalam penelitian ini, sejumlah *library* digunakan untuk mendukung pelatihan dan evaluasi model segmentasi citra satelit. *Library* utama seperti PyTorch dan *segmentation_models_pytorch* berperan dalam pembangunan, pelatihan, serta pengujian model Deep Learning. Untuk meningkatkan kinerja model, *albumentations* dan *torchvision.transforms* digunakan dalam proses augmentasi data. Evaluasi metrik seperti IoU, *precision*, *recall*, dan *F1-score* dilakukan menggunakan *sklearn.metrics*. Hasil visualisasi ditampilkan melalui *matplotlib* dan *seaborn*, sedangkan *tqdm* dimanfaatkan untuk memantau perkembangan proses pelatihan. *Library* tambahan seperti *opencv-python* dan *PIL* digunakan untuk membaca dan memproses citra, sementara *torchviz* dan *torchsummary* membantu analisis mendalam terhadap struktur model. Kombinasi dari berbagai *library* ini memungkinkan eksperimen berjalan lancar mulai dari tahap *preprocessing* hingga evaluasi hasil.

4.6.1 Patching Citra

Dalam *framework* PyTorch peneliti perlu mendefinisikan kelas *dataset* terutama untuk menangani perlakuan khusus terhadap *dataset*. Oleh karena itu, peneliti melakukan modifikasi agar kelas tersebut dapat membaca file gambar dan *mask* secara tepat. Kelas ini dirancang untuk mempersiapkan data berupa gambar dan *mask* segmentasi yang sesuai dengan kebutuhan model Deep Learning. Kode ini akan disajikan pada Gambar 4.18.

```

import numpy as np
from sklearn.preprocessing import MinMaxScaler
import cv2
from torch.utils.data import Dataset
from patchify import patchify
from PIL import Image
class SegmentationDataset(Dataset):
    def __init__(self, image_paths, mask_paths, patch_size=320, transform=None):
        self.image_paths = image_paths
        self.mask_paths = mask_paths
        self.patch_size = patch_size
        self.transform = transform

```

```

        self.scaler = MinMaxScaler()
        self.image_patches = []
        self.mask_patches = []
        self.labels = [] # Label utama untuk setiap patch, digunakan untuk
stratifikasi
        self.create_patches()

    def create_patches(self):
        for img_path, mask_path in zip(self.image_paths, self.mask_paths):
            image = cv2.imread(img_path)
            image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
            mask = cv2.imread(mask_path, 1)
            mask = cv2.cvtColor(mask, cv2.COLOR_BGR2RGB)

            # Crop images and masks to the nearest size divisible by patch_size
            SIZE_X = (image.shape[1] // self.patch_size) * self.patch_size
            SIZE_Y = (image.shape[0] // self.patch_size) * self.patch_size
            image = Image.fromarray(image).crop((0, 0, SIZE_X, SIZE_Y))
            mask = Image.fromarray(mask).crop((0, 0, SIZE_X, SIZE_Y))
            image = np.array(image)
            mask = np.array(mask)

            # Patchify images and masks
            patches_img = patchify(image, (self.patch_size, self.patch_size, 3),
step=self.patch_size)
            patches_mask = patchify(mask, (self.patch_size, self.patch_size, 3),
step=self.patch_size)

            for i in range(patches_img.shape[0]):
                for j in range(patches_img.shape[1]):
                    patch_img = patches_img[i, j, :, :, :][0]
                    patch_mask = patches_mask[i, j, :, :, :][0]

                    # Scale the image patch
                    patch_img = self.scaler.fit_transform(patch_img.reshape(-1,
patch_img.shape[-1])).reshape(patch_img.shape)

                    # Convert RGB mask to label mask
                    patch_label_mask = self.rgb_to_2D_label(patch_mask)

                    # Get Label
                    unique_labels = np.unique(patch_label_mask)
                    primary_label = unique_labels[0] if len(unique_labels) == 1
else unique_labels[1]

                    # Append data to lists
                    self.image_patches.append(patch_img)
                    self.mask_patches.append(torch.tensor(patch_label_mask,
dtype=torch.long))
                    self.labels.append(primary_label) # Menyimpan label untuk
stratifikasi

    def __len__(self):
        return len(self.image_patches)

    def __getitem__(self, idx):
        return self.image_patches[idx], self.mask_patches[idx]

    def rgb_to_2D_label(self, label):
        # Updated color dictionary
        label_colors = {
            'Cropland': [112, 240, 0],
            'Permukiman': [184, 0, 240],
            'Air': [240, 184, 0],
            'Area Terbuka': [60, 0, 138],
            'Jalan': [0, 228, 240],
            'Vegetasi': [228, 240, 0],

```

```

        'Herbaceous Vegetation': [0, 98, 255],
        'Lava Dome': [255, 0, 4],
        'Jalur Lahar': [65, 0, 204],
        'Awan Panas': [204, 122, 0],
        'Endapan Material Vulkanik Baru': [255, 0, 247],
        'Endapan Material Vulkanik Lama': [120, 120, 120],
        'Awan': [255, 173, 183],
        'Lava Flow': [122, 0, 14]
    }

    label_seg = np.zeros(label.shape[:2], dtype=np.uint8)
    for label_name, rgb in label_colors.items():
        label_seg[np.all(label == rgb, axis=-1)] =
list(label_colors.keys()).index(label_name)
    return label_seg

```

Gambar 4.18 Kode kelas *dataset*

Kelas pada Gambar 4.18 dibuat untuk memecah citra dan *mask* beresolusi tinggi menjadi *patch* kecil dengan dimensi sementara sebesar 320x320 piksel. Setelah proses pemotongan citra, dilanjutkan dengan tahap normalisasi menggunakan metode MinMaxScaler. Nilai yang dinormalisasi adalah *pixel value* yang bernilai 0 sampai 255 menjadi rentang [0,1]. Teknik normalisasi ini dilakukan guna meningkatkan stabilitas dan mempercepat proses pelatihan (Sane & Agrawal, 2017). *Mask* segmentasi dalam format RGB kemudian dikonversi menjadi *array* label numerik 2D agar dapat diolah oleh model dengan lebih mudah. Dalam proses ini, setiap piksel diberi nilai kelas sesuai dengan label yang didefinisikan pada Tabel 3.3.

Proses dimulai dengan fungsi `__init__` yang bertugas menginisialisasi *dataset* menggunakan parameter seperti *image_paths*, *mask_paths*, ukuran *patch* (*patch_size*), dan transformasi (*transform*). Fungsi *create_patches* bertanggung jawab untuk memotong citra dan *mask* agar ukurannya sesuai dengan kelipatan *patch*. Setelah citra dipotong menjadi ukuran yang habis dibagi ukuran *patch* menggunakan pustaka *patchify*, citra dan *mask* yang besar diubah menjadi *patch-patch* kecil, lalu menormalisasi *patch* citra, dan yang terakhir mengkonversi *mask* RGB menjadi format numerik menggunakan *color dictionary*. Setelah itu menyimpan label utama dari setiap *patch* untuk mendukung proses stratifikasi data pada tahapan pembagian *data train* dan *data validation*.

Konversi *mask* RGB menjadi format numerik dilakukan melalui fungsi *rgb_to_2D_label*. Caranya dengan mencocokkan setiap piksel dalam *mask* RGB dengan label numerik berdasarkan *color dictionary* yang telah didefinisikan sebelumnya. Fungsi `__getitem__` digunakan untuk mengembalikan pasangan citra dan *mask* berdasarkan indeks tertentu. Sedangkan fungsi `__len__` memberikan jumlah total *patch* yang tersedia dalam *dataset*.

Pendekatan ini memungkinkan pelatihan model secara efisien meskipun menggunakan citra dengan resolusi tinggi.

Setelah pendefinisian kelas *dataset*, lalu diikuti tahap selanjutnya yaitu *patching* citra besar menjadi *patch-patch* dengan ukuran kecil. Dalam penentuan ukuran *patch size image* diambil dari Tabel 4.8. *Patch image* ini akan menjadi *input image size* ke dalam model. *Input size* juga disesuaikan dengan Tabel 4.10 yaitu 320x320. Kode proses *patching* citra ditampilkan pada Gambar 4.19.

```
# Adjust the paths to your dataset location
base_dataset_path = 'extracted_dataset/AllDataset'
tile_paths = ['Tile1', 'Tile2']

# Collect all image and mask paths
image_paths = []
mask_paths = []

for tile in tile_paths:
    dataset_path = os.path.join(base_dataset_path, tile)
    for path, subdirs, files in os.walk(dataset_path):
        dirname = path.split(os.path.sep)[-1]
        if dirname == 'images':
            for image_name in files:
                if image_name.endswith(".jpg"):
                    image_path = os.path.join(path, image_name)
                    mask_path = os.path.join(path.replace('images', 'masks'),
                    image_name.replace('.jpg', '.png'))
                    if os.path.exists(mask_path):
                        image_paths.append(image_path)
                        mask_paths.append(mask_path)

# Create the dataset
dataset = SegmentationDataset(image_paths=image_paths, mask_paths=mask_paths,
patch_size=320)

print(f"Number of images: {len(image_paths)}")
print(f"Number of masks: {len(mask_paths)}")
print(f"Number of image patches: {len(dataset.image_patches)}")
print(f"Number of mask patches: {len(dataset.mask_patches)}")
```

Gambar 4.19 Kode *patching* gambar

Gambar 4.19 adalah kode proses pemotongan citra beresolusi besar. Pengumpulan data dilakukan dengan menavigasi direktori *dataset* yang diorganisasi dalam struktur folder Tile1 dan Tile2. Proses pengolahan dilakukan secara iteratif menggunakan fungsi *os.walk()* untuk mendeteksi semua file citra berformat *.jpg* di Subfolder *images*, sekaligus memastikan keberadaan file *mask* yang sesuai di subfolder *masks* dalam format *.png*. Proses ini memanggil kelas pada Gambar 4.18 dan mengganti parameternya menjadi 320x320 pixel. Pada akhirnya *patch* itu dikumpulkan dalam variabel “*dataset*” dengan *patch* berukuran 320x320 piksel. Data mengenai jumlah citra, *mask*, serta *patch* yang dihasilkan dari proses ini dihitung dan

dilaporkan untuk memverifikasi konsistensi dan kelengkapan *dataset*. Validasi ini bertujuan untuk menjamin keakuratan pengolahan *dataset* agar dapat dimanfaatkan secara optimal dalam model segmentasi citra satelit. Hasil *patching* untuk 62 citra pada proses pelatihan ditampilkan pada Gambar 4.20. Sedangkan untuk 72 citra ditampilkan pada Gambar 4.21.

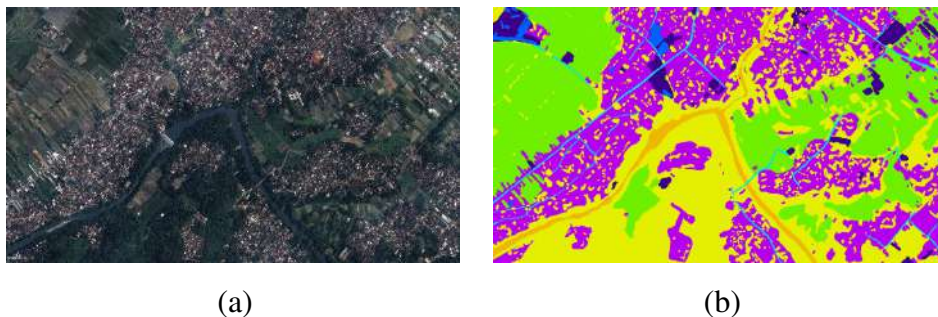
```
Number of images: 62
Number of masks: 62
Number of image patches: 2496
Number of mask patches: 2496
```

Gambar 4.20 Hasil proses *patching* 62 citra

```
Number of images: 72
Number of masks: 72
Number of image patches: 2613
Number of mask patches: 2613
```

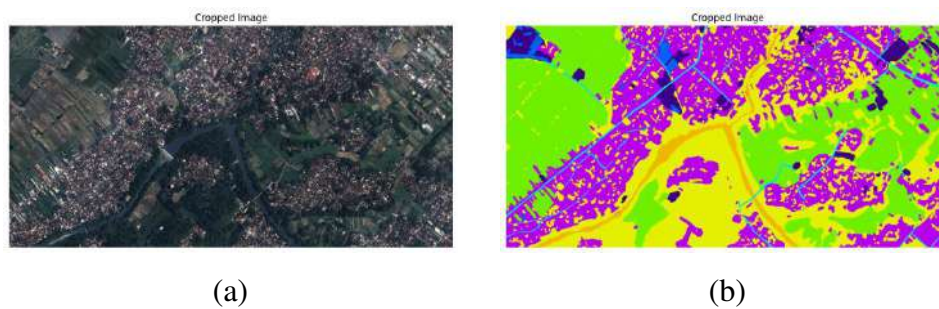
Gambar 4.21 Hasil proses *patching* 72 citra

Gambar 4.20 menunjukkan bahwa dari 62 citra yang digunakan dalam *dataset*, setelah melalui proses *patching* dihasilkan sebanyak 2496 patch berukuran 320x320 piksel. Sedangkan untuk 72 citra menghasilkan 2613 *patches*. Satu citra besar beresolusi 3840x2160 dapat menghasilkan 72 *patches*, 1920x1080 menghasilkan 18 *patches*, dan resolusi 1024x1024 berhasil memperoleh 9 *patches*. Gambar 4.22 menampilkan citra tutupan lahan beserta pasangan *mask*-nya yang akan diterapkan proses *patching*. Ukuran citra tersebut bernilai 3840x2160 piksel.



Gambar 4.22 Citra beresolusi besar, (a) Citra tutupan lahan, (b) Pasangan *mask*

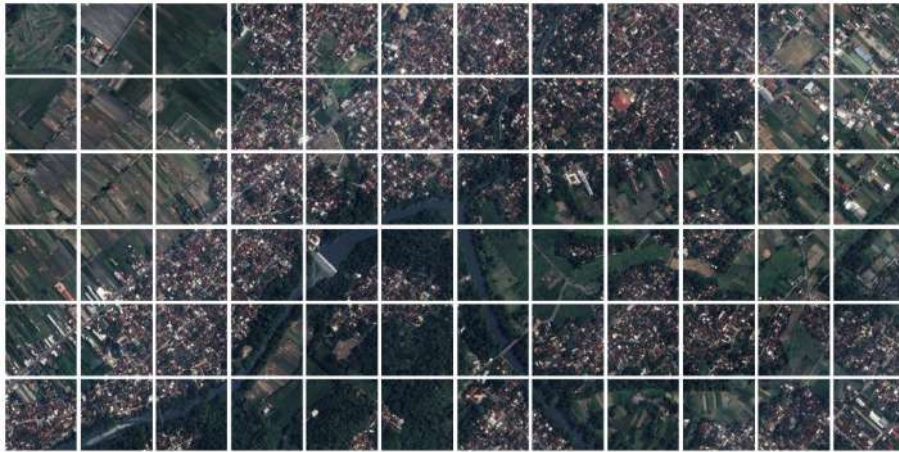
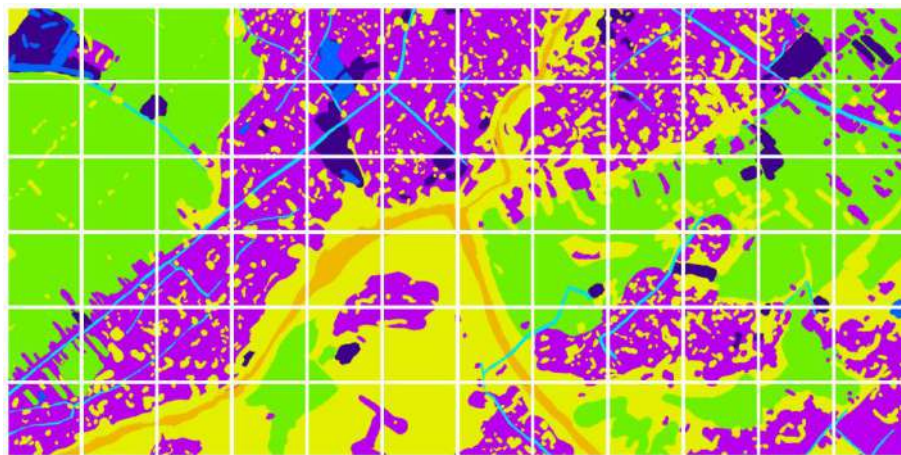
Sebelum masuk proses *patching*, citra dan *mask* terlebih dahulu melalui proses *cropping* menjadi ukuran yang habis dibagi *patch size*. Setelah melalui proses *cropping* citra berukuran 3840x2160 berubah menjadi 3840x1920, citra 1920x1080 berubah menjadi 1920x960, dan 1024x1024 berubah menjadi 960x960. Visualisasi proses *cropping* dari citra asli ditampilkan pada Gambar 4.23. Contoh pasangan citra beresolusi tinggi dan *mask* yang telah diproses menjadi *patch* ditampilkan pada Gambar 4.24. Visualisasi semua *patch* kecil dari citra beresolusi besar ditampilkan pada Gambar 4.25 dan pasangan *mask* pada Gambar 4.26.



Gambar 4.23 Hasil proses *cropping*, (a) Citra tutupan lahan, (b) Pasangan *mask*



Gambar 4.24 *Patch image* beserta labelnya

Gambar 4.25 Hasil *patching* citraGambar 4.26 Hasil *patching mask*

Pada Gambar 4.25 terlihat bahwa citra dan mask telah berhasil dipecah menjadi patch dengan resolusi setiap patch menjadi 320x320 piksel yang sebagaimana ditunjukkan pada grafik pada Gambar 2.24. Selain itu, mask yang sebelumnya dalam format RGB telah dikonversi menjadi label 2D dan kini divisualisasikan dalam bentuk grayscale atau hitam-putih. Citra dan mask yang ditampilkan juga merupakan pasangan saling berkorespondensi. Kelas-kelas pada citra terdiri dari kelas “Cropland” ditandai dengan warna hitam, kelas “Permukiman” ditandai dengan warna abu-abu, dan kelas “Vegetasi” ditandai dengan warna putih.

4.6.2 *Split Patch Data*

Setelah melalui proses *patching*, dihasilkan citra yang telah terpotong. Data tersebut dibagi dengan rasio 80% untuk *training* dan 20% untuk validasi. Kode pembagian *dataset* ini ditampilkan pada Gambar 4.27.

```

# Convert labels to one-hot encoding for MultilabelStratifiedShuffleSplit
labels = np.array(dataset.labels).reshape(-1, 1)
one_hot_encoder = OneHotEncoder(sparse=False, categories='auto')
labels_one_hot = one_hot_encoder.fit_transform(labels)

# Use MultilabelStratifiedShuffleSplit for stratified split
stratifier = MultilabelStratifiedShuffleSplit(test_size=0.2, random_state=61)
train_idx, val_idx = next(stratifier.split(dataset.image_patches,
labels_one_hot))

# Split the dataset using the stratified indices
train_images = np.array([dataset.image_patches[i].numpy() if
isinstance(dataset.image_patches[i], torch.Tensor) else dataset.image_patches[i]
for i in train_idx])
val_images = np.array([dataset.image_patches[i].numpy() if
isinstance(dataset.image_patches[i], torch.Tensor) else dataset.image_patches[i]
for i in val_idx])
train_masks = np.array([dataset.mask_patches[i].numpy() if
isinstance(dataset.mask_patches[i], torch.Tensor) else dataset.mask_patches[i]
for i in train_idx])
val_masks = np.array([dataset.mask_patches[i].numpy() if
isinstance(dataset.mask_patches[i], torch.Tensor) else dataset.mask_patches[i]
for i in val_idx])

# Print dataset statistics
print(f"Number of training patches: {len(train_images)}")
print(f"Number of validation patches: {len(val_images)}")

```

Gambar 4.27 Kode membagi *dataset*

Gambar 4.27 merupakan kode untuk memisahkan *image patch*. Kode ini digunakan untuk membagi keseluruhan *patch* menjadi data pelatihan dan validasi dengan menggunakan teknik stratifikasi. Penggunaan teknik ini bertujuan untuk menjaga distribusi kelas pada kedua *subset* ini tetap seimbang. Proses dimulai dengan mengkonversi label pada *dataset* ke dalam format yang lebih mudah diolah, yaitu *one-hot encoding*. Format ini memungkinkan setiap kelas label direpresentasikan secara unik sehingga pembagian data dapat dilakukan dengan lebih akurat.

Selanjutnya, pembagian *dataset* dilakukan menggunakan metode *Multilabel Stratified Shuffle Split* dengan proporsi 80% untuk data pelatihan dan 20% untuk data validasi. Metode ini memastikan bahwa setiap kelas dalam *dataset* terwakili secara proporsional dalam kedua *subset*. Setelah pembagian, data citra dan *mask* dipisahkan sesuai dengan indeks yang dihasilkan. Untuk mengetahui pasangan citra dan *mask* sesuai dapat menggunakan kode pada Gambar 4.28, dengan hasil visualisasi akan sama persis seperti pada Gambar 4.24.

```

import numpy as np

# Set random seed for reproducibility
np.random.seed(42)

```

```

# Get the number of sampels in the validation dataset
num_val_sampels = len(val_dataset)

# Generate random indices to shuffle the dataset
shuffled_indices = np.random.permutation(num_val_sampels)

# Select a subset of indices to visualize
selected_indices = shuffled_indices[:num_sampels_to_show]

# Visualize images directly from the shuffled validation dataset
for i in selected_indices:
    # Get the image and mask from the shuffled validation dataset
    val_image, val_mask = val_dataset[i]

    # Convert image and mask to numpy for display
    val_image = val_image.numpy().transpose(1, 2, 0) # HWC format for visualization
    val_image = np.clip(val_image, 0, 1) # Clip values between 0 and 1 if needed
    val_mask = val_mask.numpy() # Convert mask to numpy

    # Display the image and mask
    plt.figure(figsize=(12, 6))

    # Display validation image
    plt.subplot(1, 2, 1)
    plt.imshow(val_image)
    plt.title("Validation Image")

    # Display validation mask
    plt.subplot(1, 2, 2)
    plt.imshow(val_mask, cmap='gray') # Use a colormap like 'gray' for masks
    plt.title("Validation Mask (2D)")

plt.tight_layout()
plt.show()

```

Gambar 4.28 Kode untuk melihat pasangan citra dan *mask*

Tahap akhir yaitu perhitungan jumlah data pada masing-masing *subset* untuk memverifikasi keberhasilan proses pembagian. Pendekatan ini sangat penting untuk memastikan bahwa model dilatih dapat bekerja dengan baik pada semua kelas dalam *dataset*, terutama ketika *dataset* memiliki distribusi kelas tidak merata. Hasil akhir dari *split train* dan *validation* dari 62 citra dapat dilihat pada Gambar 4.29. Untuk hasil pembagian 72 citra dapat dilihat pada Gambar 2.30.

```

Number of training patches: 1996
Number of validation patches: 500

```

Gambar 4.29 Hasil pembagian data 62 citra

Number of training patches: 2090
 Number of validation patches: 523

Gambar 4.30 Hasil pembagian data 72 citra

Gambar 4.29 menampilkan jumlah *patch* yang sudah dipisahkan. Dari 2496 *patch*, dihasilkan 1996 *patches* untuk *training* dan 500 *patches* untuk *validation*. Pada Gambar 4.30 mengalami peningkatan jumlah *patch* dikarenakan adanya penambahan citra pada proses pelatihan. Tahap ini juga peneliti memaparkan class distribution pada dataset dengan menampilkan jumlah objek pada setiap kelas. Kode untuk menampilkan jumlah objek dapat dilihat pada Gambar 4.31.

```
import numpy as np
from scipy.ndimage import label, find_objects
import pandas as pd

# Define the mapping of class indices to class names
class_labels = {
    0: 'Cropland',
    1: 'Permukiman',
    2: 'Air',
    3: 'Area Terbuka',
    4: 'Jalan',
    5: 'Vegetasi',
    6: 'Herbacus Vegetation',
    7: 'Lava Dome',
    8: 'Jalur Lahar',
    9: 'Awan Panas',
    10: 'Endapan Material Vulkanik Baru',
    11: 'Endapan Material Vulkanik Lama',
    12: 'Awan',
    13: 'Lava Flow'
}

# Initialize total object counts for each class
total_counts = {class_name: 0 for class_name in class_labels.values()}

# Define a minimum size threshold for filtering
MIN_SIZE_THRESHOLD = 500 # Adjust this value as needed

# Function to calculate total object count per class for 2D label masks with
# filtering
def calculate_total_object_count_with_filtering(dataset, patch_size):
    global total_counts
    for _, mask in dataset:
        # Convert mask to numpy array if it's a tensor
        if not isinstance(mask, np.ndarray):
            mask = mask.numpy()

        # Ensure the mask is divided into patches of the desired size
        h, w = mask.shape
        for i in range(0, h, patch_size):
            for j in range(0, w, patch_size):
                patch = mask[i:i + patch_size, j:j + patch_size]

                # Process each class in the label dictionary
                unique_labels = np.unique(patch)
```

```

for label_val in unique_labels:
    # Create binary mask for the specific class
    binary_mask = (patch == label_val).astype(np.uint8)

    # Calculate connected components (objects)
    labeled_mask, num_features = label(binary_mask)

    # Filter objects by size
    object_slices = find_objects(labeled_mask)
    filtered_object_count = 0

    for k, obj_slice in enumerate(object_slices):
        if obj_slice is not None:
            # Calculate object size
            obj_size = np.sum(labeled_mask[obj_slice] == k + 1)
            if obj_size >= MIN_SIZE_THRESHOLD:
                filtered_object_count += 1

    # Update the total count for the class
    class_name = class_labels.get(label_val, f"Unknown Class {label_val}")

    total_counts[class_name] += filtered_object_count

# Example dataset to simulate (replace `dataset` with your actual dataset object)
# Example: dataset = SegmentationDataset(image_paths, mask_paths)
patch_size = 320 # Set the patch size to 320
calculate_total_object_count_with_filtering(dataset, patch_size)

# Convert results to DataFrame for display and save to CSV
total_counts_df = pd.DataFrame({
    "Class Name": list(class_labels.values()), # Maintain order from class_labels
    "Total Object Count": [total_counts[class_name] for class_name in
class_labels.values()]
})

# Save to CSV
total_counts_df.to_csv("total_object_count_filtered.csv", index=False)

# Display the DataFrame
print(total_counts_df)

```

Gambar 4.31 Kode menampilkan jumlah objek

Kode pada Gambar 4.31 bertujuan untuk menghitung total jumlah objek pada setiap kelas dalam *dataset*. Proses ini dilakukan dengan mendeteksi objek terhubung (*connected components*) pada setiap kelas dari keseluruhan *patch*. Hasil analisis ini kemudian disimpan dalam format tabel untuk mempermudah visualisasi distribusi objek pada setiap kelas. Jumlah untuk 62 citra ini divisualisasikan pada Gambar 4.32. Untuk 72 citra divisualisasikan pada Gambar 3.33.

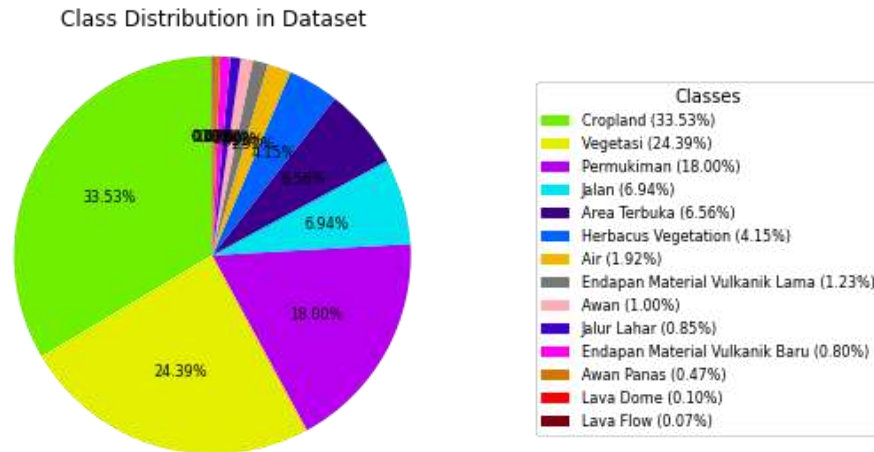
Class Name	Total Object Count
Cropland	5585
Permukiman	2998
Air	320
Area Terbuka	1093
Jalan	1156
Vegetasi	4063
Herbacus Vegetation	692
Lava Dome	17
Jalur Lahar	141
Awan Panas	78
Endapan Material Vulkanik Baru	133
Endapan Material Vulkanik Lama	205
Awan	166
Lava Flow	12

Gambar 4.32 Jumlah objek dalam *dataset* untuk 62 citra

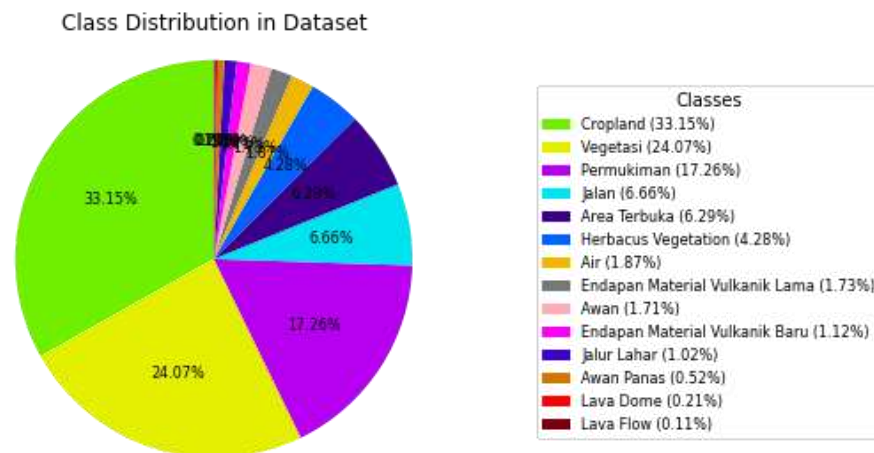
Class Name	Total Object Count
Cropland	5758
Permukiman	2998
Air	324
Area Terbuka	1093
Jalan	1156
Vegetasi	4181
Herbacus Vegetation	744
Lava Dome	36
Jalur Lahar	177
Awan Panas	91
Endapan Material Vulkanik Baru	195
Endapan Material Vulkanik Lama	301
Awan	297
Lava Flow	19

Gambar 4.33 Jumlah objek dalam *dataset* untuk 72 citra

Dari Gambar 4.32 dan Gambar 4.33 terdapat indikasi ketidakseimbangan jumlah kelas pada *dataset* yang di mana objek seperti *lava dome*, awan panas, dan *lava flow* memiliki jumlah objek kurang dari seratus. Dengan adanya penambahan citra juga tidak mengalami perubahan yang signifikan untuk beberapa kelas. Tentunya masalah ini dapat mempengaruhi hasil dari pelatihan model nantinya. Untuk memberikan pemahaman lebih lanjut, peneliti kemudian menyajikan *pie chart* dari jumlah objek pada *dataset*. Diagram jumlah objek untuk 62 citra ini dapat dilihat pada Gambar 4.34. Untuk 72 citra dapat dilihat pada Gambar 4.35.



Gambar 4.34 Distribusi kelas 62 citra



Gambar 4.35 Distribusi kelas 72 citra

Dari Gambar 4.29 bahwa indikasi ketidakseimbangan kelas dinyatakan valid. Dari 14 kelas dapat ditarik informasi bahwa 11 kelas memiliki jumlah objek dibawah 20%. Dengan adanya masalah ini, perlu untuk memastikan bahwa setiap kelas terwakili dalam data *training* dan *validation*, dengan dilakukan pengecekan jumlah piksel untuk setiap kelas. Proses ini menggunakan label dari *training patches* dan *validation patches*. Kode implementasi untuk proses ini disajikan pada Gambar 4.36.

```
# Function to count class distribution in a set of masks
def count_class_distribution(masks):
    class_counts = Counter()
    for mask in masks:
        unique, counts = np.unique(mask, return_counts=True)
        class_counts.update(dict(zip(unique, counts)))
```

```

    return class_counts

# Count class distribution in train and validation masks
train_class_counts = count_class_distribution(train_masks)
val_class_counts = count_class_distribution(val_masks)

# Combine into a DataFrame for easy comparison
classes = sorted(set(train_class_counts.keys()).union(set(val_class_counts.keys())))
train_counts = [train_class_counts.get(cls, 0) for cls in classes]
val_counts = [val_class_counts.get(cls, 0) for cls in classes]

df = pd.DataFrame({
    'Class': classes,
    'Train Count': train_counts,
    'Validation Count': val_counts
})

# Display class distribution table
print("Class Distribution in Train and Validation Masks")
print(df)

```

Gambar 4.36 Kode distribusi kelas

Fungsi pada Gambar 4.36 digunakan untuk menghitung jumlah piksel pada *dataset* dengan cara menganalisis jumlah piksel yang termasuk ke dalam setiap kelas pada *mask* segmen. Proses ini dilakukan dengan memanfaatkan fungsi `count_class_distribution` yang menggunakan metode `np.unique` untuk menghitung jumlah piksel unik di setiap *mask*. Hasil perhitungan dari *mask* pada *dataset* pelatihan (`train_masks`) dan validasi (`val_masks`) kemudian dikombinasikan ke dalam sebuah tabel *DataFrame*. *DataFrame* ini menyajikan informasi jumlah piksel dari setiap kelas secara terperinci untuk *dataset* pelatihan dan validasi. Analisis distribusi ini penting untuk memastikan kesetimbangan data antar kelas yang secara langsung dapat mempengaruhi kinerja model segmentasi dalam mendeteksi setiap kelas secara akurat. Tabel distribusi kelas ini memberikan gambaran visual yang memudahkan untuk mengidentifikasi potensi ketidakseimbangan dalam data, sehingga langkah-langkah mitigasi seperti augmentasi data atau pembobotan kelas dapat dilakukan jika diperlukan. Jumlah piksel untuk 62 citra pada *mask* ditampilkan pada Gambar 4.37. Sedangkan untuk 72 citra ditampilkan pada Gambar 4.38.

Class	Train Count	Validation Count
0	49722662	12781908
1	54332513	12896227
2	9446530	2598491
3	12069890	2981823
4	7802275	1984609
5	52012406	13809775
6	7261154	1354298
7	134465	71567
8	352642	86053
9	1628535	348565
10	3349546	696131
11	3615756	796037
12	2630163	758271
13	31863	36245

Gambar 4.37 Jumlah piksel pada 62 citra

Class	Train Count	Validation Count
0	49502030	13404680
1	54353989	12874751
2	9948275	2212846
3	12356215	2695498
4	7719676	2067208
5	55349554	14332556
6	7581562	2039332
7	350589	80983
8	443506	128611
9	1537437	492069
10	4822504	1000822
11	5508368	1392375
12	4454788	832402
13	87507	1067

Gambar 4.38 Jumlah piksel pada 72 citra

Gambar 4.37 dan Gambar 4.38 menunjukkan jumlah piksel yang dihasilkan dari proses pembagian *dataset*. Hasil ini mengkonfirmasi bahwa pembagian data telah dilakukan dengan benar. indikasi ketidakseimbangan kelas ditunjukkan dengan beberapa kelas memiliki jumlah piksel yang sangat sedikit. Hal ini terlihat dari representasi jumlah piksel untuk seluruh kelas dalam data latih dan validasi. *Dataset* ini mencakup empat belas kelas yang diidentifikasi menggunakan indeks numerik mulai dari 0. Informasi kelas lebih lengkap disajikan dalam Tabel 4.12.

Tabel 4.12 Indeks kelas

Indeks	Kelas
0	Cropland
1	Permukiman
2	Air
3	Area Terbuka

4	Jalan
5	Vegetasi
6	Herbaceous Vegetation
7	Lava Dome
8	Jalur Lahar
9	Awan Panas
10	Endapan Material Vulkanik Baru
11	Endapan Material Vulkanik Lama
12	Awan
13	Lava Flow

4.6.3 Data Augmentation

Proses augmentasi data diterapkan pada *data train* untuk mengurangi risiko *overfitting* atau *underfitting*. Kedua hal tersebut dapat mempengaruhi kemampuan model dalam menggeneralisasi data. Teknik augmentasi ini dipilih karena jumlah citra dalam *dataset* terbatas, sehingga diperlukan penambahan variasi data dengan memodifikasi citra asli. Proses augmentasi dilakukan satu kali dan diterapkan secara acak pada seluruh *data train* menggunakan teknik seperti *flip*, *rotate*, blur, serta penyesuaian kecerahan dan kontras. Kode proses augmentasi ditampilkan pada Gambar 4.39.

```
class PatchDataset(Dataset):
    def __init__(self, images, masks, patch_size=256, transform=None,
augmentations=None, apply_augmentations=False):
        self.images = images
        self.masks = masks
        self.patch_size = patch_size
        self.transform = transform
        self.augmentations = augmentations
        self.apply_augmentations = apply_augmentations

    def __len__(self):
        return len(self.images)

    def __getitem__(self, idx):
        # Retrieve the image and mask and ensure they are in float32 for
compatibility with albumentations
        image = np.array(self.images[idx]).astype(np.float32) # Convert to
float32 for compatibility
        mask = np.array(self.masks[idx]).astype(np.float32) # Convert to
float32 for compatibility

        # Ensure the patch size is not larger than the image size
        if image.shape[0] < self.patch_size or image.shape[1] < self.patch_size:
            raise ValueError("Image size must be larger than patch size")

        # Apply augmentations if specified
```

```

        if self.apply_augmentations and self.augmentations:
            # Albumentations expects images and masks in float32
            augmented = self.augmentations(image=image, mask=mask)
            image = augmented['image']
            mask = augmented['mask']

        # Apply transformations (e.g., ToTensorV2) after augmentations
        if self.transform:
            transformed = self.transform(image=image, mask=mask)
            image = transformed['image'] # This will be a PyTorch tensor if
ToTensorV2 is used
            mask = transformed['mask'].long() # Convert mask to long for
compatibility with PyTorch

        return image, mask

# Image Augmentation
train_augmentations = A.Compose([
    A.VerticalFlip(p=0.5),
    A.HorizontalFlip(p=0.5),
    A.Rotate(limit=60, p=0.5),
    A.Blur(blur_limit=7, p=0.5),
    A.RandomRotate90(p=0.75),
    A.RandomBrightnessContrast(brightness_limit=0.3, contrast_limit=0.3, p=0.7),
])

# Define transformations for training and validation
train_transform = A.Compose([
    ToTensorV2() # Convert image and mask to tensor
])

val_transform = A.Compose([
    ToTensorV2() # Only convert to tensor, no augmentation
])

# Create original and augmented datasets
train_dataset_original = PatchDataset(train_images, train_masks, patch_size=320,
transform=train_transform, apply_augmentations=False)
train_dataset_augmented = PatchDataset(train_images, train_masks, patch_size=320,
transform=train_transform, augmentations=train_augmentations,
apply_augmentations=True)

# Combine original and augmented datasets
combined_train_dataset = ConcatDataset([train_dataset_original,
train_dataset_augmented])
val_dataset = PatchDataset(val_images, val_masks, patch_size=320,
transform=val_transform)

# Print dataset sizes
print(f"Number of training patches: {len(combined_train_dataset)}")
print(f"Number of validation patches: {len(val_dataset)}")

```

Gambar 4.39 Kode proses augmentasi

Gambar 4.39 adalah kode untuk memproses pembentukan *dataset* dengan tambahan augmentasi dalam penelitian ini. Pemrosesan ini melibatkan penggabungan data asli dan data yang telah melalui proses augmentasi. Untuk meningkatkan variasi data, augmentasi dilakukan pada *dataset* pelatihan menggunakan pustaka *Albumentations*, dengan teknik seperti *vertical flip*, *horizontal flip*, rotasi hingga 90 derajat, blur, serta penyesuaian kecerahan dan kontras. *Dataset* pelatihan kemudian dibagi menjadi dua yaitu *dataset* asli tanpa augmentasi dan *dataset*

dengan augmentasi. Kedua *dataset* ini digabungkan menggunakan kelas *ConcatDataset* dari PyTorch untuk membentuk *dataset* pelatihan gabungan, lalu pada *dataset* validasi hanya menggunakan data asli tanpa augmentasi,

Untuk keseluruhan *patch* baik untuk *train patch*, *val patch*, dan *test patch*. Keseluruhan *patch* ditransformasi menjadi format tensor menggunakan *ToTensorV2*. Tujuan dari konversi *patch* menjadi format tensor yaitu untuk mempermudah model dalam mempelajari citra. Jumlah total data pada *dataset* pelatihan dan validasi dihitung untuk memastikan ketersediaan data yang cukup dalam melatih dan mengevaluasi model segmentasi semantik. Total *patch* dan *mask* untuk 62 citra setelah proses augmentasi dapat dilihat pada Gambar 4.40. Untuk 72 citra divisualisasikan pada Gambar 4.41.

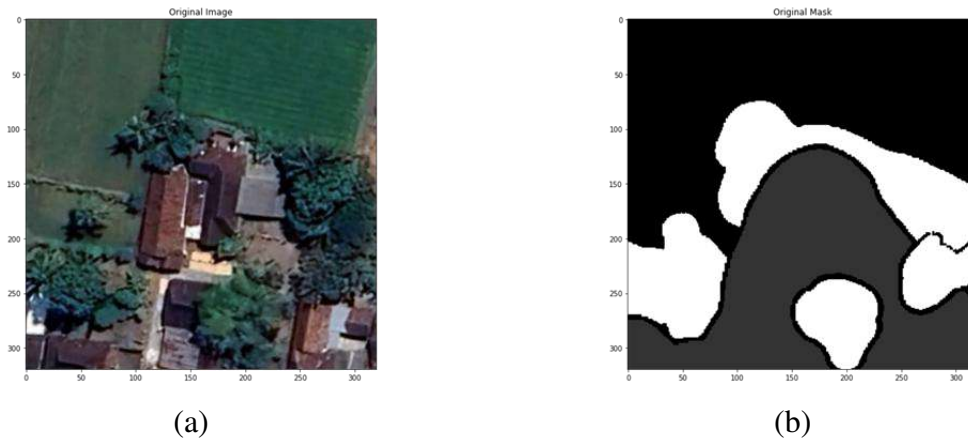
```
Number of training patches: 3992
Number of validation patches: 500
```

Gambar 4.40 Hasil proses augmentasi pada 62 citra

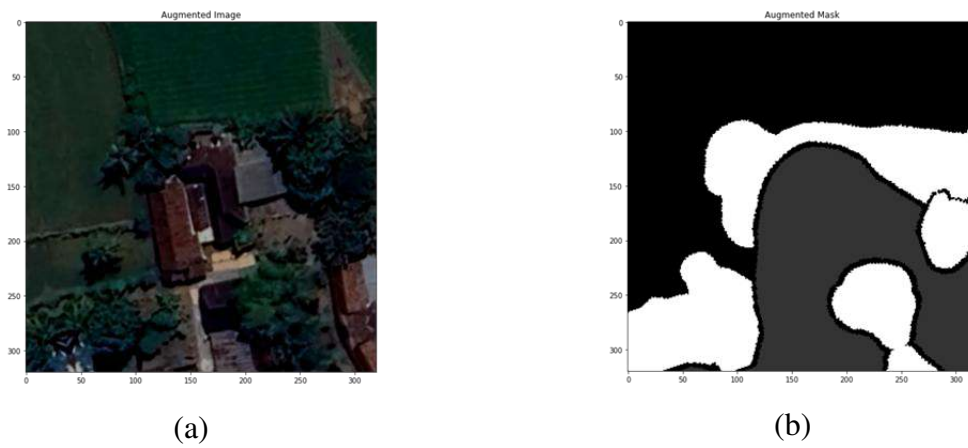
```
Number of training patches: 4180
Number of validation patches: 523
```

Gambar 4.41 Hasil proses augmentasi pada 72 citra

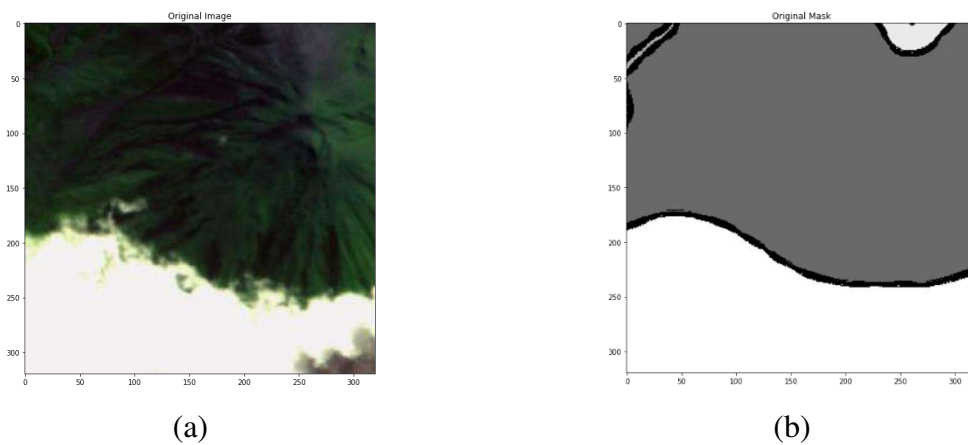
Dari Gambar 4.40, terlihat bahwa terdapat 3992 patches untuk *training images* dan 500 patches untuk *validation images*. Penambahan jumlah *training images* ini sejalan dengan proses augmentasi yang dilakukan satu kali dari 1996 *patch* citra asli, sehingga total gambar pelatihan menjadi 3992. Hal ini yang sama dilakukan untuk 72 citra pada Gambar 4.41. Keputusan untuk melakukan augmentasi hanya satu kali didasarkan pada pertimbangan efisiensi waktu pelatihan dan penggunaan memori GPU, mengingat banyaknya gambar yang harus diproses. Sampel gambar yang telah diaugmentasi dapat dilihat pada Gambar 4.43 dan Gambar 4.45.



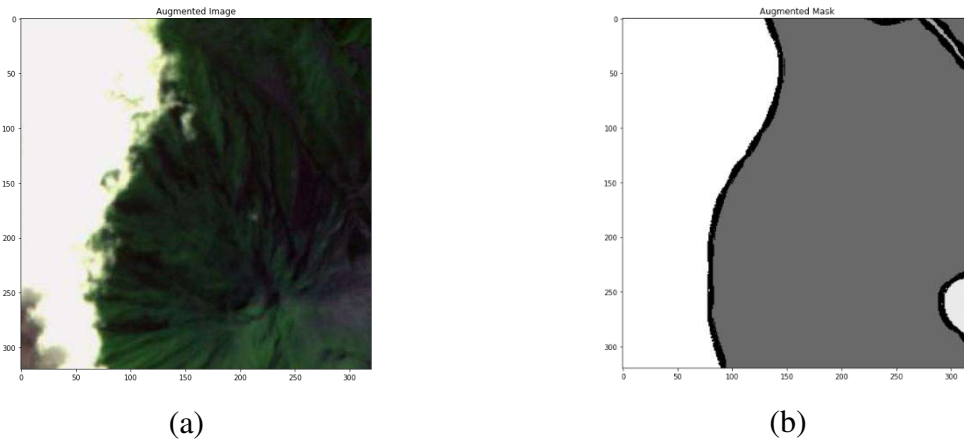
Gambar 4.42 *Patch* tutupan lahan, (a) *Patch* asli, (b) *Mask* asli



Gambar 4.43 *Patch* tutupan lahan, (a) *Patch* hasil augmentasi, (b) *mask* hasil augmentasi



Gambar 4.44 *Patch* gunung api, (a) *Patch* asli, (b) *Mask* asli



Gambar 4.45 Gambar 4.46 *Patch* tutupan lahan, (a) *Patch* hasil augmentasi, (b) *mask* hasil augmentasi

Gambar 4.43 dan Gambar 4.45 memperlihatkan hasil augmentasi dari *patch* asli pada *data train*. Gambar 4.43 menunjukkan permukaan area tutupan lahan, di mana proses augmentasi dilakukan dengan berbagai teknik seperti *flipping*, rotasi, dan penurunan kontras. Sementara itu, Gambar 4.45 menggambarkan area gunung api, citra tersebut menerima jumlah augmentasi lebih sedikit dibandingkan citra Gambar 4.43. Hal ini disebabkan oleh penerapan augmentasi yang bersifat probabilistik secara acak pada setiap gambar. Selain itu, dapat diamati bahwa proses augmentasi dilakukan secara berpasangan, mencakup gambar asli dan *mask* yang berpasangan.

4.6.4 Train Model

Setelah menemukan konfigurasi dan *hyperparameter* terbaik pada Subbab 4.5. Model dilatih untuk 3 skenario yang telah ditentukan pada Tabel 4.11 dengan *hyperparameter* terbaik pada Tabel 4.8. Untuk tahapan pada keseluruhan skenario pelatihan hampir sama yang membedakan hanya pembobotan kelas, fungsi loss, dan perubahan nilai pada variable “criterion” pada Gambar 4.49 untuk skenario pelatihan di Tabel 4.13.

Tahapan awal dalam proses ini adalah membuat *DataLoader* untuk memuat gambar *train* dan *validation*. *DataLoader* dalam PyTorch merupakan modul penting yang berfungsi untuk memudahkan pengelolaan data selama pelatihan model, khususnya dalam pengaturan *batch*. Modul ini terdapat pada *library* `torch.utils.data`, bertugas untuk memastikan data dimuat dan diproses dengan efisien sebelum dilatih model. Selain itu, *DataLoader* juga memungkinkan pengaturan *batch size* sebagai bagian dari proses pelatihan. Contoh implementasi kode *DataLoader* ditampilkan pada Gambar 4.47.

```
# Create data loaders
train_loader = DataLoader(combined_train_dataset, batch_size=64, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=64, shuffle=False)
```

Gambar 4.47 Kode menginisialisasi *dataloader*

Pada Gambar 4.47, ditampilkan tahapan proses yang diawali dengan mendefinisikan *train loader* dan *validation loader* untuk mempermudah pengelolaan data selama pelatihan dan validasi model. *Dataset* pelatihan yang digunakan adalah *combined_train_dataset*, mencakup data asli serta data augmentasi yang dihasilkan melalui proses pada Subbab 4.6.3. Proses pelatihan dilakukan dengan ukuran *batch* sebesar 64, sesuai hasil konfigurasi parameter pada Tabel 4.10. Untuk data train diacak (*shuffle=True*) sebelum dimuat terhadap model. Hal ini bertujuan agar model menerima data yang beragam di setiap iterasi, sehingga performa dapat lebih optimal.

Untuk validasi, *dataset* yang digunakan adalah *val_dataset*. Data ini mencakup data validasi tanpa augmentasi. Selama tahap ini ukuran *batch* ditetapkan sebesar 64 dan data tidak diacak (*shuffle=False*) agar menghindari data yang beragam pada setiap iterasi. Hal ini juga berguna agar menjaga konsistensi evaluasi model sesuai urutan data dalam *dataset*.

Pada skenario ketiga dalam Tabel 4.11, peneliti menghitung bobot kelas untuk menyeimbangkan *dataset*. Langkah ini dilakukan sebagai bagian dari eksplorasi penelitian karena *dataset* memiliki ketidakseimbangan yang signifikan. Kode perhitungan bobot kelas ini ditampilkan pada Gambar 4.48. Kode ini juga mengakomodasi untuk penggabungan fungsi *loss*.

```
import torch
import torch.nn as nn
from segmentation_models_pytorch.losses import DiceLoss
from sklearn.utils.class_weight import compute_class_weight
import numpy as np

# Definisikan fungsi loss gabungan
class CombinedLoss(nn.Module):
    def __init__(self, alpha=0.5, beta=0.5, class_weights=None):
        """
        Kombinasi Dice Loss dan Cross-Entropy Loss dengan class weights.
        Args:
            alpha (float): Bobot untuk Cross-Entropy Loss.
            beta (float): Bobot untuk Dice Loss.
            class_weights (torch.Tensor): Bobot kelas untuk Cross-Entropy Loss.
        """
        super(CombinedLoss, self).__init__()
        self.alpha = alpha
        self.beta = beta
        self.dice_loss = DiceLoss(mode='multiclass') # Dice loss untuk multi-kelas
```

```

        self.ce_loss = nn.CrossEntropyLoss(weight=class_weights) # Tambahkan
class weights

    def forward(self, pred, target):
        """
        Menghitung gabungan loss.
        Args:
            pred (torch.Tensor): Output model (logit) dengan dimensi (batch,
num_classes, height, width).
            target (torch.Tensor): Label ground truth dengan dimensi (batch,
height, width).
        Returns:
            torch.Tensor: Total loss.
        """
        dice = self.dice_loss(pred, target)
        ce = self.ce_loss(pred, target)
        return self.alpha * ce + self.beta * dice

# Hitung bobot kelas dari dataset
if __name__ == "__main__":
    # Nama kelas
    class_names = [
        "Cropland", "Permukiman", "Air", "Area Terbuka", "Jalan", "Vegetasi",
        "Herbaceous Vegetation", "Lava Dome", "Jalur Lahar", "Awan Panas",
        "Endapan Material Vulkanik Baru", "Endapan Material Vulkanik Lama",
        "Awan", "Lava Flow"
    ]

    # Asumsikan dataset memiliki len() dan dapat diakses per indeks
    all_masks = torch.cat([dataset[idx][1].flatten() for idx in
range(len(dataset))])

    # Hitung class weights
    labels = all_masks.numpy()
    class_weights = compute_class_weight(
        class_weight='balanced', # Metode balancing
        classes=np.unique(labels), # Kelas unik
        y=labels # Label dalam bentuk array 1D
    )
    # Konversi ke tensor PyTorch
    class_weights_tensor = torch.tensor(class_weights,
dtype=torch.float32).to('cuda')

    # Dictionary nama kelas dan bobotnya
    class_weights_dict = {class_names[i]: class_weights[i] for i in
range(len(class_weights))}

    # Tampilkan bobot kelas
    print("Class Weights:")
    for class_name, weight in class_weights_dict.items():
        print(f"{class_name}: {weight:.4f}")

```

Gambar 4.48 Kode untuk pembobotan kelas dan penggabungan fungsi *loss*

Gambar 4.48 berfungsi untuk menggabungkan fungsi *loss* dan perhitungan bobot kelas. Fungsi *CombinedLoss* dirancang untuk menggabungkan Dice Loss dan Cross-Entropy Loss dalam model segmentasi semantik multi-kelas. Pendekatan ini memberikan fleksibilitas dalam menghitung *loss* dengan mempertimbangkan bobot dari masing-masing kelas, di mana Dice Loss digunakan untuk mengevaluasi *overlap* antara area hasil segmentasi dengan *ground truth*, sedangkan Cross-Entropy Loss berperan dalam klasifikasi per piksel. Untuk menangani

ketidakseimbangan data pada *dataset*, bobot kelas dihitung secara otomatis menggunakan metode `class_weight='balanced'` dari *library* `sklearn`. Bobot ini bekerja dengan menggabungkan semua label dari *dataset* dalam bentuk tensor, kemudian dihitung frekuensi setiap kelasnya. Bobot yang dihasilkan dikonversi menjadi tensor PyTorch agar dapat digunakan dalam proses pelatihan model. Dengan cara ini, kelas dengan jumlah data yang lebih sedikit mendapatkan bobot lebih besar sehingga model tidak bias terhadap kelas mayoritas. Untuk hasil pembobotan kelas ditampilkan pada Gambar 4.49.

```
Class Weights:
Cropland: 0.3038
Permukiman: 0.2843
Air: 1.5716
Area Terbuka: 1.2698
Jalan: 1.9528
Vegetasi: 0.2743
Herbacus Vegetation: 1.9865
Lava Dome: 44.2851
Jalur Lahar: 33.4062
Awan Panas: 9.4172
Endapan Material Vulkanik Baru: 3.2820
Endapan Material Vulkanik Lama: 2.7696
Awan: 3.6148
Lava Flow: 215.7770
```

Gambar 4.49 Hasil pembobotan kelas

Dari Gambar 4.49 dapat dilihat bahwa kelas minoritas diberi nilai lebih besar dibandingkan kelas mayoritas pada *dataset*. Setelah menyelesaikan tahap pendefinisian `DataLoader`, langkah berikutnya adalah mendefinisikan model `DeepLabV3+` beserta konfigurasi *hyperparameter* optimalnya. Langkah ini mencakup perancangan arsitektur `DeepLabV3+`, pemilihan *optimizer*, penyesuaian *scheduler*, pengaturan *weight decay*, dan penerapan *dropout*. Representasi dari proses pendefinisian model tersebut ditampilkan pada Gambar 4.50.

```
# Custom class untuk DeepLabV3+ dengan Dropout
class DeepLabV3PlusWithDropout(torch.nn.Module):
    def __init__(self, encoder_name="ResNet34", encoder_weights="imagenet",
in_channels=3, classes=14):
        super(DeepLabV3PlusWithDropout, self).__init__()
        # Inisialisasi model DeepLabV3+
        self.model = smp.DeepLabV3Plus(
            encoder_name=encoder_name,
            encoder_weights=encoder_weights,
            in_channels=in_channels,
            classes=classes
        )
        # Tambahkan Dropout dengan nilai default p=0.5
```

```

        self.dropout = torch.nn.Dropout(p=0.1) #Probabilitas Dropout

    def forward(self, x):
        # Forward pass melalui model
        x = self.model(x)
        # Terapkan Dropout
        x = self.dropout(x)
        return x

# Device untuk training (CPU atau GPU)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Buat instance model dengan Dropout
model = DeepLabV3PlusWithDropout(
    encoder_name="ResNet34",          # Encoder ResNet34
    encoder_weights="imagenet",      # Pretrained weights
    in_channels=3,                   # Input channels (RGB)
    classes=14                       # Jumlah kelas output
).to(device)

# Loss function
criterion = torch.nn.CrossEntropyLoss()

# Optimizer
optimizer = torch.optim.Adam(model.parameters(), lr=0.001, weight_decay=0.0001)

# Scheduler StepLR untuk menurunkan learning rate setiap 25 epoch
scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=25, gamma=0.1)

# Cetak ringkasan model untuk verifikasi
summary(model, input_size=(3, 320, 320))

```

Gambar 4.50 Kode pendefinisian model dan *hyperparameter*

Gambar 4.50 mendefinisikan arsitektur DeepLabV3+ dan dimodifikasi dengan penambahan layer *dropout* untuk meningkatkan generalisasi model. Model dirancang untuk menerima input berupa citra RGB dengan 3 *channel* (`in_channels=3`) dan menghasilkan prediksi untuk 14 kelas *output* (`classes=14`). *Encoder* yang digunakan adalah ResNet34 dengan bobot telah dilatih sebelumnya pada *dataset* ImageNet (*pretrained weights*) (He et al., 2015). Untuk skenario ketiga pada Tabel 4.11 hanya mengganti nilai dari variable “criterion” dengan memanggil kelas pada Gambar 4.48. Variabel ini dapat dilihat pada Gambar 4.51.

```

criterion = CombinedLoss(alpha=0.7, beta=0.3, class_weights=class_weights_tensor)

```

Gambar 4.51 Perubahan nilai pada variabel *criterion*

Nilai *alpha* dan *beta* ini merepresentasikan pembagian bobot fungsi *loss* antara Cross-Entropy Loss dan Dice Loss. Nilai *alpha* mewakili Cross-Entropy Loss dan *beta* mewakili Dice Loss. Nilai ini perlu dikaji lebih mendalam untuk menghasilkan model dengan performa yang tinggi. Namun dikarenakan keterbatasan waktu penelitian, nilai ini diselaraskan dengan artikel (Galdran et al., 2022).

Model diinisialisasi pada perangkat yang tersedia, baik GPU maupun CPU menggunakan fungsi `torch.device` untuk memastikan kompatibilitas saat pelatihan. Fungsi *loss* yang digunakan adalah `CrossEntropyLoss` di mana cocok untuk masalah segmentasi multi-kelas. Dice loss dipilih karena cocok untuk tugas segmentasi karena menghitung nilai *loss* dari area *overlap* antara *predicted label* dan *ground truth* (Galdran et al., 2022). Untuk mengoptimalkan parameter model, digunakan *optimizer* Adam dalam pelatihan model ini. Strategi pengaturan *learning rate* dilakukan menggunakan *scheduler* StepLR. *Scheduler* ini akan menurunkan *learning rate* ketika pelatihan sudah memenuhi kelipatan 25. Tahap terakhir yaitu arsitektur model ditampilkan menggunakan fungsi *summary* untuk memverifikasi arsitektur. Kode ini juga memuat jumlah parameter yang terlibat dan struktur *layer* pada model. Hasilnya yaitu visualisasi arsitektur DeepLabV3+ pada Gambar 4.52.

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 64, 160, 160]	9,408
BatchNorm2d-2	[-1, 64, 160, 160]	128
ReLU-3	[-1, 64, 160, 160]	0
MaxPool2d-4	[-1, 64, 80, 80]	0
Conv2d-5	[-1, 64, 80, 80]	36,864
BatchNorm2d-6	[-1, 64, 80, 80]	128
ReLU-7	[-1, 64, 80, 80]	0
Conv2d-8	[-1, 64, 80, 80]	36,864
BatchNorm2d-9	[-1, 64, 80, 80]	128
ReLU-10	[-1, 64, 80, 80]	0
BasicBlock-11	[-1, 64, 80, 80]	0
Conv2d-12	[-1, 64, 80, 80]	36,864
BatchNorm2d-13	[-1, 64, 80, 80]	128
ReLU-14	[-1, 64, 80, 80]	0
Conv2d-15	[-1, 64, 80, 80]	36,864
BatchNorm2d-16	[-1, 64, 80, 80]	128
ReLU-17	[-1, 64, 80, 80]	0
BasicBlock-18	[-1, 64, 80, 80]	0
Conv2d-19	[-1, 64, 80, 80]	36,864
BatchNorm2d-20	[-1, 64, 80, 80]	128
ReLU-21	[-1, 64, 80, 80]	0
Conv2d-22	[-1, 64, 80, 80]	36,864
BatchNorm2d-23	[-1, 64, 80, 80]	128
ReLU-24	[-1, 64, 80, 80]	0
BasicBlock-25	[-1, 64, 80, 80]	0
Conv2d-26	[-1, 128, 40, 40]	73,728
BatchNorm2d-27	[-1, 128, 40, 40]	256
ReLU-28	[-1, 128, 40, 40]	0
Conv2d-29	[-1, 128, 40, 40]	147,456
BatchNorm2d-30	[-1, 128, 40, 40]	256
Conv2d-31	[-1, 128, 40, 40]	8,192
BatchNorm2d-32	[-1, 128, 40, 40]	256
ReLU-33	[-1, 128, 40, 40]	0
BasicBlock-34	[-1, 128, 40, 40]	0
Conv2d-35	[-1, 128, 40, 40]	147,456
BatchNorm2d-36	[-1, 128, 40, 40]	256
ReLU-37	[-1, 128, 40, 40]	0
Conv2d-38	[-1, 128, 40, 40]	147,456
BatchNorm2d-39	[-1, 128, 40, 40]	256

ReLU-40	[-1, 128, 40, 40]	0
BasicBlock-41	[-1, 128, 40, 40]	0
Conv2d-42	[-1, 128, 40, 40]	147,456
BatchNorm2d-43	[-1, 128, 40, 40]	256
ReLU-44	[-1, 128, 40, 40]	0
Conv2d-45	[-1, 128, 40, 40]	147,456
BatchNorm2d-46	[-1, 128, 40, 40]	256
ReLU-47	[-1, 128, 40, 40]	0
BasicBlock-48	[-1, 128, 40, 40]	0
Conv2d-49	[-1, 128, 40, 40]	147,456
BatchNorm2d-50	[-1, 128, 40, 40]	256
ReLU-51	[-1, 128, 40, 40]	0
Conv2d-52	[-1, 128, 40, 40]	147,456
BatchNorm2d-53	[-1, 128, 40, 40]	256
ReLU-54	[-1, 128, 40, 40]	0
BasicBlock-55	[-1, 128, 40, 40]	0
Conv2d-56	[-1, 256, 20, 20]	294,912
BatchNorm2d-57	[-1, 256, 20, 20]	512
ReLU-58	[-1, 256, 20, 20]	0
Conv2d-59	[-1, 256, 20, 20]	589,824
BatchNorm2d-60	[-1, 256, 20, 20]	512
Conv2d-61	[-1, 256, 20, 20]	32,768
BatchNorm2d-62	[-1, 256, 20, 20]	512
ReLU-63	[-1, 256, 20, 20]	0
BasicBlock-64	[-1, 256, 20, 20]	0
Conv2d-65	[-1, 256, 20, 20]	589,824
BatchNorm2d-66	[-1, 256, 20, 20]	512
ReLU-67	[-1, 256, 20, 20]	0
Conv2d-68	[-1, 256, 20, 20]	589,824
BatchNorm2d-69	[-1, 256, 20, 20]	512
ReLU-70	[-1, 256, 20, 20]	0
BasicBlock-71	[-1, 256, 20, 20]	0
Conv2d-72	[-1, 256, 20, 20]	589,824
BatchNorm2d-73	[-1, 256, 20, 20]	512
ReLU-74	[-1, 256, 20, 20]	0
Conv2d-75	[-1, 256, 20, 20]	589,824
BatchNorm2d-76	[-1, 256, 20, 20]	512
ReLU-77	[-1, 256, 20, 20]	0
BasicBlock-78	[-1, 256, 20, 20]	0
Conv2d-79	[-1, 256, 20, 20]	589,824
BatchNorm2d-80	[-1, 256, 20, 20]	512
ReLU-81	[-1, 256, 20, 20]	0
Conv2d-82	[-1, 256, 20, 20]	589,824
BatchNorm2d-83	[-1, 256, 20, 20]	512
ReLU-84	[-1, 256, 20, 20]	0
BasicBlock-85	[-1, 256, 20, 20]	0
Conv2d-86	[-1, 256, 20, 20]	589,824
BatchNorm2d-87	[-1, 256, 20, 20]	512
ReLU-88	[-1, 256, 20, 20]	0
Conv2d-89	[-1, 256, 20, 20]	589,824
BatchNorm2d-90	[-1, 256, 20, 20]	512
ReLU-91	[-1, 256, 20, 20]	0
BasicBlock-92	[-1, 256, 20, 20]	0
Conv2d-93	[-1, 256, 20, 20]	589,824
BatchNorm2d-94	[-1, 256, 20, 20]	512
ReLU-95	[-1, 256, 20, 20]	0
Conv2d-96	[-1, 256, 20, 20]	589,824
BatchNorm2d-97	[-1, 256, 20, 20]	512
ReLU-98	[-1, 256, 20, 20]	0
BasicBlock-99	[-1, 256, 20, 20]	0
Conv2d-100	[-1, 512, 20, 20]	1,179,648
BatchNorm2d-101	[-1, 512, 20, 20]	1,024
ReLU-102	[-1, 512, 20, 20]	0
Conv2d-103	[-1, 512, 20, 20]	2,359,296
BatchNorm2d-104	[-1, 512, 20, 20]	1,024
Conv2d-105	[-1, 512, 20, 20]	131,072
BatchNorm2d-106	[-1, 512, 20, 20]	1,024

ReLU-107	[-1, 512, 20, 20]	0
BasicBlock-108	[-1, 512, 20, 20]	0
Conv2d-109	[-1, 512, 20, 20]	2,359,296
BatchNorm2d-110	[-1, 512, 20, 20]	1,024
ReLU-111	[-1, 512, 20, 20]	0
Conv2d-112	[-1, 512, 20, 20]	2,359,296
BatchNorm2d-113	[-1, 512, 20, 20]	1,024
ReLU-114	[-1, 512, 20, 20]	0
BasicBlock-115	[-1, 512, 20, 20]	0
Conv2d-116	[-1, 512, 20, 20]	2,359,296
BatchNorm2d-117	[-1, 512, 20, 20]	1,024
ReLU-118	[-1, 512, 20, 20]	0
Conv2d-119	[-1, 512, 20, 20]	2,359,296
BatchNorm2d-120	[-1, 512, 20, 20]	1,024
ReLU-121	[-1, 512, 20, 20]	0
BasicBlock-122	[-1, 512, 20, 20]	0
ResNetEncoder-123	[[[-1, 3, 320, 320], [-1, 64, 160, 160], [-1, 64, 80, 80], [-1, 128, 40, 40], [-1, 256, 20, 20], [-1, 512, 20, 20]]	0
Conv2d-124	[-1, 256, 20, 20]	131,072
BatchNorm2d-125	[-1, 256, 20, 20]	512
ReLU-126	[-1, 256, 20, 20]	0
Conv2d-127	[-1, 512, 20, 20]	4,608
Conv2d-128	[-1, 256, 20, 20]	131,072
BatchNorm2d-129	[-1, 256, 20, 20]	512
ReLU-130	[-1, 256, 20, 20]	0
Conv2d-131	[-1, 512, 20, 20]	4,608
Conv2d-132	[-1, 256, 20, 20]	131,072
BatchNorm2d-133	[-1, 256, 20, 20]	512
ReLU-134	[-1, 256, 20, 20]	0
Conv2d-135	[-1, 512, 20, 20]	4,608
Conv2d-136	[-1, 256, 20, 20]	131,072
BatchNorm2d-137	[-1, 256, 20, 20]	512
ReLU-138	[-1, 256, 20, 20]	0
AdaptiveAvgPool2d-139	[-1, 512, 1, 1]	0
Conv2d-140	[-1, 256, 1, 1]	131,072
BatchNorm2d-141	[-1, 256, 1, 1]	512
ReLU-142	[-1, 256, 1, 1]	0
Conv2d-143	[-1, 256, 20, 20]	327,680
BatchNorm2d-144	[-1, 256, 20, 20]	512
ReLU-145	[-1, 256, 20, 20]	0
Dropout-146	[-1, 256, 20, 20]	0
ASPP-147	[-1, 256, 20, 20]	0
Conv2d-148	[-1, 256, 20, 20]	2,304
Conv2d-149	[-1, 256, 20, 20]	65,536
BatchNorm2d-150	[-1, 256, 20, 20]	512
ReLU-151	[-1, 256, 20, 20]	0
UpsamplingBilinear2d-152	[-1, 256, 80, 80]	0
Conv2d-153	[-1, 48, 80, 80]	3,072
BatchNorm2d-154	[-1, 48, 80, 80]	96
ReLU-155	[-1, 48, 80, 80]	0
Conv2d-156	[-1, 304, 80, 80]	2,736
Conv2d-157	[-1, 256, 80, 80]	77,824
BatchNorm2d-158	[-1, 256, 80, 80]	512
ReLU-159	[-1, 256, 80, 80]	0
DeepLabV3PlusDecoder-160	[-1, 256, 80, 80]	0
Conv2d-161	[-1, 14, 80, 80]	3,598
UpsamplingBilinear2d-162	[-1, 14, 320, 320]	0
Identity-163	[-1, 14, 320, 320]	0
Activation-164	[-1, 14, 320, 320]	0
DeepLabV3Plus-165	[-1, 14, 320, 320]	0
Dropout-166	[-1, 14, 320, 320]	0
=====		
Total params: 22,440,798		
Trainable params: 22,440,798		
Non-trainable params: 0		

Input size (MB): 1.17
Forward/backward pass size (MB): 384.29
Params size (MB): 85.60
Estimated Total Size (MB): 471.06

Gambar 4.52 Arsitektur DeepLabV3+

Gambar 4.52 adalah arsitektur dari Model DeepLabV3+. Model ini dirancang untuk segmentasi citra dengan gambar input berukuran 320x320 piksel dengan input gambar memiliki 3 saluran warna (RGB). Proses diawali dengan *encoder* ResNet34 yang terdiri dari lapisan-lapisan Conv2d, BatchNorm2d, dan ReLU, serta blok residual untuk menjaga kestabilan gradien selama pelatihan. Dimensi spasial secara bertahap diperkecil melalui MaxPooling, menghasilkan fitur padat dan informatif.

Fitur dihasilkan dari *encoder* yang diproses oleh modul Atrous Spatial Pyramid Pooling (ASPP) menggunakan konvolusi dengan dilasi berbeda-beda untuk menangkap informasi dari berbagai skala objek. Setelah itu, hasil dari ASPP diintegrasikan dengan fitur *encoder* sebelumnya melalui concatenation, dengan bantuan Upsampling Bilinear untuk menyelaraskan dimensi. Tahap ini memastikan kombinasi informasi spasial dan kontekstual yang optimal untuk segmentasi.

Pada bagian *decoder*, fitur-fitur tersebut diproses lebih lanjut menggunakan lapisan konvolusi dan *dropout*. Penggunaan pendekatan ini berfungsi mencegah *overfitting* dengan mengurangi ketergantungan pada neuron tertentu. Hasil akhir dari model ini adalah peta segmentasi dengan 14 saluran, di mana setiap saluran menunjukkan probabilitas untuk masing-masing kelas. Dimensi peta segmentasi ini dikembalikan ke ukuran aslinya menggunakan metode Upsampling Bilinear (L.-C. Chen et al., 2018). Model ini memiliki sekitar 22,4 juta parameter dan membutuhkan memori sebesar 471 MB.

Setelah semua pendefinisian selesai, model dilatih sesuai dengan jumlah *epoch* yang telah ditentukan pada Tabel 4.10. Pelatihan berlangsung selama 80 *epoch* menggunakan Hanabi dengan rata-rata pelatihan memakan durasi sekitar 35 menit per *epoch*. Sebagai ilustrasi, Gambar 4.53 menunjukkan waktu pelatihan pada *epoch* ke-71 yaitu 35 menit 10 detik, dengan rata-rata 33,49 detik per *batch*. Secara keseluruhan, proses pelatihan membutuhkan waktu lebih dari satu hari. Lamanya waktu ini sebanding dengan jumlah citra yang diproses serta kompleksitas dari model DeepLabV3+. Namun, durasi pelatihan secara *real-time* sulit untuk dihitung secara akurat karena adanya gangguan jaringan sehingga menyebabkan

pelatihan terhenti sementara. Untuk itu, pencatatan waktu *real-time* tidak dapat dilakukan secara menyeluruh.

```

100%|██████████| 63/63 [35:10<00:00, 33.49s/it]
Epoch 71/80
  Train Loss:      0.4196, Val Loss:      0.5496
  Train IoU:       0.5648, Val IoU:       0.5353
  Train Accuracy:  0.8592, Val Accuracy:  0.8270
  Train Dice:      0.6643, Val Dice:      0.6519
  Train mAP:       0.6729, Val mAP:       0.6743
Checkpoint saved at epoch 71 to ./Model/checkpoint.pth
Scheduler updated learning rate: 1e-05
100%|██████████| 63/63 [35:13<00:00, 33.55s/it]
Epoch 72/80
  Train Loss:      0.4209, Val Loss:      0.5492
  Train IoU:       0.5696, Val IoU:       0.5545
  Train Accuracy:  0.8588, Val Accuracy:  0.8278
  Train Dice:      0.6714, Val Dice:      0.6745
  Train mAP:       0.6813, Val mAP:       0.6978
Checkpoint saved at epoch 72 to ./Model/checkpoint.pth
Scheduler updated learning rate: 1e-05

```

Gambar 4.53 Proses pelatihan model

Untuk mengatasi kendala ini, peneliti menggunakan teknik *checkpoint* untuk melacak kemajuan pelatihan. *Checkpoint* dibuat setiap kali nilai *validation loss* mengalami penurunan, dan model terbaik dipilih berdasarkan checkpoint tersebut. Selain itu, *checkpoint* untuk setiap *epoch* juga disimpan untuk memastikan keamanan data pelatihan. Kode yang digunakan dalam proses pelatihan ditampilkan pada Gambar 4.54.

```

# Define the directory for saving checkpoints
CHECKPOINT_DIR = './Model/'
if not os.path.exists(CHECKPOINT_DIR):
    os.makedirs(CHECKPOINT_DIR)
# Function to calculate mean IoU, accuracy, Dice coefficient, and mAP
def calculate_metrics(preds, masks, num_classes):
    preds_flat = preds.flatten()
    masks_flat = masks.flatten()
    iou = jaccard_score(masks_flat, preds_flat, average='macro', zero_division=1)
    accuracy = (preds_flat == masks_flat).mean()
    dice = f1_score(masks_flat, preds_flat, average='macro', zero_division=1)
    precision = precision_score(masks_flat, preds_flat, average='macro',
    zero_division=1)
    recall = recall_score(masks_flat, preds_flat, average='macro',
    zero_division=1)
    mAP = precision
    return iou, accuracy, dice, mAP
# Function to save checkpoint
def save_checkpoint(epoch, model, optimizer, scheduler, train_losses, val_losses,
train_iou, val_iou, train_accuracies, val_accuracies, train_dices, val_dices,
train_maps, val_maps, learning_rates, filename='checkpoint.pth'):
    filepath = os.path.join(CHECKPOINT_DIR, filename)
    state = {
        'epoch': epoch,
        'model_state_dict': model.state_dict(),
        'optimizer_state_dict': optimizer.state_dict(),
        'scheduler_state_dict': scheduler.state_dict(),

```

```

        'train_losses': train_losses,
        'val_losses': val_losses,
        'train_iious': train_iious,
        'val_iious': val_iious,
        'train_accuracies': train_accuracies,
        'val_accuracies': val_accuracies,
        'train_dices': train_dices,
        'val_dices': val_dices,
        'train_maps': train_maps,
        'val_maps': val_maps,
        'learning_rates': learning_rates
    }
    torch.save(state, filepath)
    print(f'Checkpoint saved at epoch {epoch} to {filepath}')

# Function to load the best validation loss from best_model.pth
def load_best_val_loss(filepath):
    if os.path.exists(filepath):
        checkpoint = torch.load(filepath)
        return checkpoint.get('best_val_loss', float('inf')) # Default to inf if
not found
    return float('inf') # Return inf if file does not exist

# Function to save the best model based on early stopping
def save_best_model(epoch, model, val_loss, best_val_loss, is_new_session,
filename='best_model.pth'):
    os.makedirs(CHECKPOINT_DIR, exist_ok=True) # Ensure checkpoint directory
exists
    filepath = os.path.join(CHECKPOINT_DIR, filename)

    # If it's a new session and the first epoch, only save if val_loss improves
over the loaded best_val_loss
    if is_new_session and epoch == 1 and val_loss >= best_val_loss:
        print(f"Skipping model save for epoch {epoch} as validation loss
({val_loss:.4f}) didn't improve.")
        return best_val_loss

    # Save model if validation loss improves
    if val_loss < best_val_loss:
        best_val_loss = val_loss
        torch.save({'model_state_dict': model.state_dict(), 'best_val_loss':
best_val_loss}, filepath)
        print(f'Best model saved at epoch {epoch} with validation loss:
{val_loss:.4f}')
    return best_val_loss

# Function to load checkpoint
def load_checkpoint(filename, model, optimizer, scheduler):
    filepath = os.path.join(CHECKPOINT_DIR, filename)
    checkpoint = torch.load(filepath)
    model.load_state_dict(checkpoint['model_state_dict'])
    optimizer.load_state_dict(checkpoint['optimizer_state_dict'])
    scheduler.load_state_dict(checkpoint['scheduler_state_dict'])
    start_epoch = checkpoint['epoch'] + 1
    train_losses = checkpoint.get('train_losses', [])
    val_losses = checkpoint.get('val_losses', [])
    train_iious = checkpoint.get('train_iious', [])
    val_iious = checkpoint.get('val_iious', [])
    train_accuracies = checkpoint.get('train_accuracies', [])
    val_accuracies = checkpoint.get('val_accuracies', [])
    train_dices = checkpoint.get('train_dices', [])
    val_dices = checkpoint.get('val_dices', [])
    train_maps = checkpoint.get('train_maps', [])
    val_maps = checkpoint.get('val_maps', [])
    learning_rates = checkpoint.get('learning_rates', [])
    print(f'Checkpoint loaded from {filepath}, resuming training from epoch
{start_epoch}')

```

```

    return start_epoch, train_losses, val_losses, train_iou, val_iou,
    train_accuracies, val_accuracies, train_dices, val_dices, train_maps, val_maps,
    learning_rates

# Main training function
def train_model(model, train_loader, val_loader, loss_fn, optimizer, scheduler,
num_epochs=50, resume_from_checkpoint=None, num_classes=7, min_lr=1e-6):
    train_losses, val_losses = [], []
    train_iou, val_iou = [], []
    train_accuracies, val_accuracies = [], []
    train_dices, val_dices = [], []
    train_maps, val_maps = [], []
    learning_rates = []

    start_epoch, is_new_session = 1, True
    best_val_loss = float('inf')

    # Load checkpoint if resuming training
    if resume_from_checkpoint:
        start_epoch, train_losses, val_losses, train_iou, val_iou,
        train_accuracies, val_accuracies, train_dices, val_dices, train_maps, val_maps,
        learning_rates = load_checkpoint(resume_from_checkpoint, model, optimizer,
        scheduler)
        scheduler.last_epoch = start_epoch - 1
        is_new_session = False
        # Load best_val_loss from the best model
        best_val_loss = load_best_val_loss(os.path.join(CHECKPOINT_DIR,
        'best_model.pth'))

    total_epochs = start_epoch + num_epochs - 1

    for epoch in range(start_epoch, total_epochs + 1):
        model.train()
        train_loss, train_iou, train_correct, train_dice, train_map,
        total_train_pixels = 0, 0, 0, 0, 0, 0

        for images, masks in tqdm(train_loader):
            images, masks = images.to(device).float(), masks.to(device).long()
            optimizer.zero_grad()
            outputs = model(images)
            loss = loss_fn(outputs, masks)
            loss.backward()
            optimizer.step()

            train_loss += loss.item() * images.size(0)
            preds = torch.argmax(outputs, dim=1).cpu().numpy()
            masks = masks.cpu().numpy()

            iou, accuracy, dice, mAP = calculate_metrics(preds, masks,
num_classes)
            train_iou += iou * images.size(0)
            train_correct += (preds == masks).sum().item()
            train_dice += dice * images.size(0)
            train_map += mAP * images.size(0)
            total_train_pixels += masks.size

        train_losses.append(train_loss / len(train_loader.dataset))
        train_iou.append(train_iou / len(train_loader.dataset))
        train_accuracies.append(train_correct / total_train_pixels)
        train_dices.append(train_dice / len(train_loader.dataset))
        train_maps.append(train_map / len(train_loader.dataset))

        model.eval()
        val_loss, val_iou, val_correct, val_dice, val_map, total_val_pixels = 0,
0, 0, 0, 0, 0

        with torch.no_grad():

```

```

        for images, masks in val_loader:
            images, masks = images.to(device).float(),
masks.to(device).long()
            outputs = model(images)
            loss = loss_fn(outputs, masks)

            val_loss += loss.item() * images.size(0)
            preds = torch.argmax(outputs, dim=1).cpu().numpy()
            masks = masks.cpu().numpy()

            iou, accuracy, dice, mAP = calculate_metrics(preds, masks,
num_classes)
            val_iou += iou * images.size(0)
            val_correct += (preds == masks).sum().item()
            val_dice += dice * images.size(0)
            val_map += mAP * images.size(0)
            total_val_pixels += masks.size

        val_losses.append(val_loss / len(val_loader.dataset))
        val_iou.append(val_iou / len(val_loader.dataset))
        val_accuracies.append(val_correct / total_val_pixels)
        val_dices.append(val_dice / len(val_loader.dataset))
        val_maps.append(val_map / len(val_loader.dataset))

        print(f'Epoch {epoch}/{total_epochs}')
        print(f'  Train Loss:      {train_losses[-1]:.4f}, Val
Loss:      {val_losses[-1]:.4f}')
        print(f'  Train IoU:      {train_iou[-1]:.4f}, Val
IoU:      {val_iou[-1]:.4f}')
        print(f'  Train Accuracy: {train_accuracies[-1]:.4f}, Val
Accuracy: {val_accuracies[-1]:.4f}')
        print(f'  Train Dice:     {train_dices[-1]:.4f}, Val
Dice:     {val_dices[-1]:.4f}')
        print(f'  Train mAP:      {train_maps[-1]:.4f}, Val
mAP:      {val_maps[-1]:.4f}')

        # Save regular checkpoint
        save_checkpoint(epoch, model, optimizer, scheduler,
                        train_losses, val_losses,
                        train_iou, val_iou,
                        train_accuracies, val_accuracies,
                        train_dices, val_dices,
                        train_maps, val_maps, learning_rates)

        # Scheduler step
        scheduler.step()
        print(f"Scheduler updated learning rate:
{scheduler.optimizer.param_groups[0]['lr']}")

        # Save the best model
        best_val_loss = save_best_model(epoch, model, val_losses[-1],
best_val_loss, is_new_session, filename='best_model.pth')

        return train_losses, val_losses, train_iou, val_iou, train_accuracies,
val_accuracies, train_dices, val_dices, train_maps, val_maps, learning_rates

# Example to start training or resume from a checkpoint
resume_from_checkpoint = 'checkpoint.pth' #Set to None in the first training
train_losses, val_losses, train_iou, val_iou, train_accuracies, val_accuracies,
train_dices, val_dices, train_maps, val_maps, learning_rates = train_model(
    model, train_loader, val_loader, criterion, optimizer, scheduler,
    num_epochs=80, resume_from_checkpoint=resume_from_checkpoint, num_classes=14
)

```

Gambar 4.54 Kode pelatihan model

Gambar 4.54 adalah implementasi untuk melatih model. Fungsi utama `train_model`, dirancang untuk melatih model selama 80 *epoch* dengan menghitung berbagai metrik evaluasi, seperti nilai *loss*, IoU, *accuracy*, *precision*, *f1 score*, dan mAP untuk menilai kinerja model sesuai dengan batasan masalah pada Subbab 1.3. Selama proses pelatihan, fungsi ini mencatat metrik pada data pelatihan dan validasi, serta secara otomatis menyimpan *checkpoint* model dan parameter pelatihan untuk melanjutkan proses di sesi berikutnya. Jika validasi menunjukkan perbaikan metrik pada *val loss*, model terbaik disimpan terpisah dalam file bernama “best_model.pth”. Untuk *checkpoint* dari keseluruhan pelatihan disimpan dengan “checkpoint.pth”.

Selain itu, kode ini memungkinkan melanjutkan pelatihan dari *checkpoint* sebelumnya dengan memuat parameter model, *optimizer*, dan *scheduler*. Fungsi ini juga menggunakan *scheduler* untuk menyesuaikan *learning rate* secara dinamis selama proses pelatihan. Hasil model terbaik dari pelatihan dan *checkpoint* dari seluruh proses pelatihan dapat dilihat pada Gambar 4.55.

best_model.pth	8:03 PM	:
checkpoint.pth	8:03 PM	:

Gambar 4.55 Hasil pelatihan model

4.7 Evaluasi Model

Setelah proses pelatihan selesai, dihasilkan file “checkpoint.pth” dan “best_model.pth”. File ini dapat diekstraksi untuk mendapatkan metrik-metrik yang diperlukan guna mengevaluasi performa model. Proses ekstraksi tersebut dapat dilihat pada Gambar 4.56.

```
# Path untuk checkpoint dan CSV
checkpoint_path = './Model/checkpoint.pth'
output_dir = './Model' # Direktori untuk file CSV
csv_file = os.path.join(output_dir, 'training_metrics_filled.csv') # File CSV yang
akan disimpan

# Buat direktori jika belum ada
os.makedirs(output_dir, exist_ok=True)

# Load checkpoint
checkpoint = torch.load(checkpoint_path)

# Extract metrics dari checkpoint
train_losses = checkpoint['train_losses']
val_losses = checkpoint.get('val_losses', [])
```

```

train_iious = checkpoint['train_iious']
val_iious = checkpoint.get('val_iious', [])
train_accuracies = checkpoint['train_accuracies']
val_accuracies = checkpoint.get('val_accuracies', [])
train_dices = checkpoint['train_dices']
val_dices = checkpoint.get('val_dices', [])
train_maps = checkpoint['train_maps']
val_maps = checkpoint.get('val_maps', [])

# Kolom CSV
csv_columns = ['Epoch', 'Train Loss', 'Val Loss', 'Train IoU', 'Val IoU', 'Train
Accuracy', 'Val Accuracy',
               'Train Dice', 'Val Dice', 'Train mAP', 'Val mAP']

# Tulis data ke file CSV
with open(csv_file, mode='w', newline='') as file:
    writer = csv.writer(file)
    writer.writerow(csv_columns) # Header
    for epoch in range(len(train_losses)):
        row = [
            epoch + 1,
            train_losses[epoch],
            val_losses[epoch] if epoch < len(val_losses) else None,
            train_iious[epoch],
            val_iious[epoch] if epoch < len(val_iious) else None,
            train_accuracies[epoch],
            val_accuracies[epoch] if epoch < len(val_accuracies) else None,
            train_dices[epoch],
            val_dices[epoch] if epoch < len(val_dices) else None,
            train_maps[epoch],
            val_maps[epoch] if epoch < len(val_maps) else None
        ]
        writer.writerow(row)

print(f'Metrics saved to {csv_file}')

# Load CSV dan tampilkan sebagai tabel
if os.path.exists(csv_file):
    metrics_df = pd.read_csv(csv_file)
    display(metrics_df)

# Generate link untuk download file
display(HTML(f'<a href="{csv_file}">download</a>Click here to download the CSV
file</a>'))

```

Gambar 4.56 Kode mengekstraksi metrik pelatihan

Gambar 4.56 ini bertujuan untuk mendokumentasikan metrik pelatihan model dalam format file CSV guna keperluan analisis dan arsip data. Pertama, menentukan lokasi file *checkpoint* (*checkpoint.pth*) dan file CSV. Direktori untuk menyimpan file CSV akan dibuat secara otomatis jika belum tersedia menggunakan fungsi *os.makedirs()*. *Checkpoint* berisi informasi metrik pelatihan dan validasi, seperti *train loss*, *validation loss*, *IoU*, *accuracy*, *Dice score*, dan *mAP*, dimuat menggunakan *torch.load()*. Metrik-metrik ini kemudian diekstrak dan disusun dalam bentuk tabel dengan kolom-kolom dan merepresentasikan masing-masing metrik. File CSV dibuat menggunakan modul *csv*, di mana setiap baris merepresentasikan metrik untuk tiap *epoch*. Setelah pembuatan file selesai, data dapat dilihat dalam bentuk tabel

dengan menggunakan library *pandas*, dan tautan unduhan disediakan untuk mempermudah aksesibilitas.

Peneliti dapat menggunakan file CSV yang telah diunduh untuk membuat grafik, sehingga mempermudah dalam menganalisis bagaimana proses pelatihan berlangsung. Dari data tersebut, peneliti bisa menemukan titik di mana *validation loss* mencapai nilai terendah. Informasi ini sangat penting karena menjadi dasar dalam menentukan *best model* yang nantinya akan digunakan saat proses pengujian. Kode untuk menampilkan grafik ini akan didokumentasikan pada Gambar 4.57.

```
# Plot training metrics after training
import os
import pandas as pd
import matplotlib.pyplot as plt

metrics = {
    'Loss': ('Train Loss', 'Val Loss'),
    'IoU': ('Train IoU', 'Val IoU'),
    'Accuracy': ('Train Accuracy', 'Val Accuracy'),
    'Dice': ('Train Dice', 'Val Dice'),
    'mAP': ('Train mAP', 'Val mAP'),
}

if os.path.exists(csv_file):
    metrics_df = pd.read_csv(csv_file)
    for metric, (train_column, val_column) in metrics.items():
        plt.figure(figsize=(20, 8))

        # Plot the training metric
        plt.plot(metrics_df['Epoch'], metrics_df[train_column], label=f'Train {metric}', linewidth=3)

        # Plot the validation metric if available
        if val_column and val_column in metrics_df.columns:
            plt.plot(metrics_df['Epoch'], metrics_df[val_column], label=f'Validation {metric}', linewidth=3)

        # Label axes and title
        plt.xlabel('Epochs', fontsize=14)
        plt.ylabel(metric, fontsize=14)
        plt.title(f'{metric} vs. Epochs', fontsize=14)

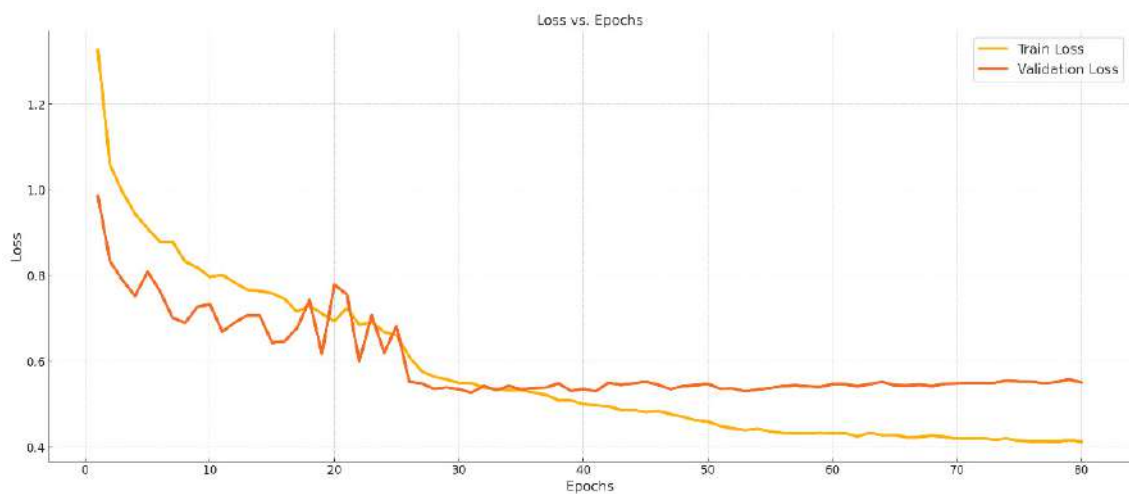
        # Add legend and grid
        plt.legend(fontsize=14)
        plt.grid(True)

        # Show the plot
        plt.show()
```

Gambar 4.57 Kode untuk membuat grafik

Kode pada Gambar 4.57 berfungsi untuk memvisualisasikan metrik dari hasil pelatihan model yang tersimpan dalam file CSV. Proses ini melibatkan *plotting* metrik pelatihan (*train metrics*) dan validasi (*validation metrics*) terhadap jumlah *epoch*, sehingga memudahkan

analisis performa pelatihan. File CSV dibaca menggunakan pustaka *pandas* dan setiap metrik seperti *Loss*, *IoU*, *Accuracy*, *Dice*, dan *mAP* divisualisasikan secara terpisah. File CSV ini dapat dilihat pada bagian lampiran. Grafik dibuat dengan sumbu X yang menunjukkan jumlah *epoch*, sementara sumbu Y menggambarkan nilai metrik terkait. Setiap grafik dilengkapi dengan label sumbu, judul, legenda, dan grid untuk membantu dalam interpretasi hasil secara visual. Grafik pertama menampilkan nilai *loss* pada proses pelatihan pada skenario pelatihan pertama. Grafik ini dapat dilihat pada Gambar 4.58.



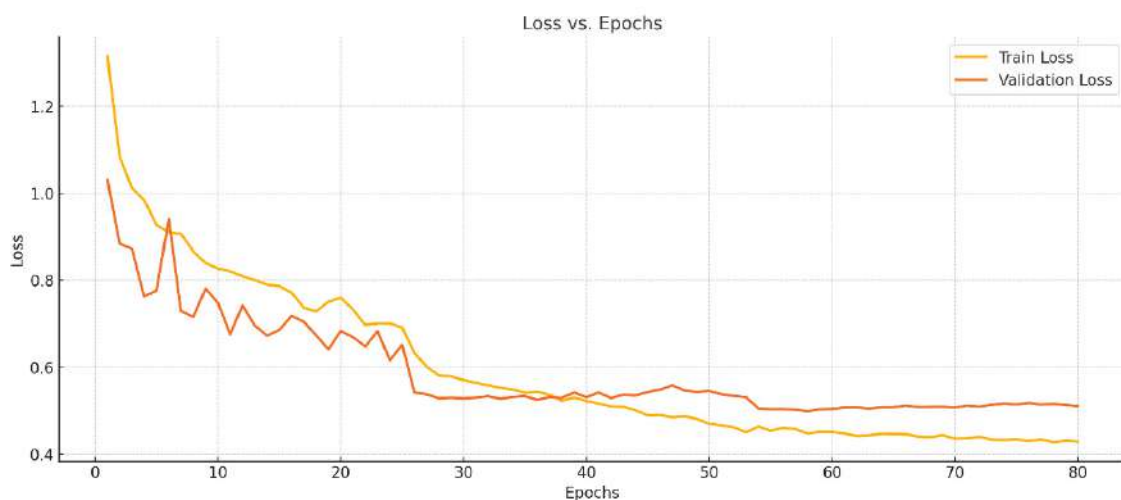
Gambar 4.58 Grafik nilai *loss* skenario pelatihan pertama

Gambar 4.58 menunjukkan grafik *loss* selama pelatihan model DeepLabV3+. Grafik berwarna kuning mewakili *train loss*, sementara grafik berwarna merah menggambarkan *validation loss*. Pada *epoch* pertama, baik *train loss* maupun *validation loss* menghasilkan nilai di atas angka 0.9. Namun, seiring dengan berjalannya pelatihan, grafik mulai menunjukkan penurunan. Penurunan ini masih belum stabil hingga mencapai *epoch* ke-24. Model masih mengalami osilasi, terlihat dari fluktuasi *validation loss* pada grafik.

Pada *epoch* ke-25, penurunan *train loss* dan *validation loss* menjadi lebih stabil. Hal ini terjadi karena *scheduler* mulai aktif bekerja menurunkan *learning rate* dari 0.001 menjadi 0.0001 dengan mengalikan faktor penurunan sebesar 10%, seperti telah ditentukan pada variabel *scheduler*. Pada *epoch* ke-31, model terbaik disimpan karena *validation loss* mencapai nilai terendah. Berdasarkan file CSV, *best model* menghasilkan nilai terbaik dengan *train loss* sebesar 0.55 dan *validation loss* sebesar 0.53.

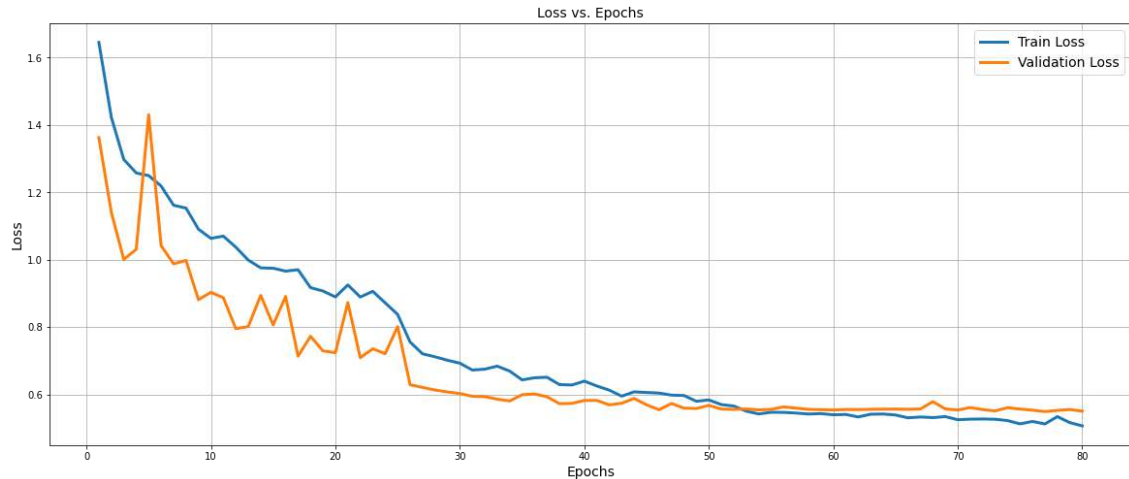
Selama pelatihan model, setelah penyimpanan *best model* terakhir. Tidak ada model terbaik lain yang disimpan. Meskipun *scheduler* diaktifkan kembali pada *epoch* ke-50 dan ke-

75, tidak ada model yang memenuhi syarat untuk disimpan. Berdasarkan grafik, *train loss* terus menurun, sedangkan *validation loss* mengalami sedikit peningkatan. Kondisi ini menunjukkan bahwa model semakin baik dalam mempelajari data pelatihan, namun kinerjanya pada data validasi mulai menurun. Fenomena ini menjadi indikasi bahwa model mulai mengalami *overfitting*. Selanjutnya grafik *loss* untuk skenario pelatihan kedua ditampilkan pada Gambar 4.59.



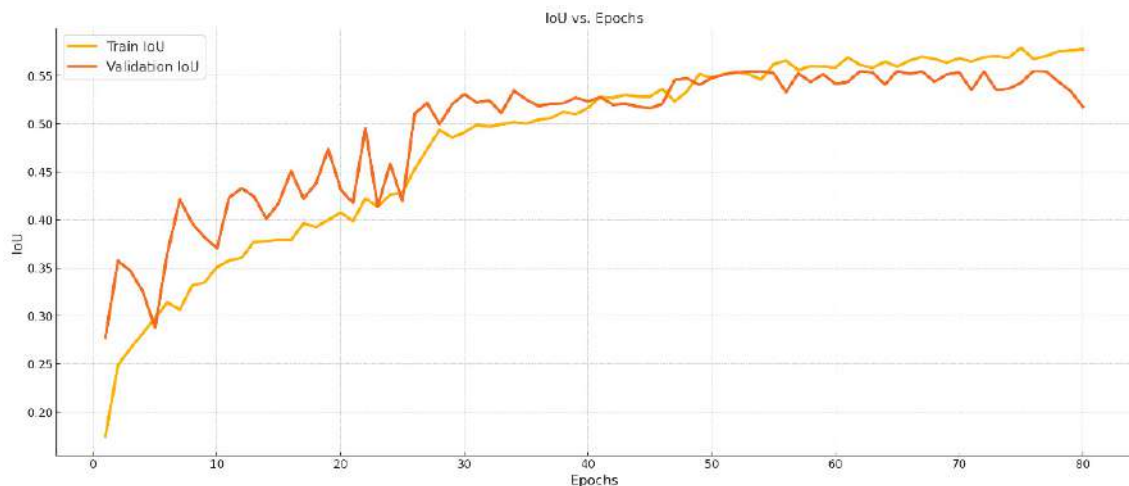
Gambar 4.59 Grafik nilai *loss* skenario pelatihan kedua

Dari Gambar 4.59 dapat dilihat grafik *loss* pada pelatihan skenario kedua. Dapat dilihat bahwa model mengalami osilasi sebelum *scheduler* diaktifkan pada *epoch* ke-25 pada *validation loss*. Pada rentang *epoch* 40-50 model mengalami kenaikan *validation loss*. Namun setelah diaktifkannya *scheduler* di *epoch* ke-50, nilai *validation loss* dapat diturunkan. Hal ini terjadi dikarenakan *learning rate* bernilai besar dapat berpengaruh terhadap konvergensi pelatihan. *Best model* disimpan di *epoch* ke-58 dan proses ini lebih lama daripada skenario pertama dikarenakan adanya penambahan citra. Nilai *train loss* pada *epoch* ke-58 adalah 0.45 dan *validation loss* pada angka 0.50. Untuk grafik *loss* pada skenario ketiga dapat dilihat pada Gambar 4.60.



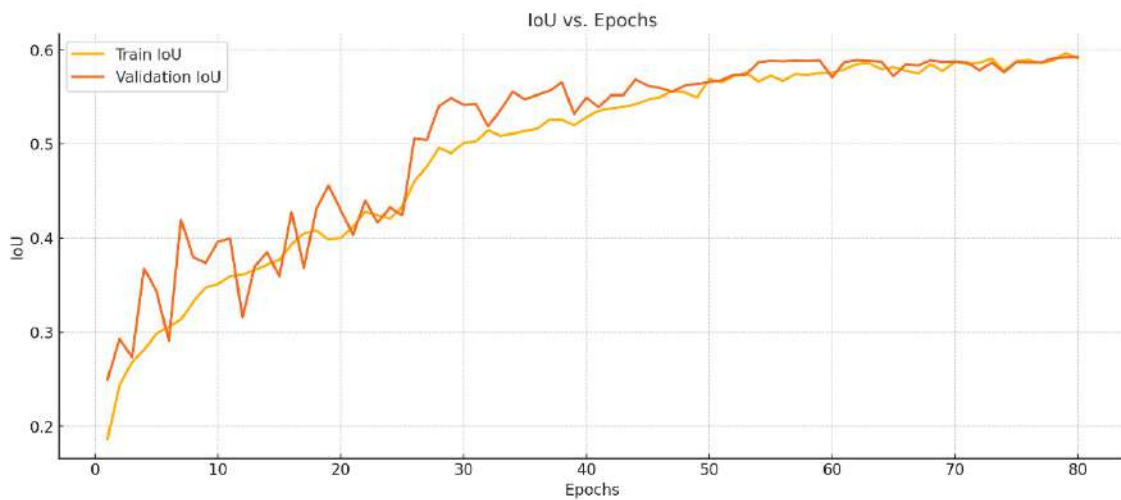
Gambar 4.60 Grafik nilai *loss* skenario pelatihan ketiga

Grafik pada Gambar 4.60 menunjukkan bahwa model masih mengalami fluktuasi pada *validation loss* sebelum *scheduler* mulai diaktifkan pada *epoch* ke-25. Skenario pelatihan ketiga menghasilkan model dengan selisih *train loss* dan *validation loss* paling minim dibandingkan dengan dua pelatihan sebelumnya. Hal ini disebabkan oleh penerapan strategi penyeimbangan *dataset* serta modifikasi pada fungsi *loss*. Dalam skenario penelitian ini, *validation loss* masih berpotensi untuk menurun karena belum menunjukkan kondisi stagnan. Oleh karena itu, diperlukan penambahan jumlah *epoch* pelatihan untuk mengamati hal ini lebih lanjut. Namun, karena keterbatasan waktu penelitian, proses tersebut tidak dilanjutkan oleh peneliti. Model terbaik diperoleh pada *epoch* ke-77, dengan *train loss* sebesar 0.51 dan *validation loss* sebesar 0.55. Selanjutnya menampilkan grafik IoU pada Gambar 4.61.



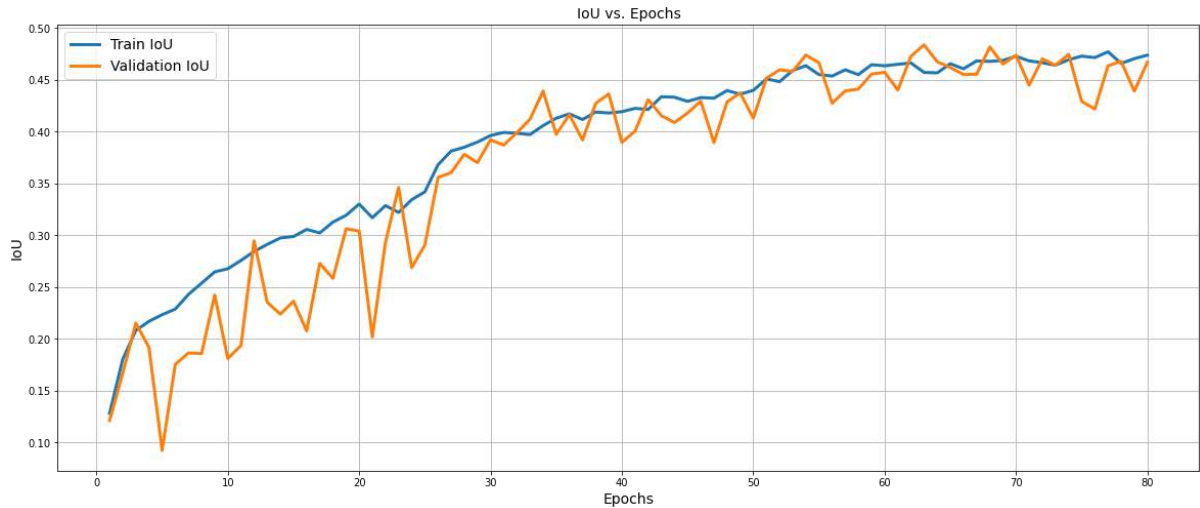
Gambar 4.61 Grafik *Intersection over union* skenario pelatihan pertama

Gambar 4.61 menampilkan visualisasi metrik *Intersection over Union* (IoU) pada skenario pelatihan pertama. Terlihat bahwa performa model yang diukur melalui IoU, menunjukkan peningkatan seiring bertambahnya jumlah *epoch* selama pelatihan. Pada tahap awal dari *epoch* pertama hingga ke-24, model mengalami kesulitan untuk mencapai nilai IoU yang konsisten. Namun, dengan diaktifkannya *scheduler*, model mampu menstabilkan nilai IoU, sehingga perbedaan antara *train* IoU dan *validation* IoU menjadi lebih kecil. Model terbaik yang disimpan pada *epoch* ke-31 mencapai nilai *train* IoU sebesar 0.50 dan *validation* IoU sebesar 0.52. Selanjutnya untuk grafik IoU skenario pelatihan kedua diperlihatkan pada Gambar 4.62.



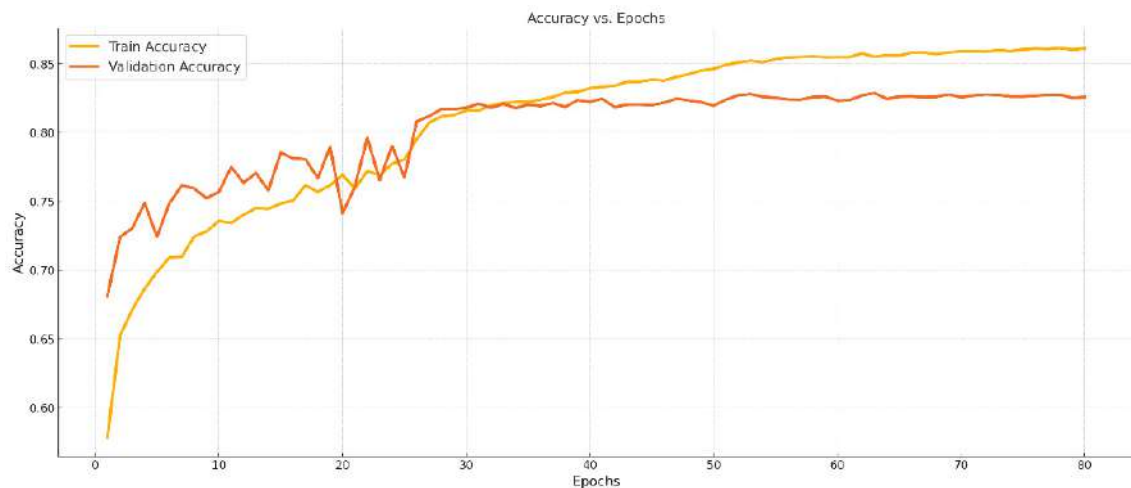
Gambar 4.62 Grafik *Intersection over union* skenario pelatihan kedua

Berdasarkan Gambar 4.62, terlihat bahwa IoU menunjukkan peningkatan meskipun osilasi masih terjadi pada *validation IoU* di skenario pelatihan kedua ini. Penambahan jumlah citra dalam *dataset* turut berkontribusi pada tingginya nilai IoU dibandingkan pada skenario pelatihan pertama. Penambahan citra ini juga berhasil menurunkan selisih *train* IoU dan *validation* IoU pada rentang *epoch* 60-80. Pada *epoch* ke-58 nilai *train* IoU menghasilkan nilai 0.57 dan *validation* IoU 0.59. Grafik IoU pada skenario pelatihan ketiga ditampilkan pada Gambar 4.63.



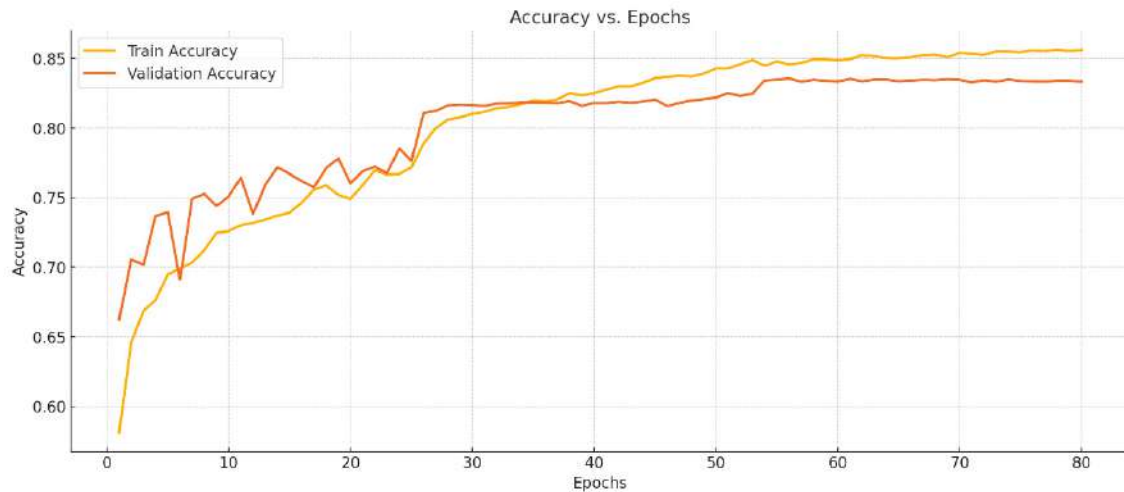
Gambar 4.63 Grafik *Intersection over union* skenario pelatihan ketiga

Grafik pelatihan untuk skenario ketiga ditampilkan pada Gambar 4.63. Dari grafik tersebut, terlihat bahwa nilai IoU mengalami peningkatan seiring bertambahnya *epoch*. Selain itu, selisih antara *train* IoU dan *validation* IoU semakin kecil seiring bertambahnya *epoch*. Namun, baik *train* IoU maupun *validation* IoU masih belum setinggi hasil pada skenario pertama dan kedua. Grafik juga menunjukkan bahwa belum ada indikasi *overfitting* dikarenakan model masih kesulitan mencapai konvergensi. Oleh karena itu, diperlukan penambahan jumlah *epoch* untuk menemukan *best model* yang ideal. Meskipun demikian, dari pelatihan selama 80 *epoch*, model terbaik diperoleh pada *epoch* ke-77 dengan *train* IoU sebesar 0.48 dan *validation* IoU sebesar 0.46. Grafik akurasi pada skenario pertama akan ditampilkan pada Gambar 4.64.



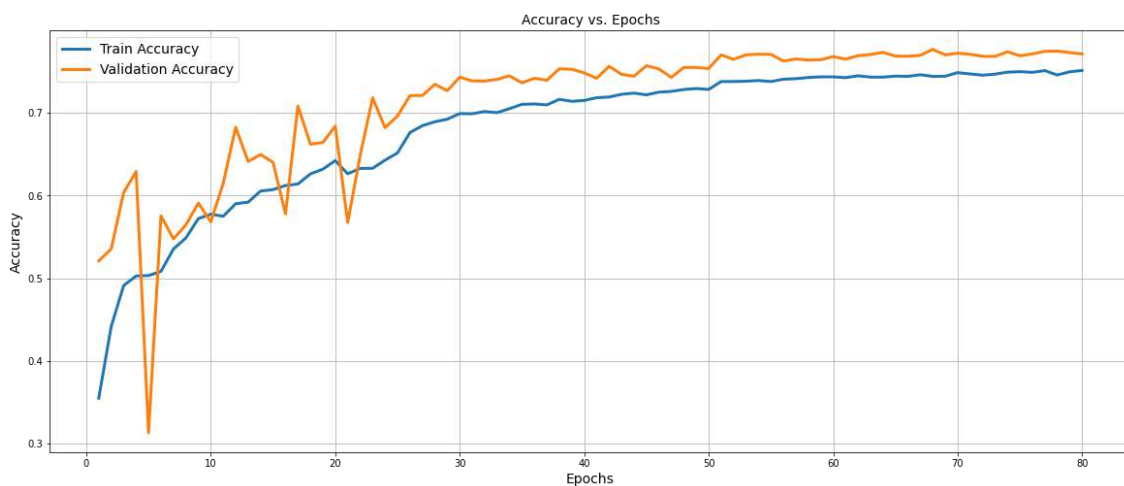
Gambar 4.64 Grafik akurasi skenario pelatihan pertama

Berdasarkan Gambar 4.64, terlihat tren peningkatan akurasi selama proses pelatihan model. Baik akurasi pada data latih maupun validasi menunjukkan kenaikan, dengan akurasi validasi cenderung lebih tinggi pada tahap awal pelatihan. Namun, osilasi masih terjadi serupa dengan yang terlihat pada metrik *loss* dan IoU. Ketika *scheduler* digunakan untuk menurunkan *learning rate*, grafik mulai menunjukkan kestabilan. Seiring berjalannya proses pelatihan, model menunjukkan tanda-tanda *overfitting*. Dapat dilihat dengan adanya peningkatan akurasi pada grafik *data train*, sementara akurasi validasi tetap stagnan. Akurasi diperoleh pada *best model* di *epoch* ke-31 bernilai 0.82 untuk data latih dan 0.82 untuk data validasi. Berikutnya ditampilkan untuk grafik akurasi pada pelatihan skenario kedua. Grafik ini disajikan pada Gambar 4.65.



Gambar 4.65 Grafik akurasi skenario pelatihan kedua

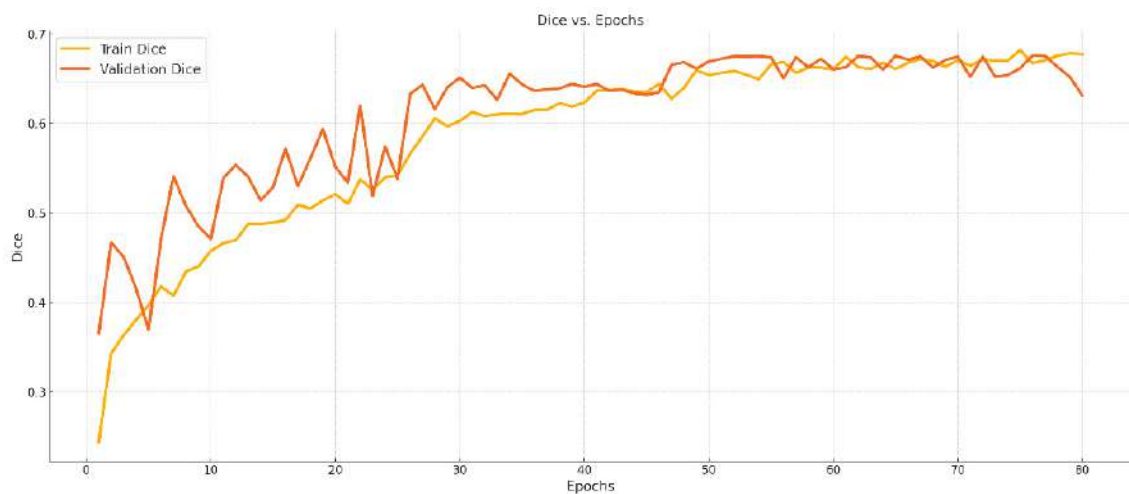
Gambar 4.65 menampilkan grafik akurasi selama proses skenario pelatihan kedua. Baik *train accuracy* dan *validation accuracy* mengalami kenaikan seiring bertambahnya jumlah *epoch*. Penambahan citra juga berpengaruh dengan kenaikan nilai *validation accuracy* dibandingkan dengan skenario pertama. Baik *train accuracy* dan *validation accuracy* mampu mencapai nilai diatas 80%. Pada *epoch* ke-58 didapatkan nilai *train accuracy* sebesar 0.85 dan *validation accuracy* sebesar 0.83. Selanjutnya disajikan grafik *accuracy* untuk skenario pelatihan ketiga pada Gambar 4.66.



Gambar 4.66 Grafik akurasi skenario pelatihan ketiga

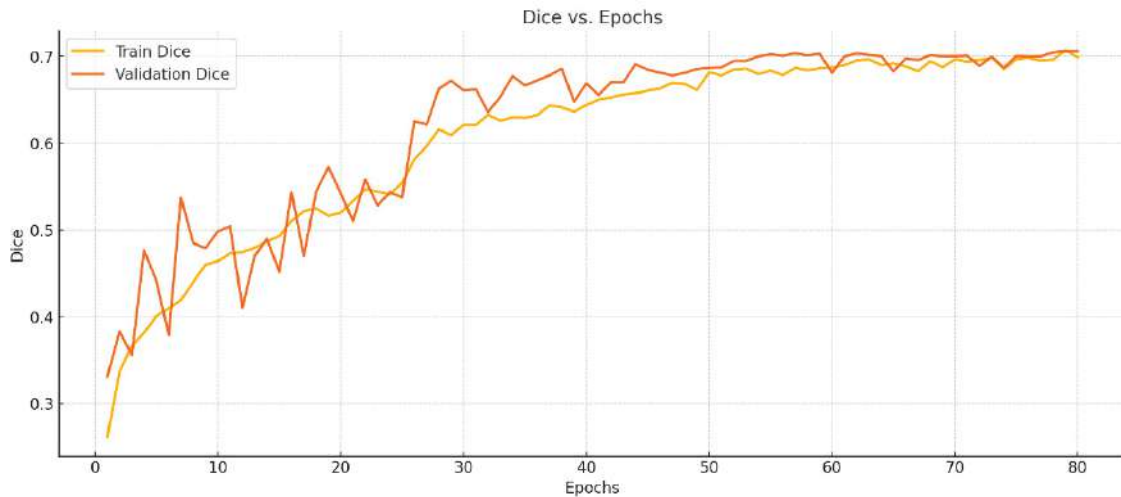
Dari Gambar 4.66 dapat diperoleh wawasan bahwa model mengalami peningkatan akurasi dengan osilasi cukup ekstrim pada rentang *epoch* ke 1-25. Dengan diaktifkannya

scheduler pada *epoch* ke-25 akurasi dari model mulai stabil. Dengan adanya penambahan pembobotan kelas dan modifikasi fungsi *loss* dapat memberikan performa yang stabil. Namun terdapat indikasi model masih dapat memberikan performa yang lebih baik dikarenakan belum terlihat adanya *overfitting* pada grafik. Dibutuhkan penambahan jumlah *epoch* untuk mencari *best model* selanjutnya. Pelatihan skenario ketiga mendapatkan nilai *train accuracy* yaitu 0.75 dan *validation accuracy* 0.77 di epoch ke-77. Garfik *f1-score (dice)* pada skenario pelatihan pertama diperlihatkan pada Gambar 4.67.



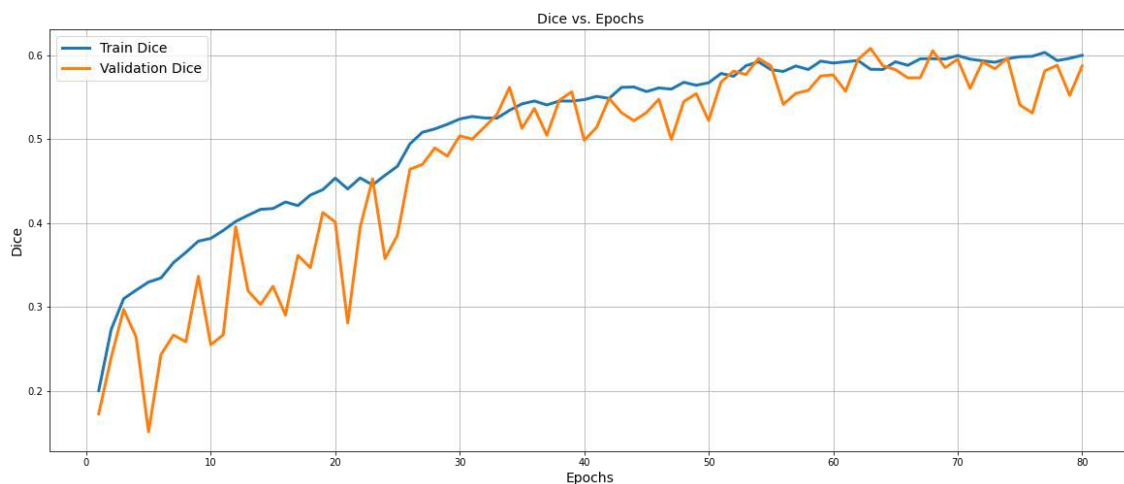
Gambar 4.67 Grafik *f1-score (dice)* skenario pelatihan pertama

Gambar 4.67 memperlihatkan grafik nilai *dice* selama proses pelatihan skenario pertama. Pada awal pelatihan, terlihat bahwa tren peningkatan dimulai dengan perbedaan nilai yang cukup mencolok, di mana nilai *val loss* lebih tinggi dibandingkan *train loss* pada *epoch* pertama. Selanjutnya, metrik ini mengalami fluktuasi signifikan selama 30 *epoch* pertama, dengan osilasi cukup ekstrem. Seiring bertambahnya jumlah *epoch*, perbedaan nilai *dice* antara *train dice* dan *validation dice* semakin kecil. Nilai *dice* dari model terbaik dicapai pada *epoch* ke-31 memperoleh nilai 0.61 untuk *train dice* dan 0.64 untuk *validation dice*. Grafik *dice* untuk skenario pelatihan kedua diperlihatkan pada Gambar 4.68.



Gambar 4.68 Grafik $f1$ -score ($dice$) skenario pelatihan kedua

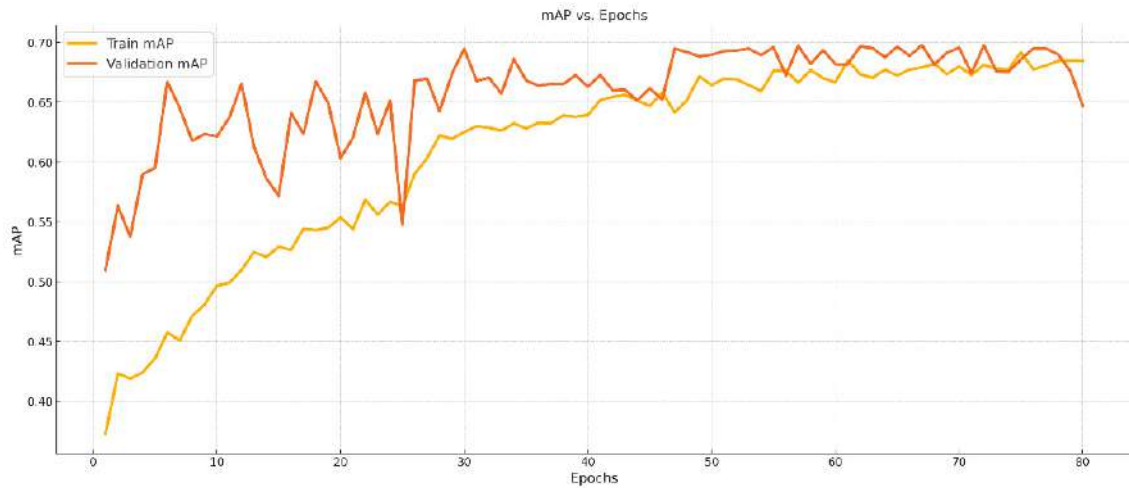
Gambar 4.68 menunjukkan tren nilai $dice$ selama pelatihan pada skenario kedua. Pada grafik tersebut, osilasi terjadi dalam rentang epoch 1-40. Namun, seiring bertambahnya $epoch$, performa $dice$ menjadi lebih stabil, ditandai dengan grafik yang tidak lagi fluktuatif. Model terbaik disimpan pada $epoch$ ke-58 dengan nilai $train\ dice$ sebesar 0.68 dan $validation\ dice$ sebesar 0.70. Untuk grafik $dice$ pada skenario pelatihan ketiga ditampilkan pada Gambar 4.69.



Gambar 4.69 Grafik $f1$ -score ($dice$) skenario pelatihan ketiga

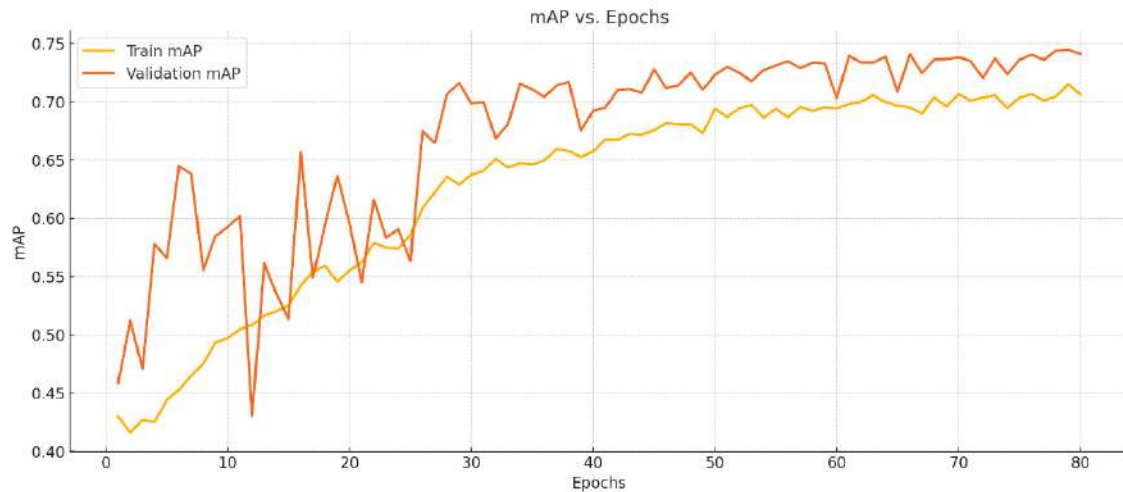
Gambar 4.69 menampilkan grafik pada skenario pelatihan ketiga. Perbedaan nilai $dice$ antara $train\ dice$ dan $validation\ dice$ cukup besar pada rentang epoch 1–25 diakibatkan penerapan pembobotan kelas dan modifikasi fungsi loss. Namun, penggunaan scheduler dapat mengatasi osilasi yang ekstrim. Best model menghasilkan $train\ dice$ 0.60 dan $validation\ dice$

0.58 pada epoch ke-77. Gambar 4.70 menampilkan metrik mAP untuk skenario pelatihan pertama.



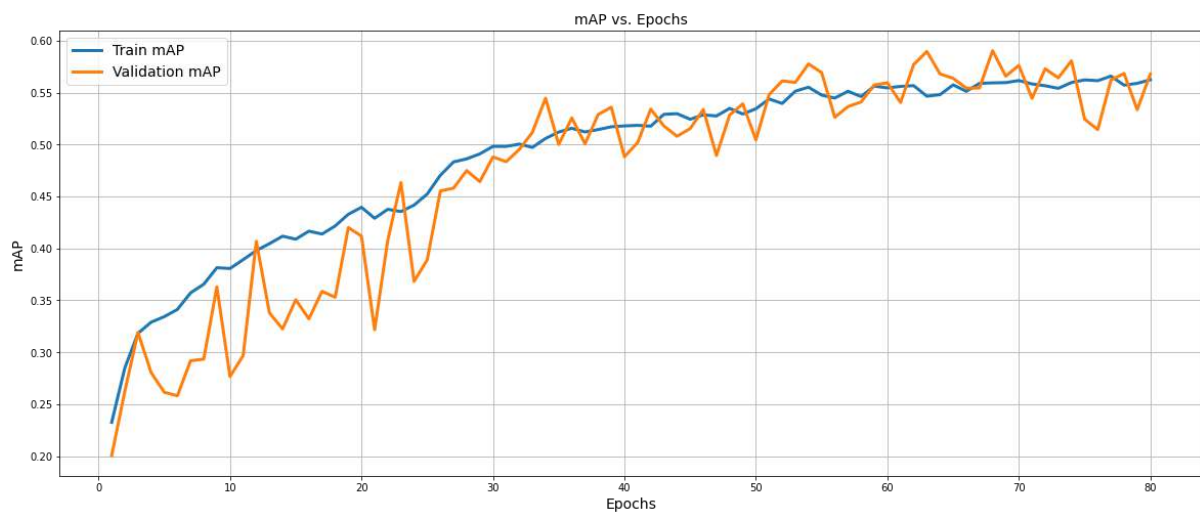
Gambar 4.70 Grafik mAP skenario pelatihan pertama

Dari Gambar 4.70, terlihat bahwa tren mAP validasi mengalami osilasi cukup tajam sebelum mencapai epoch ke-25. Pemanfaatan *scheduler* berkontribusi dalam penurunan *learning rate*, memungkinkan model untuk mencapai stabilitas lebih baik. Grafik ini disajikan dalam skala 0.05, berbeda dari metrik lainnya sehingga memberikan representasi yang lebih rinci terhadap nilai mAP. Seiring berlangsungnya pelatihan, model menunjukkan peningkatan stabilitas, ditunjukkan oleh perbedaan grafik semakin kecil. Pada model terbaik di *epoch* ke-31, nilai mAP tercatat sebesar 0.63 untuk data pelatihan dan 0.67 untuk data validasi. Untuk grafik mAP skenario pelatihan kedua ditampilkan pada Gambar 4.71.



Gambar 4.71 Grafik mAP skenario pelatihan kedua

Dapat dilihat pada Gambar 4.71 bahwa osilasi terjadi sangat ekstrim pada epoch 1-40. Model kesulitan dalam mencapai konvergensi ditunjukkan dengan *validation* mAP yang mengalami fluktuatif pada rentang tersebut. *Best model* menghasilkan nilai *train* mAP 0.69 dan *validation* mAP 0.73 pada *epoch* ke-58.. Lalu untuk grafik mAP skenario ketiga ditampilkan pada Gambar 4.71.



Gambar 4.72 Grafik mAP skenario pelatihan ketiga

Tren pada Gambar 4.71 menunjukkan bahwa nilai mAP yang diperoleh lebih rendah dibandingkan dengan skenario pelatihan pertama maupun kedua. Penurunan ini disebabkan oleh penerapan pembobotan kelas serta modifikasi pada fungsi *loss*. *Best model* mampu meraih nilai 0.57 untuk *train* mAP dan 0.56 untuk *validation* mAP.

4.8 Tes Model

Setelah model selesai dievaluasi, langkah terakhir adalah menguji performanya menggunakan data baru. Pengujian ini dilakukan dengan memanfaatkan 8 citra untuk skenario pelatihan pertama dan 9 citra untuk skenario pelatihan kedua dan ketiga pada Tabel 4.10. Citra yang telah dikumpulkan pada data tes di Tabel 4.1 dan diberi anotasi pada Subbab 4.3 digunakan sebagai data inferensi untuk mengevaluasi kemampuan model dalam menangani kasus-kasus di dunia nyata. Pengujian dilakukan pada *notebook* yang terpisah. Oleh karena itu, diperlukan *environment setup*, sama seperti dijelaskan pada Subbab 4.6.

Proses pengujian dilakukan dengan menyesuaikan ukuran *patch size*, *input image size* dan *batch size* agar sesuai dengan parameter yang diterapkan selama pelatihan dan validasi. Tahapan pengujian ini juga disesuaikan dengan penelitian (Akçay et al., 2022). Lalu, tahapan ini serupa dengan proses pelatihan dan pengujian di mana DataLoader digunakan untuk mengelola data. Pada citra *training* diterapkan teknik *cropping* untuk bisa habis dibagi dengan *patch size* sedangkan pada citra tes menggunakan teknik *resizing*. Gambar beresolusi tinggi terlebih dahulu disesuaikan ukurannya (*resize*) agar dapat dibagi secara sempurna ke dalam *patch* dengan ukuran 320x320 piksel. *Input image size* diselaraskan dengan proses *training* yaitu ukuran 320x320 piksel. Hal ini bertujuan untuk melihat performa model dengan ukuran gambar yang sama baik pada proses *training* maupun proses *testing*.

Namun pada penelitian ini dilakukan juga proses inferensi terhadap ukuran *patch* yang lebih besar untuk mengetahui performa model. Ukuran yang dipilih adalah 512x512 dan 1024x1024 menyesuaikan ukuran pada *patch size* proses di Subbab 4.5. Performa model ini ditentukan dengan menghitung mIoU dan mAP di setiap nilai *patch size* yang ditentukan. Hasil percobaan ini diperlihatkan pada Tabel 4.13.

Tabel 4.13 Skenario *testing model*

Test Patch Size	Skenario Pelatihan 1			Skenario Pelatihan 2			Skenario Pelatihan 3		
	mIoU	mAP	Akurasi	mIoU	mAP	Akurasi	mIoU	mAP	Akurasi
320x320	0.39	0.34	0.72	0.41	0.36	0.72	0.39	0.34	0.66
512x512	0.38	0.33	0.72	0.41	0.36	0.72	0.43	0.39	0.68
1024x1024	0.38	0.33	0.72	0.42	0.37	0.71	0.44	0.39	0.67

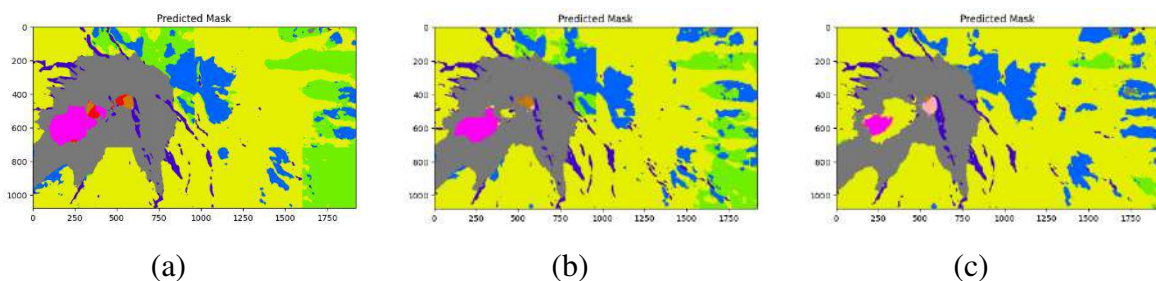
Tabel 4.13 menampilkan hasil pengujian model pada data tes dengan berbagai ukuran *patch* yang menggunakan *best model* dari setiap tahap skenario pelatihan. Hasil yang diperoleh menunjukkan bahwa DeepLabV3+ tetap dapat menghasilkan peta segmentasi, bahkan untuk ukuran *patch* yang berbeda dari *patch size* data *training* yaitu 320x320. Kemampuan ini disebabkan oleh kemampuan DeepLabV3+ dalam memahami citra pada berbagai skala (*multiscale*). Sampel untuk hasil prediksi pada beberapa skenario *patch* diperlihatkan pada Gambar 4.75.



Gambar 4.73 Citra data tes



Gambar 4.74 *Ground truth*



Gambar 4.75 *Predicted mask*, (a) Ukuran 320x320, (b) Ukuran 512x512, (c) Ukuran 1024x1024

Gambar 4.75 adalah sampel dari berbagai *patch* dari skenario pelatihan kedua. Dari Tabel 4.13 ukuran *patch* 1024x1024 pixel memberikan nilai tertinggi pada skenario pelatihan kedua dan ketiga. Hal ini dikarenakan model dapat menangkap konteks lebih luas dengan area deteksi yang lebih besar dibandingkan *patch* kecil. Namun, hal ini tidak berlaku pada skenario pelatihan pertama akibat kegagalan model dalam mendeteksi kelas *lava dome* pada data tes, serta belum adanya penambahan data *training* dan data tes seperti yang dilakukan pada skenario kedua dan ketiga. Lalu, pada skenario kedua tidak ada peningkatan yang signifikan untuk pengujian pada berbagai skenario ukuran *patch test*.

Data yang dimuat melalui *test loader* sebelumnya dipecah menjadi *patch-patch* kecil untuk disegmentasi menggunakan model terlatih sebagaimana dijelaskan pada Subbab 4.6. Hasil segmentasi dari masing-masing *patch* kemudian digabungkan kembali untuk membentuk gambar utuh. Selain itu, waktu yang dibutuhkan untuk melakukan segmentasi juga diukur dalam proses ini. Kode implementasi dari proses ini ditampilkan pada Gambar 4.76.

```
# Define transformations for test without normalization
test_transform = A.Compose([ToTensorV2()])

# Function to convert label to RGB
def label_to_rgb(label):
    label_rgb = np.zeros((label.shape[0], label.shape[1], 3), dtype=np.uint8)
    label_colors = {
        0: [112, 240, 0], 1: [184, 0, 240], 2: [240, 184, 0], 3: [60, 0, 138],
        4: [0, 228, 240], 5: [228, 240, 0], 6: [0, 98, 255], 7: [255, 0, 4],
        8: [65, 0, 204], 9: [204, 122, 0], 10: [255, 0, 247], 11: [120, 120, 120],
        12: [255, 173, 183], 13: [122, 0, 14]
    }
    for label_idx, rgb in label_colors.items():
        label_rgb[label == label_idx] = rgb
    return label_rgb

# Define the SegmentationDataset class
class SegmentationDataset(Dataset):
    def __init__(self, image_paths, mask_paths, patch_size=1024, transform=None):
        self.image_paths = image_paths
        self.mask_paths = mask_paths
        self.patch_size = patch_size
        self.transform = transform
        self.scaler = MinMaxScaler()
        self.image_patches = []
        self.mask_patches = []
        self.create_patches()

    def create_patches(self):
        for img_path, mask_path in zip(self.image_paths, self.mask_paths):
            image = cv2.imread(img_path)
            image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
            mask = cv2.imread(mask_path, 1)
            mask = cv2.cvtColor(mask, cv2.COLOR_BGR2RGB)

            # Resize dimensions to the nearest multiple of patch_size
            new_height = round(image.shape[0] / self.patch_size) * self.patch_size
            new_width = round(image.shape[1] / self.patch_size) * self.patch_size
```

```

        resized_image = cv2.resize(image, (new_width, new_height))
        resized_mask = cv2.resize(mask, (new_width, new_height),
interpolation=cv2.INTER_NEAREST)

        # Patchify resized images and masks
        patches_img = patchify(resized_image, (self.patch_size,
self.patch_size, 3), step=self.patch_size)
        patches_mask = patchify(resized_mask, (self.patch_size,
self.patch_size, 3), step=self.patch_size)

        for i in range(patches_img.shape[0]):
            for j in range(patches_img.shape[1]):
                patch_img = patches_img[i, j, 0, :, :, :]
                patch_mask = patches_mask[i, j, 0, :, :, :]

                # Scale the image patch
                patch_img = self.scaler.fit_transform(patch_img.reshape(-1,
patch_img.shape[-1])).reshape(patch_img.shape)

                # Convert RGB mask to label mask
                patch_mask = self.rgb_to_2D_label(patch_mask)

                if self.transform:
                    patch_img = self.transform(image=patch_img)['image']
                    patch_mask = torch.tensor(patch_mask, dtype=torch.long)

                self.image_patches.append(patch_img)
                self.mask_patches.append(patch_mask)

def __len__(self):
    return len(self.image_patches)

def __getitem__(self, idx):
    return self.image_patches[idx], self.mask_patches[idx]

def rgb_to_2D_label(self, label):
    label_colors = {
        (112, 240, 0): 0, (184, 0, 240): 1, (240, 184, 0): 2, (60, 0, 138): 3,
        (0, 228, 240): 4, (228, 240, 0): 5, (0, 98, 255): 6, (255, 0, 4): 7,
        (65, 0, 204): 8, (204, 122, 0): 9, (255, 0, 247): 10, (120, 120, 120):
11,
        (255, 173, 183): 12, (122, 0, 14): 13
    }
    label_seg = np.zeros(label.shape[:2], dtype=np.uint8)
    for rgb, idx in label_colors.items():
        label_seg[np.all(label == rgb, axis=-1)] = idx
    return label_seg

# Function to visualize predictions using a DataLoader
def visualize_predictions(model, loader, original_size, resized_size,
patch_size=1024):
    model.eval()

    predictions = []
    all_masks = []

    for images, masks in loader:
        images = images.to(device).float()
        masks = masks.to(device).long()

        with torch.no_grad():
            outputs = model(images)
            preds = torch.argmax(outputs, dim=1).cpu().numpy()

        predictions.extend(preds)
        all_masks.extend(masks.cpu().numpy())

```

```

predictions = np.array(predictions)
all_masks = np.array(all_masks)

# Reshape patches back to resized image size
num_patches_x = resized_size[1] // patch_size
num_patches_y = resized_size[0] // patch_size

pred_image = unpatchify(predictions.reshape(num_patches_y, num_patches_x,
patch_size, patch_size), resized_size)
mask_image = unpatchify(all_masks.reshape(num_patches_y, num_patches_x,
patch_size, patch_size), resized_size)

# Resize back to the original image size
pred_image = cv2.resize(pred_image, (original_size[1], original_size[0]),
interpolation=cv2.INTER_NEAREST)
mask_image = cv2.resize(mask_image, (original_size[1], original_size[0]),
interpolation=cv2.INTER_NEAREST)

return pred_image, mask_image

# Load the model and weights
best_model = smp.DeepLabV3Plus(
    encoder_name="ResNet34",
    encoder_weights="imagenet",
    in_channels=3,
    classes=14
).to(device)

# Load the checkpoint and fix the key mismatch issue
checkpoint_path = '/content/drive/MyDrive/skripsi segmentasi citra satelit
/Code/CodeAllDataset25November/best_model (21).pth'
checkpoint = torch.load(checkpoint_path, map_location=device)

if "model_state_dict" in checkpoint:
    state_dict = checkpoint["model_state_dict"]
else:
    state_dict = checkpoint

# Remove "model." prefix if present
new_state_dict = {k.replace("model.", ""): v for k, v in state_dict.items()}

# Load the state dictionary
best_model.load_state_dict(new_state_dict)
best_model.eval()

# Set up paths and dataset
base_path = '/content/drive/MyDrive/skripsi segmentasi citra satelit
/AllDataset/TestFolder'
image_paths = [os.path.join(base_path, 'images', img) for img in
os.listdir(os.path.join(base_path, 'images')) if img.endswith('.jpg')]
mask_paths = [os.path.join(base_path, 'masks', img.replace('.jpg', '.png')) for
img in os.listdir(os.path.join(base_path, 'images')) if img.endswith('.jpg')]
patchsize = 320 # Patch size

# Create the dataset and loader for test images
dataset = SegmentationDataset(image_paths=image_paths, mask_paths=mask_paths,
patch_size=patchsize, transform=test_transform)
patch_loader = DataLoader(dataset, batch_size=64, shuffle=False)

# Visualization and timing
prediction_times = []
for image_path, mask_path in zip(image_paths, mask_paths):
    image = cv2.imread(image_path)
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    mask = cv2.imread(mask_path)

```

```

original_size = image.shape[:2]
new_height = round(original_size[0] / patchsize) * patchsize
new_width = round(original_size[1] / patchsize) * patchsize
resized_size = (new_height, new_width)

single_image_dataset = SegmentationDataset(
    image_paths=[image_path],
    mask_paths=[mask_path],
    patch_size=patchsize,
    transform=test_transform
)

single_patch_loader = DataLoader(single_image_dataset, batch_size=64,
shuffle=False)

start_time = time.time()
pred_image, mask_image = visualize_predictions(
    best_model, single_patch_loader, original_size, resized_size,
    patch_size=patchsize
)
prediction_time = time.time() - start_time
prediction_times.append(prediction_time)

print(f"Prediction time for {image_path}: {prediction_time:.4f} seconds")

pred_rgb = label_to_rgb(pred_image)
mask_rgb = label_to_rgb(mask_image)

plt.figure(figsize=(24, 8))
plt.subplot(131)
plt.title('Original Image')
plt.imshow(image)

plt.subplot(132)
plt.title('Ground Truth Mask')
plt.imshow(mask_rgb)

plt.subplot(133)
plt.title('Predicted Mask')
plt.imshow(pred_rgb)
plt.show()

print(f"Average prediction time per image: {np.mean(prediction_times):.4f}
seconds")

```

Gambar 4.76 Kode untuk tes model

Kode pada Gambar 4.76 bertujuan untuk melakukan proses inferensi pada model segmentasi citra satelit dengan menggunakan framework PyTorch. Model yang diterapkan adalah DeepLabV3+ dengan *encoder* ResNet34. Model ini telah dilatih sebelumnya (*pretrained weights*) menggunakan *dataset* ImageNet (He et al., 2015). Proses dimulai dengan menetapkan perangkat komputasi yang digunakan, termasuk konversi citra menjadi tensor. Fungsi `label_to_rgb` dirancang untuk mengonversi label prediksi numerik menjadi representasi RGB sesuai skema warna yang telah didefinisikan pada Tabel 3.3 guna memudahkan interpretasi hasil.

Kelas `SegmentationDataset` dirancang untuk mengelola data gambar dan *mask* secara efisien. *Dataset* ini mendukung proses *patching*, di mana citra dan *mask* dibagi menjadi *patch* berukuran 320x320 sama dengan proses *training* untuk memenuhi *input image size*. Gambar pada setiap *patch* dinormalisasi menggunakan metode `MinMaxScaler`, sedangkan *mask* dikonversi dari format RGB menjadi label numerik 2D sama seperti proses *training* model.

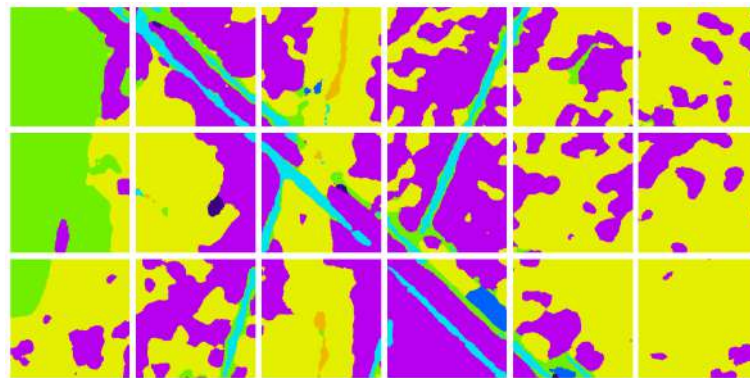
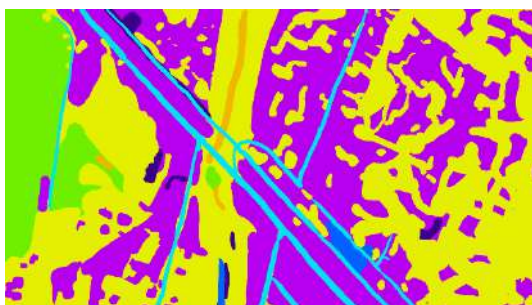
Proses inferensi dilakukan melalui fungsi `visualize_predictions` yang bertugas memprediksi *patch* gambar secara bertahap. Prediksi diawali dengan *resizing* citra berukuran besar untuk bisa habis dibagi *patch size*, ukuran *resizing* diubah berdasarkan nilai terdekat. Sampel dari citra ini ditunjukkan pada Gambar 4.77 dengan ukuran 1920x1080 dan hasil *resizing* ditampilkan pada Gambar 4.78 dengan ukuran 1920x960. Untuk citra ukuran 1024x1024 setelah melalui proses *resizing* ukurannya menjadi 960x960. Citra yang sudah melalui proses *resizing*, dilanjutkan proses *patching* dan visualisasi proses ini ditampilkan pada Gambar 4.79. Setelah proses *patching*, lalu model memprediksi setiap *patch* yang ditampilkan pada Gambar 4.80. Selesai proses prediksi selanjutnya diikuti dengan *patch* prediksi disatukan dan dikembalikan menjadi ukuran semula. Hasil akhir dari proses visualisasi prediksi ditampilkan pada Gambar 4.81.



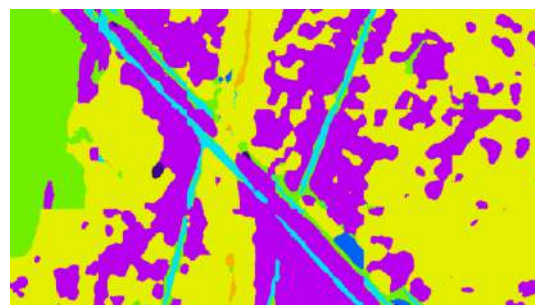
Gambar 4.77 Citra asli sebelum proses *resizing*



Gambar 4.78 Hasil *resizing* citra

Gambar 4.79 Hasil *patching* citraGambar 4.80 Hasil prediksi tiap *patch*

(a)



(b)

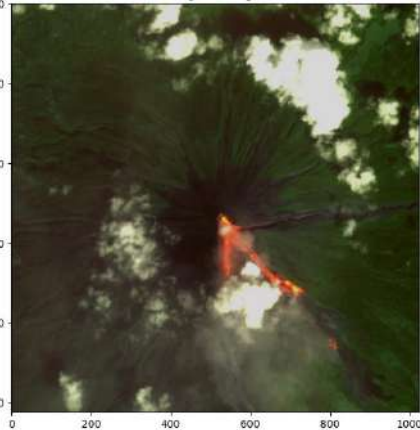
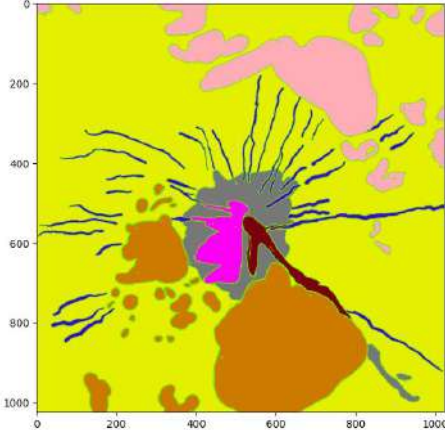
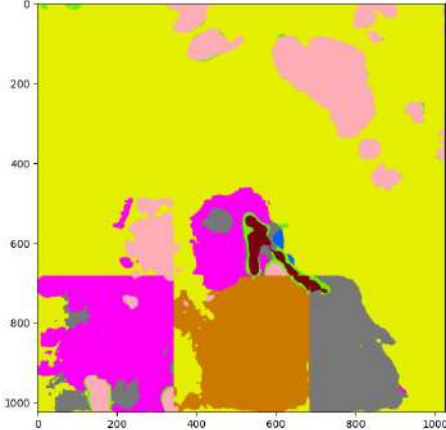
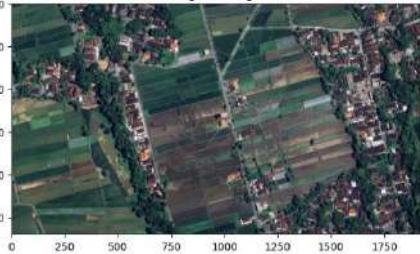
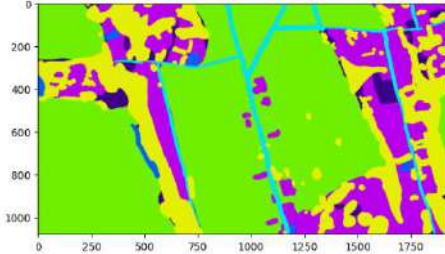
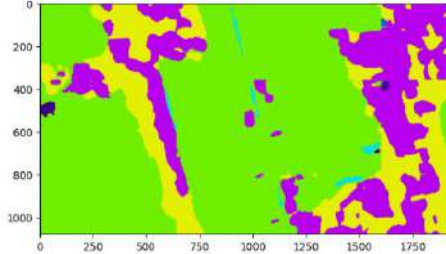
Gambar 4.81 Hasil prediksi model, (a) Ground truth, (b) Predicted mask

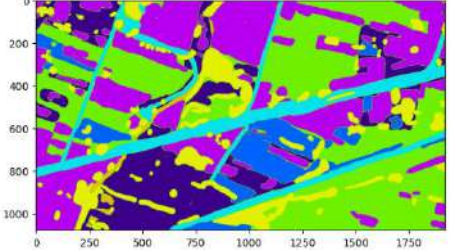
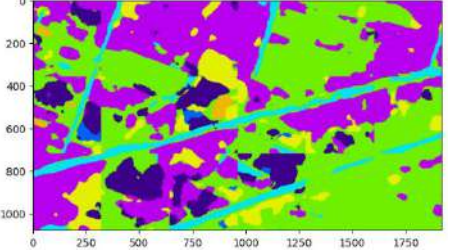
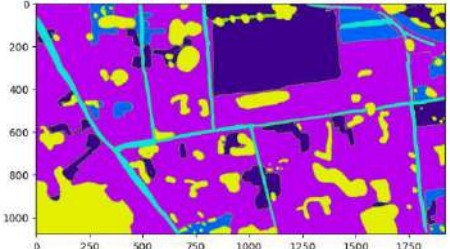
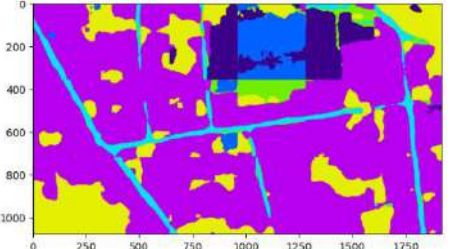
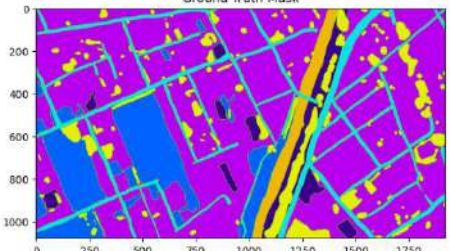
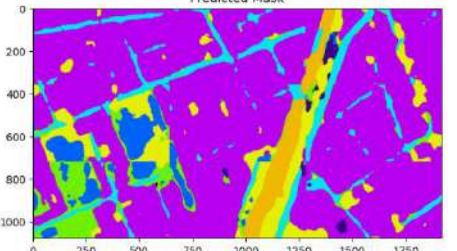
Dari Gambar 4.81 dapat dilihat bahwa citra dikembalikan menjadi rasio awal sebelum proses *resizing* menggunakan teknik *near interpolation*. Teknik ini digunakan untuk menjaga kualitas citra agar tidak menurun setelah dilakukannya proses *resizing* (Rukundo & Cao, 2012). Metode ini mengisi piksel yang kosong dengan nilai dari piksel terdekat yang memiliki

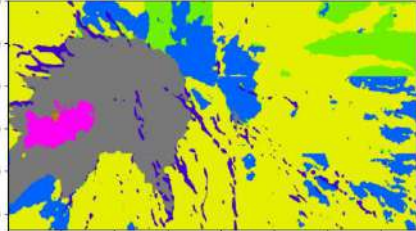
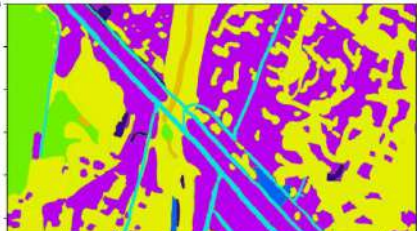
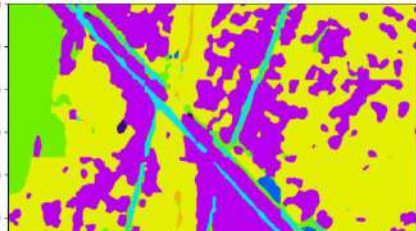
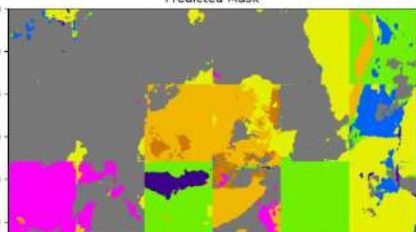
informasi, sehingga area kosong dapat terisi secara optimal berdasarkan nilai piksel sekitarnya. Setelah itu, model dimuat dari *directory* file *best_model.pth*. *Dataset* uji kemudian dibangun dari jalur direktori gambar dan *mask*.

Proses visualisasi dilakukan untuk menampilkan citra asli, *mask ground truth*, dan *mask* hasil prediksi secara berdampingan yang bertujuan untuk memvalidasi kemampuan dan menghitung metrik tes ketika model mensegmentasi citra. Pengukuran waktu inferensi untuk satu citra dilakukan dengan menjalankan beberapa kali iterasi pada data uji untuk mengevaluasi kestabilan performa model. Hasil dari visualisasi pengujian skenario pelatihan pertama terhadap 8 citra dan waktu inferensinya ditampilkan pada Tabel 4.14. Hasil prediksi yang ditampilkan berasal dari *patch* dengan nilai mAP dan mIoU tertinggi yang tercatat dalam Tabel 4.13.

Tabel 4.14 Prediksi model skenario pelatihan pertama

Nomor	Prediksi Gambar			Waktu Prediksi		
				Iterasi Ke-1	Iterasi Ke-2	Iterasi Ke-3
1				1.5s	0.04s	0.04s
2				0.3s	0.08s	0.08s

3	Original Image 	Ground Truth Mask 	Predicted Mask 	0.1s	0.08s	0.08s
4	Original Image 	Ground Truth Mask 	Predicted Mask 	0.1s	0.08s	0.08s
5	Original Image 	Ground Truth Mask 	Predicted Mask 	0.1s	0.08s	0.08s

6	Original Image 	Ground Truth Mask 	Predicted Mask 	0.1s	0.08s	0.08s
7	Original Image 	Ground Truth Mask 	Predicted Mask 	0.09s	0.08s	0.08s
8	Original Image 	Ground Truth Mask 	Predicted Mask 	0.1s	0.09s	0.08s

Dari Tabel 4.14 divisualisasikan prediksi dari skenario pelatihan pertama dengan patch terbaik pada 320x320 piksel. Model berhasil melakukan segmentasi pada citra dari data uji, dengan menampilkan *mask* hasil prediksi yang merepresentasikan warna-warna kelas pada Tabel 3.3. Namun, beberapa hasil segmentasi masih menunjukkan ketidaksempurnaan, dengan pola tampak kotak-kotak yang cukup jelas pada citra nomor 1, 6, dan 8. Hal ini disebabkan oleh proses pelatihan model yang di mana terdapat *patch* mengandung satu kelas dalam satu frame *patch*. Akibatnya, saat diuji pada citra baru, hasil segmentasi menjadi kurang halus. Pada citra nomor 8, model tidak dapat mendeteksi kelas *lava dome* karena citra telah kehilangan konteks yang diperlukan untuk dideteksi oleh model serta jumlah objek yang terbatas dalam data pelatihan.

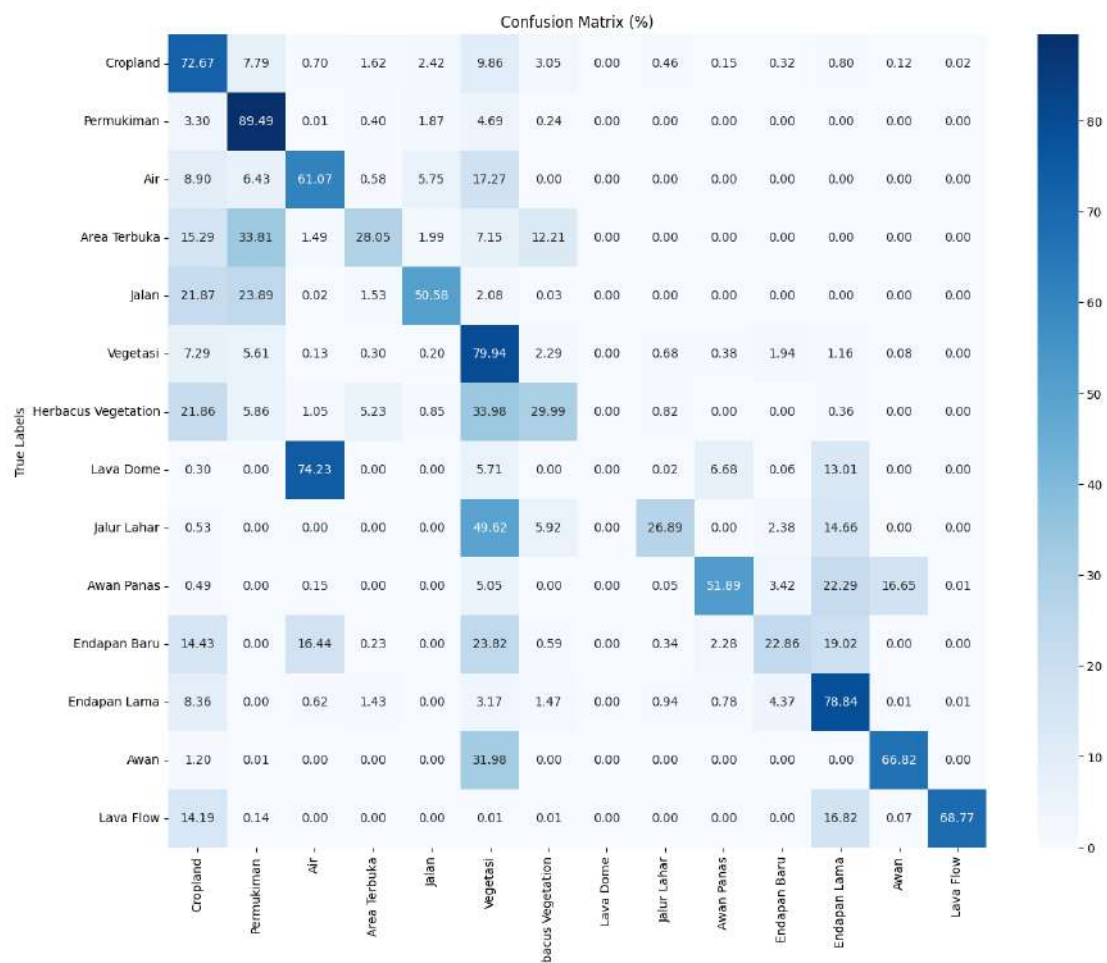
Pada citra pertama, waktu inferensi model cenderung lebih lama karena model membutuhkan proses *warming up* terlebih dahulu untuk mendeteksi citra. Namun, seiring dengan proses inferensi pada gambar berikutnya, waktu yang dibutuhkan semakin menurun. Dapat dilihat juga ketika bertambahnya iterasi pada data tes, waktu inferensi model juga semakin stabil dan lebih cepat dibandingkan dengan proses inferensi pada iterasi pertama. Setelah model menghasilkan prediksi, hasil tersebut dibandingkan dengan *ground truth* untuk menghasilkan *confusion matrix* yang menghitung perbandingan dari piksel antara *predicted mask* dan juga *ground truth mask*. *Confusion matrix* ini kemudian divisualisasikan pada Gambar 4.83.

Berdasarkan Gambar 4.83 peneliti mempersingkat penyebutan untuk kelas “Endapan Vulkanik Lama” menjadi kelas “Endapan Lama” dan kelas “Endapan Vulkanik Baru” menjadi “Endapan Baru”. Langkah ini diambil oleh peneliti dikarenakan keterbatasan tempat untuk menampilkan *confusion matrix*. Kelas “Endapan Lama” lebih mudah dikenali dibandingkan kelas “Endapan Baru” dikarenakan karakteristik warna pada endapan lama sangat mencolok, ditandai dengan warna abu-abu. Warna abu-abu ini tidak ada dalam kelas yang lain sehingga membantu model untuk mengenalinya. Karakteristik endapan lama dan endapan baru dapat dilihat pada Gambar 4.82.



Gambar 4.82 Tampak visual, (a) Endapan vulkanik lama, (b) Endapan vulkanik baru

Dapat dilihat juga bahwa sembilan kelas berhasil disegmentasi dengan tingkat keberhasilan di atas 50%. Kelas “Permukiman” memiliki tingkat keberhasilan tertinggi, mencapai 89%. Hal ini dapat dipahami karena kelas “Permukiman” mendominasi *dataset* dan memiliki ciri visual yang khas, seperti keberadaan atap bangunan yang ditampilkan pada Gambar 4.84. Tentunya hal ini dapat mempermudah model dalam mengenali pola tersebut.



Gambar 4.83 *Confusion matrix* data tes skenario pelatihan pertama



Gambar 4.84 Tampak visual atap bangunan

Lalu kelas “Lava Flow” yang berhasil dideteksi dengan nilai 68% walaupun dengan jumlah kelas pada *dataset* yang sedikit. Keberhasilan ini dipengaruhi oleh fitur dari *lava flow* itu sendiri yang mencolok, ditandai dengan adanya warna merah yang menyala setelah di *highlight* dengan *band* NIR dan berbentuk memanjang. Sehingga hal ini memudahkan model untuk mengenali objek *lava flow*.

Sebaliknya, deteksi untuk kelas “Area Terbuka” dan “Herbaceous Vegetation” masih kurang optimal. Kondisi ini disebabkan oleh jumlah sampel untuk kedua kelas tersebut lebih sedikit dibandingkan kelas lainnya, dan secara tampak visual kedua kelas tersebut sangat identik. Hal ini menyebabkan model sering kali memprediksi keduanya secara keliru. Perbandingan visual dari kedua kelas ini ditampilkan pada Gambar 4.85.



(a)



(b)

Gambar 4.85 Tampak visual, (a) *Herbaceous vegetation*, (b) Area terbuka

Gambar 4.85 adalah tampak visual dari kelas “Herbaceous Vegetation” dan “Area Terbuka”. Bisa dilihat bahwa secara visual warna dan tekstur dari kedua gambar tersebut hampir identik. Tentunya dengan adanya kasus seperti ini menyebabkan model kesulitan untuk membedakan kelasnya.

Selain itu, kelas “Area Terbuka” lebih sering terdeteksi sebagai kelas “Permukiman”. Contoh kasus ini dapat dijumpai pada citra nomor lima pada Tabel 4.14. Citra tersebut ditampilkan pada Gambar 4.86.

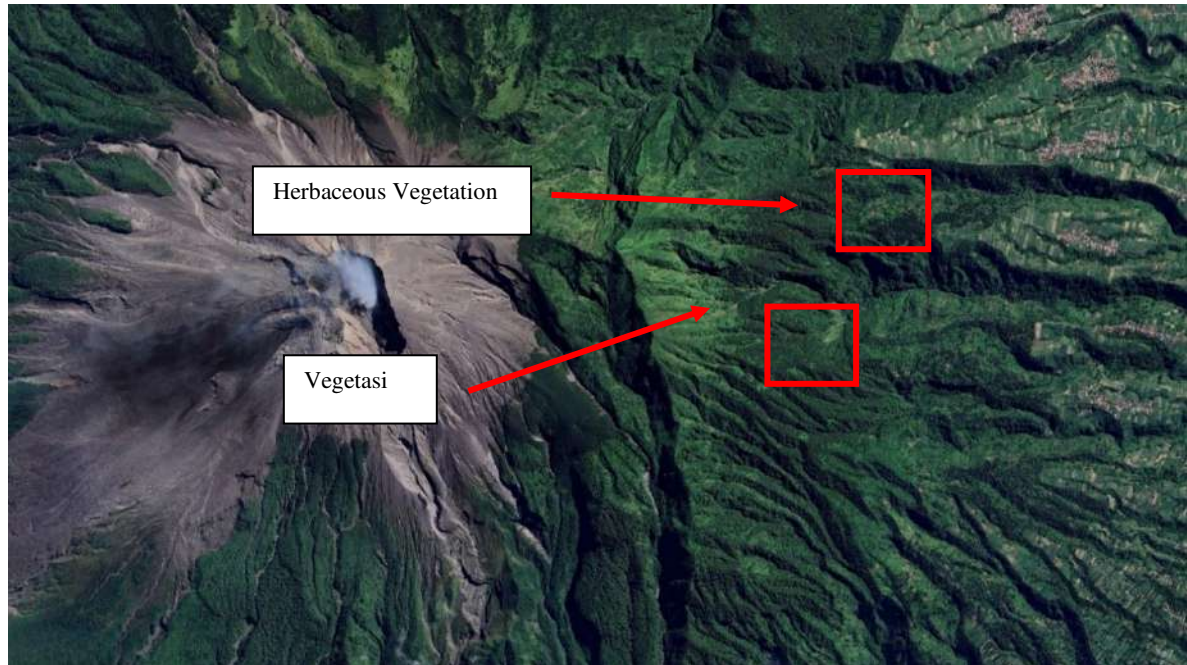


Gambar 4.86 Citra tes nomor lima

Dari Gambar 4.83 kelas “Area Terbuka” diwakili oleh lapangan olahraga. Model gagal mensegmentasi lapangan tersebut dikarenakan kelas “Area Terbuka” sering kali dikelilingi oleh bangunan sehingga menimbulkan kesalahan prediksi. Tentunya dengan adanya masalah ini juga dapat menurunkan performa model secara keseluruhan.

Kelas “Herbaceous Vegetation” masih memprediksi kelas yang salah, dengan kesalahan tertinggi memprediksi kelas “Vegetasi”. Hal ini dapat dipahami mengingat bahwa kedua kelas tersebut memiliki karakteristik hampir mirip. Di mana diantara kedua kelas tersebut memiliki ciri seperti warna hijau dari tumbuh-tumbuhan. Kesalahan prediksi ini dapat ditemui pada citra nomor enam di Tabel 4.14. Citra nomor enam ditampilkan pada Gambar 4.87. Untuk

memberikan pemahaman lebih mendalam disajikan pula area yang ditunjuk panah berwarna merah dan kedua area tersebut disandingkan bersama pada Gambar 4.87.



Gambar 4.87 Citra tes nomor enam



(a)



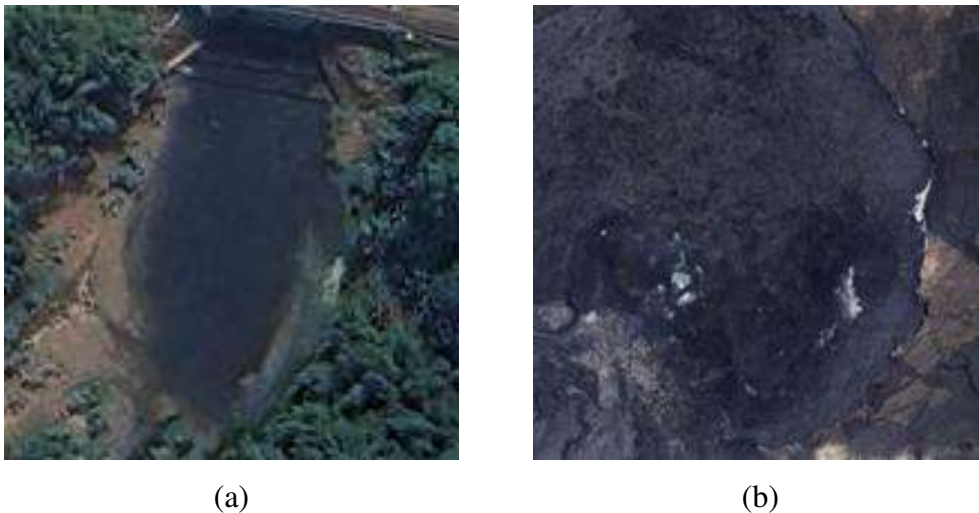
(b)

Gambar 4.88 Tampak visual, (a) *Herbaceous vegetation*, (b) *Vegetasi*

Pada Gambar 4.88 terlihat bahwa kedua kelas memiliki kesamaan tampilan visual. Kedua citra menunjukkan keberadaan pohon-pohon, meskipun pada area “Herbaceous Vegetation” jumlahnya lebih sedikit dibandingkan dengan area vegetasi. Area “Herbaceous Vegetation”

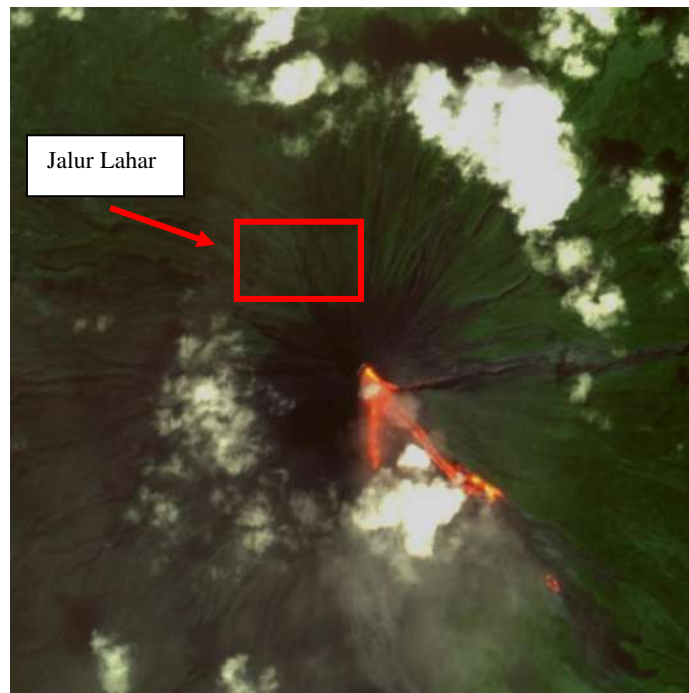
juga masih tampak terlihat sedikit keberadaan *bare soil*. Namun, kehadiran pohon-pohon di area “Herbaceous Vegetation” meningkatkan kesalahan prediksi menjadi “Vegetasi”.

Kelas “Lava Dome” mengalami kendala deteksi oleh model, di mana keseluruhan wilayah yang seharusnya termasuk dalam kelas “Lava Dome” disegmentasi sebagai kelas “Air”. Hal ini dapat terjadi disebabkan oleh kemiripan karakteristik seperti warna gelap antara kedua kelas sehingga menyulitkan model untuk membedakannya. Selain itu faktor lain yang mempengaruhi kegagalan deteksi untuk kelas “Lava Dome” yaitu jumlah kelas yang sedikit dalam *dataset*. Tampak visual dari kedua kelas ini ditampilkan pada Gambar 4.89.



Gambar 4.89 Tampak visual, (a) Kelas air, (b) Kelas *lava dome*

Masalah lain juga muncul yaitu kelas “Jalur Lahar” menunjukkan tingkat deteksi rendah, seperti terjadi pada citra nomor satu pada Tabel 4.14. Kendala ini muncul dikarenakan detail penting hilang akibat proses *resizing* di Subbab 4.1. Pada citra tersebut, kelas “Jalur Lahar” tidak dapat terdeteksi sama sekali. Model memprediksi keseluruhan kelas “Jalur Lahar” menjadi kelas “Vegetasi”. Citra nomor satu ini ditampilkan pada Gambar 4.90.



Gambar 4.90 Citra tes nomor satu

Tampak pada Gambar 4.90 terlihat bahwa kelas “Jalur Lahar” kehilangan teksturnya. Jalur lahar pada citra tersebut juga tampak sangat kecil diantara vegetasi sekelilingnya. Untuk memberikan pengetahuan lebih lanjut peneliti memperbesar area yang ditunjuk panah yang berwarna merah. Area ini ditunjukkan pada Gambar 4.91.



Gambar 4.91 Area jalur lahar

Gambar 4.91 memperlihatkan hilangnya detail pada kelas jalur lahar yang ditandai dengan warna hitam. Tekstur pada area vegetasi yang semestinya menunjukkan keberadaan pohon juga tidak terlihat. Hal ini disebabkan oleh perbedaan satelit yang digunakan dalam

pengambilan citra pada Google Earth dan penelitian (Del Rosso et al., 2021). Akibatnya model lebih condong memprediksi kelas dominan, sehingga kelas jalur lahar disegmentaasi sebagai kelas vegetasi.

Pada citra nomor enam, kelas “Jalur Lahar” sulit dibedakan dari kelas “Endapan Vulkanik Lama” karena lokasinya berada di area yang didominasi oleh endapan vulkanik, sehingga hal ini dapat meningkatkan kesalahan prediksi. Tampak visual dari dominasi kelas endapan vulkanik pada jalur lahar ditampilkan pada Gambar 4.92.



Gambar 4.92 Tampak visual kelas jalur lahar diantara kelas endapan vulkanik lama

Selanjutnya, kelas “Endapan Vulkanik Lama” sering mengalami salah deteksi. Misalnya dalam satu *frame patch*, model cenderung mensegmentasi seluruh frame sebagai kelas “Cropland”. Kesalahan ini disebabkan oleh kemiripan fitur visual antara “Endapan Vulkanik Lama” dengan lahan pertanian. Kondisi visual ini pada beberapa kasus seperti kelas “Endapan Vulkanik Lama” menyerupai lahan pertanian yang luas terutama setelah masa panen. Visualisasi dari kelas “Cropland” dan “Endapan Vulkanik Lama” ditampilkan pada Gambar 4.93.



Gambar 4.93 Tampak visual, (a) Kelas *cropland*, (b) Kelas endapan vulkanik lama

Kelas “Endapan Baru” juga masih mengalami kesalahan deteksi di mana model memprediksi menjadi kelas “Vegetasi”. Kesalahan ini dikarenakan kualitas yang kurang memadai pada data *training*. Tampak visual ini ditampilkan pada Gambar 4.94.



Gambar 4.94 Tampak visual, (a) Vegetasi pada data *training*, (b) Endapan baru

Dari Gambar 4.94, diketahui bahwa objek vegetasi pada data *training* secara visual sangat mirip dengan citra endapan baru pada data *testing*. Kemiripan ini terjadi karena vegetasi pada citra data *training* tertutupi oleh bayangan awan dan tampak menjadi gelap. Selain itu, rendahnya tingkat detail pada citra tersebut membuat model kesulitan mempelajari informasi selama proses pelatihan yang pada akhirnya menyebabkan kesalahan prediksi pada data *testing*.

Adanya kelas yang terdeteksi rendah serta satu kelas gagal disegmentasi menjadi faktor utama nilai metrik yang rendah. Untuk memberikan pemahaman lebih mendalam, peneliti

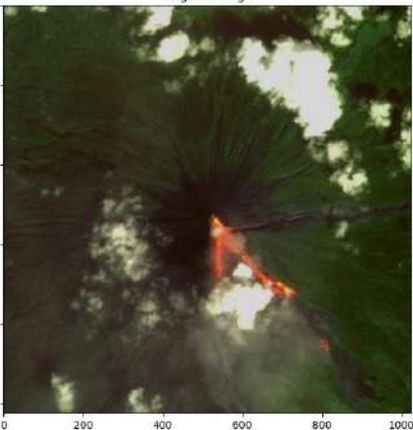
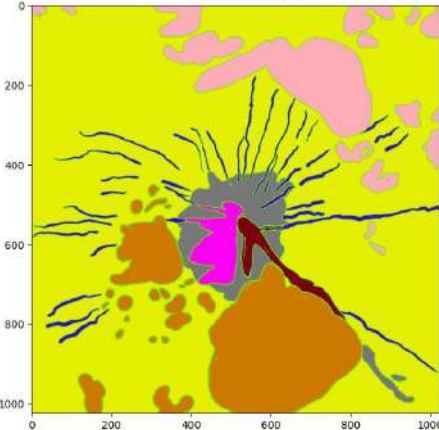
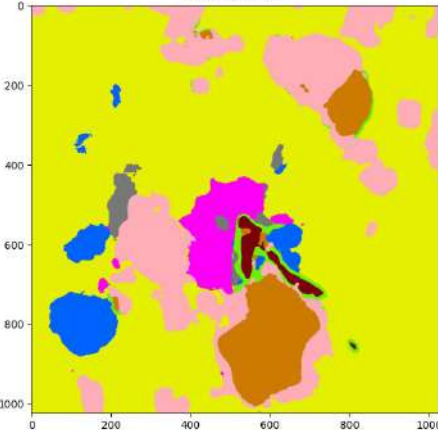
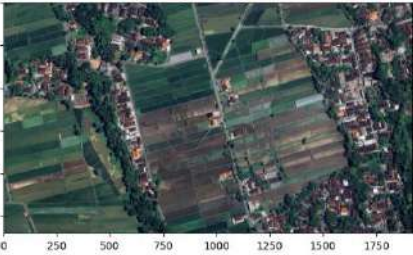
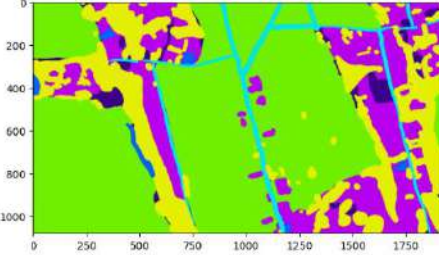
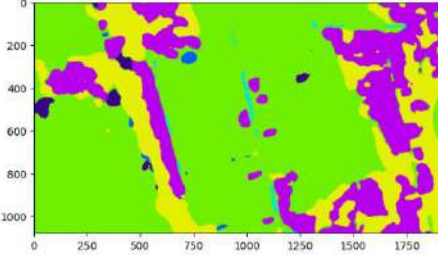
memaparkan analisis metrik per kelas, meliputi IoU, *precision*, *recall*, dan *F1-score (dice)*. Hasi metrik ini disajikan secara visual pada Gambar 4.95.

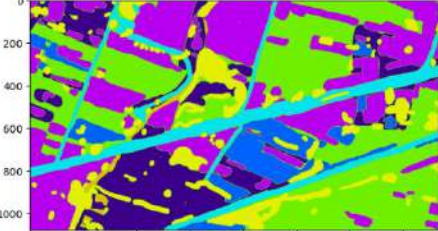
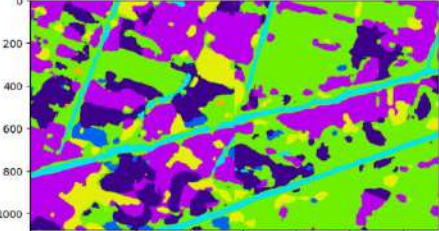
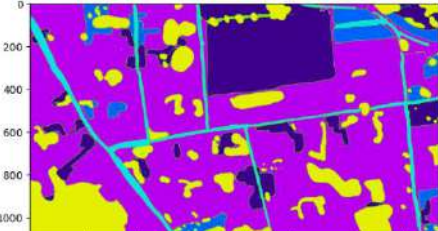
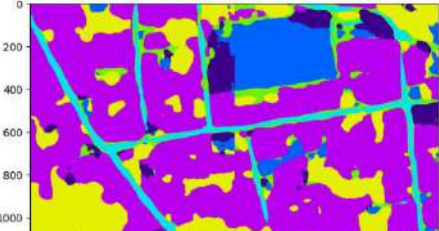
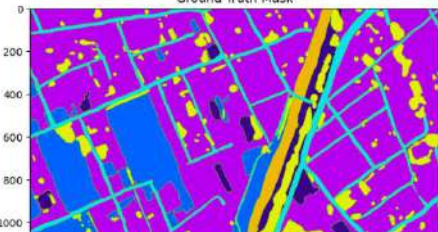
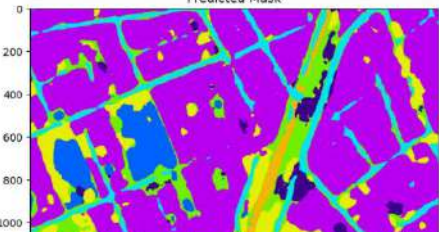
Class	IoU	Precision	Recall	F1 Score
Cropland	0.519193	0.645196	0.726665	0.683511
Permukiman	0.737975	0.807977	0.894934	0.849236
Air	0.152146	0.168489	0.610670	0.264109
Area Terbuka	0.234031	0.585585	0.280485	0.379295
Jalan	0.393370	0.638978	0.505782	0.564631
Vegetasi	0.619052	0.732929	0.799369	0.764709
Herbaceous Vegetation	0.220171	0.452973	0.299913	0.360885
Lava Dome	0.000000	0.000000	0.000000	0.000000
Jalur Lahar	0.151689	0.258204	0.268851	0.263420
Awan Panas	0.374115	0.572763	0.518928	0.544518
Endapan Baru	0.181877	0.470953	0.228579	0.307777
Endapan Lama	0.679426	0.830982	0.788372	0.809117
Awan	0.507874	0.679176	0.668173	0.673629
Lava Flow	0.636060	0.894389	0.687712	0.777551

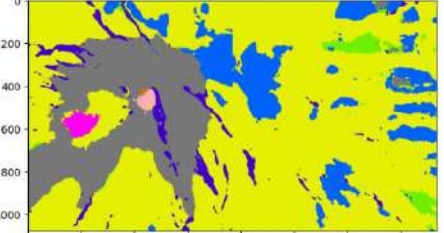
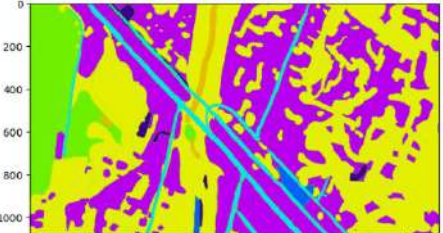
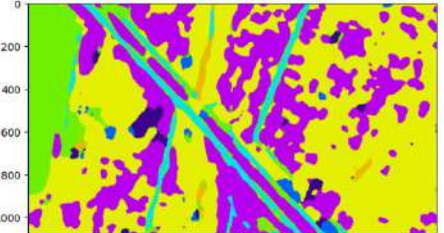
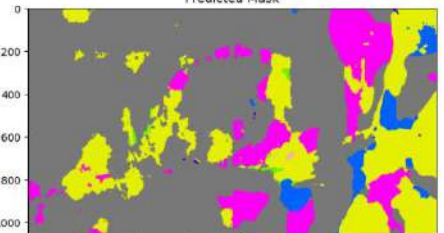
Gambar 4.95 Metrik per kelas

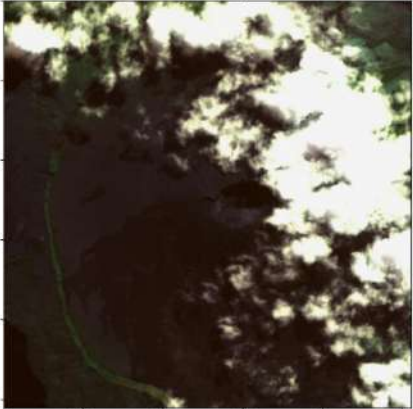
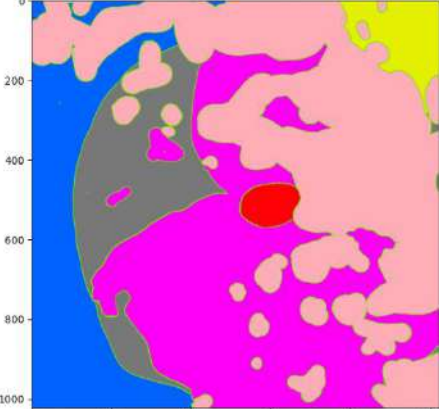
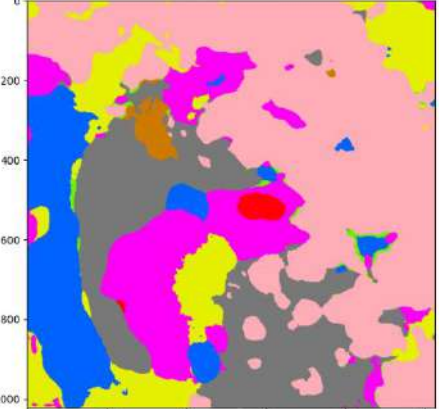
Gambar 4.95 menyajikan hasil analisis metrik untuk setiap kelas. Terlihat bahwa nilai metrik untuk kelas "Lava Dome" sepenuhnya bernilai 0. Indikasi ini secara jelas menunjukkan bahwa model tidak mampu mengenali kelas tersebut. Temuan ini menjadi perhatian penting untuk penelitian selanjutnya, khususnya dalam meningkatkan performa model pada kelas-kelas dengan nilai metrik yang masih rendah. Selanjutnya divisualisasikan juga hasil prediksi dari skenario pelatihan kedua pada Tabel 4.15 pada ukuran *patch* 1024x1024.

Tabel 4.15 Prediksi model skenario pelatihan kedua

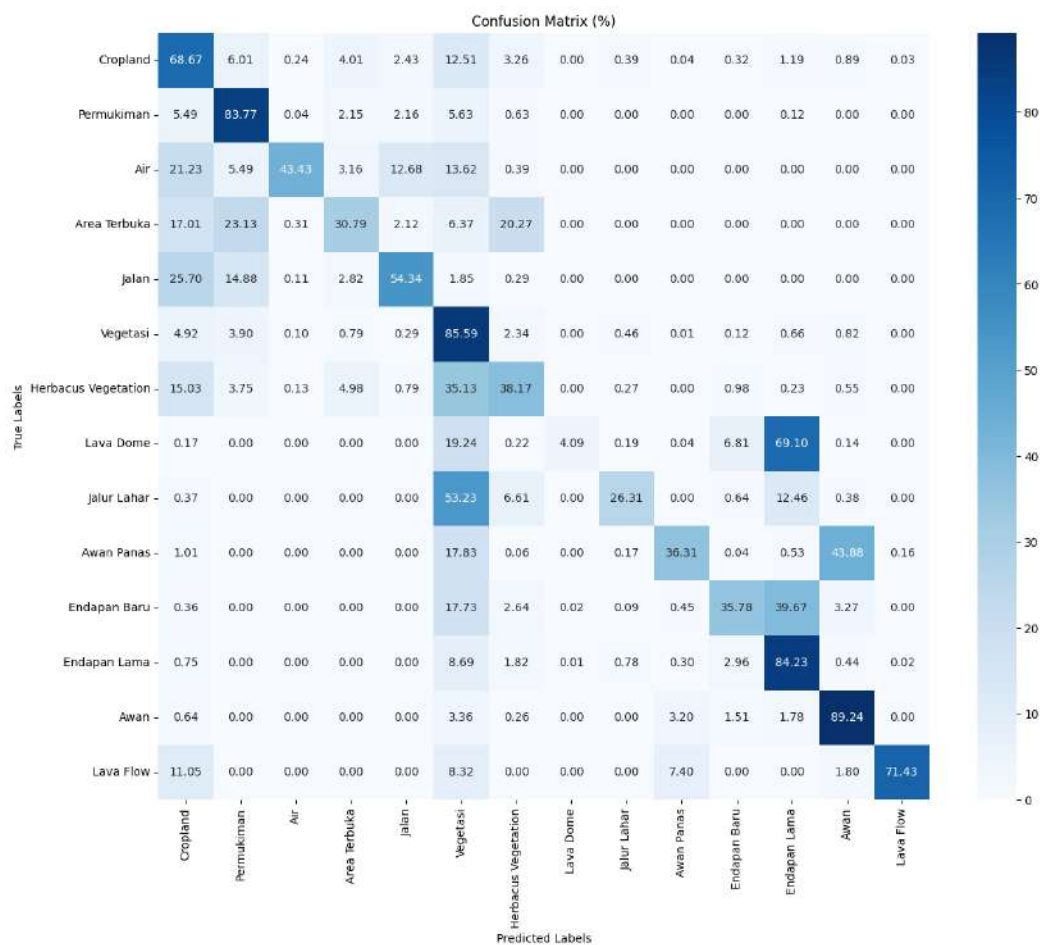
Nomor	Prediksi Gambar			Waktu Prediksi		
				Iterasi Ke-1	Iterasi Ke-2	Iterasi Ke-3
1				1.5s	0.04s	0.04s
2				0.3s	0.08s	0.08s

3	 Original Image	 Ground Truth Mask	 Predicted Mask	0.1s	0.08s	0.08s
4	 Original Image	 Ground Truth Mask	 Predicted Mask	0.1s	0.08s	0.08s
5	 Original Image	 Ground Truth Mask	 Predicted Mask	0.1s	0.08s	0.08s

6	 Original Image	 Ground Truth Mask	 Predicted Mask	0.1s	0.08s	0.08s
7	 Original Image	 Ground Truth Mask	 Predicted Mask	0.09s	0.08s	0.08s
8	 Original Image	 Ground Truth Mask	 Predicted Mask	0.1s	0.09s	0.08s

9	Original image 	Ground Truth Mask 	Predicted Mask 	0.5s	0.04s	0.04s
---	---	---	---	------	-------	-------

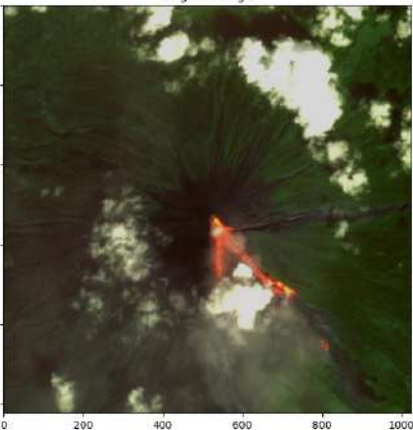
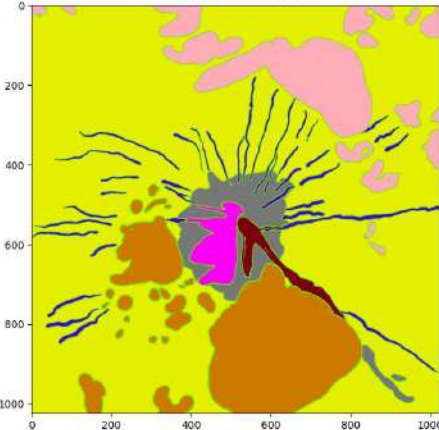
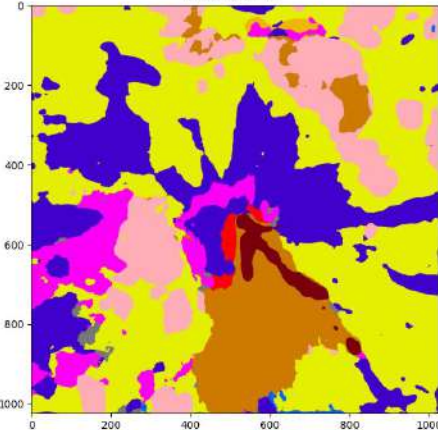
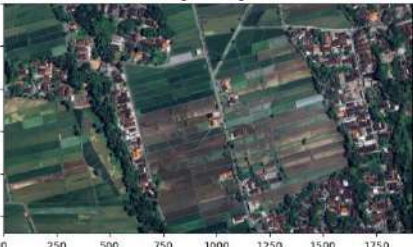
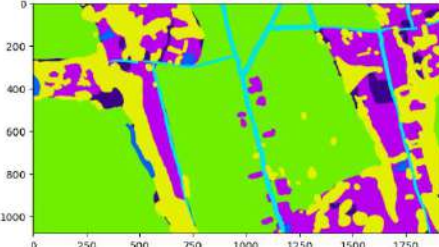
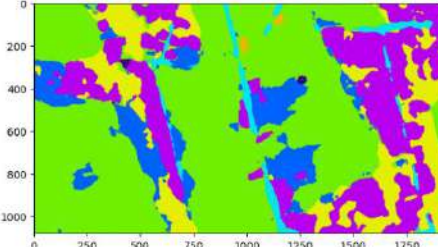
Dari Tabel 4.15 divisualisasikan prediksi dari skenario pelatihan kedua dengan *patch* terbaik pada 1024x1024 piksel. Dengan adanya penambahan citra pada data *train*, kelas *lava dome* berhasil terdeteksi oleh model yang dapat dilihat pada citra nomor 9 di Tabel 4.15. Dari keseluruhan citra juga tidak tampak pola kotak-kotak dikarenakan luasnya area yang dapat dideteksi oleh model. Untuk mendapatkan pemahaman yang lebih mendalam ditampilkan *confusion matrix* pada Gambar 4.96.

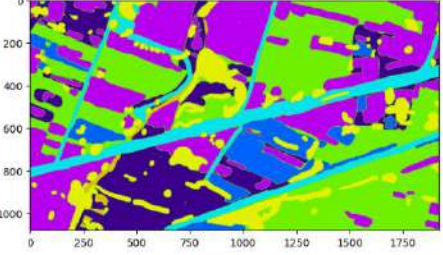
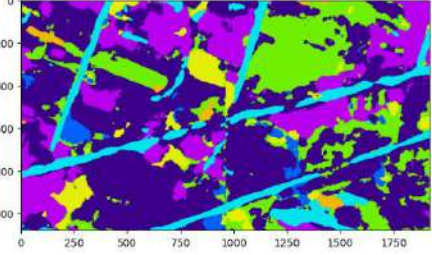
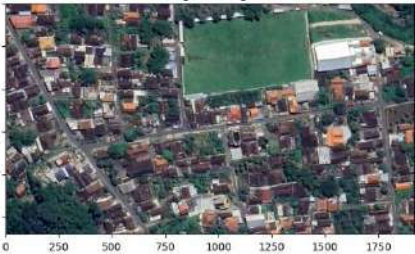
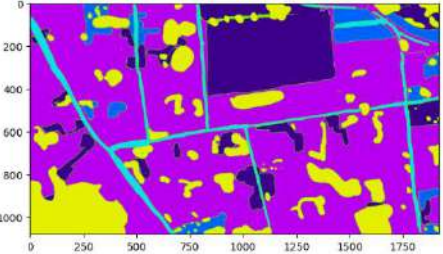
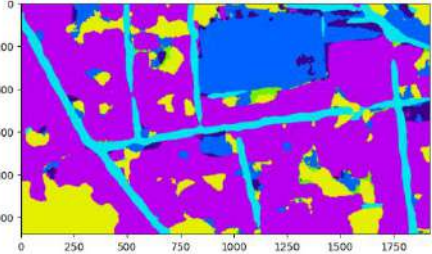
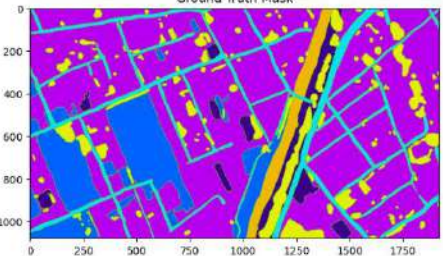
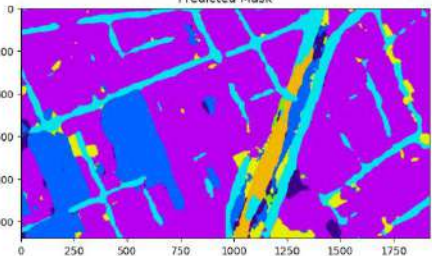


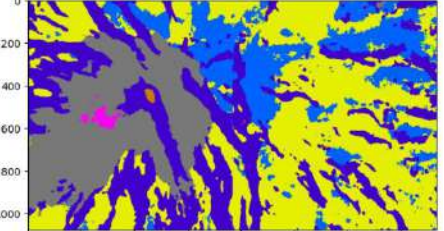
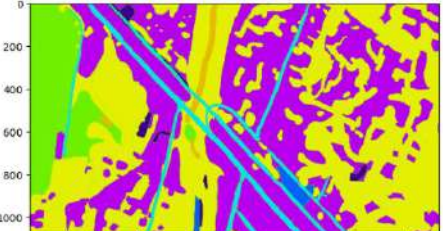
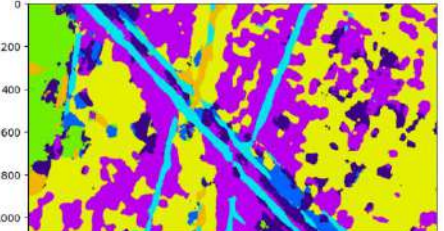
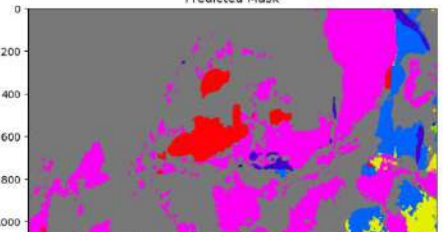
Gambar 4.96 *Confusion matrix* data tes skenario pelatihan kedua

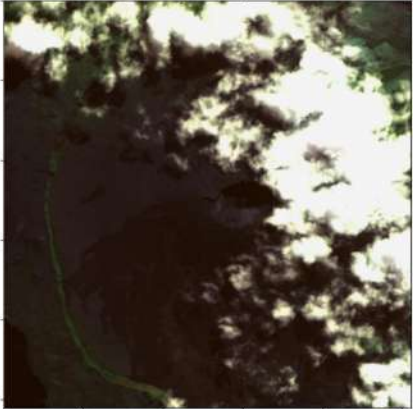
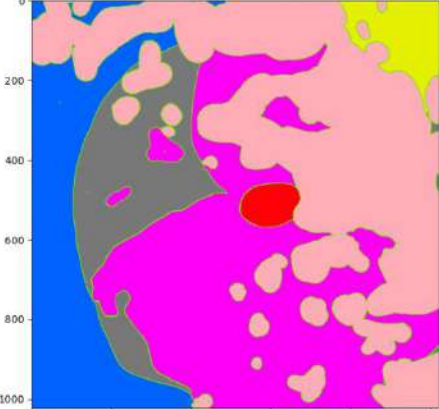
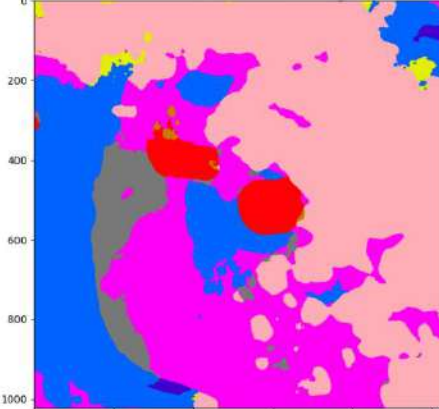
Gambar 4.96 menunjukkan bahwa model berhasil mengenali kelas *lava dome* dengan tingkat prediksi sebesar 4% yang meningkat dibandingkan dengan skenario pelatihan pertama. Meskipun demikian, model masih lebih sering mengklasifikasikan area tersebut sebagai endapan lama daripada *lava dome*. Hal ini disebabkan oleh letak *lava dome* yang dikelilingi oleh endapan lama, sehingga model kesulitan membedakan kedua kelas tersebut. Berikutnya ditampilkan visualisasi untuk skenario pelatihan ketiga pada Tabel 4.16 untuk ukuran *patch* 1024x1024.

Tabel 4.16 Prediksi model skenario pelatihan ketiga

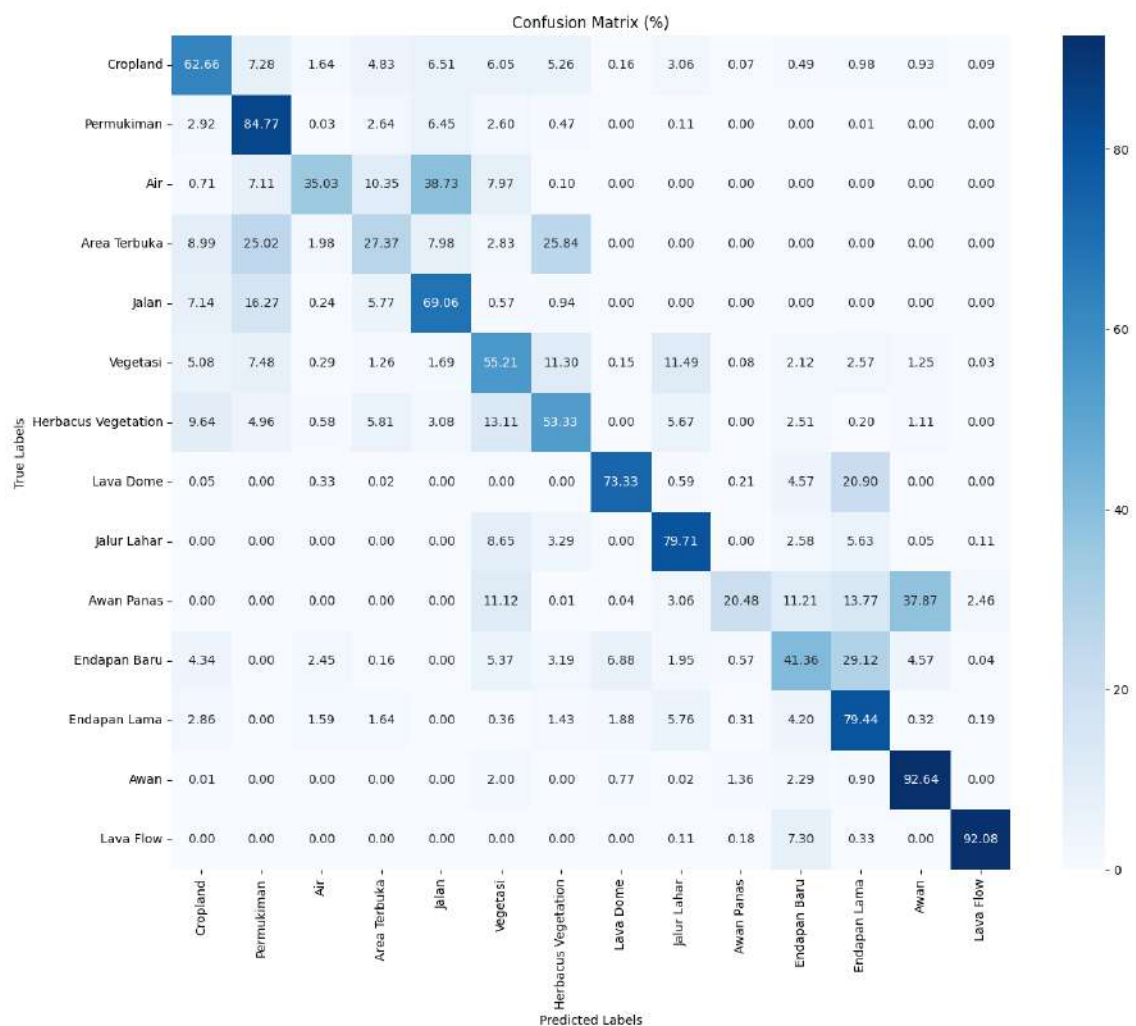
Nomor	Prediksi Gambar			Waktu Prediksi		
				Iterasi Ke-1	Iterasi Ke-2	Iterasi Ke-3
1				1.5s	0.04s	0.04s
2				0.3s	0.08s	0.08s

3	Original Image 	Ground Truth Mask 	Predicted Mask 	0.1s	0.08s	0.08s
4	Original Image 	Ground Truth Mask 	Predicted Mask 	0.1s	0.08s	0.08s
5	Original Image 	Ground Truth Mask 	Predicted Mask 	0.1s	0.08s	0.08s

6	Original Image 	Ground Truth Mask 	Predicted Mask 	0.1s	0.08s	0.08s
7	Original Image 	Ground Truth Mask 	Predicted Mask 	0.09s	0.08s	0.08s
8	Original Image 	Ground Truth Mask 	Predicted Mask 	0.1s	0.09s	0.08s

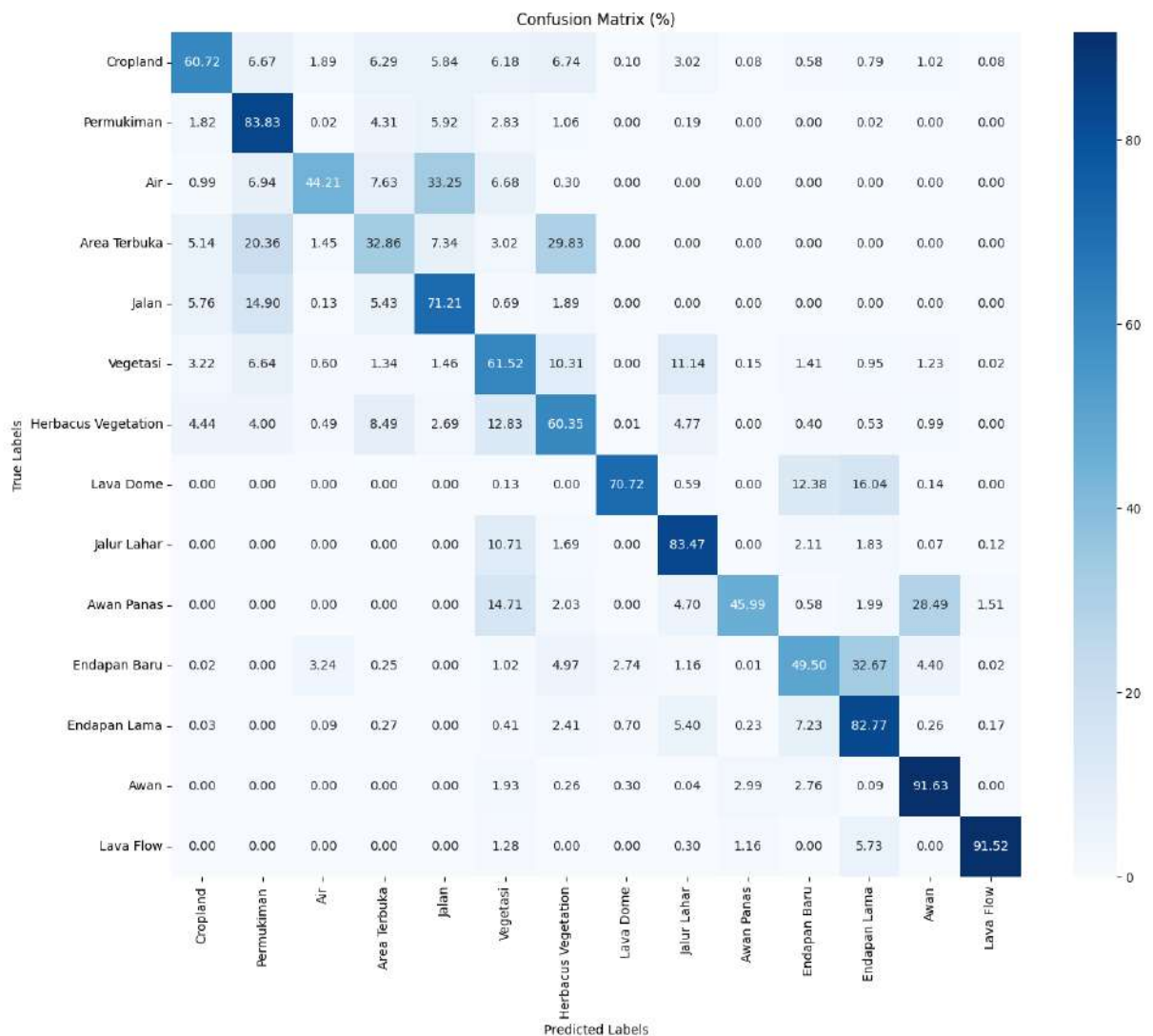
9	Original image 	Ground Truth Mask 	Predicted Mask 	0.5s	0.04s	0.04s
---	---	---	---	------	-------	-------

Berdasarkan Tabel 4.16, hasil prediksi model dengan penambahan pembobotan kelas serta modifikasi fungsi *loss* menunjukkan peningkatan secara visual. Misalnya, pada citra nomor 8, kelas *lava dome* berhasil terdeteksi lebih baik dibandingkan dengan skenario pertama dan kedua. Lalu jalur lahar berhasil terdeteksi pada citra nomor 1 dan nomor 6 walaupun dengan hasil yang belum tampak baik. Untuk memahami lebih mendalam ditampilkan *confusion matrix* untuk tes *patch* ukuran 320x320 piksel pada Gambar 4.97.



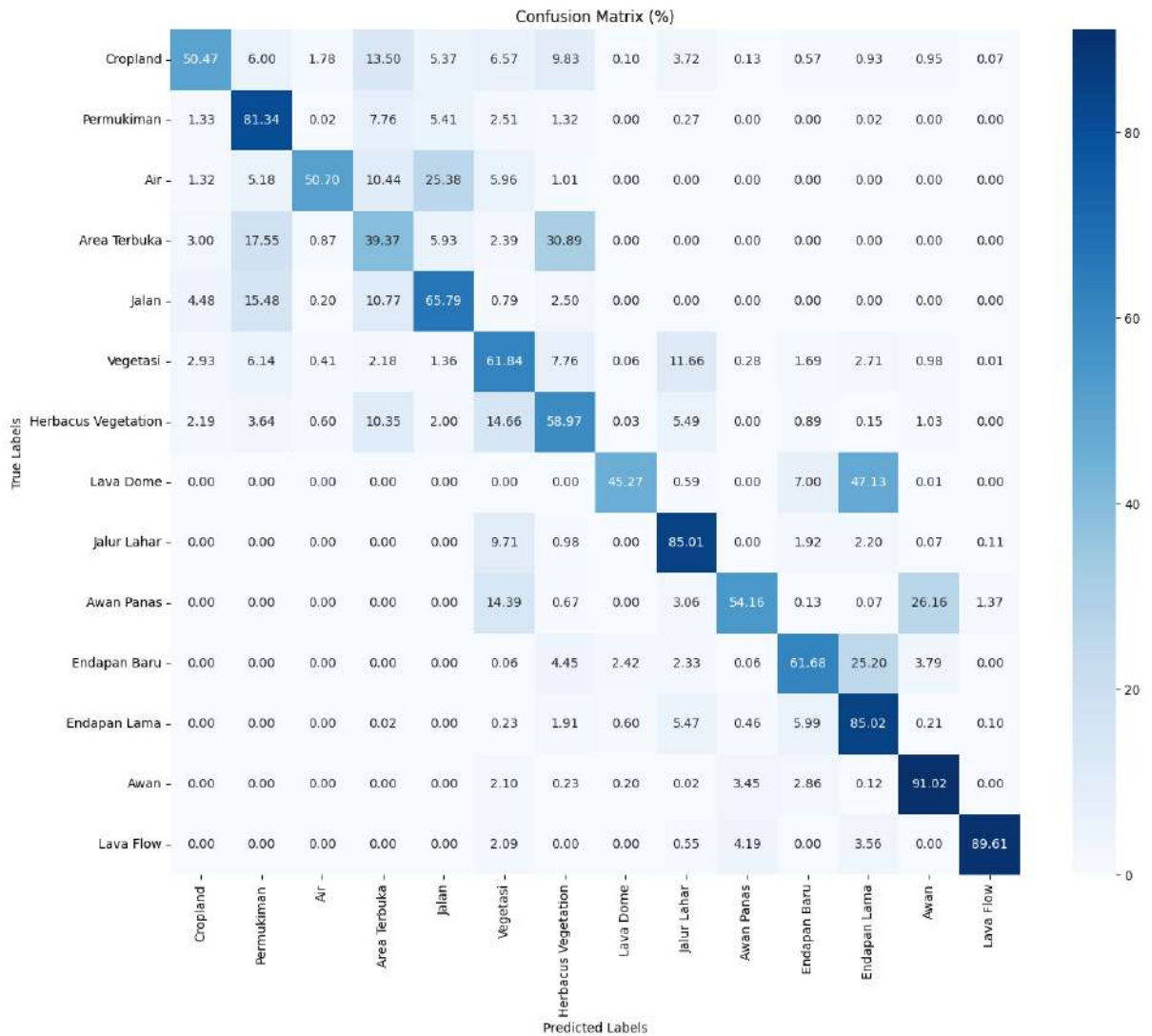
Gambar 4.97 *Confusion matrix* data tes skenario pelatihan ketiga ukuran *patch* 320x320

Pada Gambar 4.97 terlihat bahwa dengan ukuran tes *patch* 320x320, model mampu mendeteksi kelas *lava dome* hingga 79%. Penambahan pembobotan kelas serta modifikasi fungsi *loss* terbukti berkontribusi dalam meningkatkan performa model. Diperlukan eksplorasi lebih lanjut terhadap parameter tes *patch size* untuk mencapai hasil yang optimal. Untuk hasil prediksi pada ukuran *patch* 512x512 piksel ditunjukkan pada Gambar 4.98.



Gambar 4.98 *Confusion matrix* data tes skenario pelatihan ketiga ukuran *patch* 512x512

Dari Gambar 4.98 dengan adanya penambahan ukuran pada tes *patch* mengalami sedikit peningkatan dari ukuran 320. Misalnya, beberapa kelas mengalami peningkatan 5%-10% diantaranya kelas air, jalan, area terbuka dan beberapa kelas lainnya. Untuk memberikan pemahaman yang lebih mendalam diperlihatkan *confusion matrix* untuk ukuran *patch* 1024x1024 pada Gambar 4.99.

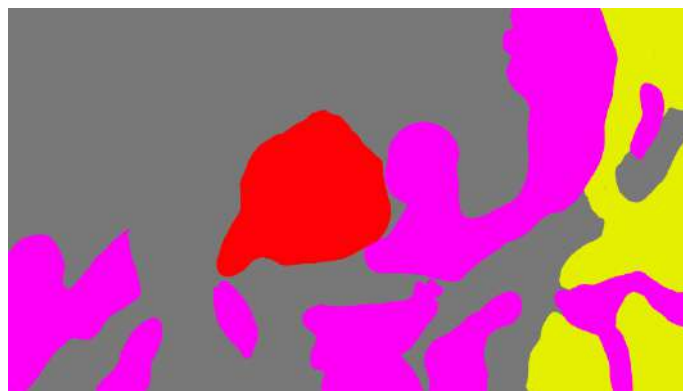


Gambar 4.99 *Confusion matrix* data tes skenario pelatihan ketiga ukuran *patch* 1024x1024

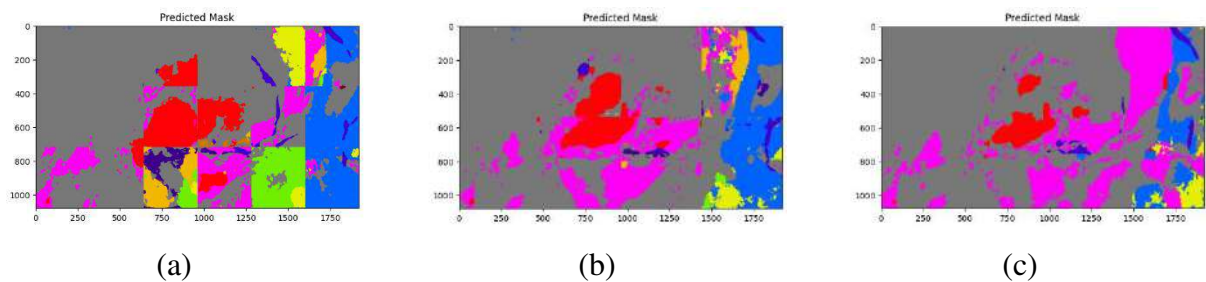
Gambar 4.99 menampilkan *confusion matrix* yang menunjukkan bahwa segmentasi untuk kelas *lava dome* mengalami penurunan yang signifikan dibandingkan dengan *patch size* sebelumnya. Hal ini mengindikasikan adanya *trade-off* antara ukuran *patch* dan kemampuan model dalam mendeteksi kelas *lava dome*. Semakin besar ukuran *patch*, semakin sulit bagi model untuk membedakan *lava dome* dari endapan lama karena lokasinya yang dikelilingi oleh endapan lama. Untuk meninjau *trade-off* ini divisualisasikan citra data tes nomor 8 di Tabel 4.16 pada Gambar 4.102.



Gambar 4.100 Citra tes nomor delapan



Gambar 4.101 *Ground truth* citra nomor delapan



Gambar 4.102 Hasil prediksi *lava dome*, (a) Ukuran patch 320x320, (b) Ukuran patch 512x512, (c) Ukuran patch 1024x1024

Gambar 4.102 menunjukkan hasil prediksi dari berbagai *patch* untuk Gambar 4.100. Bisa dilihat adanya *trade-off* yaitu semakin besar *patch size* maka area *lava dome* makin sulit terdeteksi ditandai dengan makin mengecilnya area berwarna merah pada *patch size* 1024x1024. Namun semakin besar ukuran *patch* dapat menghilangkan pola kotak-kotak dikarenakan area deteksi yang luas.

4.9 Analisis Hasil

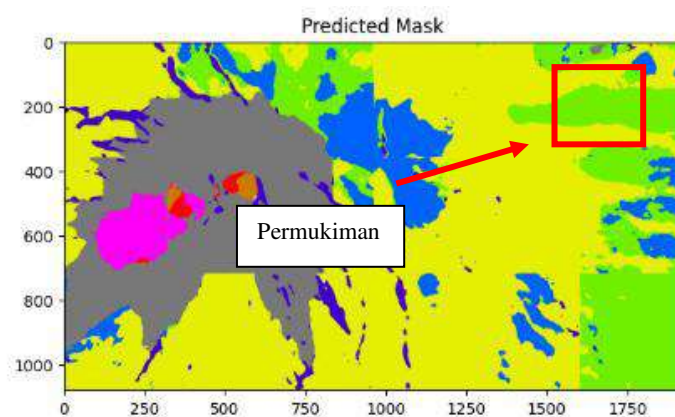
Penelitian ini menyoroti beberapa aspek penting. Pertama, dari kelebihan dan kekurangan model. Kelebihan dari penggunaan DeepLabV3+ dan konfigurasi parameter yang optimal, ditambah penerapan *weight decay* serta *dropout*, berhasil menghasilkan model dengan kemampuan generalisasi yang cukup. Penambahan pembobotan kelas dan juga modifikasi fungsi *loss* mampu meningkatkan performa model. Namun diperlukan eksplorasi dalam modifikasi fungsi *loss* untuk mendapatkan model yang benar-benar optimal. Penerapan *transfer learning* serta pemilihan *backbone* yang tepat juga memberikan pengaruh signifikan terhadap performa model dalam segmentasi citra. Namun, kompleksitas arsitektur DeepLabV3+ dengan 22.4 juta parameter menyebabkan penggunaan memori yang cukup tinggi. Waktu inferensi masih tergolong lama dan membutuhkan beberapa iterasi untuk mencapai kestabilan model, hal ini tentunya akan menjadi tantangan tersendiri dalam penerapannya di dunia nyata. Oleh karena itu, dibutuhkan pengembangan lebih lanjut untuk meningkatkan efisiensi model ini tanpa mengorbankan kinerjanya.

Dalam pengumpulan citra satelit, sulit ditemukan citra RGB gunung api yang secara simultan menampilkan aliran lava (*lava flow*) dan bebas dari gangguan awan. Tantangan ini mendorong pengembangan segmentasi citra berbasis citra hiperspektral yang memanfaatkan berbagai band spektral dari satelit. Selain itu, keterbatasan ketersediaan citra secara *real time* menjadi kendala pada proses pengumpulan citra, hal ini dikarenakan satelit hanya dapat merekam gambar pada interval waktu tertentu. Misalnya untuk satelit Sentinel 2 milik European Space Agency hanya mampu merekam pada koordinat yang sama dalam rentang 5 hari sekali (European Space Agency, n.d.). Masalah ini menuntut penelitian lebih lanjut, terutama mengingat pentingnya kecepatan informasi dalam pemantauan gunung api, di mana respon yang cepat merupakan hal krusial.

Kemudian pemilihan ukuran *patch* dan ukuran input citra perlu diperhatikan dengan cermat, karena hal ini berpengaruh pada kualitas prediksi akhir. Oleh karena itu, eksplorasi resolusi diperlukan tanpa mengorbankan efisiensi penggunaan memori. Selain itu, perlu adanya eksplorasi lebih lanjut untuk melihat hasil pengujian terhadap nilai *eye altitude* yang berbeda misalnya dalam satu lokasi yang sama untuk menguji nilai *eye altitude* dari rentang 500 mdpl – 2000 mdpl. Namun, penelitian ini mengungkapkan bahwa semakin tinggi *eye altitude*, model mengalami kesulitan dalam mendeteksi objek-objek kecil seperti contohnya kelas permukiman yang sebagaimana ditampilkan pada Gambar 4.103



Gambar 4.103 Citra Tes

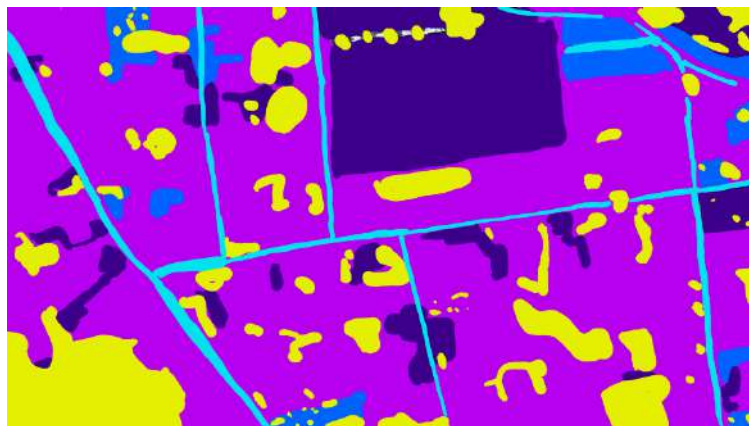
Gambar 4.104 *Ground truth*Gambar 4.105 *Predicted mask*

Pada Gambar 4.105, model gagal mendeteksi kelas permukiman. Hal ini disebabkan oleh ukuran objek permukiman yang sangat kecil, sehingga model cenderung mensegmentasi seluruh area permukiman sebagai kelas *cropland* karena objek permukiman ini dikelilingi oleh

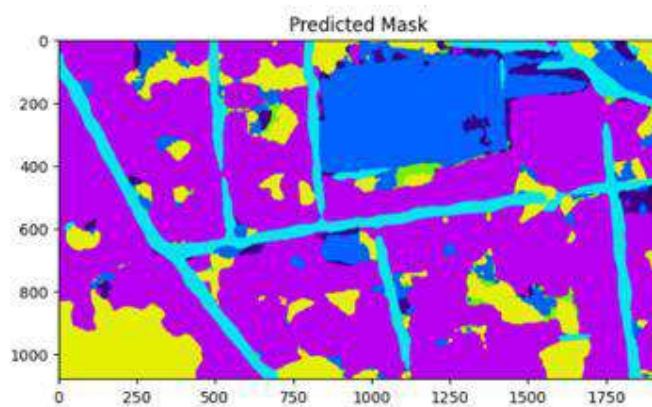
area *cropland*. Peneliti juga menampilkan citra lainya untuk dapat melihat perbandingan objek permukiman yang memiliki *eye altitude* yang rendah pada Gambar 4.106.



Gambar 4.106 Citra tes



Gambar 4.107 *Ground truth*



Gambar 4.108 *Predicted mask*

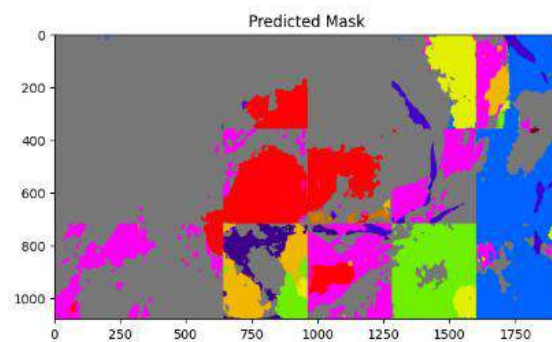
Dari Gambar 4.108. Kelas permukiman yang ditandai dengan warna ungu dapat terdeteksi dengan baik dikarenakan objek permukiman ini sangat jelas dan dapat dibedakan dari kelas lainya. Untuk mengetahui *trade off eye altitude* ini peneliti juga menyajikan citra untuk melihat perbedaan pada kelas *lava dome* pada Gambar 4.109.



Gambar 4.109 Citra tes



Gambar 4.110 *Ground truth*



Gambar 4.111 *predicted label*

Gambar 4.111 memperlihatkan objek *lava dome* yang berhasil terdeteksi oleh model. Namun, karena citra tersebut memiliki *eye altitude* yang rendah, hasil deteksi model menjadi kurang optimal. Hal ini disebabkan oleh proses segmentasi objek yang dilakukan berdasarkan *patch*, sehingga model kehilangan konteks untuk mendeteksi keseluruhan objek *lava dome*. Akibatnya, *patch-patch* tersebut ditampilkan dengan pola yang tidak rapi. Diperlukan percobaan dengan menambahkan data dengan berbagai *eye altitude* untuk mengatasi permasalahan *eye altitude* ini dan mengeksplorasi solusi lebih lanjut.

Pengembangan metode lebih lanjut diperlukan untuk menghasilkan prediksi yang lebih halus tanpa munculnya pola berupa kotak-kotak. Walaupun belum ada penelitian lebih lanjut mengenai prediksi halus pada citra satelit, namun pada *repository* Github sudah ada yang membahas kasus ini dengan judul *repository* “Make Smooth Predictions by Blending Image Patches, Such as for Image Segmentation” (Guillaume Chevalier., n.d.) . Teknik yang telah diterapkan dalam *repository* tersebut telah mampu menghasilkan segmentasi citra yang halus dengan kualitas memadai untuk tahap awal.

Salah satu tantangan lainnya yang dihadapi adalah hasil metrik rendah atau bahkan nihil pada beberapa kelas. Tentunya masalah ini dapat mempengaruhi performa model ketika diterapkan dalam sistem dunia nyata. Untuk mengatasi hal ini, diperlukan modifikasi model agar mampu mendeteksi seluruh kelas secara lebih efektif. Selain itu, penambahan data untuk menyeimbangkan *dataset* juga menjadi langkah penting agar model mampu mengenali seluruh kelas dan meningkatkan performa dari model. Berbagai penyesuaian tambahan perlu dilakukan guna memastikan model siap untuk diaplikasikan dalam skenario dunia nyata.

BAB V

KESIMPULAN DAN SARAN

5.1 Kesimpulan

Berdasarkan hasil penelitian yang telah dilaksanakan, peneliti dapat menyimpulkan beberapa hal. Poin-poin kesimpulan tersebut adalah sebagai berikut:

- a. Penelitian ini telah berhasil menerapkan model DeepLabV3+ untuk segmentasi semantik citra satelit pada area tutupan lahan dan objek sekitar gunung api. Model diuji dalam tiga skenario pelatihan, yang masing-masing menggunakan augmentasi data dengan konfigurasi yang sama. Augmentasi yang digunakan mencakup flipping, rotasi, perubahan kontras, brightness, gaussian blur, dan elastic transform. Hasil evaluasi menunjukkan bahwa setiap skenario memberikan peningkatan bertahap dalam performa model.
- b. Penggunaan berbagai parameter dan pembobotan kelas menunjukan perbaikan performa.
- c. Pada skenario pertama, model dilatih dengan konfigurasi dasar menggunakan augmentasi standar. Hasil tes menunjukkan bahwa model mencapai mIoU sebesar 0.39 dan mAP sebesar 0.34.
- d. Pada skenario kedua, augmentasi yang sama diterapkan, dan hasil tes menunjukkan peningkatan dengan nilai mIoU sebesar 0.41 dan mAP sebesar 0.36.
- e. Skenario ketiga menggunakan strategi pelatihan yang sama namun dengan tambahan pembobotan kelas dan optimasi fungsi *loss*. Hasil tes pada skenario ini menunjukkan peningkatan lebih lanjut dengan nilai mIoU sebesar 0.44, mAP sebesar 0.39.
- f. Performa model masih memerlukan peningkatan agar dapat diterapkan pada *real world case*. Model juga mengalami kesulitan dalam mendeteksi beberapa kelas yang kurang dominan di dalam *dataset*. Meskipun hasil ini belum sepenuhnya memuaskan, nilai tersebut sudah cukup memadai untuk tahap awal penelitian.

5.2 Saran

Tentunya peneliti menyadari bahwa penelitian ini masih memiliki celah yang harus ditingkatkan pada penelitian selanjutnya. Beberapa saran untuk penelitian selanjutnya diajabrkan sebagai berikut :

- a. Menerapkan arsitektur model yang lebih ringan, seperti MobileNet atau ShuffleNet, untuk meningkatkan efisiensi memori.
- b. Pelatihan dan pengujian menggunakan citra dari satelit yang sama untuk memaksimalkan hasil penelitian.
- c. Proses anotasi data dikerjakan langsung oleh ahli untuk mengurangi *noise*.
- d. Menggunakan citra hiperspektral untuk meningkatkan akurasi segmentasi dengan memanfaatkan informasi spektral tambahan.
- e. Menggunakan *overlapping patches* dan teknik *post processing* seperti *Conditional Random Field* (CRF) untuk menghasilkan prediksi yang lebih halus.
- f. Menambahkan jumlah citra dalam *dataset* untuk meningkatkan variasi data dan representasi setiap kelas.
- g. Menyeimbangkan *dataset* melalui *synthetic data generation* untuk meningkatkan representasi kelas minoritas.
- h. Menggunakan teknik *oversampling* untuk kelas minoritas.
- i. Melakukan modifikasi arsitektur untuk meningkatkan deteksi kelas yang memiliki performa rendah seperti penambahan *attention block*.
- j. Membuat *pipeline end-to-end* untuk memastikan model siap diterapkan dalam skenario dunia nyata, termasuk *preprocessing* otomatis.
- k. Mengevaluasi performa arsitektur model lain, seperti U-Net++, SegNet, atau FPN, untuk membandingkan performa segmentasi pada *dataset* yang digunakan.

DAFTAR PUSTAKA

- Akcay, O., Kinaci, A. C., Avsar, E. O., & Aydar, U. (2022a). Semantic Segmentation of High-Resolution Airborne Images with Dual-Stream DeepLabV3+. *ISPRS International Journal of Geo-Information*, 11(1). <https://doi.org/10.3390/ijgi11010023>
- Akcay, O., Kinaci, A. C., Avsar, E. O., & Aydar, U. (2022b). Semantic Segmentation of High-Resolution Airborne Images with Dual-Stream DeepLabV3+. *ISPRS International Journal of Geo-Information*, 11(1). <https://doi.org/10.3390/ijgi11010023>
- Alexander, J. C., Romito, B. T., & Çobanoğlu, M. C. (2020). The present and future role of artificial intelligence and machine learning in anesthesiology. *International Anesthesiology Clinics*, 58(4), 7–16. <https://doi.org/10.1097/AIA.0000000000000294>
- Alsheibani, S., Cheung, Y., & Messom, C. (2018). Association for Information Systems AIS Electronic Library (AISeL) Artificial Intelligence Adoption: AI-readiness at Firm-Level. <https://aisel.aisnet.org/pacis2018/37>
- Amboage, J. P. G., Wulff, E., Girone, M., & Pena, T. F. (2023). Model Performance Prediction for Hyperparameter Optimization of Deep Learning Models Using High Performance Computing and Quantum Annealing. <http://arxiv.org/abs/2311.17508>
- Badan Informasi Geospasial. (n.d.). BIG Serahkan Peta NKRI Kepada Kemenkokesra.
- Badan Meteorologi, K. dan G. (2018, August 13). BMKG: Kajian Ilmiah LIPI Soal Gempa Jawa Dipelintir. <https://www.bmkg.go.id/siaran-pers/bmkg-kajian-ilmiah-lipi-soal-gempa-jawa-dipelintir>
- Bartz-Beielstein, T., Rehbach, F., Sen, A., & Zaefferer, M. (2021). Surrogate Model Based Hyperparameter Tuning for Deep Learning with SPOT. <http://arxiv.org/abs/2105.14625>
- Chen, D., Lu, Y., & Hsu, C. Y. (2022). Measurement Invariance Investigation for Performance of Deep Learning Architectures. *IEEE Access*, 10, 78070–78087. <https://doi.org/10.1109/ACCESS.2022.3192468>
- Chen, L.-C., Papandreou, G., Schroff, F., & Adam, H. (2017). Rethinking Atrous Convolution for Semantic Image Segmentation. <http://arxiv.org/abs/1706.05587>
- Chen, L.-C., Zhu, Y., Papandreou, G., Schroff, F., & Adam, H. (2018). Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation. <http://arxiv.org/abs/1802.02611>
- Chen, M., Bai, Y., Lee, J. D., Zhao, T., Wang, H., Xiong, C., & Socher, R. (2020). Towards Understanding Hierarchical Learning: Benefits of Neural Representations. <http://arxiv.org/abs/2006.13436>

- Cheng, L., & Zhao, M. (2022). *Human Instance Segmentation and Tracking via Data Association and Single-stage Detector*. <http://arxiv.org/abs/2203.16966>
- Copernicus Global Land Cover Layers: CGLS-LC100 Collection 3*. (n.d.). Retrieved December 30, 2024, from https://developers.google.com/earth-engine/datasets/catalog/COPERNICUS_Landcover_100m_Proba-V-C3_Global
- Del Rosso, M. P., Sebastianelli, A., Spiller, D., Mathieu, P. P., & Ullo, S. L. (2021a). On-Board Volcanic Eruption Detection through CNNs and Satellite Multispectral Imagery. *Remote Sensing*, 13(17). <https://doi.org/10.3390/rs13173479>
- Del Rosso, M. P., Sebastianelli, A., Spiller, D., Mathieu, P. P., & Ullo, S. L. (2021b). On-Board Volcanic Eruption Detection through CNNs and Satellite Multispectral Imagery. *Remote Sensing*, 13(17). <https://doi.org/10.3390/rs13173479>
- Deshmukh, O. M. (2021). Computer Vision. *International Journal for Research in Applied Science and Engineering Technology*, 9(VII), 1237–1239. <https://doi.org/10.22214/ijraset.2021.35926>
- Devarakonda, A., Naumov, M., & Garland, M. (2017). *AdaBatch: Adaptive Batch Sizes for Training Deep Neural Networks*. <http://arxiv.org/abs/1712.02029>
- Dokumentasi PyTorch. (n.d.-a). *Source code for torchvision.models.segmentation.deeplabv3*. Retrieved January 1, 2025, from https://pytorch.org/vision/stable/_modules/torchvision/models/segmentation/deeplabv3.html
- Dokumentasi PyTorch. (n.d.-b). *StepLR*. Retrieved January 1, 2025, from https://pytorch.org/docs/stable/generated/torch.optim.lr_scheduler.StepLR.html
- Du, S., Du, S., Liu, B., & Zhang, X. (2021). Incorporating DeepLabv3+ and object-based image analysis for semantic segmentation of very high resolution remote sensing images. *International Journal of Digital Earth*, 14(3), 357–378. <https://doi.org/10.1080/17538947.2020.1831087>
- Earth Resources Observation and Science (EROS) Center. (n.d.). *Satellite Imagery*. United States Geological Survey. Retrieved December 30, 2024, from <https://www.usgs.gov/centers/eros/satellite-imagery>
- Ekong, F., Yu, Y., Patamia, R. A., Feng, X., Tang, Q., Mazumder, P., & Cai, J. (2022). Bayesian Depth-Wise Convolutional Neural Network Design for Brain Tumor MRI Classification. *Diagnostics*, 12(7). <https://doi.org/10.3390/diagnostics12071657>

- Eropean Space Agency. (n.d.). *Sentinel 2*. Retrieved January 2, 2025, from https://www.esa.int/Applications/Observing_the_Earth/Copernicus/Sentinel-2
- Fu, J., Yi, X., Wang, G., Mo, L., Wu, P., & Kapula, K. E. (2022). Research on Ground Object Classification Method of High Resolution Remote-Sensing Images Based on Improved DeeplabV3+. *Sensors*, 22(19). <https://doi.org/10.3390/s22197477>
- Galdran, A., Carneiro, G., & Ballester, M. Á. G. (2022). *On the Optimal Combination of Cross-Entropy and Soft Dice Losses for Lesion Segmentation with Out-of-Distribution Robustness*. <http://arxiv.org/abs/2209.06078>
- Google Earth Pro. (n.d.). Retrieved December 30, 2024, from <https://earth.google.com/intl/earth/download/ge/agree.html>
- Google Image. (n.d.). Retrieved February 7, 2025, from https://www.google.com/search?sca_esv=0d5cec6ac40c2f16&q=satellite+image+volcano+eruption&udm=2&fbs=ABzOT_DbUoXV9d3f6LsLPD3ftI-OuFzxdZzYMMXqlS54eCDeR1aioGyySPibBdKbMy29L8PE-62Oxa_5NIiKi7qjeB_RDt-saUN1AYorRPh4VEPCGKf3jMYGbC1k_GhJ-G5-PXSMU_ODX0xmJvnnwDDRgjHNxcuVU2kcz4Vuddbdlg-JiuiIhXAfkk6FNryPrfGCg894nRSHQoEYgLqp2k58_TtT_CjPA&sa=X&ved=2ahUKEwi8gL2Z_bCLAxVV1DgGHZ2UGAoQtKgLegQIFRAB&biw=1422&bih=650&dpr=1.35
- Grover, P., Kar, A. K., & Dwivedi, Y. K. (2022). Understanding artificial intelligence adoption in operations management: insights from the review of academic literature and social media discussions. *Annals of Operations Research*, 308(1–2), 177–213. <https://doi.org/10.1007/s10479-020-03683-9>
- Guillaume Chevalier. (n.d.). *Make smooth predictions by blending image patches, such as for image segmentation*. Retrieved January 2, 2025, from <https://github.com/Vooban/Smoothly-Blend-Image-Patches?tab=readme-ov-file>
- Guo, Y., Liu, Y., Georgiou, T., & Lew, M. S. (2018). A review of semantic segmentation using deep neural networks. *International Journal of Multimedia Information Retrieval*, 7(2), 87–93. <https://doi.org/10.1007/s13735-017-0141-z>
- Haenlein, M., & Kaplan, A. (2019). A brief history of artificial intelligence: On the past, present, and future of artificial intelligence. *California Management Review*, 61(4), 5–14. <https://doi.org/10.1177/0008125619864925>

- Hamdan, B. A. (2022). Neural Network Principles and its Application. *Webology*, 19(1), 3955–3970. <https://doi.org/10.14704/web/v19i1/web19261>
- He, K., Zhang, X., Ren, S., & Sun, J. (2015a). *Deep Residual Learning for Image Recognition*. <http://arxiv.org/abs/1512.03385>
- He, K., Zhang, X., Ren, S., & Sun, J. (2015b). *Deep Residual Learning for Image Recognition*. <http://arxiv.org/abs/1512.03385>
- Heckel, R., & Yilmaz, F. F. (2020). *Early Stopping in Deep Networks: Double Descent and How to Eliminate it*. <http://arxiv.org/abs/2007.10099>
- Helmhold, D. P., & Long, P. M. (2016). *Surprising properties of dropout in deep networks*. <http://arxiv.org/abs/1602.04484>
- Hu, X., Chu, L., Pei, J., Liu, W., & Bian, J. (2021). *Model Complexity of Deep Learning: A Survey*. <http://arxiv.org/abs/2103.05127>
- Huang, J., Rathod, V., Sun, C., Zhu, M., Korattikara, A., Fathi, A., Fischer, I., Wojna, Z., Song, Y., Guadarrama, S., & Murphy, K. (2016). *Speed/accuracy trade-offs for modern convolutional object detectors*. <http://arxiv.org/abs/1611.10012>
- Indhurani, A., Manimegalai, A., Arunpandiyam, I., Ramachandran, M., & Sathiyaraj, C. (2022). Exploring Recent Trends in Computer Vision. *Electrical and Automation Engineering*, 1(1), 33–39. <https://doi.org/10.46632/eae/1/1/6>
- Kadra, A., Janowski, M., Wistuba, M., & Grabocka, J. (2023). *Scaling Laws for Hyperparameter Optimization*. <http://arxiv.org/abs/2302.00441>
- Khryashchev, V., Kotov, N., Larionov, R., & Priorov, A. (2022). Semantic Segmentation of Satellite Sar-C Images Using Deep Learning Based Algorithms. *2022 Systems of Signal Synchronization, Generating and Processing in Telecommunications (SYNCHROINFO)*, 1–5. <https://doi.org/10.1109/SYNCHROINFO55067.2022.9840934>
- Kiran, M., & Ozyildirim, M. (2022). *Hyperparameter Tuning for Deep Reinforcement Learning Applications*. <http://arxiv.org/abs/2201.11182>
- Kuhn, M., & Johnson, K. (2013). *Applied Predictive Modeling*. Springer New York. <https://doi.org/10.1007/978-1-4614-6849-3>
- Lengerich, B., Xing, E. P., & Caruana, R. (2020). On dropout, overfitting, and interaction effects in deep neural networks. *ArXiv Preprint ArXiv:2007.00823*, 2.
- Li, C., & Huang, H. (2023). Emergence of hierarchical modes from deep learning. *Physical Review Research*, 5(2). <https://doi.org/10.1103/PhysRevResearch.5.L022011>

- Li, Q., Yang, Y., Guo, Y., Li, W., Liu, Y., Liu, H., & Kang, Y. (2021). Performance Evaluation of Deep Learning Classification Network for Image Features. *IEEE Access*, 9, 9318–9333. <https://doi.org/10.1109/ACCESS.2020.3048956>
- Li, W., & Zhao, S. (2022a). Semantic segmentation of buildings in high-resolution remote sensing images based on DeepLabV3+ algorithm. *Journal of Physics: Conference Series*, 2400(1). <https://doi.org/10.1088/1742-6596/2400/1/012037>
- Li, W., & Zhao, S. (2022b). Semantic segmentation of buildings in high-resolution remote sensing images based on DeepLabV3+ algorithm. *Journal of Physics: Conference Series*, 2400(1). <https://doi.org/10.1088/1742-6596/2400/1/012037>
- Li, Z., Liu, F., Yang, W., Peng, S., & Zhou, J. (2022). A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects. *IEEE Transactions on Neural Networks and Learning Systems*, 33(12), 6999–7019. <https://doi.org/10.1109/TNNLS.2021.3084827>
- Liu, X., Xu, B., Gu, W., Yin, Y., & Wang, H. (2022). Plant leaf veins coupling feature representation and measurement method based on DeepLabV3+. *Frontiers in Plant Science*, 13. <https://doi.org/10.3389/fpls.2022.1043884>
- Mertikas, S. P., Partsinevelos, P., Mavrocordatos, C., & Maximenko, N. A. (2020). Environmental applications of remote sensing. In *Pollution Assessment for Sustainable Practices in Applied Sciences and Engineering* (pp. 107–163). Elsevier. <https://doi.org/10.1016/B978-0-12-809582-9.00003-7>
- Mishra, P. (2019). Introduction to PyTorch, Tensors, and Tensor Operations. In *PyTorch Recipes* (pp. 1–27). Apress. https://doi.org/10.1007/978-1-4842-4258-2_1
- Ovi, T. B., Mosharrof, S., Bashree, N., Islam, M. S., & Islam, M. N. (2023). *DeepTriNet: A Tri-Level Attention Based DeepLabv3+ Architecture for Semantic Segmentation of Satellite Images*. <http://arxiv.org/abs/2310.06848>
- Pandey, Mrs. A. (2023). Computer Vision. *International Journal for Research in Applied Science and Engineering Technology*, 11(7), 510–514. <https://doi.org/10.22214/ijraset.2023.54701>
- Putri, A. (2020). *Sistem Pakar*.
- Reader, A. J., Corda, G., Mehranian, A., Costa-Luis, C. da, Ellis, S., & Schnabel, J. A. (2020). Deep Learning for PET Image Reconstruction. *IEEE Transactions on Radiation and Plasma Medical Sciences*, 5(1), 1–25. <https://doi.org/10.1109/trpms.2020.3014786>

- Rezatofighi, H., Tsoi, N., Gwak, J., Sadeghian, A., Reid, I., & Savarese, S. (2019). *Generalized Intersection over Union: A Metric and A Loss for Bounding Box Regression*. <http://arxiv.org/abs/1902.09630>
- Rukundo, O., & Cao, H. (2012). Nearest Neighbor Value Interpolation. In *IJACSA) International Journal of Advanced Computer Science and Applications* (Vol. 3, Issue 4). www.ijacsa.thesai.org
- Saleem, M. A., Senan, N., Wahid, F., Aamir, M., Samad, A., & Khan, M. (2022). Comparative Analysis of Recent Architecture of Convolutional Neural Network. *Mathematical Problems in Engineering*, 2022. <https://doi.org/10.1155/2022/7313612>
- Sane, P., & Agrawal, R. (2017). *Pixel Normalization from Numeric Data as Input to Neural Networks*. <http://arxiv.org/abs/1705.01809>
- Sarker, I. H. (2021). Machine Learning: Algorithms, Real-World Applications and Research Directions. In *SN Computer Science* (Vol. 2, Issue 3). Springer. <https://doi.org/10.1007/s42979-021-00592-x>
- Segmentation Models. (n.d.). *Welcome to Segmentation Models's documentation!* Retrieved January 22, 2025, from <https://segmentation-models-pytorch.readthedocs.io/en/latest/index.html>
- Sertel, E., Ekim, B., Ettehadi Osgouei, P., & Kabadayi, M. E. (2022). Land Use and Land Cover Mapping Using Deep Learning Based Segmentation Approaches and VHR Worldview-3 Images. *Remote Sensing*, 14(18). <https://doi.org/10.3390/rs14184558>
- Shreya M. Shelke, Indrayani S. Pathak, Aniket P. Sangai, Dipali V. Lunge, Kalyani A. Shahale, & Harsha R. Vyawahare. (2023). A Review Paper on Computer Vision. *International Journal of Advanced Research in Science, Communication and Technology*, 673–677. <https://doi.org/10.48175/ijarsct-8901>
- Singh, A., Kumar, M., & Saxena, A. (2023). Analysis of Computer Vision for Graphics and Animation. *2023 9th International Conference on Advanced Computing and Communication Systems (ICACCS)*, 804–807. <https://doi.org/10.1109/ICACCS57279.2023.10112718>
- Sunandini, G., Sivanpillai, R., Sowmya, V., & Sajith Variyar, V. V. (2023). Significance of Atrous Spatial Pyramid Pooling (ASPP) in Deeplabv3+ for Water Body Segmentation. *Proceedings of the 10th International Conference on Signal Processing and Integrated Networks, SPIN 2023*, 744–749. <https://doi.org/10.1109/SPIN57001.2023.10116882>

- Thapa, A., Horanont, T., Neupane, B., & Aryal, J. (2023). Deep Learning for Remote Sensing Image Scene Classification: A Review and Meta-Analysis. In *Remote Sensing* (Vol. 15, Issue 19). Multidisciplinary Digital Publishing Institute (MDPI). <https://doi.org/10.3390/rs15194804>
- Wang, Z., Berman, M., Rannen-Triki, A., Torr, P. H. S., Tuia, D., Tuytelaars, T., Van Gool, L., Yu, J., & Blaschko, M. B. (2023). *Revisiting Evaluation Metrics for Semantic Segmentation: Optimization and Evaluation of Fine-grained Intersection over Union*. <http://arxiv.org/abs/2310.19252>
- Wood, L., & Chollet, F. (2022). *Efficient Graph-Friendly COCO Metric Computation for Train-Time Model Evaluation*. <http://arxiv.org/abs/2207.12120>
- Xiao, X., Yan, M., Basodi, S., Ji, C., & Pan, Y. (2020). *Efficient Hyperparameter Optimization in Deep Learning Using a Variable Length Genetic Algorithm*. <http://arxiv.org/abs/2006.12703>
- Xu, S., Wang, J., Shou, W., Ngo, T., Sadick, A. M., & Wang, X. (2021). Computer Vision Techniques in Construction: A Critical Review. *Archives of Computational Methods in Engineering*, 28(5), 3383–3397. <https://doi.org/10.1007/s11831-020-09504-3>
- Yang, M. Y., Kumaar, S., Lyu, Y., & Nex, F. (2021). Real-time Semantic Segmentation with Context Aggregation Network. *ISPRS Journal of Photogrammetry and Remote Sensing*, 178, 124–134. <https://doi.org/10.1016/j.isprsjprs.2021.06.006>
- Yang, S., Xiao, W., Zhang, M., Guo, S., Zhao, J., & Shen, F. (n.d.). *Image Data Augmentation for Deep Learning: A Survey*.
- Yu, L., Zeng, Z., Liu, A., Xie, X., Wang, H., Xu, F., & Hong, W. (2022). A Lightweight Complex-Valued DeepLabv3+ for Semantic Segmentation of PolSAR Image. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 15, 930–943. <https://doi.org/10.1109/JSTARS.2021.3140101>
- Zaim Awang Pon, M., Prakash K, K. K., Tun Razak, J., & Lampur, K. (2021). Hyperparameter Tuning of Deep learning Models in Keras. In *Sparklinglight Transactions on Artificial Intelligence and Quantum Computing (STAIQC)* (Vol. 1, Issue 1). <https://sparklinglightpublisher.com/>
- Zheng, T. (2020). Semantic Segmentation Based on Deeplabv3+ with Fractional-Order Image Enhancement Convolution Kernel. *ACM International Conference Proceeding Series*, 34–39. <https://doi.org/10.1145/3425577.3425584>

LAMPIRAN

1. File Pelatihan Model

```
1. # -*- coding: utf-8 -*-
2. """FinalTrainingTutupanLahan25Nov.ipynb
3.
4. Automatically generated by Colab.
5.
6. Original file is located at
7.     https://colab.research.google.com/drive/1-kzT4JwjOdlyvWOkYgt5vO4Ql490a0k0
8.
9. ***Connect Google Drive**
10.
11. ***Install Library**
12.
13. ***Import Libraries**
14. """
15.
16. !pip install numpy==1.21.4 matplotlib==3.4.3 seaborn==0.11.2 scikit-
    learn==0.24.2 torch==1.10.0 torchvision==0.11.1 torchsummary==1.5.1 torchviz==0.0.2 opencv-
```

```
python-headless==4.5.5.64 pillow==8.4.0 segmentation-models-  
pytorch==0.2.1 albumentations==1.1.0 patchify==0.2.3 tqdm ipywidgets  
17.  
18. !pip install tensorflow==2.4.0  
19.  
20.  
21.  
22. from google.colab import drive  
23. drive.mount('/content/drive')  
24.  
25. import numpy as np  
26. import torch  
27. import torchvision  
28. import segmentation_models_pytorch as smp  
29. import albumentations as A  
30. import tensorflow as tf  
31.  
32. print("Numpy version:", np.__version__)  
33. print("PyTorch version:", torch.__version__)  
34. print("Torchvision version:", torchvision.__version__)  
35. print("Segmentation Models PyTorch version:", smp.__version__)  
36. print("Albumentations version:", A.__version__)
```

```
37.print("TensorFlow version:", tf.__version__)
38.
39.import torch
40.import segmentation_models_pytorch as smp
41.import torch.nn as nn
42.import torch.optim as optim
43.import numpy as np
44.import matplotlib.pyplot as plt
45.from sklearn.utils.class_weight import compute_class_weight
46.from collections import Counter
47.import os
48.from torch.utils.data import Dataset, DataLoader, ConcatDataset
49.import torchvision.transforms as transforms
50.import cv2
51.from PIL import Image
52.from sklearn.preprocessing import MinMaxScaler
53.from sklearn.model_selection import train_test_split
54.from tqdm import tqdm
55.from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay, jaccard_score, precision_score, recall_score, f1_score
56.import seaborn as sns
57.import albumentations as A
```

```
58.from albumentations.pytorch import ToTensorV2
59.from torch.optim.lr_scheduler import ReduceLROnPlateau
60.from patchify import patchify, unpatchify
61.from torchsummary import summary
62.from torchviz import make_dot
63.import matplotlib.ticker as mticker
64.from IPython.display import display, HTML
65.import random
66.
67."""#**Set Device**"""
68.
69.# Set device
70.device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
71.print(f'Using device: {device}')
72.torch.backends.cudnn.deterministic = True
73.torch.backends.cudnn.benchmark = True
74.torch.backends.cudnn.enabled = True
75.
76."""#**Open Dataset**"""
77.
78.import os
79.import random
```

```
80.from PIL import Image
81.import matplotlib.pyplot as plt
82.
83.# Definisikan path ke dataset
84.base_dataset_path = 'extracted_dataset/AllDataset'
85.tile_paths = ['Tile1', 'Tile3']
86.
87.tile1 = {
88.    'images_path': os.path.join(base_dataset_path, 'Tile1', 'images'),
89.    'masks_path': os.path.join(base_dataset_path, 'Tile1', 'masks')
90.}
91.tile3 = {
92.    'images_path': os.path.join(base_dataset_path, 'Tile3', 'images'),
93.    'masks_path': os.path.join(base_dataset_path, 'Tile3', 'masks')
94.}
95.tiles = [tile1, tile3]
96.
97.# Fungsi untuk membaca gambar dan mask dari tile
98.def get_random_image_and_mask_from_tile(tile_paths):
99.    # Daftar file gambar di tile
100.        image_files = os.listdir(tile_paths['images_path'])
```

```
101.     image_files = [f for f in image_files if f.endswith('.jpg')] # Filter hanya file .j
pg
102.
103.     # Pilih file secara acak dari tile yang dipilih
104.     selected_image_file = random.choice(image_files)
105.
106.     # Path gambar dan mask
107.     image_path = os.path.join(tile_paths['images_path'], selected_image_file)
108.
109.     # Nama mask yang sesuai dengan gambar (ubah ekstensi menjadi .png)
110.     selected_mask_file = selected_image_file.replace('.jpg', '.png')
111.     mask_path = os.path.join(tile_paths['masks_path'], selected_mask_file)
112.
113.     # Baca gambar dan mask
114.     image = Image.open(image_path)
115.     mask = Image.open(mask_path)
116.
117.     return image, mask
118.
119. # Fungsi untuk menampilkan gambar dan mask secara acak
120. def show_images_and_masks(tiles, n=5):
```

```
121.     fig, axes = plt.subplots(n, 2, figsize=(15, 5 * n)) # Adjusting figure size based o
    n the number of sampels
122.     for i in range(n):
123.         # Pilih tile secara acak
124.         selected_tile = random.choice(tiles)
125.         image, mask = get_random_image_and_mask_from_tile(selected_tile)
126.
127.         axes[i, 0].imshow(image)
128.         axes[i, 0].set_title(f'Image {i+1}')
129.         axes[i, 0].axis('off')
130.
131.         axes[i, 1].imshow(mask, cmap='gray')
132.         axes[i, 1].set_title(f'Mask {i+1}')
133.         axes[i, 1].axis('off')
134.
135.     plt.tight_layout()
136.     plt.show()
137.
138. # Tampilkan 5 gambar dan mask secara acak
139. show_images_and_masks(tiles, n=5)
140.
141. """**Custom Class Dataset**"""
```

```
142.
143. import numpy as np
144. from sklearn.preprocessing import MinMaxScaler
145. import cv2
146. from torch.utils.data import Dataset
147. from patchify import patchify
148. from PIL import Image
149. import torch
150.
151. class SegmentationDataset(Dataset):
152.     def __init__(self, image_paths, mask_paths, patch_size=256, transform=None):
153.         self.image_paths = image_paths
154.         self.mask_paths = mask_paths
155.         self.patch_size = patch_size
156.         self.transform = transform
157.         self.scaler = MinMaxScaler()
158.         self.image_patches = []
159.         self.mask_patches = []
160.         self.labels = [] # Label utama untuk setiap patch, digunakan untuk stratifikasi

161.         self.create_patches()
162.
```

```
163.     def create_patches(self):
164.         for img_path, mask_path in zip(self.image_paths, self.mask_paths):
165.             image = cv2.imread(img_path)
166.             image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
167.             mask = cv2.imread(mask_path, 1)
168.             mask = cv2.cvtColor(mask, cv2.COLOR_BGR2RGB)
169.
170.             # Crop images and masks to the nearest size divisible by patch_size
171.             SIZE_X = (image.shape[1] // self.patch_size) * self.patch_size
172.             SIZE_Y = (image.shape[0] // self.patch_size) * self.patch_size
173.             image = Image.fromarray(image).crop((0, 0, SIZE_X, SIZE_Y))
174.             mask = Image.fromarray(mask).crop((0, 0, SIZE_X, SIZE_Y))
175.             image = np.array(image)
176.             mask = np.array(mask)
177.
178.             # Patchify images and masks
179.             patches_img = patchify(image, (self.patch_size, self.patch_size, 3), step=self.patch_size)
180.             patches_mask = patchify(mask, (self.patch_size, self.patch_size, 3), step=self.patch_size)
181.
182.             for i in range(patches_img.shape[0]):
```

```

183.         for j in range(patches_img.shape[1]):
184.             patch_img = patches_img[i, j, :, :, :][0]
185.             patch_mask = patches_mask[i, j, :, :, :][0]
186.
187.             # Scale the image patch
188.             patch_img = self.scaler.fit_transform(patch_img.reshape(-
190.             1, patch_img.shape[-1])).reshape(patch_img.shape)
189.
190.             # Convert RGB mask to label mask
191.             patch_label_mask = self.rgb_to_2D_label(patch_mask)
192.
193.             # Ambil label utama untuk stratifikasi
194.             unique_labels = np.unique(patch_label_mask)
195.             primary_label = unique_labels[0] if len(unique_labels) == 1 else uni
196.             que_labels[1]
197.
198.             # Append data to lists
199.             self.image_patches.append(patch_img)
200.             self.mask_patches.append(torch.tensor(patch_label_mask, dtype=torch.
long))
200.             self.labels.append(primary_label) # Menyimpan label untuk stratifik
asi

```

```
201.
202.     def __len__(self):
203.         return len(self.image_patches)
204.
205.     def __getitem__(self, idx):
206.         return self.image_patches[idx], self.mask_patches[idx]
207.
208.     def rgb_to_2D_label(self, label):
209.         # Updated color dictionary
210.         label_colors = {
211.             'Cropland': [112, 240, 0],
212.             'Permukiman': [184, 0, 240],
213.             'Air': [240, 184, 0],
214.             'Area Terbuka': [60, 0, 138],
215.             'Jalan': [0, 228, 240],
216.             'Vegetasi': [228, 240, 0],
217.             'Herbaceous Vegetation': [0, 98, 255],
218.             'Lava Dome': [255, 0, 4],
219.             'Jalur Lahar': [65, 0, 204],
220.             'Awan Panas': [204, 122, 0],
221.             'Endapan Material Vulkanik Baru': [255, 0, 247],
222.             'Endapan Material Vulkanik Lama': [120, 120, 120],
```

```
223.         'Awan': [255, 173, 183],
224.         'Lava Flow': [122, 0, 14]
225.     }
226.
227.     label_seg = np.zeros(label.shape[:2], dtype=np.uint8)
228.     for label_name, rgb in label_colors.items():
229.         label_seg[np.all(label == rgb, axis=-
1)] = list(label_colors.keys()).index(label_name)
230.     return label_seg
231.
232. """#**Create Dataset**"""
233.
234. import os
235. import numpy as np
236. from iterstrat.ml_stratifiers import MultilabelStratifiedShuffleSplit
237. from sklearn.preprocessing import OneHotEncoder
238. from torch.utils.data import DataLoader
239.
240. # Adjust the paths to your dataset location
241. base_dataset_path = 'extracted_dataset/AllDataset'
242. tile_paths = ['Tile1', 'Tile3']
243.
```

```
244. # Collect all image and mask paths
245. image_paths = []
246. mask_paths = []
247.
248. for tile in tile_paths:
249.     dataset_path = os.path.join(base_dataset_path, tile)
250.     for path, subdirs, files in os.walk(dataset_path):
251.         dirname = path.split(os.path.sep)[-1]
252.         if dirname == 'images':
253.             for image_name in files:
254.                 if image_name.endswith(".jpg"):
255.                     image_path = os.path.join(path, image_name)
256.                     mask_path = os.path.join(path.replace('images', 'masks'), image_name
257.                                                .replace('.jpg', '.png'))
258.                     if os.path.exists(mask_path):
259.                         image_paths.append(image_path)
260.                         mask_paths.append(mask_path)
261. # Create the dataset
262. dataset = SegmentationDataset(image_paths=image_paths, mask_paths=mask_paths, patch_size
263.                               =320)
```

```
264. # Convert labels to one-hot encoding for MultilabelStratifiedShuffleSplit
265. labels = np.array(dataset.labels).reshape(-1, 1)
266. one_hot_encoder = OneHotEncoder(sparse=False, categories='auto')
267. labels_one_hot = one_hot_encoder.fit_transform(labels)
268.
269. # Use MultilabelStratifiedShuffleSplit for stratified split
270. stratifier = MultilabelStratifiedShuffleSplit(test_size=0.2, random_state=61)
271. train_idx, val_idx = next(stratifier.split(dataset.image_patches, labels_one_hot))
272.
273. # Split the dataset using the stratified indices
274. train_images = np.array([dataset.image_patches[i].numpy() if isinstance(dataset.image_patches[i], torch.Tensor) else dataset.image_patches[i] for i in train_idx])
275. val_images = np.array([dataset.image_patches[i].numpy() if isinstance(dataset.image_patches[i], torch.Tensor) else dataset.image_patches[i] for i in val_idx])
276. train_masks = np.array([dataset.mask_patches[i].numpy() if isinstance(dataset.mask_patches[i], torch.Tensor) else dataset.mask_patches[i] for i in train_idx])
277. val_masks = np.array([dataset.mask_patches[i].numpy() if isinstance(dataset.mask_patches[i], torch.Tensor) else dataset.mask_patches[i] for i in val_idx])
278.
279. # Print dataset statistics
280. print(f"Number of images: {len(image_paths)}")
281. print(f"Number of masks: {len(mask_paths)}")
```

```
282. print(f"Number of image patches: {len(dataset.image_patches)}")
283. print(f"Number of mask patches: {len(dataset.mask_patches)}")
284. print(f"Number of training patches: {len(train_images)}")
285. print(f"Number of validation patches: {len(val_images)}")
286.
287. import numpy as np
288. import pandas as pd
289. from collections import Counter
290. import matplotlib.pyplot as plt
291.
292. # Function to count class distribution in a set of masks
293. def count_class_distribution(masks):
294.     class_counts = Counter()
295.     for mask in masks:
296.         unique, counts = np.unique(mask, return_counts=True)
297.         class_counts.update(dict(zip(unique, counts)))
298.     return class_counts
299.
300. # Count class distribution in train and validation masks
301. train_class_counts = count_class_distribution(train_masks)
302. val_class_counts = count_class_distribution(val_masks)
303.
```

```
304. # Combine into a DataFrame for easy comparison
305. classes = sorted(set(train_class_counts.keys()).union(set(val_class_counts.keys())))
306. train_counts = [train_class_counts.get(cls, 0) for cls in classes]
307. val_counts = [val_class_counts.get(cls, 0) for cls in classes]
308.
309. df = pd.DataFrame({
310.     'Class': classes,
311.     'Train Count': train_counts,
312.     'Validation Count': val_counts
313. })
314.
315. # Display class distribution table
316. print("Class Distribution in Train and Validation Masks")
317. print(df)
318.
319. # Plot class distribution as bar chart
320. plt.figure(figsize=(20, 15))
321. bar_width = 0.35
322. index = np.arange(len(classes))
323.
324. plt.bar(index, train_counts, bar_width, label='Train')
325. plt.bar(index + bar_width, val_counts, bar_width, label='Validation')
```

```
326. plt.xlabel('Class')
327. plt.ylabel('Pixel Count')
328. plt.title('Class Distribution in Train and Validation Masks')
329. plt.xticks(index + bar_width / 2, classes)
330. plt.legend()
331. plt.show()
332.
333. """#**Plot Class Distribution**"""
334.
335. import numpy as np
336. from collections import Counter
337. import pandas as pd
338. from tabulate import tabulate
339. import matplotlib.pyplot as plt
340.
341. # Class labels and colors
342. class_labels = {
343.     0: 'Cropland',
344.     1: 'Permukiman',
345.     2: 'Air',
346.     3: 'Area Terbuka',
347.     4: 'Jalan',
```

```
348.     5: 'Vegetasi',
349.     6: 'Herbaceous Vegetation',
350.     7: 'Lava Dome',
351.     8: 'Jalur Lahar',
352.     9: 'Awan Panas',
353.    10: 'Endapan Material Vulkanik Baru',
354.    11: 'Endapan Material Vulkanik Lama',
355.    12: 'Awan',
356.    13: 'Lava Flow'
357. }
358.
359. class_colors = {
360.     0: '#70F000', # Cropland
361.     1: '#B800F0', # Permukiman
362.     2: '#F0B800', # Air
363.     3: '#3C008A', # Area Terbuka
364.     4: '#00E4F0', # Jalan
365.     5: '#E4F000', # Vegetasi
366.     6: '#0062FF', # Herbaceous Vegetation
367.     7: '#FF0004', # Lava Dome
368.     8: '#4100CC', # Jalur Lahar
369.     9: '#CC7A00', # Awan Panas
```

```
370.     10: '#FF00F7', # Endapan Material Vulkanik Baru
371.     11: '#787878', # Endapan Material Vulkanik Lama
372.     12: '#FFADB7', # Awan
373.     13: '#7A000E'  # Lava Flow
374. }
375.
376. # Function to count classes in the dataset
377. def count_classes_in_dataset(dataset):
378.     class_counts = Counter()
379.     total_pixels = 0
380.     for _, mask in dataset: # Assuming dataset is a list of tuples (image, mask)
381.         # Convert mask to numpy array if it's not already
382.         if not isinstance(mask, np.ndarray):
383.             mask = mask.numpy()
384.             unique, counts = np.unique(mask, return_counts=True)
385.             class_counts.update(dict(zip(unique, counts)))
386.             total_pixels += mask.size
387.     return class_counts, total_pixels
388.
389.
390. # Function to display class distribution as a table
391. def display_class_distribution_table(class_counts, total_pixels):
```

```
392.     classes = list(class_counts.keys())
393.     counts = [count / total_pixels * 100 for count in class_counts.values()]
394.     labels = [class_labels[cls] for cls in classes]
395.
396.     # Create a DataFrame for better display of the table
397.     df = pd.DataFrame({
398.         'Class': labels,
399.         'Pixel Count': [class_counts[cls] for cls in classes],
400.         'Percentage (%)': [f'{count:.2f}' for count in counts]
401.     })
402.
403.     # Display the table using tabulate for better formatting
404.     print(tabulate(df, headers='keys', tablefmt='pretty', showindex=False))
405.
406. # Function to display class distribution as a pie chart
407. def display_class_distribution_pie_chart(class_counts, total_pixels):
408.     labels = [class_labels[cls] for cls in class_counts.keys()]
409.     sizes = [count / total_pixels * 100 for count in class_counts.values()]
410.     colors = [class_colors[cls] for cls in class_counts.keys()]
411.
412.     # Plot the pie chart
413.     plt.figure(figsize=(16, 16))
```

```
414.     plt.pie(sizes, labels=labels, autopct='%1.1f%%', startangle=90, colors=colors)
415.     plt.axis('equal') # Equal aspect ratio ensures that pie is drawn as a circle.
416.     plt.title('Class Distribution in Dataset')
417.     plt.show()
418.
419. # Assuming 'dataset' is available and contains (image, mask) pairs
420. # Count the classes in the dataset
421. class_counts, total_pixels = count_classes_in_dataset(dataset)
422.
423.
424. # Display the pie chart
425. display_class_distribution_pie_chart(class_counts, total_pixels)
426.
427. """##**Create Data Loader**"""
428.
429. import torch
430. from torch.utils.data import Dataset, DataLoader, ConcatDataset
431. import torchvision.transforms as transforms
432. import albumentations as A
433. from albumentations.pytorch import ToTensorV2
434. import numpy as np
435. import matplotlib.pyplot as plt
```

```
436.
437.
438. import numpy as np
439. import torch
440. from torch.utils.data import Dataset
441.
442. class PatchDataset(Dataset):
443.     def __init__(self, images, masks, patch_size=256, transform=None, augmentations=None
    , apply_augmentations=False):
444.         """
445.         Initializes the PatchDataset.
446.
447.         Args:
448.             images (list or np.array): List or array of images.
449.             masks (list or np.array): List or array of corresponding masks.
450.             patch_size (int): Size of patches to be extracted from images.
451.             transform (albumentations.Compose): Transformation to apply after augmentati
    ons (e.g., ToTensorV2).
452.             augmentations (albumentations.Compose): Augmentations to apply to images and
    masks.
453.             apply_augmentations (bool): Whether to apply augmentations.
454.         """
```

```
455.         self.images = images
456.         self.masks = masks
457.         self.patch_size = patch_size
458.         self.transform = transform
459.         self.augmentations = augmentations
460.         self.apply_augmentations = apply_augmentations
461.
462.     def __len__(self):
463.         return len(self.images)
464.
465.     def __getitem__(self, idx):
466.         # Retrieve the image and mask and ensure they are in float32 for compatibility w
ith albumentations
467.         image = np.array(self.images[idx]).astype(np.float32) # Convert to float32 for
compatibility
468.         mask = np.array(self.masks[idx]).astype(np.float32) # Convert to float32 for
compatibility
469.
470.         # Ensure the patch size is not larger than the image size
471.         if image.shape[0] < self.patch_size or image.shape[1] < self.patch_size:
472.             raise ValueError("Image size must be larger than patch size")
473.
```

```
474.         # Apply augmentations if specified
475.         if self.apply_augmentations and self.augmentations:
476.             # Albumentations expects images and masks in float32
477.             augmented = self.augmentations(image=image, mask=mask)
478.             image = augmented['image']
479.             mask = augmented['mask']
480.
481.         # Apply transformations (e.g., ToTensorV2) after augmentations
482.         if self.transform:
483.             transformed = self.transform(image=image, mask=mask)
484.             image = transformed['image'] # This will be a PyTorch tensor if ToTensorV2
            is used
485.             mask = transformed['mask'].long() # Convert mask to long for compatibility
            with PyTorch
486.
487.         return image, mask
488.
489. # Image Augmentation
490. train_augmentations = A.Compose([
491.     A.VerticalFlip(p=0.5),
492.     A.HorizontalFlip(p=0.5),
493.     A.Rotate(limit=60, p=0.5),
```

```
494.     A.Blur(blur_limit=7, p=0.5),
495.     A.RandomRotate90(p=0.75),
496.     A.RandomBrightnessContrast(brightness_limit=0.3, contrast_limit=0.3, p=0.7),
497. ])
498.
499. # Define transformations for training and validation
500. train_transform = A.Compose([
501.     ToTensorV2() # Convert image and mask to tensor
502. ])
503.
504. val_transform = A.Compose([
505.     ToTensorV2() # Only convert to tensor, no augmentation
506. ])
507.
508. # Create original and augmented datasets
509. train_dataset_original = PatchDataset(train_images, train_masks, patch_size=320, transform=train_transform, apply_augmentations=False)
510. train_dataset_augmented = PatchDataset(train_images, train_masks, patch_size=320, transform=train_transform, augmentations=train_augmentations, apply_augmentations=True)
511.
512. # Combine original and augmented datasets
```

```
513. combined_train_dataset = ConcatDataset([train_dataset_original, train_dataset_augmented]
    )
514. val_dataset = PatchDataset(val_images, val_masks, patch_size=320, transform=val_transfor
    m)
515.
516. # Create data loaders
517. train_loader = DataLoader(combined_train_dataset, batch_size=64, shuffle=True)
518. val_loader = DataLoader(val_dataset, batch_size=64, shuffle=False)
519.
520. # Print dataset sizes
521. print(f"Number of training patches: {len(combined_train_dataset)}")
522. print(f"Number of validation patches: {len(val_dataset)}")
523.
524. """#**Sanity Few Images**"""
525.
526. import random
527. import numpy as np
528. import matplotlib.pyplot as plt
529.
530. # Shuffle dataset indices for original dataset
531. num_sampels_to_show = 10
```

```
532. indices = list(range(len(train_dataset_original))) # Buat list dari indeks dataset orig
    inal
533. random.shuffle(indices) # Acak urutan indeks
534.
535. for i in range(num_sampels_to_show):
536.     # Get the image and mask from the original dataset
537.     idx = indices[i] # Ambil indeks acak dari hasil shuffle
538.     original_image, original_mask = train_dataset_original[idx]
539.     augmented_image, augmented_mask = train_dataset_augmented[idx] # Ambil yang sama da
        ri augmented
540.
541.     # Convert image and mask to numpy for display
542.     original_image = original_image.numpy().transpose(1, 2, 0) # HWC format for visuali
        zation
543.     original_image = np.clip(original_image, 0, 1) # Clip values between 0 and 1 if nee
        ded
544.     original_mask = original_mask.numpy() # Convert mask to numpy
545.
546.     augmented_image = augmented_image.numpy().transpose(1, 2, 0) # HWC format for visua
        lization
547.     augmented_image = np.clip(augmented_image, 0, 1) # Clip values between 0 and 1 if n
        eeded
```

```
548.     augmented_mask = augmented_mask.numpy()    # Convert mask to numpy
549.
550.     # Display the image and mask
551.     plt.figure(figsize=(15, 15))
552.
553.     # Display original image
554.     plt.subplot(2, 2, 1)
555.     plt.imshow(original_image)
556.     plt.title("Original Image")
557.
558.     # Display original mask
559.     plt.subplot(2, 2, 2)
560.     plt.imshow(original_mask, cmap='gray')
561.     plt.title("Original Mask")
562.     # Display augmented image
563.     plt.subplot(2, 2, 3)
564.     plt.imshow(augmented_image)
565.     plt.title("Augmented Image")
566.
567.     # Display augmented mask
568.     plt.subplot(2, 2, 4)
569.     plt.imshow(augmented_mask, cmap='gray')
```

```
570.     plt.title("Augmented Mask")
571.
572.     plt.tight_layout()
573.     plt.show()
574.
575. import numpy as np
576.
577. # Set random seed for reproducibility
578. np.random.seed(42)
579.
580. # Get the number of sampels in the validation dataset
581. num_val_sampels = len(val_dataset)
582.
583. # Generate random indices to shuffle the dataset
584. shuffled_indices = np.random.permutation(num_val_sampels)
585.
586. # Select a subset of indices to visualize
587. selected_indices = shuffled_indices[:num_sampels_to_show]
588.
589. # Visualize images directly from the shuffled validation dataset
590. for i in selected_indices:
591.     # Get the image and mask from the shuffled validation dataset
```

```
592.     val_image, val_mask = val_dataset[i]
593.
594.     # Convert image and mask to numpy for display
595.     val_image = val_image.numpy().transpose(1, 2, 0)  # HWC format for visualization
596.     val_image = np.clip(val_image, 0, 1)  # Clip values between 0 and 1 if needed
597.     val_mask = val_mask.numpy()  # Convert mask to numpy
598.
599.     # Display the image and mask
600.     plt.figure(figsize=(12, 6))
601.
602.     # Display validation image
603.     plt.subplot(1, 2, 1)
604.     plt.imshow(val_image)
605.     plt.title("Validation Image")
606.
607.     # Display validation mask
608.     plt.subplot(1, 2, 2)
609.     plt.imshow(val_mask, cmap='gray')  # Use a colormap like 'gray' for masks
610.     plt.title("Validation Mask (2D)")
611.
612.     plt.tight_layout()
613.     plt.show()
```

```
614.
615. """#**Training Model**"""
616.
617. import torch
618. import segmentation_models_pytorch as smp
619. from torchsummary import summary
620. import torch.optim as optim
621.
622. # Definisikan custom class untuk DeepLabV3+ dengan Dropout
623. class DeepLabV3PlusWithDropout(torch.nn.Module):
624.     def __init__(self, encoder_name="resnet34", encoder_weights="imagenet", in_channels=
        3, classes=14):
625.         super(DeepLabV3PlusWithDropout, self).__init__()
626.         # Inisialisasi model DeepLabV3+
627.         self.model = smp.DeepLabV3Plus(
628.             encoder_name=encoder_name,
629.             encoder_weights=encoder_weights,
630.             in_channels=in_channels,
631.             classes=classes
632.         )
633.         # Tambahkan Dropout dengan nilai default p=0.5
634.         self.dropout = torch.nn.Dropout(p=0.1)
```

```
635.
636.     def forward(self, x):
637.         # Forward pass melalui model
638.         x = self.model(x)
639.         # Terapkan Dropout
640.         x = self.dropout(x)
641.         return x
642.
643. # Tentukan device untuk training (CPU atau GPU)
644. device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
645.
646. # Buat instance model dengan Dropout
647. model = DeepLabV3PlusWithDropout(
648.     encoder_name="resnet34",          # Encoder ResNet34
649.     encoder_weights="imagenet",       # Pretrained weights
650.     in_channels=3,                    # Input channels (RGB)
651.     classes=14                        # Jumlah kelas output
652. ).to(device)
653.
654. # Loss function
655. criterion = torch.nn.CrossEntropyLoss()
656.
```

```
657. # Optimizer
658. optimizer = torch.optim.Adam(model.parameters(), lr=0.001, weight_decay=0.0001)
659.
660. # Scheduler StepLR untuk menurunkan learning rate setiap 25 epoch
661. scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=25, gamma=0.1)
662.
663. # Cetak ringkasan model untuk verifikasi
664. summary(model, input_size=(3, 320, 320))
665.
666. """#**Latihan Sesi Ke-1**"""
667.
668. # Define the directory for saving checkpoints
669. CHECKPOINT_DIR = './Model/'
670. if not os.path.exists(CHECKPOINT_DIR):
671.     os.makedirs(CHECKPOINT_DIR)
672.
673. # Function to calculate mean IoU, accuracy, Dice coefficient, and mAP
674. def calculate_metrics(preds, masks, num_classes):
675.     preds_flat = preds.flatten()
676.     masks_flat = masks.flatten()
677.     iou = jaccard_score(masks_flat, preds_flat, average='macro', zero_division=1)
678.     accuracy = (preds_flat == masks_flat).mean()
```

```
679.     dice = f1_score(masks_flat, preds_flat, average='macro', zero_division=1)
680.     precision = precision_score(masks_flat, preds_flat, average='macro', zero_division=1
    )
681.     recall = recall_score(masks_flat, preds_flat, average='macro', zero_division=1)
682.     mAP = precision
683.     return iou, accuracy, dice, mAP
684.
685. # Function to save checkpoint
686. def save_checkpoint(epoch, model, optimizer, scheduler, train_losses, val_losses, train_
    ious, val_ious, train_accuracies, val_accuracies, train_dices, val_dices, train_maps, val_m
    aps, learning_rates, filename='checkpoint.pth'):
687.     filepath = os.path.join(CHECKPOINT_DIR, filename)
688.     state = {
689.         'epoch': epoch,
690.         'model_state_dict': model.state_dict(),
691.         'optimizer_state_dict': optimizer.state_dict(),
692.         'scheduler_state_dict': scheduler.state_dict(),
693.         'train_losses': train_losses,
694.         'val_losses': val_losses,
695.         'train_ious': train_ious,
696.         'val_ious': val_ious,
697.         'train_accuracies': train_accuracies,
```

```

698.         'val_accuracies': val_accuracies,
699.         'train_dices': train_dices,
700.         'val_dices': val_dices,
701.         'train_maps': train_maps,
702.         'val_maps': val_maps,
703.         'learning_rates': learning_rates
704.     }
705.     torch.save(state, filepath)
706.     print(f'Checkpoint saved at epoch {epoch} to {filepath}')
707.
708. # Function to load the best validation loss from best_model.pth
709. def load_best_val_loss(filepath):
710.     if os.path.exists(filepath):
711.         checkpoint = torch.load(filepath)
712.         return checkpoint.get('best_val_loss', float('inf')) # Default to inf if not fo
und
713.     return float('inf') # Return inf if file does not exist
714.
715. # Function to save the best model based on early stopping
716. def save_best_model(epoch, model, val_loss, best_val_loss, is_new_session, filename='bes
t_model.pth'):
717.     os.makedirs(CHECKPOINT_DIR, exist_ok=True) # Ensure checkpoint directory exists

```

```
718.     filepath = os.path.join(CHECKPOINT_DIR, filename)
719.
720.     # If it's a new session and the first epoch, only save if val_loss improves over the
    loaded best_val_loss
721.     if is_new_session and epoch == 1 and val_loss >= best_val_loss:
722.         print(f"Skipping model save for epoch {epoch} as validation loss ({val_loss:.4f}
    ) didn't improve.")
723.         return best_val_loss
724.
725.     # Save model if validation loss improves
726.     if val_loss < best_val_loss:
727.         best_val_loss = val_loss
728.         torch.save({'model_state_dict': model.state_dict(), 'best_val_loss': best_val_lo
    ss}, filepath)
729.         print(f'Best model saved at epoch {epoch} with validation loss: {val_loss:.4f}')

730.     return best_val_loss
731.
732. # Function to load checkpoint
733. def load_checkpoint(filename, model, optimizer, scheduler):
734.     filepath = os.path.join(CHECKPOINT_DIR, filename)
735.     checkpoint = torch.load(filepath)
```

```
736.     model.load_state_dict(checkpoint['model_state_dict'])
737.     optimizer.load_state_dict(checkpoint['optimizer_state_dict'])
738.     scheduler.load_state_dict(checkpoint['scheduler_state_dict'])
739.     start_epoch = checkpoint['epoch'] + 1
740.     train_losses = checkpoint.get('train_losses', [])
741.     val_losses = checkpoint.get('val_losses', [])
742.     train_iious = checkpoint.get('train_iious', [])
743.     val_iious = checkpoint.get('val_iious', [])
744.     train_accuracies = checkpoint.get('train_accuracies', [])
745.     val_accuracies = checkpoint.get('val_accuracies', [])
746.     train_dices = checkpoint.get('train_dices', [])
747.     val_dices = checkpoint.get('val_dices', [])
748.     train_maps = checkpoint.get('train_maps', [])
749.     val_maps = checkpoint.get('val_maps', [])
750.     learning_rates = checkpoint.get('learning_rates', [])
751.     print(f'Checkpoint loaded from {filepath}, resuming training from epoch {start_epoch
        }')
752.     return start_epoch, train_losses, val_losses, train_iious, val_iious, train_accuracies
        , val_accuracies, train_dices, val_dices, train_maps, val_maps, learning_rates
753.
754. # Main training function
```

```
755. def train_model(model, train_loader, val_loader, loss_fn, optimizer, scheduler, num_epoc
    hs=50, resume_from_checkpoint=None, num_classes=7, min_lr=1e-6):
756.     train_losses, val_losses = [], []
757.     train_iious, val_iious = [], []
758.     train_accuracies, val_accuracies = [], []
759.     train_dices, val_dices = [], []
760.     train_maps, val_maps = [], []
761.     learning_rates = []
762.
763.     start_epoch, is_new_session = 1, True
764.     best_val_loss = float('inf')
765.
766.     # Load checkpoint if resuming training
767.     if resume_from_checkpoint:
768.         start_epoch, train_losses, val_losses, train_iious, val_iious, train_accuracies, v
            al_accuracies, train_dices, val_dices, train_maps, val_maps, learning_rates = load_checkpoi
            nt(resume_from_checkpoint, model, optimizer, scheduler)
769.         scheduler.last_epoch = start_epoch - 1
770.         is_new_session = False
771.         # Load best_val_loss from the best model
772.         best_val_loss = load_best_val_loss(os.path.join(CHECKPOINT_DIR, 'best_model.pth'
            ))
```

```
773.
774.     total_epochs = start_epoch + num_epochs - 1
775.
776.     for epoch in range(start_epoch, total_epochs + 1):
777.         model.train()
778.         train_loss, train_iou, train_correct, train_dice, train_map, total_train_pixels
       = 0, 0, 0, 0, 0, 0
779.
780.         for images, masks in tqdm(train_loader):
781.             images, masks = images.to(device).float(), masks.to(device).long()
782.             optimizer.zero_grad()
783.             outputs = model(images)
784.             loss = loss_fn(outputs, masks)
785.             loss.backward()
786.             optimizer.step()
787.
788.             train_loss += loss.item() * images.size(0)
789.             preds = torch.argmax(outputs, dim=1).cpu().numpy()
790.             masks = masks.cpu().numpy()
791.
792.             iou, accuracy, dice, mAP = calculate_metrics(preds, masks, num_classes)
793.             train_iou += iou * images.size(0)
```

```
794.         train_correct += (preds == masks).sum().item()
795.         train_dice += dice * images.size(0)
796.         train_map += mAP * images.size(0)
797.         total_train_pixels += masks.size
798.
799.         train_losses.append(train_loss / len(train_loader.dataset))
800.         train_iou.append(train_iou / len(train_loader.dataset))
801.         train_accuracies.append(train_correct / total_train_pixels)
802.         train_dices.append(train_dice / len(train_loader.dataset))
803.         train_maps.append(train_map / len(train_loader.dataset))
804.
805.     model.eval()
806.     val_loss, val_iou, val_correct, val_dice, val_map, total_val_pixels = 0, 0, 0, 0
807.     , 0, 0
808.
809.     with torch.no_grad():
810.         for images, masks in val_loader:
811.             images, masks = images.to(device).float(), masks.to(device).long()
812.             outputs = model(images)
813.             loss = loss_fn(outputs, masks)
814.
815.             val_loss += loss.item() * images.size(0)
```

```

815.         preds = torch.argmax(outputs, dim=1).cpu().numpy()
816.         masks = masks.cpu().numpy()
817.
818.         iou, accuracy, dice, mAP = calculate_metrics(preds, masks, num_classes)
819.
820.         val_iou += iou * images.size(0)
821.         val_correct += (preds == masks).sum().item()
822.         val_dice += dice * images.size(0)
823.         val_map += mAP * images.size(0)
824.         total_val_pixels += masks.size
825.
826.         val_losses.append(val_loss / len(val_loader.dataset))
827.         val_iou.append(val_iou / len(val_loader.dataset))
828.         val_acc.append(val_correct / total_val_pixels)
829.         val_dice.append(val_dice / len(val_loader.dataset))
830.         val_map.append(val_map / len(val_loader.dataset))
831.
832.         print(f'Epoch {epoch}/{total_epochs}')
833.         print(f'  Train Loss:      {train_losses[-1]:.4f}, Val Loss:      {val_losses[-1]:.4f}')
834.         print(f'  Train IoU:      {train_iou[-1]:.4f}, Val IoU:      {val_iou[-1]:.4f}')

```

```
834.         print(f'  Train Accuracy:  {train_accuracies[-1]:.4f}, Val Accuracy:  {val_accuracies[-1]:.4f}')
835.         print(f'  Train Dice:      {train_dices[-1]:.4f}, Val Dice:      {val_dices[-1]:.4f}')
836.         print(f'  Train mAP:       {train_maps[-1]:.4f}, Val mAP:       {val_maps[-1]:.4f}')
837.
838.         # Save regular checkpoint
839.         save_checkpoint(epoch, model, optimizer, scheduler,
840.                         train_losses, val_losses,
841.                         train_iious, val_iious,
842.                         train_accuracies, val_accuracies,
843.                         train_dices, val_dices,
844.                         train_maps, val_maps, learning_rates)
845.
846.         # Scheduler step
847.         scheduler.step()
848.         print(f"Scheduler updated learning rate: {scheduler.optimizer.param_groups[0]['lr']}")
849.
850.         # Save the best model
```

```
851.         best_val_loss = save_best_model(epoch, model, val_losses[-
    1], best_val_loss, is_new_session, filename='best_model.pth')
852.
853.         return train_losses, val_losses, train_iious, val_iious, train_accuracies, val_accurac
    ies, train_dices, val_dices, train_maps, val_maps, learning_rates
854.
855. # Example to start training or resume from a checkpoint
856. resume_from_checkpoint = 'checkpoint.pth'
857. train_losses, val_losses, train_iious, val_iious, train_accuracies, val_accuracies, train_
    dices, val_dices, train_maps, val_maps, learning_rates = train_model(
858.     model, train_loader, val_loader, criterion, optimizer, scheduler, num_epochs=30, res
    ume_from_checkpoint=resume_from_checkpoint, num_classes=14
859. )
860.
861. """#**Plot Evaluation**"""
862.
863. import csv
864. import torch
865. import os
866. import pandas as pd
867. from IPython.display import display, HTML
868.
```

```
869. # Path untuk checkpoint dan CSV
870. checkpoint_path = '/content/drive/MyDrive/skripsi segmentasi citra satelit /Code/CodeAll
    Dataset25November/checkpoint.pth' # Sesuaikan path checkpoint Anda
871. output_dir = '/content' # Direktori untuk file CSV
872. csv_file = os.path.join(output_dir, 'training_metrics_filled.csv') # File CSV yang akan
    disimpan
873.
874. # Buat direktori jika belum ada
875. os.makedirs(output_dir, exist_ok=True)
876.
877. # Load checkpoint
878. checkpoint = torch.load(checkpoint_path)
879.
880. # Extract metrics dari checkpoint
881. train_losses = checkpoint['train_losses']
882. val_losses = checkpoint.get('val_losses', [])
883. train_iious = checkpoint['train_iious']
884. val_iious = checkpoint.get('val_iious', [])
885. train_accuracies = checkpoint['train_accuracies']
886. val_accuracies = checkpoint.get('val_accuracies', [])
887. train_dices = checkpoint['train_dices']
888. val_dices = checkpoint.get('val_dices', [])
```

```
889. train_maps = checkpoint['train_maps']
890. val_maps = checkpoint.get('val_maps', [])
891.
892. # Kolom CSV
893. csv_columns = ['Epoch', 'Train Loss', 'Val Loss', 'Train IoU', 'Val IoU', 'Train Accurac
    y', 'Val Accuracy',
894.                 'Train Dice', 'Val Dice', 'Train mAP', 'Val mAP']
895.
896. # Tulis data ke file CSV
897. with open(csv_file, mode='w', newline='') as file:
898.     writer = csv.writer(file)
899.     writer.writerow(csv_columns) # Header
900.     for epoch in range(len(train_losses)):
901.         row = [
902.             epoch + 1,
903.             train_losses[epoch],
904.             val_losses[epoch] if epoch < len(val_losses) else None,
905.             train_iious[epoch],
906.             val_iious[epoch] if epoch < len(val_iious) else None,
907.             train_accuracies[epoch],
908.             val_accuracies[epoch] if epoch < len(val_accuracies) else None,
909.             train_dices[epoch],
```

```
910.         val_dices[epoch] if epoch < len(val_dices) else None,
911.         train_maps[epoch],
912.         val_maps[epoch] if epoch < len(val_maps) else None
913.     ]
914.     writer.writerow(row)
915.
916. print(f'Metrics saved to {csv_file}')
917.
918. # Load CSV dan tampilkan sebagai tabel
919. if os.path.exists(csv_file):
920.     metrics_df = pd.read_csv(csv_file)
921.     display(metrics_df)
922.
923. # Generate link untuk download file
924. display(HTML(f'<a href="{csv_file}" download>Click here to download the CSV file</a>'))
925.
926. # Plot training metrics after training
927. import os
928. import pandas as pd
929. import matplotlib.pyplot as plt
930.
```

```
931. metrics = {
932.     'Loss': ('Train Loss', 'Val Loss'),
933.     'IoU': ('Train IoU', 'Val IoU'),
934.     'Accuracy': ('Train Accuracy', 'Val Accuracy'),
935.     'Dice': ('Train Dice', 'Val Dice'),
936.     'mAP': ('Train mAP', 'Val mAP'),
937. }
938.
939. if os.path.exists(csv_file):
940.     metrics_df = pd.read_csv(csv_file)
941.     for metric, (train_column, val_column) in metrics.items():
942.         plt.figure(figsize=(20, 8))
943.
944.         # Plot the training metric
945.         plt.plot(metrics_df['Epoch'], metrics_df[train_column], label=f'Train {metric}',
946.                  linewidth=3)
947.
948.         # Plot the validation metric if available
949.         if val_column and val_column in metrics_df.columns:
950.             plt.plot(metrics_df['Epoch'], metrics_df[val_column], label=f'Validation {me
```

```
951.         # Label axes and title
952.         plt.xlabel('Epochs', fontsize=14)
953.         plt.ylabel(metric, fontsize=14)
954.         plt.title(f'{metric} vs. Epochs', fontsize=14)
955.
956.         # Add legend and grid
957.         plt.legend(fontsize=14)
958.         plt.grid(True)
959.
960.         # Show the plot
961.         plt.show()
962.
963. import torch
964.
965. # Path ke file best_model.pth
966. file_path = '/content/drive/MyDrive/skripsi segmentasi citra satelit /Code/CodeAllDatase
    t25November/best_model.pth'
967.
968. # Memuat file checkpoint
969. checkpoint = torch.load(file_path)
970.
971. # Menampilkan semua isi file
```

```
972. print("Isi file best_model.pth:")
973. for key, value in checkpoint.items():
974.     print(f"{key}: {value}")
```

2. File Pengujian Model

```
1. # -*- coding: utf-8 -*-
2. """FinalTesting25NovemberNoSmoothing320.ipynb
3.
4. Automatically generated by Colab.
5.
6. Original file is located at
7.     https://colab.research.google.com/drive/1eYMf-wlv7ng6GS2xfpDegJgdeASQ9wHh
8.
9. Connect to GDrive
10. """
11.
12. from google.colab import drive
13. drive.mount('/content/drive')
14.
15. """Install Library"""
16.
17. # Install necessary libraries
18. !pip install patchify
19. !pip install segmentation_models
20. !pip install tensorflow
```

```
21. !pip install torch torchvision
22. !pip install segmentation_models_pytorch
23. !pip install pytorchtools
24. !pip install torchsummary
25. !pip install torchviz
26. !apt-get install graphviz
27.
28. """#**Import Library**"""
29.
30. import os
31. import numpy as np
32. import cv2
33. import torch
34. from torch.utils.data import Dataset, DataLoader
35. import albumentations as A
36. from albumentations.pytorch import ToTensorV2
37. import segmentation_models_pytorch as smp
38. import matplotlib.pyplot as plt
39. from sklearn.preprocessing import MinMaxScaler
40. from patchify import patchify, unpatchify
41.
42. """#**Set Device**"""
```

```
43.
44.# Set device
45.device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
46.print(f'Using device: {device}')
47.torch.backends.cudnn.deterministic = True
48.torch.backends.cudnn.benchmark = True
49.torch.backends.cudnn.enabled = True
50.
51."""#**Predicting Image**"""
52.
53.import os
54.import cv2
55.import numpy as np
56.import torch
57.from torch.utils.data import Dataset, DataLoader
58.import albumentations as A
59.from albumentations.pytorch import ToTensorV2
60.from sklearn.preprocessing import MinMaxScaler
61.from patchify import patchify, unpatchify
62.import matplotlib.pyplot as plt
63.import segmentation_models_pytorch as smp
64.import time
```

```
65.
66.# Set device
67.device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
68.
69.# Define transformations for test without normalization
70.test_transform = A.Compose([ToTensorV2()])
71.
72.# Function to convert label to RGB
73.def label_to_rgb(label):
74.    label_rgb = np.zeros((label.shape[0], label.shape[1], 3), dtype=np.uint8)
75.    label_colors = {
76.        0: [112, 240, 0], 1: [184, 0, 240], 2: [240, 184, 0], 3: [60, 0, 138],
77.        4: [0, 228, 240], 5: [228, 240, 0], 6: [0, 98, 255], 7: [255, 0, 4],
78.        8: [65, 0, 204], 9: [204, 122, 0], 10: [255, 0, 247], 11: [120, 120, 120],
79.        12: [255, 173, 183], 13: [122, 0, 14]
80.    }
81.    for label_idx, rgb in label_colors.items():
82.        label_rgb[label == label_idx] = rgb
83.    return label_rgb
84.
85.# Define the SegmentationDataset class
86.class SegmentationDataset(Dataset):
```

```
87.     def __init__(self, image_paths, mask_paths, patch_size=1024, transform=None):
88.         self.image_paths = image_paths
89.         self.mask_paths = mask_paths
90.         self.patch_size = patch_size
91.         self.transform = transform
92.         self.scaler = MinMaxScaler()
93.         self.image_patches = []
94.         self.mask_patches = []
95.         self.create_patches()
96.
97.     def create_patches(self):
98.         for img_path, mask_path in zip(self.image_paths, self.mask_paths):
99.             image = cv2.imread(img_path)
100.             image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
101.             mask = cv2.imread(mask_path, 1)
102.             mask = cv2.cvtColor(mask, cv2.COLOR_BGR2RGB)
103.
104.             # Resize dimensions to the nearest multiple of patch_size
105.             new_height = round(image.shape[0] / self.patch_size) * self.patch_size
106.             new_width = round(image.shape[1] / self.patch_size) * self.patch_size
107.
108.             resized_image = cv2.resize(image, (new_width, new_height))
```

```
109.         resized_mask = cv2.resize(mask, (new_width, new_height), interpolation=cv2.I
    NTER_NEAREST)
110.
111.         # Patchify resized images and masks
112.         patches_img = patchify(resized_image, (self.patch_size, self.patch_size, 3),
    step=self.patch_size)
113.         patches_mask = patchify(resized_mask, (self.patch_size, self.patch_size, 3),
    step=self.patch_size)
114.
115.         for i in range(patches_img.shape[0]):
116.             for j in range(patches_img.shape[1]):
117.                 patch_img = patches_img[i, j, 0, :, :, :]
118.                 patch_mask = patches_mask[i, j, 0, :, :, :]
119.
120.                 # Scale the image patch
121.                 patch_img = self.scaler.fit_transform(patch_img.reshape(-
    1, patch_img.shape[-1])).reshape(patch_img.shape)
122.
123.                 # Convert RGB mask to label mask
124.                 patch_mask = self.rgb_to_2D_label(patch_mask)
125.
126.                 if self.transform:
```

```
127.             patch_img = self.transform(image=patch_img)['image']
128.             patch_mask = torch.tensor(patch_mask, dtype=torch.long)
129.
130.             self.image_patches.append(patch_img)
131.             self.mask_patches.append(patch_mask)
132.
133.     def __len__(self):
134.         return len(self.image_patches)
135.
136.     def __getitem__(self, idx):
137.         return self.image_patches[idx], self.mask_patches[idx]
138.
139.     def rgb_to_2D_label(self, label):
140.         label_colors = {
141.             (112, 240, 0): 0, (184, 0, 240): 1, (240, 184, 0): 2, (60, 0, 138): 3,
142.             (0, 228, 240): 4, (228, 240, 0): 5, (0, 98, 255): 6, (255, 0, 4): 7,
143.             (65, 0, 204): 8, (204, 122, 0): 9, (255, 0, 247): 10, (120, 120, 120): 11,
144.             (255, 173, 183): 12, (122, 0, 14): 13
145.         }
146.         label_seg = np.zeros(label.shape[:2], dtype=np.uint8)
147.         for rgb, idx in label_colors.items():
148.             label_seg[np.all(label == rgb, axis=-1)] = idx
```

```
149.         return label_seg
150.
151. # Function to visualize predictions using a DataLoader
152. def visualize_predictions(model, loader, original_size, resized_size, patch_size=1024):

153.     model.eval()
154.
155.     predictions = []
156.     all_masks = []
157.
158.     for images, masks in loader:
159.         images = images.to(device).float()
160.         masks = masks.to(device).long()
161.
162.         with torch.no_grad():
163.             outputs = model(images)
164.             preds = torch.argmax(outputs, dim=1).cpu().numpy()
165.
166.             predictions.extend(preds)
167.             all_masks.extend(masks.cpu().numpy())
168.
169.     predictions = np.array(predictions)
```

```
170.     all_masks = np.array(all_masks)
171.
172.     # Reshape patches back to resized image size
173.     num_patches_x = resized_size[1] // patch_size
174.     num_patches_y = resized_size[0] // patch_size
175.
176.     pred_image = unpatchify(predictions.reshape(num_patches_y, num_patches_x, patch_size
        , patch_size), resized_size)
177.     mask_image = unpatchify(all_masks.reshape(num_patches_y, num_patches_x, patch_size,
        patch_size), resized_size)
178.
179.     # Resize back to the original image size
180.     pred_image = cv2.resize(pred_image, (original_size[1], original_size[0]), interpolat
        ion=cv2.INTER_NEAREST)
181.     mask_image = cv2.resize(mask_image, (original_size[1], original_size[0]), interpolat
        ion=cv2.INTER_NEAREST)
182.
183.     return pred_image, mask_image
184.
185. # Load the model and weights
186. best_model = smp.DeepLabV3Plus(
187.     encoder_name="resnet34",
```

```
188.     encoder_weights="imagenet",
189.     in_channels=3,
190.     classes=14
191. ).to(device)
192.
193. # Load the checkpoint and fix the key mismatch issue
194. checkpoint_path = '/content/drive/MyDrive/skripsi segmentasi citra satelit /Code/CodeAll
    Dataset25November/best_model.pth'
195. checkpoint = torch.load(checkpoint_path, map_location=device)
196.
197. if "model_state_dict" in checkpoint:
198.     state_dict = checkpoint["model_state_dict"]
199. else:
200.     state_dict = checkpoint
201.
202. # Remove "model." prefix if present
203. new_state_dict = {k.replace("model.", ""): v for k, v in state_dict.items()}
204.
205. # Load the state dictionary
206. best_model.load_state_dict(new_state_dict)
207. best_model.eval()
208.
```

```
209. # Set up paths and dataset
210. base_path = '/content/drive/MyDrive/skripsi segmentasi citra satelit /AllDataset/TestFolder'
211. image_paths = [os.path.join(base_path, 'images', img) for img in os.listdir(os.path.join(base_path, 'images')) if img.endswith('.jpg')]
212. mask_paths = [os.path.join(base_path, 'masks', img.replace('.jpg', '.png')) for img in os.listdir(os.path.join(base_path, 'images')) if img.endswith('.jpg')]
213. patchsize = 320 # Patch size
214.
215. # Create the dataset and loader for test images
216. dataset = SegmentationDataset(image_paths=image_paths, mask_paths=mask_paths, patch_size=patchsize, transform=test_transform)
217. patch_loader = DataLoader(dataset, batch_size=64, shuffle=False)
218.
219. # Visualization and timing
220. prediction_times = []
221. for image_path, mask_path in zip(image_paths, mask_paths):
222.     image = cv2.imread(image_path)
223.     image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
224.     mask = cv2.imread(mask_path)
225.
226.     original_size = image.shape[:2]
```

```
227.     new_height = round(original_size[0] / patchsize) * patchsize
228.     new_width = round(original_size[1] / patchsize) * patchsize
229.     resized_size = (new_height, new_width)
230.
231.     single_image_dataset = SegmentationDataset(
232.         image_paths=[image_path],
233.         mask_paths=[mask_path],
234.         patch_size=patchsize,
235.         transform=test_transform
236.     )
237.     single_patch_loader = DataLoader(single_image_dataset, batch_size=64, shuffle=False)
238.
239.     start_time = time.time()
240.     pred_image, mask_image = visualize_predictions(
241.         best_model, single_patch_loader, original_size, resized_size, patch_size=patchsi
ze
242.     )
243.     prediction_time = time.time() - start_time
244.     prediction_times.append(prediction_time)
245.
246.     print(f"Prediction time for {image_path}: {prediction_time:.4f} seconds")
```

```
247.
248.     pred_rgb = label_to_rgb(pred_image)
249.     mask_rgb = label_to_rgb(mask_image)
250.
251.     plt.figure(figsize=(24, 8))
252.     plt.subplot(131)
253.     plt.title('Original Image')
254.     plt.imshow(image)
255.
256.     plt.subplot(132)
257.     plt.title('Ground Truth Mask')
258.     plt.imshow(mask_rgb)
259.
260.     plt.subplot(133)
261.     plt.title('Predicted Mask')
262.     plt.imshow(pred_rgb)
263.     plt.show()
264.
265.     print(f"Average prediction time per image: {np.mean(prediction_times):.4f} seconds")
266.
267.     """#**Calculate Metrics**
268.
```

```
269. Confusion Matrix
270. """
271.
272. from sklearn.metrics import confusion_matrix
273. import seaborn as sns
274.
275. # Function to compute the confusion matrix
276. def compute_confusion_matrix(model, loader):
277.     model.eval()
278.
279.     all_preds = []
280.     all_labels = []
281.
282.     for images, masks in loader:
283.         images = images.to(device).float()
284.         masks = masks.to(device).long()
285.
286.         with torch.no_grad():
287.             outputs = model(images)
288.             preds = torch.argmax(outputs, dim=1).cpu().numpy()
289.
290.             all_preds.extend(preds.flatten()) # Flatten the predictions
```

```
291.         all_labels.extend(masks.cpu().numpy().flatten()) # Flatten the ground truth mas
ks
292.
293.     # Compute confusion matrix
294.     cm = confusion_matrix(all_labels, all_preds, labels=list(range(14))) # Assuming 14
classes
295.     cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis] * 100 # Convert to percenta
ges
296.     return cm
297.
298. # Function to plot the confusion matrix
299. def plot_confusion_matrix(cm, class_names):
300.     plt.figure(figsize=(15, 12))
301.     sns.heatmap(cm, annot=True, fmt='.2f', cmap='Blues', xticklabels=class_names, ytickl
abels=class_names)
302.     plt.title("Confusion Matrix (%)")
303.     plt.ylabel("True Labels")
304.     plt.xlabel("Predicted Labels")
305.     plt.show()
306.
307. # Define your class names (corresponding to the label IDs)
308. class_names = ['Cropland', 'Permukiman', 'Air', 'Area Terbuka', 'Jalan', 'Vegetasi',
```

```
309.             'Herbaceous Vegetation', 'Lava Dome', 'Jalur Lahar', 'Awan Panas',
310.             'Endapan Baru', 'Endapan Lama', 'Awan', 'Lava Flow']
311.
312. # Compute confusion matrix
313. cm = compute_confusion_matrix(best_model, patch_loader)
314.
315. # Plot the confusion matrix
316. plot_confusion_matrix(cm, class_names)
317.
318. """Metrics Per Class"""
319.
320. import numpy as np
321. import torch
322. from sklearn.metrics import confusion_matrix, precision_score, recall_score, f1_score
323. import pandas as pd
324.
325. # Function to compute Intersection over Union (IoU) for each class
326. def compute_iou_per_class(conf_matrix):
327.     iou_per_class = np.zeros(conf_matrix.shape[0])
328.
329.     for c in range(conf_matrix.shape[0]):
330.         intersection = conf_matrix[c, c]
```

```
331.         ground_truth = conf_matrix[c, :].sum()
332.         predicted = conf_matrix[:, c].sum()
333.         union = ground_truth + predicted - intersection
334.         iou_per_class[c] = intersection / union if union != 0 else 0
335.
336.     return iou_per_class
337.
338. # Function to compute mIoU, mAP, and F1 Score
339. def compute_metrics(model, loader, num_classes=14):
340.     model.eval()
341.
342.     all_preds = []
343.     all_labels = []
344.
345.     for images, masks in loader:
346.         images = images.to(device).float()
347.         masks = masks.to(device).long()
348.
349.         with torch.no_grad():
350.             outputs = model(images)
351.             preds = torch.argmax(outputs, dim=1).cpu().numpy()
352.
```

```
353.         all_preds.extend(preds.flatten()) # Flatten predictions
354.         all_labels.extend(masks.cpu().numpy().flatten()) # Flatten ground truth masks
355.
356.     # Compute confusion matrix
357.     conf_matrix = confusion_matrix(all_labels, all_preds, labels=list(range(num_classes)
    ))
358.
359.     # Calculate IoU for each class
360.     iou_per_class = compute_iou_per_class(conf_matrix)
361.     mean_iou = np.mean(iou_per_class)
362.
363.     # Compute precision, recall, and F1 Score for each class
364.     precision_per_class = precision_score(all_labels, all_preds, average=None, labels=list(range(num_classes)))
365.     recall_per_class = recall_score(all_labels, all_preds, average=None, labels=list(range(num_classes)))
366.     f1_per_class = f1_score(all_labels, all_preds, average=None, labels=list(range(num_classes)))
367.
368.     # Calculate mAP (mean of average precision across classes)
369.     mean_ap = np.mean(precision_per_class)
370.
```

```
371.         return mean_iou, mean_ap, iou_per_class, precision_per_class, recall_per_class, fl_p
           er_class
372.
373.  # Define class names (for better understanding of results)
374.  class_names = ['Cropland', 'Permukiman', 'Air', 'Area Terbuka', 'Jalan', 'Vegetasi',
375.                 'Herbaceous Vegetation', 'Lava Dome', 'Jalur Lahar', 'Awan Panas',
376.                 'Endapan Baru', 'Endapan Lama', 'Awan', 'Lava Flow']
377.
378.  # Compute metrics
379.  mean_iou, mean_ap, iou_per_class, precision_per_class, recall_per_class, fl_per_class =
           compute_metrics(best_model, patch_loader)
380.
381.  # Create a DataFrame for better visualization
382.  metrics_df = pd.DataFrame({
383.      'Class': class_names,
384.      'IoU': iou_per_class,
385.      'Precision': precision_per_class,
386.      'Recall': recall_per_class,
387.      'F1 Score': fl_per_class
388.  })
389.
390.  # Display the DataFrame
```

```
391. print(metrics_df)
392.
393. """mAP and mIoU"""
394.
395. import numpy as np
396. import torch
397. from sklearn.metrics import confusion_matrix
398.
399. # Function to compute Intersection over Union (IoU) for each class
400. def compute_iou_per_class(conf_matrix):
401.     iou_per_class = np.zeros(conf_matrix.shape[0])
402.
403.     for c in range(conf_matrix.shape[0]):
404.         intersection = conf_matrix[c, c]
405.         ground_truth = conf_matrix[c, :].sum()
406.         predicted = conf_matrix[:, c].sum()
407.         union = ground_truth + predicted - intersection
408.         iou_per_class[c] = intersection / union if union != 0 else 0
409.
410.     return iou_per_class
411.
412. # Function to compute mean Average Precision (mAP)
```

```
413. def compute_ap_per_class(conf_matrix):
414.     ap_per_class = np.zeros(conf_matrix.shape[0])
415.
416.     for c in range(conf_matrix.shape[0]):
417.         tp = conf_matrix[c, c]
418.         fp = conf_matrix[:, c].sum() - tp
419.         fn = conf_matrix[c, :].sum() - tp
420.         precision = tp / (tp + fp) if (tp + fp) != 0 else 0
421.         recall = tp / (tp + fn) if (tp + fn) != 0 else 0
422.         ap_per_class[c] = precision * recall # Simplified AP calculation
423.
424.     return ap_per_class
425.
426. # Function to compute mIoU and mAP
427. def compute_metrics(model, loader, num_classes=14):
428.     model.eval()
429.
430.     all_preds = []
431.     all_labels = []
432.
433.     for images, masks in loader:
434.         images = images.to(device).float()
```

```
435.         masks = masks.to(device).long()
436.
437.         with torch.no_grad():
438.             outputs = model(images)
439.             preds = torch.argmax(outputs, dim=1).cpu().numpy()
440.
441.             all_preds.extend(preds.flatten()) # Flatten predictions
442.             all_labels.extend(masks.cpu().numpy().flatten()) # Flatten ground truth masks
443.
444.         # Compute confusion matrix
445.         conf_matrix = confusion_matrix(all_labels, all_preds, labels=list(range(num_classes)
446.                                     ))
447.
448.         # Calculate IoU for each class
449.         iou_per_class = compute_iou_per_class(conf_matrix)
450.         mean_iou = np.mean(iou_per_class)
451.
452.         # Calculate AP for each class (used to calculate mAP)
453.         ap_per_class = compute_ap_per_class(conf_matrix)
454.         mean_ap = np.mean(ap_per_class)
455.
456.     return mean_iou, mean_ap
```

```
456.
457. # Compute mIoU and mAP
458. mean_iou, mean_ap = compute_metrics(best_model, patch_loader)
459.
460. # Display the results
461. print(f"Mean IoU (mIoU): {mean_iou}")
462. print(f"Mean Average Precision (mAP): {mean_ap}")
463.
464. """Plot Metrics"""
465.
466. from sklearn.metrics import classification_report
467.
468. # Function to generate a classification report
469. def generate_classification_report(model, loader, num_classes=14):
470.     model.eval()
471.
472.     all_preds = []
473.     all_labels = []
474.
475.     for images, masks in loader:
476.         images = images.to(device).float()
477.         masks = masks.to(device).long()
```

```
478.
479.         with torch.no_grad():
480.             outputs = model(images)
481.             preds = torch.argmax(outputs, dim=1).cpu().numpy()
482.
483.             all_preds.extend(preds.flatten()) # Flatten predictions
484.             all_labels.extend(masks.cpu().numpy().flatten()) # Flatten ground truth masks
485.
486.         # Generate classification report
487.         report = classification_report(all_labels, all_preds, target_names=class_names, label
            ls=list(range(num_classes)), digits=4)
488.         return report
489.
490. # Define class names (corresponding to label IDs)
491. class_names = [
492.     'Cropland', 'Permukiman', 'Air', 'Area Terbuka', 'Jalan', 'Vegetasi',
493.     'Herbaceous Vegetation', 'Lava Dome', 'Jalur Lahar', 'Awan Panas',
494.     'Endapan Material Vulkanik Baru', 'Endapan Material Vulkanik Lama',
495.     'Awan', 'Lava Flow'
496. ]
497.
498. # Generate and print the classification report
```

```
499. classification_report_str = generate_classification_report(best_model, patch_loader)
500. print(classification_report_str)
501.
502. import matplotlib.pyplot as plt
503. import numpy as np
504. import re
505.
506. # Function to generate and plot precision, recall, and F1-score
507. def plot_metrics_bar_chart(report, class_names):
508.     # Extract precision, recall, and f1-score values from the report
509.     lines = report.split("\n")
510.     class_metrics = []
511.
512.     # Iterate over lines and collect class metrics
513.     for line in lines[2:]:
514.         if line.strip() == "" or any(avg in line.lower() for avg in ["accuracy", "macro
            avg", "weighted avg"]):
515.             continue
516.
517.         try:
518.             # Use regex to find the numeric values in the line
519.             values = re.findall(r"\d+\.\d+", line)
```

```

520.         if len(values) >= 3:  # Ensure we have at least precision, recall, and f1-
    score
521.             class_metrics.append([float(v) for v in values[:3]])
522.         except ValueError:
523.             continue
524.
525.     # Convert to numpy array
526.     class_metrics = np.array(class_metrics)
527.
528.     # Ensure that we have the correct number of classes parsed
529.     if len(class_metrics) != len(class_names):
530.         raise ValueError(f"Number of parsed classes ({len(class_metrics)}) does not matc
    h number of class names ({len(class_names)}).")
531.
532.     precision = class_metrics[:, 0]
533.     recall = class_metrics[:, 1]
534.     f1_score = class_metrics[:, 2]
535.
536.     # Plot the bar chart
537.     x = np.arange(len(class_names))  # label locations
538.     width = 0.25  # width of the bars
539.

```

```
540.     fig, ax = plt.subplots(figsize=(15, 10))
541.
542.     # Plot bars for precision, recall, and f1-score
543.     ax.bar(x - width, precision, width, label='Precision', color='skyblue')
544.     ax.bar(x, recall, width, label='Recall', color='orange')
545.     ax.bar(x + width, f1_score, width, label='F1-Score', color='green')
546.
547.     # Add labels, title, and custom x-axis tick labels
548.     ax.set_xlabel('Class')
549.     ax.set_ylabel('Score (%)')
550.     ax.set_title('Precision, Recall, and F1-Score for Each Class (Percentage)')
551.     ax.set_xticks(x)
552.     ax.set_xticklabels(class_names, rotation=45, ha='right')
553.     ax.legend()
554.
555.     fig.tight_layout()
556.     plt.show()
557.
558. # Assuming you have already generated the classification report
559. classification_report_str = generate_classification_report(best_model, patch_loader)
560.
561. # Plot metrics
```

```
562. plot_metrics_bar_chart(classification_report_str, class_names)
```