

**IMPLEMENTASI METODE *DEEP LEARNING* MENGGUNAKAN
ALGORITMA YOLOV5 DAN *DEEPSORT* UNTUK
MENGHITUNG KENDARAAN PADA CCTV DALAM KONDISI
PENCAHAYAAN RENDAH STUDI KASUS DI LINGKUNGAN
UNIVERSITAS BRAWIJAYA**

SKRIPSI

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun oleh:
Firman Afrialdy
NIM: 205150200111034



PROGRAM STUDI TEKNIK INFORMATIKA
DEPARTEMEN TEKNIK INFORMATIKA
FAKULTAS ILMU KOMPUTER
UNIVERSITAS BRAWIJAYA
MALANG
2024

PENGESAHAN

IMPLEMENTASI METODE *DEEP LEARNING* MENGGUNAKAN ALGORITMA YOLOV5
DAN *DEEPSORT* UNTUK MENGHITUNG KENDARAAN PADA CCTV DALAM KONDISI
PENCAHAYAAN RENDAH STUDI KASUS DI LINGKUNGAN UNIVERSITAS BRAWIJAYA

SKRIPSI

Diajukan untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun Oleh :
Firman Afrialdy
NIM: 205150200111034

Skripsi ini telah diuji dan dinyatakan lulus pada
26 April 2024
Telah diperiksa dan disetujui oleh:

Dosen Pembimbing I



Rizal Setya Perdana, S.Kom., M.Kom., Ph.D.
NIK: 2016039101181001

Dosen Pembimbing 2



Dr. Candra Dewi, S.Kom., M.Sc.
NIP: 197711142003122001

Mengetahui

Ketua Departemen Teknik Informatika



Achmad Basuki, S.T. M.MG., Ph.D.
NIP: 197411182003121002

PERNYATAAN ORISINALITAS

Saya menyatakan dengan sebenar-benarnya bahwa sepanjang pengetahuan saya, di dalam naskah skripsi ini tidak terdapat karya ilmiah yang pernah diajukan oleh orang lain untuk memperoleh gelar akademik di suatu perguruan tinggi, dan tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang lain, kecuali yang secara tertulis disitasi dalam naskah ini dan disebutkan dalam daftar referensi.

Apabila ternyata didalam naskah skripsi ini dapat dibuktikan terdapat unsur-unsur plagiasi, saya bersedia skripsi ini digugurkan dan gelar akademik yang telah saya peroleh (sarjana) dibatalkan, serta diproses sesuai dengan peraturan perundang-undangan yang berlaku (UU No. 20 Tahun 2003, Pasal 25 ayat 2 dan Pasal 70).

Malang, 26 April 2024



Firman Afrialdy

NIM: 205150200111034

PRAKATA

Puji syukur ke hadirat Allah SWT, yang telah melimpahkan rahmat, taufik, serta hidayah-Nya sehingga laporan skripsi yang berjudul “Implementasi Metode Deep Learning Menggunakan Algoritma YOLOv5 dan DeepSORT untuk Menghitung Kendaraan pada CCTV dalam Kondisi Pencahayaan Rendah Studi Kasus di Lingkungan Universitas Brawijaya” ini dapat terselesaikan.

Segala rintangan, tantangan, dan hambatan yang penulis hadapi selama proses penulisan berhasil diatasi dan dihadapi berkat dukungan yang diberikan oleh berbagai pihak. Penulis ingin menyampaikan rasa terima kasih kepada pihak-pihak yang telah memberikan dukungan, sebagai berikut.

1. Bapak Rizal Setya Perdana, S.Kom., M.Kom., Ph.D. selaku dosen pembimbing I yang telah mengusulkan topik skripsi, membimbing penulis dari awal hingga akhir dengan sabar hingga penulis dapat menyelesaikan skripsi ini.
2. Ibu Dr. Candra Dewi, S.Kom., M.Sc. selaku dosen pembimbing II yang telah membimbing penulis dengan sabar sehingga penulis dapat menyelesaikan skripsi ini.
3. Kedua orang tua penulis, Bapak Sutikno dan Ibu Ely Fitrina, atas dukungannya, sehingga penulis mampu melewati berbagai ujian yang ditempuh dalam pengerjaan skripsi.
4. Kakak-kakak penulis, atas segala bantuan dan dukungannya.
5. Teman-teman dekat seperjuangan penulis yang sama-sama menempuh skripsi di semester akhir ini atas segala bantuan dan dukungannya.
6. Teman-teman mahasiswa Teknik Informatika angkatan 2020 yang telah menjadi motivasi dan pendukung bagi penulis.
7. Bapak Eko Sakti Pramukantoro, S.Kom., M.Kom., Ph.D. selaku Ketua Program Studi Teknik Informatika Fakultas Ilmu Komputer Universitas Brawijaya.
8. Bapak Achmad Basuki, S.T., M.Mg., Ph.D. selaku Ketua Departemen Teknik Informatika Fakultas Ilmu Komputer Universitas Brawijaya.
9. Bapak Wayan Firdaus Mahmudy, S.Si., M.T., Ph.D. selaku Dekan Fakultas Ilmu Komputer Universitas Brawijaya.
10. Keluarga besar Fakultas Ilmu Komputer Universitas Brawijaya yang telah menyediakan bantuan baik secara langsung maupun tidak langsung selama penulisan skripsi ini.

Malang, 26 April 2024



Firman Afrialdy

firmen@student.ub.ac.id

ABSTRAK

Firman Afrialdy, Implementasi Metode *Deep Learning* Menggunakan Algoritma YOLOv5 dan *DeepSORT* untuk Menghitung Kendaraan pada CCTV dalam Kondisi Pencahayaan Rendah Studi Kasus di Lingkungan Universitas Brawijaya.

Pembimbing: Rizal Setya Perdana, S.Kom., M.Kom., Ph.D. dan Dr. Candra Dewi, S.Kom., M.Sc.

CCTV telah diterapkan untuk memantau berbagai aktivitas di lingkungan Universitas Brawijaya, termasuk lalu lintas kendaraan di gerbang kampus. Pengawasan pada malam hari dalam kondisi intensitas cahaya yang rendah merupakan tantangan tersendiri dalam penggunaan CCTV. Hal ini dikarenakan kualitas gambar yang rendah sehingga menghambat kemampuan sistem untuk mendeteksi dan mengidentifikasi objek dengan tepat. Salah satu permasalahan yang timbul dalam kasus kurangnya pencahayaan adalah munculnya flare atau kesilauan yang disebabkan oleh lampu kendaraan yang mengarah langsung ke CCTV. Oleh karena itu, pada penelitian ini digunakan model deteksi objek menggunakan metode deep learning dengan YOLOv5 dan *DeepSORT*. Selain itu, digunakan segmentasi U-Net dan restorasi inpaint untuk preproses data sebelum dilakukan deteksi objek menggunakan framework YOLOv5. Hasil pengujian deteksi objek, didapatkan model yang dilatih pada gambar yang dipreproses mendapatkan hasil yang lebih unggul dengan nilai *precision* 0.942, *recall* 0.873, dan *F1 score* 0.88. Sedangkan pada pengujian pelacakan dan perhitungan objek, model yang dengan gambar tanpa preproses jauh lebih unggul. Dengan nilai rata-rata MOTA sebesar 0.697 dan MOTP sebesar 0.215 untuk evaluasi pelacakan objek. Sedangkan pada evaluasi perhitungan objek didapatkan rata-rata akurasi sebesar 0.987 dan *F1 score* sebesar 0.9935.

Kata Kunci: visi komputer, deteksi objek, pelacakan objek, perhitungan objek, *DeepSORT*, YOLOV5

ABSTRACT

Firman Afrialdy, *Implementation of Deep Learning Methods Using YOLOv5 and DeepSORT Algorithms to Count Vehicles on CCTV in Low Lighting Conditions Case Study in Brawijaya University Environment.*

Supervisors: Rizal Setya Perdana, S.Kom., M.Kom., Ph.D. **and** Dr. Candra Dewi, S.Kom., M.Sc.

CCTV has been implemented to monitor various activities within Brawijaya University, including vehicle traffic at the campus gate. Surveillance at night in low light intensity conditions is a challenge in the use of CCTV. This is due to the low image quality that hampers the system's ability to detect and identify objects correctly. One of the problems that arise in the case of lack of lighting is the appearance of flares or glare caused by vehicle lights that point directly to the CCTV. Therefore, in this research, an object detection model using deep learning method with YOLOv5 and DeepSORT is used. In addition, U-Net segmentation and inpaint restoration are used to preprocess data before object detection using the YOLOv5 framework. The results of object detection testing, it was found that the model trained on preprocessed images obtained superior results with a precision value of 0.942, recall 0.873, and F1 score 0.88. While in object tracking and calculation tests, models with images without preprocessing are far superior. With an average MOTA value of 0.697 and MOTP of 0.215 for object tracking evaluation. While the object calculation evaluation obtained an average accuracy of 0.987 and F1 score of 0.9935.

Keywords: computer vision, object detection, object tracking, object counting, DeepSORT, YOLOv5

DAFTAR ISI

PENGESAHAN	i
PERNYATAAN ORISINALITAS.....	ii
PRAKATA.....	iii
ABSTRAK.....	iv
ABSTRACT	v
DAFTAR ISI	vi
DAFTAR GAMBAR	ix
DAFTAR TABEL	xi
DAFTAR KODE PROGRAM.....	xiii
BAB 1 PENDAHULUAN.....	1
1.1 Latar Belakang	1
1.2 Rumusan Masalah	2
1.3 Tujuan	2
1.4 Manfaat.....	2
1.5 Batasan Masalah.....	2
1.6 Sistematika Pembahasan	3
BAB 2 LANDASAN KEPUSTAKAAN	5
2.1 Kajian Pustaka.....	5
2.2 <i>Object Detection</i> dan <i>Object Tracking</i>	7
2.3 YOLOv5	8
2.3.1 Convolutional Neural Network (CNN)	10
2.3.2 BackBone Module.....	11
2.3.3 Neck Module	12
2.3.4 Head Module	12
2.4 <i>DeepSORT</i>	12
2.5 U-Net.....	14
2.6 <i>Inpaint</i>	16
2.7 Evaluasi.....	18
BAB 3 METODOLOGI	21
3.1 Tipe Penelitian	21
3.2 Strategi Penelitian.....	21

3.2.1	Dataset	22
3.2.2	Instrumen yang Digunakan	23
3.2.2.1	Perangkat Keras.....	23
3.2.2.2	Perangkat Lunak	23
3.2.3	Artefak yang Dihasilkan	24
BAB 4	PERANCANGAN ALGORITMA	25
4.1	Alur Utama Algoritma	25
4.1.1	Alur Implementasi Preprocessing Model Machine Learning	25
4.1.2	Alur Pelatihan Model <i>Preprocessing</i> Segementasi Gambar	26
4.1.3	Alur Perhitungan Kendaraan	28
4.1.4	Alur Deteksi Objek	29
4.1.5	Alur Pelatihan Model Deteksi Objek.....	30
4.2	Perhitungan Manual	31
4.2.1	Perhitungan Manual <i>Training U-Net</i>	31
4.2.1.1	Perhitungan Manual Lapisan Konvolusi.....	32
4.2.1.2	Perhitungan Manual Fungsi Aktivasi ReLU	34
4.2.1.3	Perhitungan Manual Lapisan Max Pooling	35
4.2.1.4	Perhitungan Manual Lapisan Up Convolution	36
4.2.1.5	Perhitungan Manual Fungsi Aktivasi Sigmoid.....	37
4.2.1.6	Perhitungan Manual Fungsi Loss BCE (Binary Cross Entropy) .	38
4.2.1.7	Perhitungan Manual Propagasi Mundur	39
4.2.2	Perhitungan Manual Proses <i>Inpaint</i>	40
4.2.3	Perhitungan Manual Training YOLOv5.....	41
4.2.3.1	Perhitungan Manual Lapisan ConvBNSiLU	43
4.2.3.2	Perhitungan Manual Fungsi Loss.....	46
4.2.3.3	Perhitungan Manual Propagasi Mundur	47
4.2.4	Perhitungan Manual Kalman Filter Predict	47
4.2.5	Perhitungan Manual Kalman Filter Project	49
4.2.6	Perhitungan Manual Kalman Filter Update	51
4.2.7	Perhitungan Manual Matriks Biaya <i>Matching Cascade</i>	52
4.2.8	Perhitungan Manual <i>IoU Match</i>	54
4.3	Perancangan Pengujian.....	57
BAB 5	IMPLEMENTASI	58

5.1	Implementasi Seleksi <i>Frame</i>	58
5.2	Implementasi Anotasi Gambar	59
5.3	Implementasi Dataset Segmentasi <i>flare</i>	64
5.4	Implementasi Struktur Direktori	66
5.5	Implementasi Model U-Net.....	67
5.6	Implementasi Dataloader untuk Pelatihan Model U-Net.....	70
5.7	Implementasi Pelatihan Model, Validasi Model dan Penyimpanan Bobot Pelatihan U-Net.....	71
5.8	Implementasi Persiapan Data Pelatihan Deteksi Objek	74
5.9	Implementasi Pelatihan dan Validasi Model Deteksi Objek YOLOv5	75
5.10	Implementasi Persiapan Pelacakan Objek	77
5.11	Implementasi Fungsi Inferensi.....	82
BAB 6	PENGUJIAN DAN ANALISIS	87
6.1	Pengujian Deteksi Objek	87
6.1.1	Deteksi Objek dengan Gambar tanpa Preproses	87
6.1.2	Deteksi Objek dengan Gambar dengan Preproses.....	88
6.2	Pengujian Pelacakan Objek	91
6.3	Pengujian Perhitungan Jumlah Objek.....	93
BAB 7	PENUTUP	96
7.1	Kesimpulan	96
7.2	Saran	96
	DAFTAR REFERENSI	98

DAFTAR GAMBAR

Gambar 2.1 <i>Object Detection</i>	8
Gambar 2.2 Arsitektur Algoritma YOLOv5	9
Gambar 2.3 <i>Framework</i> dari <i>DeepSORT</i>	12
Gambar 2.4 Arsitektur U-Net	15
Gambar 2.5 Penggunaan Metode <i>Inpaint</i>	17
Gambar 3.1 Diagram Operasional Penelitian	21
Gambar 3.2 Salah satu <i>frame</i> dari <i>video CCTV</i>	23
Gambar 4.1 Alur Utama Algoritma	25
Gambar 4.2 Alur Implementasi Preprocessing Gambar	26
Gambar 4.3 Alur Pelatihan Model Segmentasi Gambar	27
Gambar 4.4 Alur perhitungan kendaraan	29
Gambar 4.5 Alur Deteksi Objek	30
Gambar 4.6 Alur Pelatihan Model Deteksi Objek.....	31
Gambar 4.7 Sampel gambar original (a),	32
Gambar 4.8 <i>Ground Truth</i> dari Gambar 4.7 (b).....	32
Gambar 4.9 Sample Gambar percobaan perhitungan manual	41
Gambar 4.10 Perubahan Ukuran Gambar	42
Gambar 5.1 Tampilan Awal dari Label Studio	60
Gambar 5.2 Pembuatan <i>Project</i> baru	60
Gambar 5.3 Tampilan <i>Data Import</i>	61
Gambar 5.4 Tampilan <i>Labeling Setup</i>	61
Gambar 5.5 Inisiasi Kelas Pelabelan	62
Gambar 5.6 Tamplan pada Projek yang Telah dibuat	62
Gambar 5.7 Tampilan Tempat Anotasi Gambar.....	63
Gambar 5.8 Anotasi Objek	63
Gambar 5.9 Pemilihan Format <i>Export</i>	64
Gambar 5.10 Struktur Direktori Data <i>Preprocessing Learning</i>	67
Gambar 5.11 Arsitektur Data Pelatihan Model YOLOv5.....	74
Gambar 6.1 Nilai <i>Loss</i> pada Pelatihan Model YOLOv5 pada Dataset tanpa Dipreproses	87
Gambar 6.2 Kurva <i>F1 score</i> Validasi Data Gambar tanpa Preproses.....	88

Gambar 6.3 Nilai <i>Loss</i> pada Pelatihan Model YOLOv5 pada Dataset yang Dipreproses	89
Gambar 6.4 Kurva <i>F1 score</i> Validasi Data Gambar dengan Preproses	89
Gambar 6.6 Hasil Video Inferensi Pengujian Tanpa Menggunakan Metode Preproses Gambar	93

DAFTAR TABEL

Tabel 2.1 Tabel Kajian Pustaka	6
Tabel 4.1 Pixel Chanel B Gambar 4.7	32
Tabel 4.2 Pixel Chanel G Gambar 4.7.....	33
Tabel 4.3 Pixel Chanel R Gambar 4.7	33
Tabel 4.4 Bobot Kernel 1 Channel R	33
Tabel 4.5 Bobot Kernel 1 Channel G	33
Tabel 4.6 Bobot Kernel 1 Channel B	34
Tabel 4.7 Bobot <i>Channel</i> pertama konvolusi	34
Tabel 4.8 Bobot <i>Channel</i> Pertama Setelah Fungsi Aktivasi ReLU	35
Tabel 4.9 Hasil <i>Max Pooling</i> pada Bobot <i>Channel</i> Pertama.....	35
Tabel 4.10 Bobot Salah Satu Chanel Percobaan <i>Up-Conv</i>	36
Tabel 4.11 Bobot Kernel pada <i>up-conv</i>	36
Tabel 4.12 Bobot Hasil Perhitungan <i>Up-conv</i>	37
Tabel 4.13 Bobot Percobaan Perhitungan Sigmoid	37
Tabel 4.14 Hasil Perhitungan Fungsi Aktivasi Sigmoid	38
Tabel 4.15 Piksel <i>Ground Truth</i>	38
Tabel 4.16 Lapisan Konvolusi	39
Tabel 4.17 Bobot Kernel.....	39
Tabel 4.18 Piksel Gambar <i>channel</i> B Percobaan Perhitungan <i>Inpaint</i>	40
Tabel 4.19 Piksel Gambar 4.7	40
Tabel 4.20 Pixel Chanel B Gambar 4.20	42
Tabel 4.21 Pixel Chanel G Gambar 4.21.....	43
Tabel 4.22 Pixel Chanel R Gambar 4.22	43
Tabel 4.23 Bobot Kernel 1 Channel R	44
Tabel 4.24 Bobot Kernel 1 Channel G	44
Tabel 4.25 Bobot Kernel 1 Channel B	44
Tabel 4.26 Bobot Chanel Pertama pada Layer Konvolusi	45
Tabel 4.27 Bobot <i>Chanel</i> Pertama Setelah <i>Batch Normalization</i>	45
Tabel 4.28 Bobot <i>Chanel</i> Pertama Setelah Fungsi Aktivasi SiLU	46
Tabel 4.29 Chanel Indeks 0.....	47
Tabel 4.30 Tabel Evaluasi Deteksi Objek.....	57

Tabel 4.31 Tabel Evaluasi Pelacakan Objek.....	57
Tabel 4.32 Tabel Evaluasi Perhitungan Objek	57
Tabel 6.1 Tabel Pengujian Deteksi Objek.....	90
Tabel 6.2 Hasil Pengujian Pelacakan Objek.....	92
Tabel 6.3 Hasil Pengujian Perhitungan Objek	95

DAFTAR KODE PROGRAM

Kode Program 5.1 Seleksi Frame	58
Kode Program 5.2 <i>Script</i> Instalasi dan Menjalankan Label Studio pada Docker ..	59
Kode Program 5.3 Kode <i>Preprocess</i> Dataset <i>Flare</i>	64
Kode Program 5.4 <i>Script</i> Penyusunan Direktori.....	66
Kode Program 5.5 Kode Model U-NET.....	67
Kode Program 5.6 Kode Penyusunan Dataloader untuk Pelatihan Model U-Net	70
Kode Program 5.7 Kode Pelatihan, Validasi, dan Penyimpanan Model U-Net.....	72
Kode Program 5.8 Penulisan <i>Script</i> File YAML	75
Kode Program 5.9 <i>Script</i> Instalasi Algoritma YOLOv5.....	75
Kode Program 5.10 <i>Script</i> Pelatihan Model YOLOv5	76
Kode Program 5.11 <i>Script</i> Pelatihan Model YOLOv5	76
Kode Program 5.12 <i>Script</i> Instalasi <i>Library DeepSORT</i>	77
Kode Program 5.13 Kode Memanggil Kelas <i>DeepSORT</i> dari <i>Library</i> ‘deep_sort_realtime’	77
Kode Program 5.14 Kode Pembuatan Kelas untuk Deteksi Objek dengan YOLO .	78
Kode Program 5.15 Kode Pembuatan Kelas untuk <i>Preprocessing Frame</i>	79
Kode Program 5.16 Implementasi Kode Fungsi Inferensi.....	82

BAB 1 PENDAHULUAN

1.1 Latar Belakang

Pada beberapa tahun terakhir, perkembangan teknologi telah membawa perubahan signifikan dalam cara kita memandang keamanan dan pengawasan. *Closed Circuit Television* (CCTV) merupakan salah satu teknologi yang telah masuk ke berbagai aspek kehidupan, dari lingkungan pribadi hingga lingkungan publik. CCTV adalah kamera yang digunakan untuk memantau, merekam, dan mengawasi kondisi suatu lokasi untuk tujuan keamanan (Pangestu, 2022). Penggunaan CCTV tidak hanya terbatas pada pengawasan di rumah atau perkantoran, tetapi juga telah merambah ke lingkungan yang lebih luas, seperti area jalan dan kampus universitas.

Pengawasan visual terutama pada area jalan dan area masuk, telah menjadi fokus penting dalam upaya menjaga keamanan dan efisiensi. Di Universitas Brawijaya, misalnya, CCTV telah diterapkan untuk memantau berbagai aktivitas di lingkungan kampus, termasuk lalu lintas kendaraan yang masuk dan keluar dari gerbang masuk kampus. Pengawasan pada malam hari dan dalam kondisi kurangnya intensitas cahaya yang merupakan tantangan tersendiri dalam penggunaan CCTV. Kualitas gambar yang rendah pada malam hari dapat menghambat kemampuan sistem untuk mendekripsi dan mengidentifikasi objek dengan tepat, seperti kendaraan pada gerbang masuk universitas. Salah satu permasalahan yang timbul dalam kasus kurangnya pencahayaan adalah munculnya *flare* atau kesilauan yang disebabkan oleh lampu kendaraan yang mengarah langsung ke arah CCTV sehingga pengenalan objek dapat terganggu. Oleh karena itu, diperlukan solusi yang efisien dan cepat dalam pemrosesan data CCTV, sehingga informasi mengenai pergerakan dan identifikasi kendaraan dapat diperoleh dengan tepat, mendukung efektivitas pengawasan serta responsivitas tindakan yang dibutuhkan.

Untuk mengatasi tantangan mendapatkan informasi pergerakan dan identifikasi kendaraan, pada penelitian ini dikembangkan model deteksi objek pada CCTV di Universitas Brawijaya menggunakan metode deep learning dengan YOLOv5 dan *DeepSORT*. YOLOv5 adalah merupakan versi ke-5 dari algoritma YOLO (*you only look once*) yang dikembangkan oleh tim Ultralytics. algoritma YOLO merupakan algoritma yang dikembangkan untuk tantangan deteksi objek (Jocher et al., 2022). Sedangkan *DeepSORT* merupakan gabungan dari deep learning dan SORT (Simple Online and Realtime Tracking). Algoritma *DeepSORT* ini digunakan untuk melakukan deteksi objek berjumlah ganda pada aplikasi yang berjalan secara online dan real time (Bewley et al., 2016). Penerapan kombinasi dari metode YOLOv5 dengan *DeepSORT* sudah banyak dilakukan khususnya pada kasus deteksi objek untuk mengidentifikasi kendaraan dan manusia di jalanan. Salah satu contoh penelitiannya adalah yang dilakukan oleh Kutlimuratov dan teman-teman (2023) dengan membuat sebuah sistem penghitung jumlah kendaraan yang

ada pada lingkungan Tashkent dan berhasil mendapatkan hasil rata-rata akurasi dalam perhitungan jumlah kendaraan sebesar 98.1%. Penelitian ini diharapkan dapat memberikan kontribusi dalam pengembangan sistem pengawasan lalu lintas yang lebih canggih dan efisien di lingkungan kampus.

1.2 Rumusan Masalah

Rumusan masalah yang diangkat pada penelitian ini adalah

1. Bagaimana pengaruh dari penggunaan metode pemrosesan citra pada performa deteksi objek dengan menggunakan metode *deep learning* YOLOv5 dalam kondisi pencahayaan rendah?
2. Bagaimana pengaruh dari penggunaan metode pemrosesan citra pada performa pelacakan objek dengan menggunakan metode algoritma *DeepSORT* dalam kondisi pencahayaan rendah?
3. Bagaimana pengaruh dari penggunaan metode pemrosesan citra pada performa perhitungan objek dengan menggunakan metode *deep learning* YOLOv5 sebagai deteksi objek dan metode algoritma *DeepSORT* sebagai pelacakan objek dalam kondisi pencahayaan rendah?

1.3 Tujuan

Tujuan diangkatnya penelitian ini adalah:

1. Untuk mengetahui pengaruh dari penggunaan metode pemrosesan citra pada performa deteksi objek dengan menggunakan metode *deep learning* YOLOv5 dalam kondisi pencahayaan rendah.
2. Untuk mengetahui pengaruh dari penggunaan metode pemrosesan citra pada performa pelacakan objek dengan menggunakan metode algoritma *DeepSORT* dalam kondisi pencahayaan rendah.
3. Untuk mengetahui pengaruh dari penggunaan metode pemrosesan citra pada performa perhitungan objek dengan menggunakan metode *deep learning* YOLOv5 sebagai deteksi objek dan metode algoritma *DeepSORT* sebagai pelacakan objek dalam kondisi pencahayaan rendah.

1.4 Manfaat

Manfaat yang diharapkan pada penelitian ini adalah:

1. Dapat mengoptimalkan penggunaan CCTV di lingkungan Universitas Brawijaya.
2. Hasil dari penelitian ini dapat menjadi referensi bagi penelitian dan pengembangan selanjutnya di bidang deteksi objek, pelacakan objek, dan perhitungan objek terkhususnya pada kasus kurangnya intensitas cahaya.

1.5 Batasan Masalah

Batasan masalah pada penelitian ini antara lain:

1. Data latih yang digunakan adalah video dari CCTV Universitas Brawijaya pada gerbang veteran pada tanggal 22 Mei 2023 pukul 17:00 hingga 21:00.
2. Objek yang dideteksi adalah motor dan mobil.
3. Kendaraan yang dihitung adalah kendaraan yang masuk dari gerbang veteran.
4. Teknik *preprocessing* yang digunakan adalah *image segmentation* dengan arsitektur U-Net dan metode *inpainting* untuk perbaikan area kesilauan pada gambar hasil segmentasi U-Net.
5. Bobot *pre-trained* (telah dilatih) yang digunakan pada model YOLOv5 adalah bobo yolov5s atau ukuran *small*.

1.6 Sistematika Pembahasan

Sistematika pembahasan dalam penyusunan laporan ini bertujuan untuk memberikan gambaran dan penjelasan dari penelitian ini. Laporan penelitian ini terdiri dari beberapa bab antara lain, sebagai berikut:

BAB 1 PENDAHULUAN

Bab ini memuat latar belakang, rumusan masalah, tujuan, manfaat, batasan masalah dan sistematika pembahasan dari penelitian.

BAB 2 LANDASAN KEPUSTAKAAN

Bab ini memuat dasar teori yang berhubungan dengan topik penelitian dan kajian pustaka dari penelitian-penelitian terdahulu tentang *object tracking*, algoritma YOLOv5, dan algoritma *DeepSORT* yang dapat menguatkan dasar teori.

BAB 3 METODOLOGI

Bab ini menjelaskan urutan aktivitas yang dilakukan serta penjelasannya secara lengkap oleh peneliti dalam melakukan penelitian.

BAB 4 PERANCANGAN ALGORITMA

Bab ini menjelaskan mengenai alur perancangan algoritma yang digunakan dalam penelitian. Bab ini juga menyertakan contoh perhitungan manual dari metode yang digunakan

BAB 5 IMPLEMENTASI

Bab ini menjelaskan mengenai implementasi kode program dalam penelitian untuk membangun sistem pelacakan kendaraan masuk pada studi kasus gerbang masuk veteran Universitas Brawijaya pada malam hari.

BAB 6 PENGUJIAN DAN ANALISIS

Bab ini menjelaskan hasil pengujian yang didapatkan dari proses implementasi beserta analisis dari hasil pengujian tersebut.

BAB 7 PENUTUP

Bab ini berisi kesimpulan yang diambil setelah melakukan penelitian dengan mengacu pada rumusan masalah, serta saran untuk penelitian selanjutnya.

BAB 2 LANDASAN KEPUSTAKAAN

2.1 Kajian Pustaka

Landasan pustaka pada penelitian ini membahas perbandingan antara penelitian ini dengan penelitian terdahulu. Pada penelitian Guo dan Xu (2022), penelitian ini menggunakan YOLOv5 untuk mendeteksi objek, *DeepSORT* untuk melakukan tracking(pelacakan), dan IPM(*Inverse perspective transformation*) untuk melakukan perhitungan kecepatan laju sebuah kendaraan. Dataset berupa video dari sebuah drone pada arus lalu lintas yang berkecepatan tinggi, yang menghasilkan 11.000 gambar kendaraan yang diekstraksi dan 150 diantaranya dijadikan data test. Hasil dari eksperimen ini menunjukkan rata-rata akurasi berjumlah 96% untuk deteksi kendaraan dan 98.36% rata-rata akurasi untuk pengukuran kecepatan kendaraan. Letak perbedaan dengan penelitian ini ada pada masalah yang diteliti dan jenis dataset yang digunakan. Dataset yang digunakan pada penelitian ini berupa rekaman CCTV dan dalam kondisi pencahayaan rendah atau bisa disebut dalam kondisi malam. Sedangkan dalam metode yang digunakan bisa dilihat cukup mirip dengan penelitian Guo dan Xu (2022) yang menggunakan algoritma YOLOv5 untuk melakukan objek deteksi dan algoritma *DeepSORT* untuk melakukan pelacakan objek.

Pada penelitian Wei (2023), penelitian ini menggunakan YOLOv5 untuk melakukan deteksi kendaraan dan *DeepSORT* untuk melakukan pelacakan kendaraan. Perbedaannya terletak pada data yang digunakan dan hasil dianalisis dengan penarikan kesimpulan kualitatif bukan kuantitatif. Data yang digunakan pada penelitian ini adalah data bayer. data bayer sendiri merupakan sebuah format gambar dari kamera digital menggunakan rangkaian filter warna bayer (CFA). warna filter terdiri dari hijau dan merah bergantian dalam satu set, dan filter hijau dan biru di set lainnya. Setiap piksel pada sensor hanya dapat menangkap salah satu dari tiga komponen warna: Merah (R), Hijau (G), dan Biru (B). Hasil dari penelitian ini diperoleh bahwa gambar bayer dapat digunakan dalam pelacakan objek. Letak perbedaan dengan penelitian ini ada pada jenis dataset yang digunakan. Dataset pada penelitian ini adalah data bayer.

Penelitian lainnya yang menggunakan metode yang sama seperti Wei (2023) dan Guo dan Xu (2022) namun dengan tujuan penelitian berbeda berupa klasifikasi pelanggaran kendaraan pada video rekaman CCTV yang melewati garis jalan. Penelitian ini dilakukan oleh Wu, Ge, dan Zhan (2023) yang mendapatkan hasil akurasi dengan rata-rata sebesar 99.5%.

Kemudian pada penelitian yang dilakukan oleh Kutlimuratov dan teman-teman (2023) yang bertujuan mengembangkan sistem real-time yang akurat dan handal untuk menghitung jumlah kendaraan guna mengurangi kemacetan lalu lintas di area yang ditentukan. Penelitian tidak hanya dilakukan pada siang hari saja namun juga pada keadaan kurangnya pencahayaan pada kasus malam hari dalam keadaan bergerimis. Video yang digunakan untuk melatih model adalah

tangkapan CCTV secara *real time* atau waktu nyata. Dari 10 *test* video, didapatkan rata-rata akurasi sebesar 98.1% dengan menggunakan metode YOLOv5 dan algoritma *DeepSORT*. Perbedaan dari penelitian ini adalah lokasi penelitian yang mana penelitian yang dilakukan oleh Kutlimuratov dan teman-teman dilaksanakan pada lokasi Tashkent.

Penelitian terakhir yang menjadi kajian pustaka pada penelitian ini adalah penelitian yang dilakukan oleh Lin dan teman-teman (2023). Penelitian ini melakukan improvisasi pada algoritma YOLOv5 yang disebut dengan YOLOv5+DSC. DSC merujuk kepada *Depth Separable Convolution* yang merupakan jenis operasi konvolusi yang memecah konvolusi standar menjadi dua operasi terpisah. Yaitu konvolusi kedalaman (*depthwise convolution*) dan konvolusi titik (*pointwise convolution*). Tujuan penggabungan metode YOLOv5 dengan DSC adalah untuk meningkatkan proses deteksi kendaraan, mengurangi jumlah parameter dan komputasi sambil tetap menjaga akurasi dan kecepatan. Hasil akurasi akhir pada penelitian ini adalah 0.980 dengan menggunakan YOLOv5_DSC dan 0.981 dengan menggunakan YOLOv5, Namun mengurangi jumlah parameter sebesar 23,5%, jumlah komputasi sebesar 32,3%, ukuran bobot model sebesar 20%, dan meningkatkan kecepatan pemrosesan rata-rata setiap gambar sebesar 18,8%. Penelitian dari Lin dan teman-temannya lebih fokus terhadap optimalisasi dan pengembangan dari algoritma YOLOv5 agar lebih ringan dari ukuran namun mengurangi akurasi seminimal mungkin atau bisa meningkatkan akurasinya.

Bentuk ringkasan dan hasil dari studi literatur yang telah dilakukan penulis dapat dilihat pada tabel 2.1.

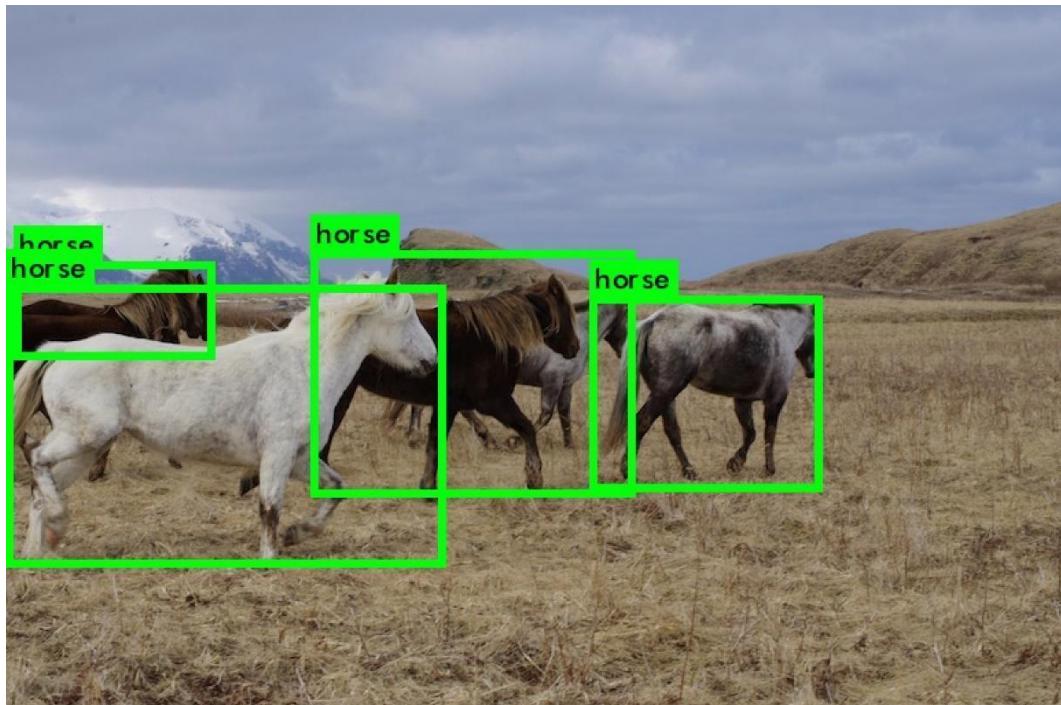
Tabel 2.1 Tabel Kajian Pustaka

No.	Pustaka	Object Penelitian	Metode	Hasil
1.	Guo dan Xu (2022)	Video dari drone diambil secara aerial untuk melakukan perhitungan dan perkiraan kecepatan kendaraan.	YOLOv5 dan <i>DeepSORT</i>	Akurasi berjumlah 96% untuk deteksi kendaraan dan 98.36% rata-rata akurasi untuk pengukuran kecepatan kendaraan.
2.	Wei (2023)	Data Bayer	YOLOv5 dan <i>DeepSORT</i>	Hasil akurasi tidak dijelaskan secara eksplisit

3.	Wu, Ge, dan Zhang (2023)	Video kendaraan lalu lintas untuk mendeteksi kendaraan melanggar marka jalan.	YOLOv5 dan <i>DeepSOR T</i>	Akurasi 99.5% dalam klasifikasi pelanggaran pada kendaraan yang melewati garis jalan.
4.	Kutlimuratov dan teman-teman (2023)	Perhitungan jumlah kendaraan pada video CCTV di lingkungan Jalan Tashkent pada siang hari dan malam dengan gerimis.	YOLOv5 dan <i>DeepSOR T</i>	Rata-rata akurasi sebesar 98.1%
5.	Lin dan teman-teman (2023)	Aplikasi dan evaluasi algoritma pelacakan multitarget <i>DeepSORT</i> dalam pelacakan kendaraan, khususnya kombinasi dengan algoritma YOLOv5s_DSC.	YOLOv5_DSC dan <i>DeepSOR T</i>	Akurasi 0.980 dengan menggunakan YOLOv5_DSC dan 0.981 dengan menggunakan YOLOv5

2.2 *Object Detection* dan *Object Tracking*

Object detection atau deteksi objek merupakan salah satu dari masalah fundamental pada visi komputer. Makna dari *object detection* atau deteksi objek menurut penelitian (Zhao et al., 2019) adalah menentukan sebuah lokasi pada sebuah target didalam sebuah gambar atau *frame video* dan menentukan kategori atau kelas pada objek yang terdeteksi. Gambar 2.1 dapat menjadi contoh bagaimana hasil dari melakukan deteksi objek berupa objek ‘Kuda’ pada sebuah gambar. Tahap dalam melakukan deteksi objek pada umumnya dapat dibagi menjadi 3 tahap, yaitu: pemilihan wilayah informatif, ekstraksi fitur, dan klasifikasi.

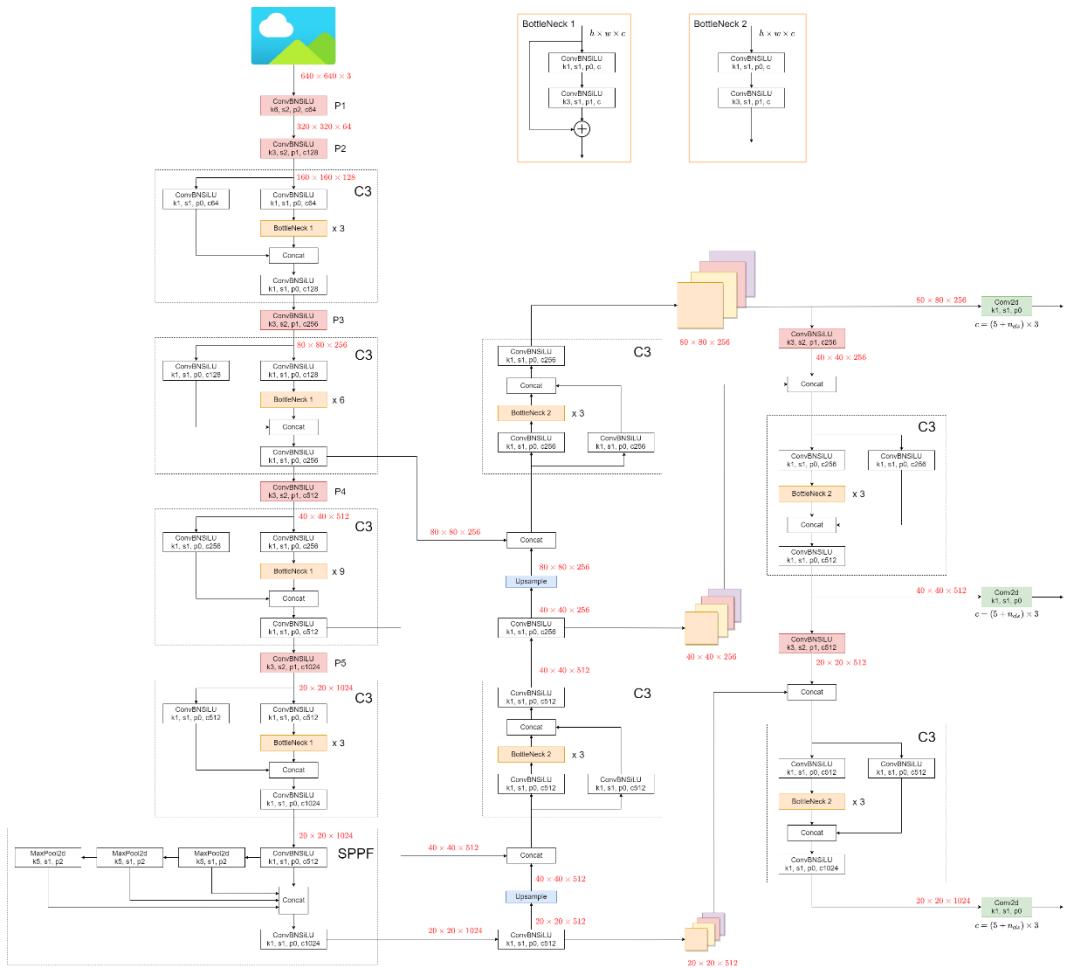


Gambar 2.1 Object Detection

Pelacakan objek atau *object tracking* adalah proses mengidentifikasi suatu wilayah yang diminati dalam urutan bingkai atau *frame*. Secara umum, proses pelacakan objek terdiri dari empat modul, termasuk inisialisasi target, pemodelan penampilan, estimasi gerakan, dan pemosisan target. Inisialisasi target adalah proses anotasi posisi objek, atau wilayah yang diminati, dengan salah satu dari representasi berikut: kotak pembatas objek, elips, pusat, kerangka objek, kontur objek, atau siluet objek (Fiaz, Mahmood and Jung, 2018).

2.3 YOLOv5

Algoritma YOLOv5 merupakan salah satu versi dari seri YOLO (You Only Look Once) versi ke-5 yang dirilis pada tahun 2020. YOLO sendiri merupakan model deteksi objek yang berbasis jaringan *Convolutional Neural Network*. YOLO memiliki pendekatan yang berbeda pada sistem deteksi objek. YOLO menyatukan komponen-komponen pada model deteksi objek menjadi sebuah single neural network. Fitur-fitur yang ada pada seluruh gambar digunakan untuk memprediksi setiap *bounding box* dan mengklasifikasikannya ke seluruh kelas secara simultan. Arsitektur dari algoritma YOLOv5 memiliki tiga bagian utama. diantaranya adalah Backbone, Neck, dan Head. Arsitektur YOLOv5 dapat dilihat pada Gambar 2.2.



Gambar 2.2 Arsitektur Algoritma YOLOv5

Keterangan parameter yang digunakan pada masing-masing layer dari arsitektur YOLOv5 yang dapat dilihat pada Tabel 2.2:

Tabel 2.2 Parameter Masing-masing Layer

No	Layer	Input Chanel	Output Chanel	Kernel	Stride	Padding
1	ConvBNSiLU(1)	3	64	6	2	2
2	ConvBNSiLU(2)	64	128	3	2	1
3	C3(1)	128	128	1	1	0
4	ConvBNSiLU(3)	128	256	3	2	1
5	C3(2)	256	256	1	1	0

6	ConvBNSiLU(4)	256	512	3	2	1
7	C3(3)	512	512	1	1	0
8	ConvBNSiLU(5)	512	1024	3	2	1
9	C3(4)	1024	1024	1	1	0
10	ConvBNSiLU(6)	1024	512	1	1	0
11	ConvBNSiLU(7)	1024	1024	1	1	0
12	MaxPool2d	-	-	5	1	2
13	ConvBNSiLU(8)	1024	512	1	1	0
14	C3_Neck(1)	1024	512	1	1	0
15	ConvBNSiLU(9)	512	256	1	1	0
16	C3_Neck(2)	512	256	1	1	0
17	ConvBNSiLU(10)	256	256	3	2	1
18	C3_Neck(3)	512	512	1	1	0
19	ConvBNSiLU(11)	512	512	3	2	1
20	C3_Neck(4)	1024	1024	1	1	0

2.3.1 Convolutional Neural Network (CNN)

Dalam model YOLOv5, *Convolutional Neural Network* (CNN) adalah komponen kunci yang digunakan untuk deteksi objek, termasuk kendaraan. CNN sendiri merupakan salah satu jaringan saraf tiruan yang biasanya digunakan dalam *deep learning* untuk menyelesaikan berbagai tugas baik itu *image* dan *video recognition*, *natural language processing*, maupun tugas lainnya (Taye, 2023). Penggunaan CNN dalam YOLOv5 memungkinkan deteksi objek yang efisien dan akurat, sehingga cocok untuk menyelesaikan tugas *computer vision*.

Pada YOLOv5, penggunaan CNN digunakan pada setiap layer dan dapat dilihat pada Gambar 2.2. dengan nama ConvBNSiLU. ConvBNSiLU merupakan urutan proses secara berurutan dari CNN, *batch normalization*, dan fungsi aktivasi SiLU.

Formulasi dari masing-masing ConvBNSiLU dapat dilihat pada persamaan (2.1), (2.4), dan (2.5).

$$h(x, y) = \left(\sum_{a=-n}^n \sum_{b=-n}^n f(a, b) \times g(x - a, y - b) \right) + Bias \quad (2.1)$$

$$\mu_i = \frac{1}{m} \sum_{i=0}^m x_{ij} \quad (2.2)$$

$$\sigma_j^2 = \frac{1}{m} \sum_{i=0}^m (x_{ij} - \mu_j)^2 \quad (2.3)$$

$$normalized = \frac{x_{ij} - \mu_j}{\sqrt{\sigma_j^2 - \epsilon}} \quad (2.4)$$

$$\hat{x}_{(0,0)} = \gamma \times normalized + bias \quad (2.5)$$

$$SiLU(x) = x \cdot \frac{1}{1 + e^{-x}} \quad (2.6)$$

Keterangan:

- $g_i(x, y)$: Bobot bernilai random yang diinisiasi saat melakukan konvolusi
- $f_i(x, y)$: Nilai pixel yang akan digunakan
- μ_i : Nilai rata-rata
- σ_j^2 : Standar deviasi
- γ : Skala
- \hat{x}_{ij} : Nilai Batch Normalisasi pada masing-masing pixel untuk setiap channel

2.3.2 BackBone Module

Modul *backbone* dalam YOLOv5, yang dikenal sebagai CSPDarknet53, adalah bagian yang mendalam dan kuat dari jaringan. Ini terdiri dari banyak lapisan konvolusi dan operasi pengolahan lainnya. Modul ini berfungsi untuk mengekstrak fitur-fitur tingkat tinggi dari gambar masukan. Proses ini mencakup konvolusi untuk mengidentifikasi pola dan fitur dalam berbagai tingkat resolusi gambar. CSPDarknet53 menggabungkan konsep *cross-stage* dan *partial channels* untuk mengoptimalkan proses ekstraksi fitur dan memitigasi masalah gradien yang tumpang tindih.

2.3.3 Neck Module

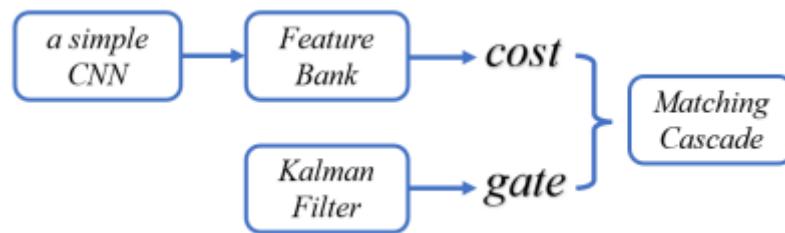
Modul *neck* dalam YOLOv5, yang disebut PANet, bertanggung jawab untuk menggabungkan dan memperkaya fitur-fitur dari berbagai skala spasial. Ini mencapai peningkatan representasi fitur dengan menyatukan informasi konteks lokal dan global. Modul ini membantu meningkatkan kemampuan model untuk mengidentifikasi objek yang memiliki skala yang berbeda dalam satu gambar.

2.3.4 Head Module

Modul *head* dalam YOLOv5 adalah bagian yang bertanggung jawab untuk membuat prediksi akhir. Ini mengambil fitur-fitur yang telah diekstraksi oleh modul *backbone* dan *neck* dan menghasilkan hasil deteksi. Modul *head* biasanya terdiri dari beberapa lapisan konvolusi, dan ini adalah tempat di mana prediksi seperti koordinat *bounding box coordinates* dan probabilitas kelas dilakukan.

2.4 DeepSORT

DeepSORT adalah algoritma tracking online yang menggunakan *framework Simple Online and Realtime Tracking* dan model CNN yang sudah dilatih terlebih dahulu. *DeepSORT* bekerja dengan metode *tracking-by-detection* yang mempertimbangkan *bounding box* hasil deteksi objek dan informasi visual objek yang sedang dilacak. Algoritma ini adalah algoritma *tracking online* yang berarti untuk membuat prediksi, algoritma ini hanya mempertimbangkan informasi citra kini dan yang di masa lalu.



Gambar 2.3 Framework dari DeepSORT

Kerangka *DeepSORT* dapat dirangkum sebagai sebuah kerangka kerja dua cabang, yaitu dengan appearance branch dan motion branch, seperti yang ditunjukkan pada Gambar 2.3. pada appearance branch, diberikan deteksi pada setiap bingkai, deskriptor deep appearance (CNN sederhana) yang telah dilatih sebelumnya pada dataset re-identifikasi MARS(cite dataset). digunakan untuk mengekstrak fitur appearance. Ini menggunakan mekanisme bank fitur untuk menyimpan fitur dari 100 bingkai terakhir untuk setiap tracklet. Saat deteksi baru masuk, jarak cosinus terkecil antara bank fitur B_i dari tracklet ke- i dan fitur f_j dari deteksi ke- j dihitung pada persamaan (2.5).

$$d(i, j) = \{1 - f_j^T f_k^i \mid f_k^i \in B_i\} \quad (2.5)$$

Keterangan:

- $d(i,j)$: Jarak antara *tracklet* i dan deteksi j.
 f_j^T : Vektor fitur dari deteksi j.
 f_k^i : Vektor fitur dari tracklet i pada bingkai k.
 B_i : Kumpulan vektor fitur dari tracklet i pada berbagai bingkai yang berbeda.

Dalam *motion branch*, algoritma filter Kalman digunakan untuk memprediksi posisi tracklet pada bingkai saat ini. Ini bekerja melalui dua tahap, yaitu prediksi keadaan (*state prediction*) dan pembaruan keadaan (*state update*). Pada tahap prediksi keadaan, algoritma ini memprediksi keadaan saat ini dapat dilihat pada persamaan (2.6) dan (2.7).

$$\hat{x}'_k = F_k \hat{x}_{k-1} \quad (2.6)$$

Keterangan:

- \hat{x}'_k : Keadaan yang diprediksi pada langkah waktu k.
 F_k : Matriks transisi keadaan pada langkah waktu k.

$$P'_k = F_k P_{k-1} F_k^T + Q_K \quad (2.7)$$

Keterangan:

- P'_k : Matriks kovariansi yang diprediksi pada langkah waktu k.
 F_k : Matriks transisi keadaan pada langkah waktu k.
 Q_K : Kovarian proses noise

Pada tahap pembaruan keadaan, gain Kalman dihitung berdasarkan kovariansi keadaan yang diestimasi P'_k dan noise observasi R_k dapat dilihat pada persamaan (2.8).

$$K = P'_k H_k^T (H_k P'_k H_k^T + R_k)^{-1} \quad (2.8)$$

Keterangan:

- K : Kalman gain matrix.
 P'_k : Matriks kovariansi yang diprediksi pada langkah waktu k.
 H_k : Matriks model observasi pada langkah waktu k.
 R_k : Matriks kovariansi noise pengukuran pada langkah waktu k.

Variabel H_k^T adalah model observasi, yang memetakan keadaan dari ruang estimasi ke ruang observasi. Kemudian, gain Kalman K digunakan untuk meng-update keadaan akhir. Persamaan yang digunakan dapat dilihat pada persamaan (2.9) dan (2.10).

$$x_k = \hat{x}'_k + K (z_k - H_K \hat{x}'_k) \quad (2.9)$$

Keterangan:

x_k : Estimasi keadaan yang diperbarui pada langkah waktu k.

\hat{x}'_k : Keadaan yang diprediksi pada langkah waktu k.

K : Kalman gain matrix.

H_K : Matriks model observasi pada langkah waktu k.

z_k : Pengukuran pada langkah waktu k.

$$P_k = (I - KH_k)P'_k \quad (2.10)$$

Keterangan:

P_k : Matriks kovariansi yang diperbarui pada langkah waktu k.

I : Matriks identitas

K : Kalman gain matrix.

H_K : Matriks model observasi pada langkah waktu k.

P'_k : Matriks kovariansi yang diprediksi pada langkah waktu k.

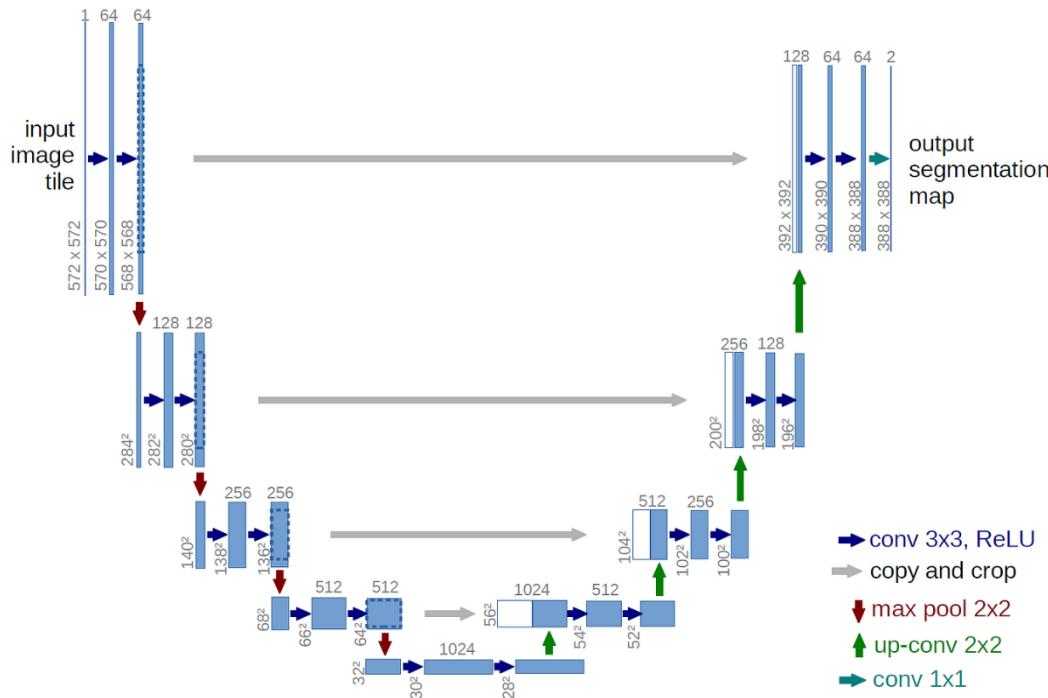
Di mana z_k adalah pengukuran pada langkah waktu k. Diberikan keadaan pergerakan *tracklet* dan deteksi yang baru datang, jarak Mahalanobis digunakan untuk mengukur ketidakmiripan spasial-temporal di antaranya. *DeepSORT* menggunakan jarak pergerakan ini sebagai filter untuk mengeluarkan asosiasi yang tidak mungkin.

Algoritma pencocokan tumpukan (*matching cascade*) diusulkan untuk menyelesaikan tugas asosiasi sebagai serangkaian submasalah daripada masalah penugasan global. Ide inti adalah memberikan prioritas pencocokan yang lebih besar kepada objek yang lebih sering terlihat. Setiap sub masalah asosiasi dipecahkan menggunakan algoritma Hungarian (Du et al., 2023).

2.5 U-Net

Arsitektur model pembelajaran mesin U-Net pertama kali diperkenalkan oleh Olaf Ronneberger, Fischer and Brox (2015) dalam melakukan segmentasi gambar terhadap citra biomedis. Arsitektur ini diberi nama U-Net dikarenakan menyerupai

gambar huruf "U" pada model yang digunakan. Arsitektur model U-Net dapat dilihat pada Gambar 2.4.



Gambar 2.4 Arsitektur U-Net

Komponen lapisan utama yang digunakan pada arsitektur ini menggunakan lapisan konvolusi dan menggunakan fungsi aktivasi ReLU. Lapisan ini bertindak sebagai *encoder* yang bertujuan untuk mengurangi dimensi citra sementara mengekstraksi fitur penting dari citra input. Pada arsitektur ini juga mengadaptasi lapisan *max poll* yang bertujuan untuk mengurangi resolusi citra dan memperluas cakupan fitur. Setelah melakukan *encoder* dengan menggunakan lapisan konvolusi dan *max pool*, maka tahapan yang dilakukan pada arsitektur ini adalah *decoder*. *Decoder* merupakan tahap yang berfungsi untuk mengembalikan resolusi citra dari masing-masing tingkat pada citra input. Hal ini dapat dicapai dengan menggunakan lapisan *upsampling*. Terakhir lapisan *output segmentation map* yang menggunakan lapisan konvolusi yang menghasilkan gambar dengan ukuran panjang dan lebar yang sama dengan input gambar pada model. Fungsi aktivasi yang digunakan pada lapisan akhir tergantung pada jumlah kelas yang akan diprediksi. Fungsi aktivasi sigmoid akan digunakan pada segmentasi biner dan fungsi aktivasi softmax akan digunakan pada segmentasi multi kelas. Formulasi yang ada pada arsitektur ini dapat dilihat pada persamaan (2.11) hingga persamaan (2.15).

$$h(x, y) = \left(\sum_{a=-n}^n \sum_{b=-n}^n f(a, b) \times g(x-a, y-b) \right) + Bias \quad (2.11)$$

Keterangan:

- $h(x, y)$: Matriks output
- $f_i(x, y)$: Input channel
- $g_i(x, y)$: Kernel
- bias : nilai bias (random pada awal inisialisasi)

$$ReLU = \max(0, x) \quad (2.12)$$

Keterangan:

jika nilai x positif, maka ReLU mengembalikan nilai x , dan jika nilai x negatif atau nol, maka ReLU mengembalikan nilai 0.

$$Upconv(X, kernel) = \sum_{i=0}^k \sum_{j=0}^k X[i, j] \cdot kernel[i, j] \quad (2.13)$$

Keterangan:

- $X[i, j]$: Matriks input
- Kernel[I,j] : kernel

$$\text{sigmoid}(x) = \frac{1}{1 + e^{-x}} \quad (2.14)$$

Keterangan:

$$e \approx 2.71828$$

$$BCE \ loss = -\frac{1}{N} \sum_{i=0}^N y \times \log(p) + (1 - y) \times \log(1 - p) \quad (2.15)$$

Keterangan:

- BCE = Binary cross entropy
- y = Ground truth
- p = prediksi

2.6 Inpaint

Inpaint merupakan salah satu teknik preprocessing gambar yang digunakan untuk melakukan perbaikan gambar (Guillemot and Meur, 2014). Menurut Yang dan teman-teman (2022), metode *inpainting* sendiri sekarang memiliki 2 jenis berupa metode tradisional dan metode pembelajaran mesin. Metode tradisional

mencakup metode berbasis difusi dan berbasis tambalan yang berfokus pada fitur tingkat rendah. Metode yang terakhir menggunakan jaringan saraf *convolutional*, terutama jaringan adversarial generatif, untuk menyimpulkan konten semantik tingkat tinggi untuk mencari daerah yang hilang. Penerapan metode ini dapat dilihat pada Gambar 2.5.



Gambar 2.5 Penggunaan Metode *Inpaint*.

Dengan menggunakan gambar *masking* sebagai penanda area yang rusak nantinya gambar original akan melakukan prediksi untuk melakukan perbaikan sehingga menghasilkan gambar yang menghilangkan efek rusak dari *area masking* yang ditunjuk. Ada banyak formulasi matematis dalam penerapan metode ini. Salah satu formulasinya dapat dilihat pada persamaan (2.15).

$$I(p) = \frac{\sum_{q \in B_\varepsilon(p)} w(p, q)[I(q) + \nabla I(q)(p - q)]}{\sum_{q \in B_\varepsilon(p)} w(p, q)} \quad (2.15)$$

Keterangan:

$I(p)$ = Hasil gambar *inpaint*

$\sum_{q \in B_\varepsilon(p)}$ = Penjumlahan dilakukan untuk setiap piksel tetangga q dalam suatu radius ε dari piksel p .

$\nabla I(q)$ = Gradien dari intensitas piksel q .

$w(p, q)$ = Fungsi bobot

Untuk fungsi bobot pada persamaan (2.15) dapat menggunakan persamaan yang mengacu pada penelitian Telea mengandung 3 Indeks yang akan diperhitungan. Persamaan dapat dilihat pada (2.16), (2.17), (2.18), dan (2.19).

$$w(p, q) = dir(p, q) \times dst(p, q) \times lev(p, q) \quad (2.16)$$

$$dir(p, q) = \frac{p - q}{\|p - q\|} \cdot N(p) \quad (2.17)$$

$$dst(p, q) = \frac{d_0}{\|p - q\|^2} \quad (2.18)$$

$$lev(p, q) = \frac{T_0}{1 + \|T(p) - T(q)\|} \quad (2.19)$$

Keterangan:

- $dir(p, q)$ = Arah dari pengaruh *inpaint* pada vektor q terhadap vektor p
- $dst(p, q)$ = Jarak dari pengaruh *inpaint* pada vektor q terhadap vektor p
- $lev(p, q)$ = Pengaruh waktu pada proses inpainting, jika piksel q berdekatan dengan informasi yang diketahui, maka pengaruhnya akan lebih besar
- $N(p)$ = Normalisasi pada nilai p
- d_0 = Nilai konstan
- T_0 = Nilai konstan
- $T(p)$ = Informasi waktu

Pada penelitian ini akan menggunakan teknik *inpaint* dalam mengurangi efek kesilauan dari cahaya pada sebuah gambar dengan tidak mengurangi kualitas dari gambar aslinya. Dengan menerapkan *masking* pada bagian yang silau dari cahaya.

2.7 Evaluasi

Penelitian ini melakukan evaluasi pada masing-masing 3 tahap. Yaitu tahap deteksi objek, pelacakan objek, dan perhitungan objek. Metrik evaluasi yang digunakan pada deteksi objek dan perhitungan objek berupa *precision*, *recall*, dan *F1 score*. *Precision* mengukur seberapa akurat model dalam memprediksi kelas positif. *Recall* mengukur seberapa banyak model mampu mengidentifikasi semua contoh positif yang sebenarnya. *F1-score* adalah metrik gabungan yang mempertimbangkan keseimbangan antara precision dan recall. Persamaan dari *precision*, *recall*, dan *F1-score* dapat dilihat pada persamaan (2.20) hingga (2.22).

$$Precision = \frac{TP}{TP + FP} \quad (2.20)$$

$$Recall = \frac{TP}{TP + FN} \quad (2.21)$$

$$F1\ score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \quad (2.22)$$

Keterangan:

- TP = *True Positive* atau keadaan kelas yang diprediksi sesuai dengan nilai kelas pada *ground truth* (nilai asli)

- FP = *False Positive* atau keadaan model memprediksi sebuah kelas pada gambar namun pada nilai *ground truth*-nya seharusnya tidak ada kelas yang terdeteksi atau model salah memprediksi kelas.
- FN = *False Negative* atau keadaan pada *ground truth* ada sebuah objek yang tersimpan namun tidak terlacak oleh model.
- TN = *True Negative* objek yang seharusnya tidak terlacak dan tidak tersimpan di *ground truth* dan tidak terdeteksi oleh model.

Untuk mendapatkan nilai TP, FP, FN, dan TN dalam kasus deteksi objek, ada sedikit penambahan persamaan untuk melihat apakah sebuah kelas yang terdeteksi masuk ke dalam salah satu variabel *confusion matrix* tersebut. Caranya adalah dengan menggunakan persamaan IoU atau singkatan dari *intersection over union*. Setelah melakukan perhitungan IoU, akan diterapkan nilai ambang batas atau *threshold* untuk menentukan apakah sebuah objek dianggap valid atau tidak terhadap area dari *ground truth* yang ada. Persamaannya dapat dilihat pada persamaan (2.23).

$$IoU = \frac{A \cap B}{A \cup B} \quad (2.22)$$

Keterangan:

- $A \cap B$ = Area tumpang tindih dari hasil area prediksi model dan area *ground truth*
- $A \cup B$ = Area gabungan pada area prediksi model dan area *ground truth*

Pada evaluasi pelacakan objek menggunakan evaluasi MOTA (*Multiple Objek Tracking Accuracy*) dan MOTP (*Multiple Objek Tracking Precision*). MOTA adalah suatu metrik evaluasi yang digunakan untuk mengukur akurasi dari sistem pelacakan objek. Sedangkan MOTP merupakan rata-rata ketidaksamaan antara semua kelas yang benar dan target *ground truth* yang sesuai (Milan et al., 2016). Persamaan MOTA dan MOTP dapat dilihat pada persamaan (2.23) dan persamaan (2.24).

$$MOTA = 1 - \frac{\sum_t (FN_t + FP_t + IDSW_t)}{\sum_t GT_t} \quad (2.23)$$

$$MOTP = \frac{\sum_{t,i} d_{t,i}}{\sum_t c_t} \quad (2.24)$$

Keterangan:

- \sum_t = Penjumlahan dilakukan sebanyak *frame* yang dievaluasi dan dimiliki pada *ground truth* yang sesuai.

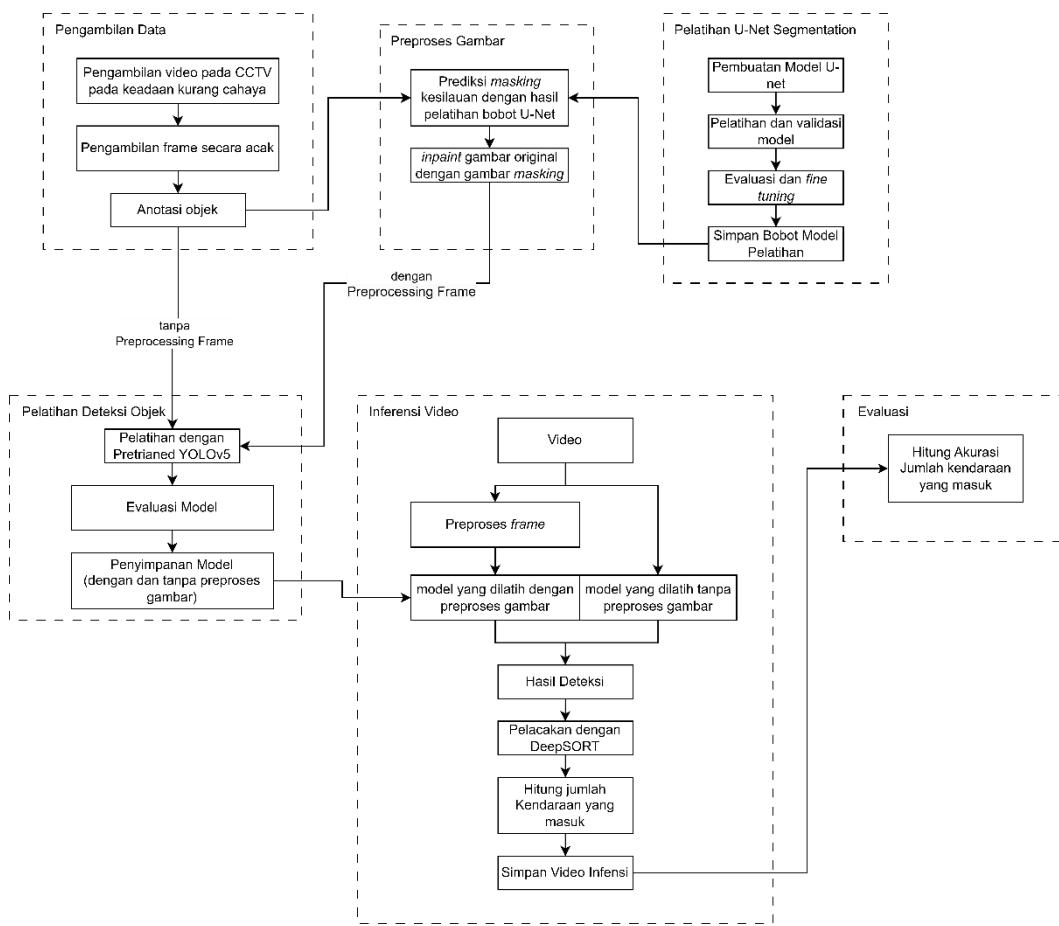
- $IDSW_t$ = Jumlah perubahan identitas yang salah atau kesalahan dalam mengaitkan suatu objek dengan objek yang sama dalam frame berikutnya.
- GT_t = Total objek sesungguhnya dalam *ground truth* (seluruh *frame*).
- $d_{t,i}$ = Tumpang tindih (IoU) area dari target i dengan objek *ground truth*-nya.
- c_t = Jumlah prediksi pelacakan yang sesuai dengan *ground truth* pada *frame* ke-t

BAB 3 METODOLOGI

3.1 Tipe Penelitian

Tipe penelitian yang diterapkan adalah analitikal non-implementatif. Dalam konteks penelitian ini, jenis penelitian ini dipilih karena fokus utama penelitian adalah pada analisis, perbandingan, dan evaluasi terhadap berbagai aspek dan parameter yang terkait dengan penghitungan kendaraan menggunakan algoritma YOLOv5 dan *DeepSORT*.

3.2 Strategi Penelitian



Gambar 3.1 Diagram Operasional Penelitian

Strategi penelitian yang digunakan adalah teknik eksperimen, teknik eksperimen digunakan untuk menemukan keterkaitan metode yang digunakan dengan objek data yang dipakai, dalam kasus ini adalah menghitung jumlah kendaraan yaitu mobil atau motor yang masuk kedalam gerbang veteran ub dalam kondisi pencahayaan rendah. Metode yang digunakan penelitian ini adalah algoritma YOLOv5 untuk melakukan deteksi objek dan algoritma *DeepSORT* untuk melakukan pelacakan objek dan menghitung jumlah kendaraan.

Penelitian ini menggunakan teknik *image preprocessing* dengan melakukan pelatihan model U-Net untuk melakukan segmentasi terhadap *flare* dengan menggunakan dataset sekunder dari penelitian Esfahani dan Wang (2021). Setelah melakukan pelatihan dengan model U-Net tersebut. Model disimpan dan digunakan saat melakukan inferensi terhadap setiap *frame* video dengan melakukan segmentasi cahaya. Setelah itu dilanjutkan dengan tahapan preproses gambar dengan metode *inpainting* terhadap cahaya yang ditandai dengan *masking*. Penggunaan *inpainting* sendiri bertujuan sebagai *image restoration* atau perbaikan gambar terhadap kesiluan yang diakibatkan oleh cahaya atau *flare* dari yang berlebihan dari sebuah kendaraan. Penggunaan teknik *preprocessing* ini juga bertujuan untuk mengetahui apakah ada peningkatan pada model deep learning yang diimplementasikan dan meningkatkan akurasi dalam perhitungan jumlah kendaraan yang masuk dalam gerbang veteran universitas Brawijaya dalam kondisi pencahayaan rendah dibanding dengan yang tidak menggunakan teknik *preprocessing*.

Perhitungan Evaluasi dilakukan terhadap perhitungan pada deteksi objek, pelacakan objek, dan perhitungan objek. Deteksi objek dievaluasi menggunakan metrik evaluasi *precision*, *recall*, dan *F1-score*. Sedangkan pada pelacakan objek akan menggunakan evaluasi metrik MOTA (*Multiple Object Tracking Accuracy*) dan MOTP (*Multiple Object Tracking Precision*). Untuk evaluasi perhitungan kendaraan akan dilakukan dengan observasi manual dengan menghitung jumlah kendaraan masing-masing motor dan mobil yang masuk dengan membandingkan hasil perhitungan kendaraan motor dan mobil yang masuk dengan menggunakan *DeepSORT* terhadap masing-masing menggunakan teknik *preprocessing* gambar dan tidak menggunakan teknik *preprocessing* gambar. Setelah itu dihitung nilai *precision*, *recall*, akurasi, dan *F1-score* dari hasil perhitungan manual.

3.2.1 Dataset

Dataset yang digunakan berupa potongan video rekaman CCTV pada gerbang Veteran Universitas Brawijaya pada tanggal 22 Mei 2023 dari pukul 6.06 pagi hingga 22.48 Malam. Video berjumlah sebanyak 71 video dengan rata-rata video berdurasi selama 15 menit. Penelitian ini hanya menggunakan video yang diambil pada rentang waktu mulai pukul 17:00 hingga 21:00. Pengambilan dibataskan pada pukul 21:00 dikarenakan gerbang masuk veteran UB ditutup. Dengan demikian jumlah cuplikan video yang diambil berjumlah 15 video. Setelah itu, masing-masing video akan diambil sampel *frame* video secara acak dengan masing-masing video berjumlah 50 *frame*. Sehingga total *frame* yang dipapatkan secara acak berjumlah 750. Dikarenakan pengambilan *frame* secara acak terdapat *frame* yang tidak menampilkan kendaraan mobil atau motor dari *frame* yang diambil, maka dilakukan pemilihan secara manual. Didapatkan hasil akhir total *frame* yang digunakan untuk melakukan pelatihan model objek deteksi berjumlah 642 *frame* dengan total objek mobil berjumlah 205 dan objek motor berjumlah 1177.

Setelah melakukan seleksi *frame*, seluruh *frame* dianotasi menggunakan aplikasi Label Studio. Objek mobil akan diberi id kelas 0 dan objek motor akan diberi id kelas 1. Setelah selesai melakukan anotasi dari setiap kendaraan yang muncul dari masing-masing *frame*, ekspor hasil pelabelan dalam format YOLO. Data ini diambil langsung dari CCTV Universitas Brawijaya dan belum pernah digunakan pada penelitian terdahulu. Sehingga data ini bersifat primer. Penelitian ini berfokus pada perhitungan kendaraan dalam kondisi pencahayaan rendah. Salah satu sampel gambar yang menjadi dataset dapat dilihat pada gambar 3.2.



Gambar 3.2 Salah satu *frame* dari video CCTV

3.2.2 Instrumen yang Digunakan

Instrumen yang digunakan pada penelitian ini meliputi perangkat keras dan perangkat lunak. Adapun Instrumen tersebut adalah sebagai berikut:

3.2.2.1 Perangkat Keras

Perangkat keras yang digunakan dalam penelitian ini meliputi:

- Komputer dengan prosesor AMD 3020e, Memory (RAM) 16GB, dan SSD 512GB
- GPU Labkc Universitas brawijaya

3.2.2.2 Perangkat Lunak

Perangkat lunak yang digunakan dalam penelitian ini meliputi:

- Sistem operasi Windows 11
- Aplikasi pengolah kata (Microsoft Word 2006)
- Aplikasi teks editor (Vs Code)
- Python 3.8

- Google Colab
- Label Studio

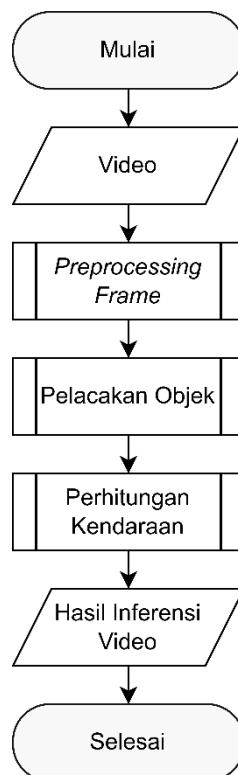
3.2.3 Artefak yang Dihasilkan

Artefak yang dihasilkan pada penelitian ini berupa sebuah model pembelajaran mesin yang dapat melakukan deteksi jumlah kendaraan dan sebuah video dari hasil inferensi dengan tambahan algoritma tracking yaitu *DeepSORT* untuk melakukan tracking kendaraan dan menghitung jumlah kendaraan pada lingkungan Universitas Brawijaya gerbang Veteran pada malam hari.

BAB 4 PERANCANGAN ALGORITMA

4.1 Alur Utama Algoritma

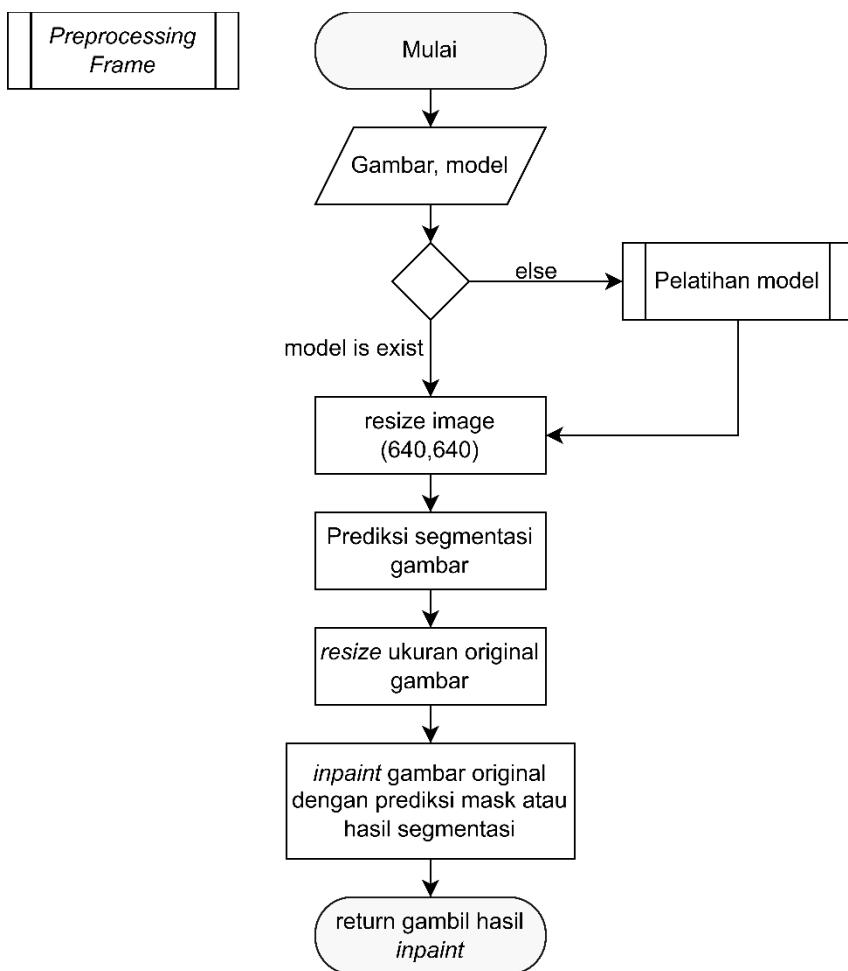
Alur utama algoritma yang dibangun terdiri dari *preprocessing frame* dari video inputan, dilanjutkan dengan melakukan pelacakan objek dengan algoritma *DeepSORT* berdasarkan hasil deteksi objek dari masing-masing *frame* video, dan perhitungan kendaraan. Inputan yang digunakan adalah video yang belum pernah digunakan pada saat pelatihan model dan diharapkan nantinya hasil akhir dari sistem ini adalah berupa video hasil inferensi yang mana akan menunjukkan deteksi objek motor dan mobil serta perhitungan masing-masing jenis kendaraan yang masuk. Gambar *flowchart* pada bagian alur utama algoritma dapat dilihat pada Gambar 4.1.



Gambar 4.1 Alur Utama Algoritma

4.1.1 Alur Implementasi Preprocessing Model Machine Learning

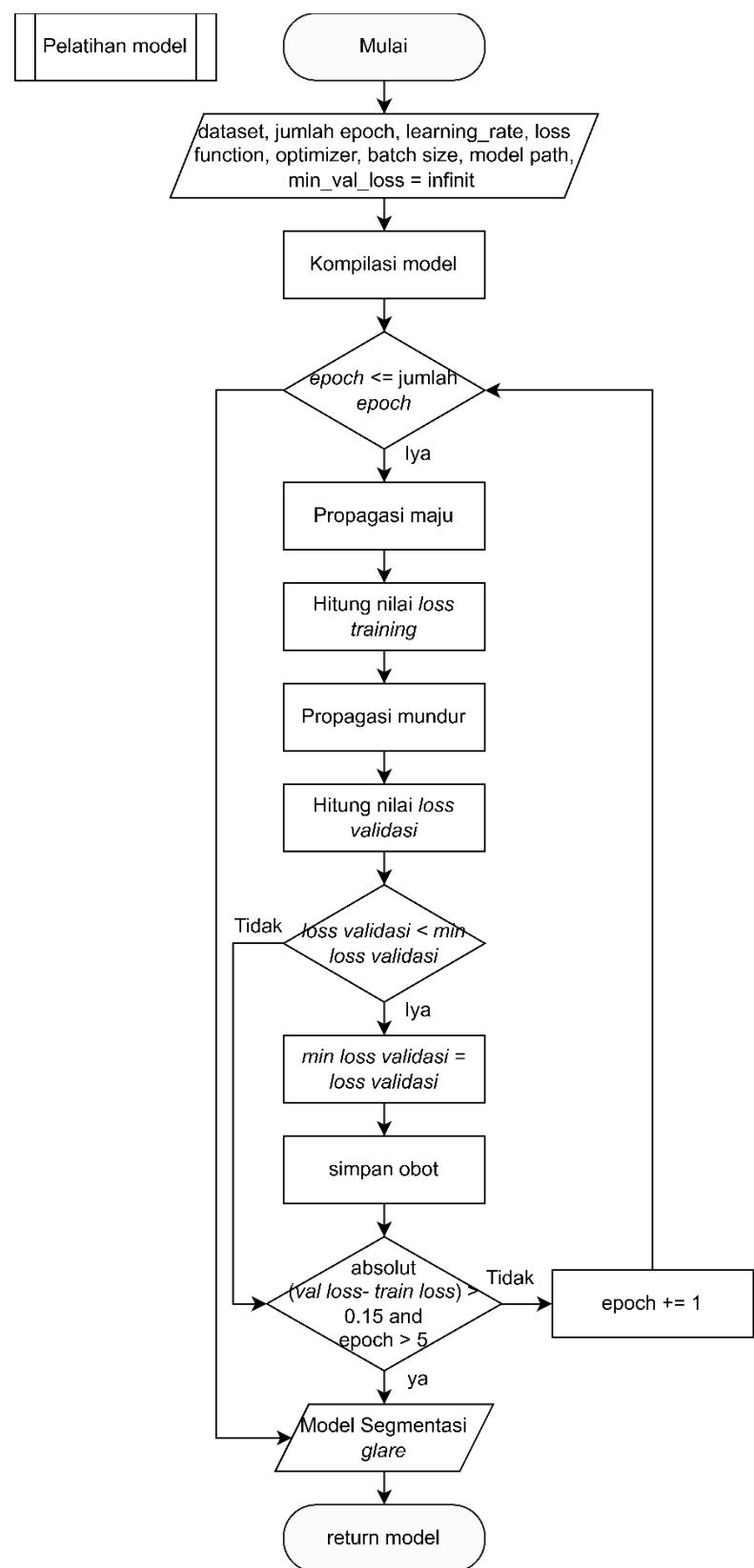
Proses ini dilakukan untuk menjawab rumusan masalah ke-2. Proses gambar pada penelitian ini menggunakan model segmentasi dari yang telah dilatih sebelumnya. Hasil prediksi nantinya berupa gambar masking dari glare atau kesilauan yang ada pada gambar masukan. Setelah mendapatkan gambar prediksi, nantinya gambar original akan dilakukan proses *inpaint* terhadap gambar maskingnya. *Flowchart* dari *preprocess frame* dapat dilihat pada gambar 4.2.



Gambar 4.2 Alur Implementasi Preprocessing Gambar

4.1.2 Alur Pelatihan Model *Preprocessing Segementasi Gambar*

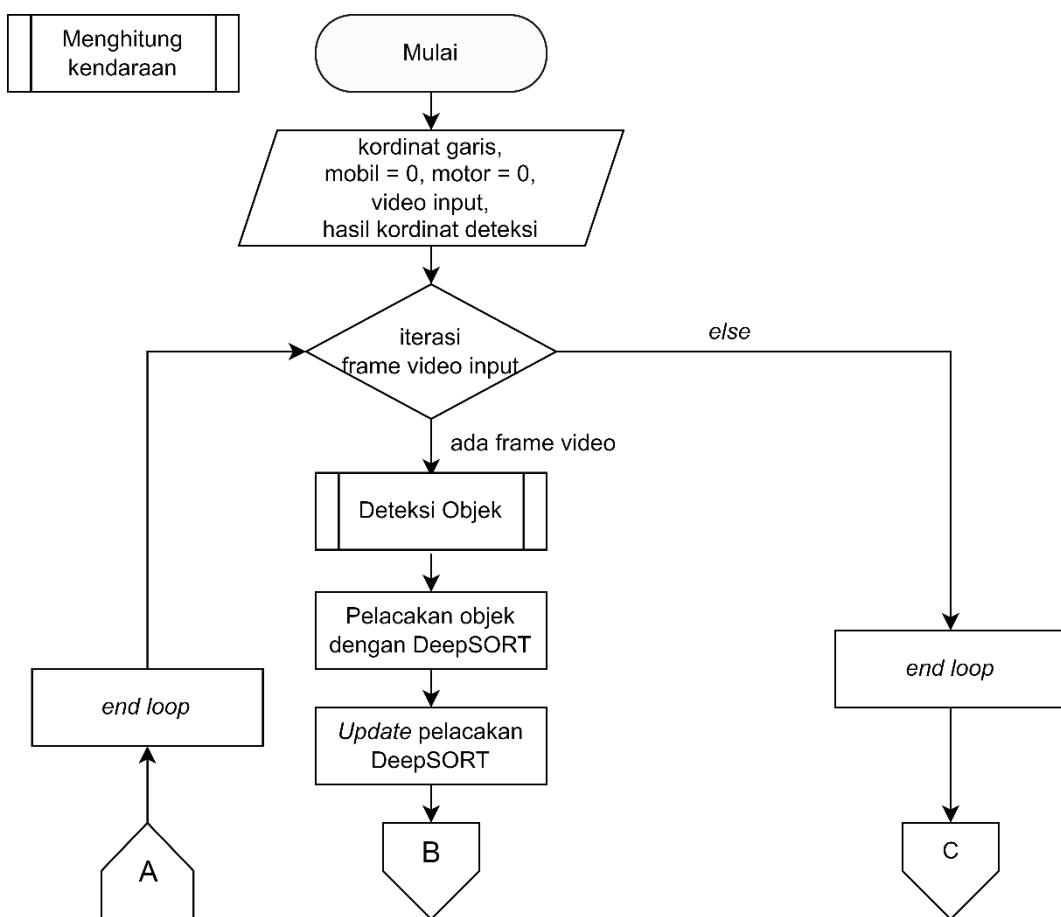
Pada alur pelatihan model *preprocessing* segmentasi gambar, arsitektur model yang digunakan adalah arsitektur segmentasi seperti pada Gambar 2.5. Jumlah epoch yang diinisiasi berupa 100. Selain itu pelatihan model diterapkan pembuatan *callback* manual dengan melihat selisih dari nilai *loss* antara model latih dan model validasi. Model yang disimpan berupa model yang memiliki nilai *loss* validasi yang paling rendah. Nilai *loss* validasi akan selalu diperbarui setiap mendapatkan nilai *loss* yang lebih rendah dari sebelumnya. Alur pelatihan model *preprocessing* untuk segmentasi gambar dapat dilihat pada Gambar 4.3.

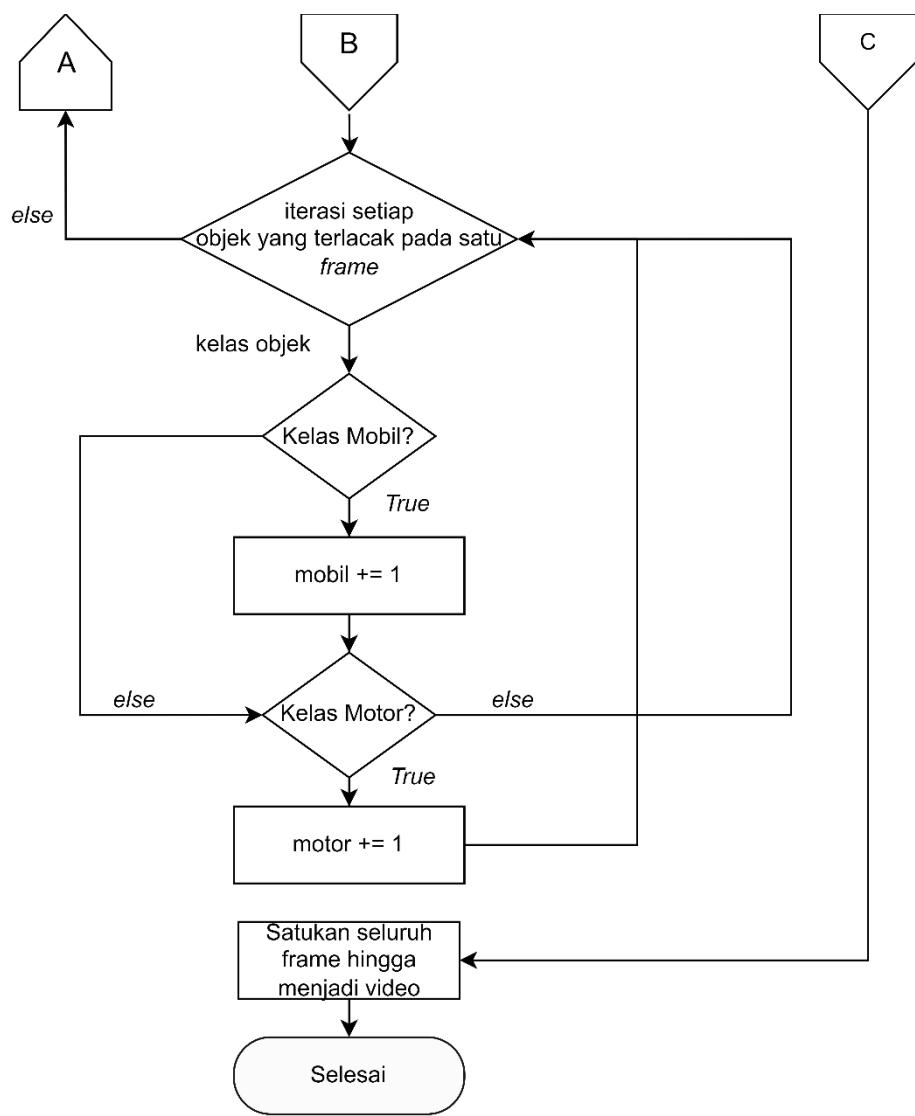


Gambar 4.3 Alur Pelatihan Model Segmentasi Gambar

4.1.3 Alur Perhitungan Kendaraan

Proses akan melakukan perulangan sebanyak frame dari video yang dimasukkan ke sistem utama. Jika percobaan pelacakan ingin menggunakan metode *preprocessing* gambar. Maka setiap *frame* dari video masukan akan dilakukan *preprocessing* gambar menggunakan metode segmentasi dan *inpaint* berdasarkan alur kerja pada Gambar 4.2. Kemudian barulah dilakukan proses pelacakan dengan menggunakan algoritma *DeepSORT*. Setelah melewati proses pelacakan objek, nantinya akan dilihat titik tengah dari kordinat pada sebuah objek yang terlacak. Jika itu mobil dan melewati garis batas, maka jumlah mobil yang terhitung bertambah 1 dan begitu juga pada objek yang bernama motor. Alur perhitungan kendaraan dari hasil pelacakan dapat dilihat pada *flowchart* Gambar 4.4.

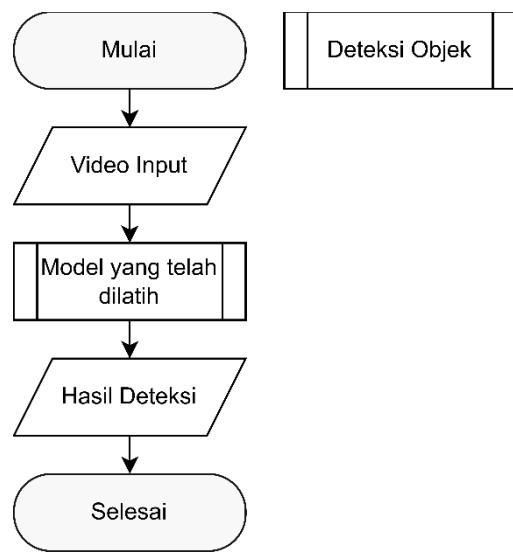




Gambar 4.4 Alur perhitungan kendaraan

4.1.4 Alur Deteksi Objek

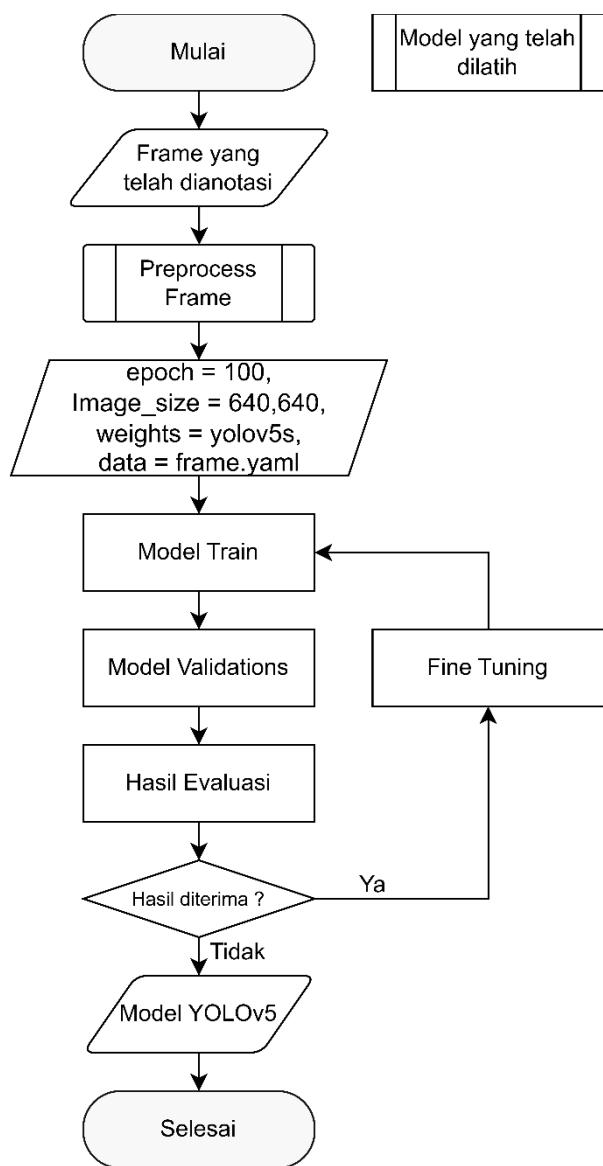
Pada process pendekripsi objek, video nantinya akan di deteksi dengan menggunakan model dari YOLOv5 yang telah dilatih sebelumnya. Model yang dilatih menyesuaikan dengan dangan dataset yang telah dimiliki berupa frame dari CCTV gerbang veteran pada Universitas brawijaya dari pukul 18.00 hingga 21.00 untuk mendekripsi khusus pada area tersebut pada permasalahan kurangnya pencahayaan. Gambar flowchart pada alur deteksi objek dapat dilihat pada Gambar 4.5.



Gambar 4.5 Alur Deteksi Objek

4.1.5 Alur Pelatihan Model Deteksi Objek

Pada proses pelatihan model, penelitian ini menggunakan pretrained dari model YOLOv5 pada ukuran *small*. Untuk data yang digunakan berupa frame dari CCTV gerbang veteran Universitas Brawijaya pada pukul 17.00 hingga 21.00 yang diambil secara acak. Nantinya frame melewati proses yang bernama preprocess frame yang bertujuan untuk mengurangi efek silau atau *glare* yang disebabkan oleh kendaraan pada malam hari. Pelatihan model dilakukan sebanyak 100 epoch dengan mengurangi *size frame* menjadi 640,640. Pada proses ini juga dilakukan *fine tuning* yang mana parameter pada pelatihan YOLOv5 disesuaikan agar mendapatkan hasil dari evaluasi yang maksimal. Alur pelatihan model dapat dilihat pada *flowchart* Gambar 4.6.

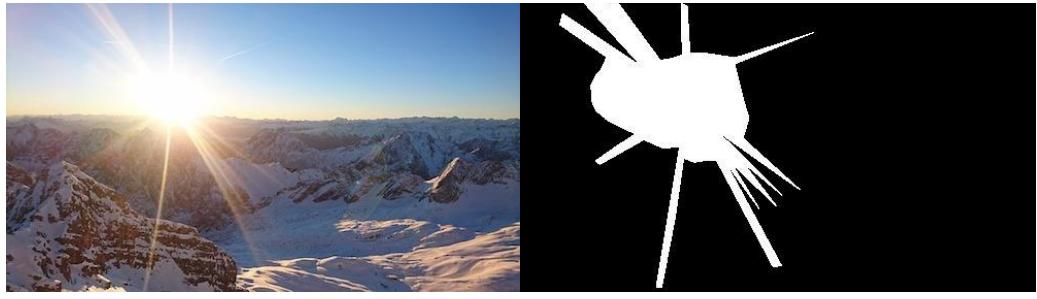


Gambar 4.6 Alur Pelatihan Model Deteksi Objek

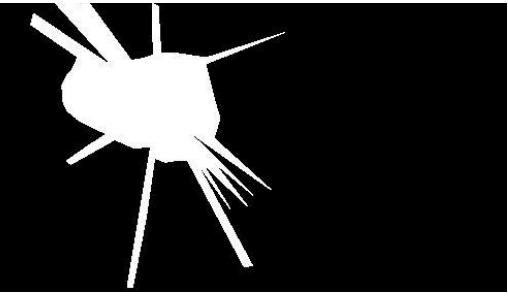
4.2 Perhitungan Manual

4.2.1 Perhitungan Manual *Training U-Net*

Pada perhitungan manual dengan menggunakan model U-Net menggunakan sebuah gambar sebagai sampel dan pelatihan dalam 1 epoch. Tampilan gambar sampel dan *ground truth* hasil segmentasi yang digunakan pada pelatihan model segmentasi arsitektur U-net pada Gambar 4.7 dan Gambar 4.8.



(a)



(b)

Gambar 4.7 Sampel gambar original (a),

Gambar 4.8 *Ground Truth* dari Gambar 4.7 (b)

4.2.1.1 Perhitungan Manual Lapisan Konvolusi

Percobaan perhitungan lapisan konvolusi akan menggunakan persamaan (2.11) dengan inputan Gambar 4.7. Sebelum melakukan perhitungan manual, nilai indeks *pixel* yang digunakan dilakukan normalisasi. Normalisasi ini umumnya diterapkan pada data gambar sebelum diteruskan ke model jaringan saraf tiruan atau algoritma pembelajaran mesin lainnya. Ini membantu model untuk belajar lebih baik karena nilai yang lebih kecil seringkali memudahkan konvergensi selama pelatihan. Persamaan yang digunakan untuk melakukan normalisasi dapat dilihat pada persamaan (4.1).

$$output[c, h, w] = \frac{input[c, h, w]}{255} \quad (4.1)$$

Selain itu, seluruh inputan gambar di-*resize* ukuran Panjang dan lebarnya menjadi ukuran 640 x 640. Sehingga indeks nilai pixel dari masing-masing *channel* yang digunakan dalam perhitungan konvolusi dalam penelitian ini dapat dilihat pada Tabel 4.1, Tabel 4.2, dan Tabel 4.3:

Tabel 4.1 Pixel Chanel B Gambar 4.7

j\i	0	1	2	...	637	638	639
0	0.8000	0.7961	0.7961	...	0.6235	0.6235	0.6196
1	0.8000	0.7961	0.7961	...	0.6235	0.6235	0.6196
2	0.7961	0.7922	0.7922	...	0.6275	0.6275	0.6235
...
637	0.2706	0.2078	0.1608	...	0.3608	0.3608	0.3608
638	0.2745	0.2353	0.2000	...	0.3451	0.3490	0.3529
639	0.2645	0.2431	0.2157	...	0.3412	0.3451	0.3490

Tabel 4.2 Pixel Chanel G Gambar 4.7

j\i	0	1	2	...	637	638	639
0	0.6745	0.6706	0.6706	...	0.4039	0.4039	0.4000
1	0.6745	0.6706	0.6706	...	0.4039	0.4039	0.4000
2	0.6706	0.6667	0.6667	...	0.4078	0.4078	0.4039
...
637	0.2667	0.2078	0.1608	...	0.2627	0.2667	0.2667
638	0.2706	0.2341	0.2000	...	0.2471	0.2510	0.2549
639	0.2706	0.2392	0.2118	...	0.2431	0.2471	0.2510

Tabel 4.3 Pixel Chanel R Gambar 4.7

j\i	0	1	2	...	637	638	639
0	0.6510	0.6471	0.6471	...	0.2196	0.2196	0.2157
1	0.6510	0.6471	0.6471	...	0.2196	0.2196	0.2157
2	0.6471	0.6431	0.6431	...	0.2235	0.2235	0.2196
...
637	0.3137	0.2510	0.2039	...	0.2627	0.2627	0.2627
638	0.3098	0.2706	0.2392	...	0.2431	0.2471	0.2471
639	0.3098	0.2784	0.2510	...	0.2353	0.2392	0.2431

Kemudian inisiasi nilai bobot kernel dengan nilai random dengan rentang -1 hingga 1. Sesuai dengan arsitektur U-Net pada Gambar 2.4, Nilai kernel konvolusi yang digunakan berukuran 3x3. Nilai kernel yang digunakan dalam perhitungan manual ini dapat dilihat pada Tabel 4.4, Tabel 4.5, dan Tabel 4.6.

Tabel 4.4 Bobot Kernel 1 Channel R

j\i	0	1	2
0	0.147	0.16	-0.045
j\i	0	1	2
1	0.177	-0.042	0.039
2	-0.094	0.113	0.17

Tabel 4.5 Bobot Kernel 1 Channel G

j\i	0	1	2
0	-0.141	0.167	0.036

1	0.142	0.026	0.093
2	-0.027	0.148	0.028

Tabel 4.6 Bobot Kernel 1 Channel B

j\i	0	1	2
0	-0.09	0.049	-0.089
1	-0.023	-0.078	0.128
2	-0.152	-0.089	-0.054

Setelah itu hitunglah dengan persamaan (2.11) dengan menggunakan bias sebesar 0.03669779. Sehingga nilai bobot pada *channel* pertama dapat dilihat pada Tabel 4.7:

$$h(1,1) = \left(\sum_{c=0}^{channel} \sum_{a=-n}^n \sum_{b=-n}^n f_c(a,b) \times g_c(x-a, y-b) \right) + Bias$$

$$\begin{aligned} h(1,1)_r &= (0.6510 \times 0.147 + 0.6471 \times 0.16 + 0.6471 \times -0.045 + 0.6510 \\ &\quad \times 0.142 + 0.6706 \times -0.042 + 0.6471 \times 0.039 + 0.6471 \\ &\quad \times -0.094 + 0.6431 \times 0.113 + 0.6431 \times 0.17) \end{aligned}$$

$$h(1,1)_r = 0.127$$

$$h(1,1) = h(1,1)_r + h(1,1)_g + h(1,1)_b + bias = 0.4408$$

Tabel 4.7 Bobot *Channel* pertama konvolusi

j\i	0	1	2	...	637	638	639
0	0.3415	0.3333	0.3332	...	0.1151	0.1135	-0.025
1	0.5208	0.4408	0.4416	...	0.1168	0.1159	0.0249
2	0.5194	0.4401	0.4408	...	0.1186	0.1177	0.0257
...
637	0.2415	0.1946	0.1679	...	0.1841	0.1856	0.1081
638	0.2486	0.2091	0.1863	...	0.1725	0.1754	0.1019
639	0.1568	0.2110	0.1926	...	0.1860	0.1870	0.1443

4.2.1.2 Perhitungan Manual Fungsi Aktivasi ReLU

[Jelaskan ReLU]. Mengikuti persamaan (2.12), Fungsi ReLU berfungsi untuk mengubah seluruh negatif dari hasil perhitungan bobot konvolusi pada Tabel 4.7 menjadi 0. Sedangkan untuk nilai yang lebih dari 0 tetap mempertahankan nilai bobotnya tersebut. Nilai bobot setelah melakukan fungsi aktivasi ReLU menjadi seperti pada Tabel 4.8:

Tabel 4.8 Bobot *Channel* Pertama Setelah Fungsi Aktivasi ReLU

j\i	0	1	2	...	637	638	639
0	0.3415	0.3333	0.3332	...	0.1151	0.1135	0
1	0.5208	0.4408	0.4416	...	0.1168	0.1159	0.0249
2	0.5194	0.4401	0.4408	...	0.1186	0.1177	0.0257
...
637	0.2415	0.1946	0.1679	...	0.1841	0.1856	0.1081
638	0.2486	0.2091	0.1863	...	0.1725	0.1754	0.1019
639	0.1568	0.2110	0.1926	...	0.1860	0.1870	0.1443

4.2.1.3 Perhitungan Manual Lapisan Max Pooling

Pada arsitektur U-Net yang diimplementasikan pada penelitian ini menggunakan lapisan *max pooling* dengan stride berjumlah 2. *Stride* (langkah) adalah parameter dalam operasi konvolusi yang mengontrol seberapa banyak kernel "bergerak" di sepanjang dimensi input setiap kali melakukan operasi konvolusi. Dengan kata lain, stride menentukan seberapa jauh kernel bergerak ketika menggeser dari satu posisi ke posisi berikutnya dalam input. Sedangkan untuk ukuran kernel yang digunakan pada percobaan perhitungan manual ini adalah ukuran 2x2. Perhitungan *max pool* akan mencari nilai tertinggi setiap 2x2 blok pixel di setiap channelnya dengan melakukan Langkah sebanyak 2 langkah bergerak. Nilai *max pooling* dapat dilihat pada Tabel 4.9 dengan menggunakan bobot pada Tabel 4.8 sebagai percobaan perhitungannya.

$$\text{Max Pooling}(0,0) = \text{MAX}(0.3415, 0.3333, 0.5208, 0.4408)$$

$$\text{Max Pooling}(0,0) = 0.5208$$

Tabel 4.9 Hasil *Max Pooling* pada Bobot *Channel* Pertama

j\i	0	1	2	...	317	318	319
0	0.5208	0.4428	0.4426	...	0.1252	0.1184	0.1159
1	0.5194	0.4432	0.4464	...	0.1259	0.1209	0.1184
2	0.5195	0.4458	0.4491	...	0.1268	0.1228	0.1204
...
317	0.2077	0.1987	0.2983	...	0.2213	0.2185	0.2171
318	0.2415	0.1799	0.2337	...	0.2038	0.1972	0.1966
319	0.2486	0.1926	0.1821	...	0.2001	0.1883	0.1870

Dari Tabel 4.9, dapat dilihat bahwa ukuran channel berubah dari yang mulaunya 640 x 640 menjadi 320 x 320. Hal ini terjadi dikarenakan jumlah *stride* yang digunakan sebesar 2 langkah.

4.2.1.4 Perhitungan Manual Lapisan Up Convolution

Lapisan *up-conv* merupakan metode yang biasanya digunakan untuk meningkatkan resolusi spatial dari *feature map* pada *deep learning* terkhususnya dalam kasus pengolahan citra. Dalam pengimplementasiannya menggunakan sebuah layer yang Bernama ConvTranspose2d yang memiliki parameter berupa filter yang akan memperbarui bobotnya sehingga *upsampling* bisa menjadi optimal. Formulasi dari perhitungan pada persamaan (2.13). Salah satu nilai bobot yang digunakan dalam perhitungan manual dapat dilihat pada Tabel 4.10 dan untuk bobot dari kernel yang digunakan dapat dilihat pada Tabel 4.11.

Tabel 4.10 Bobot Salah Satu Chanel Percobaan *Up-Conv*

j\i	0	1	2	...	37	38	39
0	0.0094	0.0059	0.0053	...	0.0053	0.0061	0.0061
1	0.0088	0.0060	0.005	...	0.005	0.0061	0.006
2	0.0094	0.0073	0.0061	...	0.0061	0.0071	0.0069
...
37	0.0095	0.0073	0.0062	...	0.0063	0.0073	0.007
38	0.0095	0.0072	0.0061	...	0.0063	0.0067	0.0071
39	0.001	0.0098	0.009	...	0.009	0.0091	0.009

Tabel 4.11 Bobot Kernel pada *up-conv*

j\i	0	1
0	0.0176	0.0108
1	0.0216	-0.004

Untuk mengetahui hasil ukuran panjang dan lebar dari sebuah perhitungan lapisan *up-conv* untuk masing-masing *channel* dapat mengikuti persamaan (4.2) dan (4.3).

$$\begin{aligned} \text{lebar output} &= (\text{lebar input} - 1) \times \text{stride} - 2 \times \text{padding} \\ &\quad + \text{kernel size} + \text{output padding} \end{aligned} \quad (4.2)$$

$$\begin{aligned} \text{panjang output} &= (\text{panjang input} - 1) \times \text{stride} - 2 \times \text{padding} \\ &\quad + \text{kernel size} + \text{output padding} \end{aligned} \quad (4.3)$$

Dalam perhitungan ini menggunakan *stride* berjumlah 2, kernel berukuran 2 x 2 dan tidak menggunakan *padding* pada masukan dan keluaran. Sehingga ukuran dari panjang dan lebar keluaran yang didapatkan adalah sebagai berikut:

$$\text{Tinggi output} = (40 - 1) \times 2 - 2 \times 0 + 2 + 0 = 80$$

$$\text{Lebar output} = (40 - 1) \times 2 - 2 \times 0 + 2 + 0 = 80$$

Untuk perhitungan nilai bobot yang didapatkan berupa perhitungan berikut:

$$h(0,0) = 0.0094 \times 0.0176 = 0.0001$$

$$h(0,1) = 0.0094 \times 0.0108 = 0.0001$$

$$h(1,0) = 0.0094 \times 0.0216 = 0.0002$$

$$h(1,1) = 0.0094 \times -0.004 = -0.00003$$

Hasil bobot perhitungan dari *up-conv* dapat dilihat dari Tabel 4.12.

Tabel 4.12 Bobot Hasil Perhitungan *Up-conv*

j\i	0	1	2	...	77	78	79
0	0.0001	0.0001	1e-5	...	6e-4	-3e-4	4e-4
1	0.0002	-1e-4	1e-3	...	-1e-3	-1e-3	-2e-4
2	6e-4	1e-3	-2e-4	...	4e-4	-5e-4	5e-4
...
77	-1e-3	-1e-3	-1e-3	...	-1e-3	-7e-4	-4e-4
78	2e-4	1e-3	-4e-4	...	4e-4	-3e-4	1e-4
79	2e-4	1e-3	-4e-3	...	-5e-4	-3e-4	-3e-4

4.2.1.5 Perhitungan Manual Fungsi Aktivasi Sigmoid

Perhitungan manual pada fungsi aktivasi sigmoid akan mengikuti persamaan (2.14). Penggunaan fungsi sigmoid dikarenakan hasil masking pixel warna gambar akan berupa 0 dan 1. Sehingga penggunaan fungsi aktivasi ini cocok dikarenakan hanya terdiri dari dua kelas. Perhitungan fungsi Sigmoid dengan menggunakan percobaan bobot yang dapat dilihat pada Tabel 4.13. Nilai bobot diambil dari hasil perhitungan pada layer terakhir dari percobaan model U-Net terhadap Gambar 4.11 dalam epoch pertama.

Tabel 4.13 Bobot Percobaan Perhitungan Sigmoid

j\i	0	1	2	...	637	638	639
0	-0.046	-0.043	-0.041	...	-0.046	-0.045	-0.049
1	-0.048	-0.046	-0.044	...	-0.048	-0.046	-0.05
2	-0.048	-0.044	-0.043	...	-0.04	-0.04	-0.05
...
637	-0.051	-0.049	-0.051	...	-0.049	-0.047	-0.051
638	-0.051	-0.05	-0.051	...	-0.05	-0.04	-0.05
639	-0.052	-0.048	-0.05	...	-0.04	-0.048	-0.051

$$sigmoid(x) = \frac{1}{1 + e^{-x}}$$

$$sigmoid(-0.046) = \frac{1}{1 + e^{0.046}}$$

$$sigmoid(-0.046) = \frac{1}{1 + 1.047} \approx 0.48849$$

Hasil dari penerapan fungsi aktivasi sigmoid terhadap seluruh bobot pada Tabel 4.13 dapat dilihat pada Tabel 4.14.

Tabel 4.14 Hasil Perhitungan Fungsi Aktivasi Sigmoid

j\i	0	1	2	...	637	638	639
0	0.4884	0.489	0.4897	...	0.4884	0.4887	0.4876
1	0.4878	0.4885	0.4889	...	0.4879	0.4884	0.4875
2	0.488	0.489	0.4891	...	0.4875	0.4885	0.4872
...
637	0.4871	0.4877	0.4872	...	0.4876	0.4881	0.487
638	0.4861	0.4873	0.4871	...	0.4872	0.4876	0.4868
639	0.4869	0.4879	0.4874	...	0.4877	0.4878	0.4871

4.2.1.6 Perhitungan Manual Fungsi Loss BCE (Binary Cross Entropy)

Perhitungan manual terhadap fungsi *loss* BCE menggunakan nilai pada Tabel 4.14 dari hasil perhitungan fungsi aktivasi sigmoid. Sedangkan untuk nilai dari *ground truth masking* yang digunakan dapat dilihat pada Tabel 4.15.

Tabel 4.15 Piksel *Ground Truth*

j\i	0	1	2	...	637	638	639
0	0	0	0	...	0	0	0
1	0	0	0	...	0	0	0
2	0	0	0	...	0	0	0
...
637	0	0	0	...	0	0	0
638	0	0	0	...	0	0	0
639	0	0	0	...	0	0	0

Perhitungan BCE akan menggunakan persamaan pada (2.15). Berikut hasil perhitungan manualnya:

$$BCE \ loss = -\frac{1}{N} \sum_{i=0}^N y \times \log(p) + (1 - y) \times \log(1 - p)$$

$$BCE_{(0,0)} = 0 \times \log(0.4884) + (1 - 0) \times \log(1 - 0.4884)$$

$$BCE_{(0,0)} = -0.29$$

$$BCE \ loss = 0.6758$$

4.2.1.7 Perhitungan Manual Propagasi Mundur

Propagasi mundur dilakukan untuk memperbarui bobot setiap data yang dimasukkan dan dilatih dari sebuah model. Pembaruan bobot menggunakan formulasi pada bagian akhir layer dengan menggunakan persamaan (4.4):

$$w'_i = w_i - \alpha \frac{\partial L}{\partial w_i} \quad (4.4)$$

$$\frac{\partial L}{\partial w_i} = \frac{\partial z}{\partial w} \times \frac{\partial L}{\partial z} \quad (4.5)$$

$$\frac{\partial L}{\partial z} = \frac{e^{-x}}{(1 + e^{-x})^2} \quad (4.6)$$

Pada perhitungan manual propagasi mundur dari model U-Net menggunakan salah satu lapisan konvolusi pada Tabel 4.16 dan bobot kernel pada Tabel 4.17.

Tabel 4.16 Lapisan Konvolusi

y\x	0	1	2	...	637	638	639
0	0.01	0.004	0.005	...	0.01	0.01	0.01
1	0	0	0	...	0.005	0.007	0.009
2	0	0	0	...	0.006	0.007	0.007
...
637	0.01	0.007	0.002	...	0.0009	0.002	0.007
638	0.01	0.004	0	...	0.002	0.002	0.0063
639	0.01	0.01	0.01	...	0.01	0.01	0.01

Tabel 4.17 Bobot Kernel

j\ni	0
0	-0.0366

Perhitungan menggunakan nilai *learning rate* (α) berjumlah 0.0001. Berikut hasil percobaan perhitungan:

$$\frac{\partial L}{\partial z} = \frac{e^{-(0.01 \times -0.0366)}}{(1 + e^{-(0.01 \times -0.0366)})^2} = 0.999$$

$$\frac{\partial L}{\partial w_i} = 0.01 \times 0.999 + f(x, y) \times \text{sigmoid}'(f(x, y) \times -0.0366) + \dots$$

$$\frac{\partial L}{\partial w_i} \approx -10.05$$

$$w'_i = -0.0366 - 0.0001(-10.05)$$

$$w'_i = -0.0368$$

4.2.2 Perhitungan Manual Proses *Inpaint*

Proses *inpaint* digunakan untuk melakukan perbaikan atau restorasi gambar pada area yang telah ditandai berupa gambar *masking*. Area yang dilakukan *masking* adalah area yang memiliki cahaya berlebih atau silau (*flare*). Persamaan yang digunakan untuk melakukan *inpaint* dapat dilihat pada persamaan (2.15) dengan percobaan perhitungan manual pada Gambar 4.6 dan *masking* pada Gambar 4.7. Piksel yang digunakan dapat dilihat pada Tabel 4.18 dengan menggunakan piksel pada kordinat (61,0) pada *channel* warna B. Sedangkan untuk piksel pada gambar *masking* dapat dilihat pada Tabel 4.19.

Tabel 4.18 Piksel Gambar *channel* B Percobaan Perhitungan *Inpaint*

y\x	...	58	59	60	61	...
0	...	210	210	211	211	...
1	...	210	210	211	211	...
2	...	210	210	210	210	...
3	...	212	212	212	212	...
..

Tabel 4.19 Piksel Gambar 4.7

y\x	...	58	59	60	61	...
0	...	0	0	0	1	...
1	...	0	0	0	0	...
2	...	0	0	0	0	...
3	...	0	0	0	0	...
..

Radius yang digunakan pada perhitungan *inpaint* adalah 3. Radius merupakan jarak antara piksel yang tertandai *inpaint* pada gambar *masking* yang pada pada percobaan ini ada pada kordinat piksel (0,61) dan radius pada piksel tersebut berupa jarak 3 langkah sebelum dan sesudah pada piksel yang akan di-*inpaint*. Percobaan perhitungan dilakukan terhadap piksel (0, 61) dan (0,58) yang mana

sesuai dengan kordinat. Perhitungan awal yang dilakukan adalah menghitung bobot. Perhitungan bobot. Pada perhitungan bobot dapat dilakukan mengikuti persamaan (2.16) hingga (2.19)

$$dir(p, q) = \frac{p - q}{\|p - q\|} \cdot N(p)$$

$$\|p - q\| = \sqrt{(0 - 0)^2 + (58 - 61)^2} = 3$$

$$dir(p, q) = \frac{|210 - 211|}{3} \cdot 0,82 = 0.27$$

$$dst(p, q) = \frac{d_0}{\|p - q\|^2}$$

$$dst(p, q) = \frac{1}{9} = 0.11$$

$$lev(p, q) = \frac{T_0}{1 + \|T(p) - T(q)\|}$$

$$lev(p, q) = \frac{1}{1 + 3} = 0.25$$

$$w(p, q) = 0.27 \times 0.11 \times 0.25 = 0.007$$

Selanjutnya adalah perhitungan antara bobot dan nilai piksel yang akan di-*inpaint* yang dapat dilihat pada persamaan berikut:

$$w(p, q)[I(q) + \nabla I(q)(p - q)] = 0.007 \times 210 = 1.47$$

Perhitungan ini dilakukan terhadap seluruh radius pada piksel yang tertandai pada gambar *masking* yang nantinya seluruhnya akan dijumlah dan dibagi dengan jumlah bobot yang dapat dilihat pada persamaan (2.15).

4.2.3 Perhitungan Manual Training YOLOv5

Pada perhitungan manual dengan menggunakan model YOLOv5 ini menggunakan sebuah gambar sebagai sampel dan pelatihan dalam 1 epoch atau 1 proses dalam seluruh gambar. Baik dalam propagasi maju dan propagasi mundur. Tampilan gambar sampel dan *bounding box*-nya dapat dilihat pada Gambar 4.9:



Gambar 4.9 Sample Gambar percobaan perhitungan manual

Sebelum melakukan perhitungan *training*. Ukuran gambar sebaiknya diubah menjadi lebih simetris dan lebih kecil dari originalnya. Hal ini dapat meningkatkan efisiensi waktu dalam melakukan pelatihan model. Sehingga gambarnya dapat berubah seperti yang ditampilkan pada Gambar 4.10:



Gambar 4.10 Perubahan Ukuran Gambar

Sebelum melakukan perhitungan, nilai indeks warna akan dilakukan normalisasi. Normalisasi dilakukan dengan menerapkan persamaan (4.1), nilai indeks warna yang digunakan memiliki rentang 0-1. Nilai indek warna yang digunakan setelah melakukan normalisasi dapat dilihat pada Tabel 4.20, Tabel 4.21, dan Tabel 4.22.

Tabel 4.20 Pixel Chanel B Gambar 4.20

j\i	0	1	2	...	637	638	639
0	0.1412	0.1451	0.1529	...	0.0784	0.0784	0.0784
1	0.1412	0.1451	0.1529	...	0.0784	0.0784	0.0784
2	0.1490	0.1529	0.1608	...	0.0784	0.0784	0.0784
...
637	0.6745	0.7647	0.7725	...	0.3216	0.3255	0.3176
638	0.7059	0.7725	0.7725	...	0.3412	0.3412	0.3608
639	0.7412	0.7804	0.7686	...	0.3608	0.3804	0.4078

Tabel 4.21 Pixel Chanel G Gambar 4.21

j\i	0	1	2	...	637	638	639
0	0.1490	0.1529	0.1608	...	0.1333	0.1333	0.1333
1	0.1412	0.1490	0.1608	...	0.1333	0.1333	0.1333
2	0.1490	0.1529	0.1608	...	0.1333	0.1333	0.1333
...
637	0.6235	0.7137	0.7216	...	0.3059	0.3098	0.2980
638	0.6549	0.7216	0.7255	...	0.3255	0.3255	0.3412
639	0.6902	0.7294	0.7176	...	0.3451	0.3647	0.3922

Tabel 4.22 Pixel Chanel R Gambar 4.22

j\i	0	1	2	...	637	638	639
0	0.1569	0.1529	0.1608	...	0.2078	0.2078	0.2078
1	0.1647	0.1647	0.1725	...	0.2078	0.2078	0.2078
2	0.1804	0.1843	0.1922	...	0.2078	0.2078	0.2078
...
637	0.6314	0.7216	0.7294	...	0.3020	0.3059	0.2941
638	0.6627	0.7294	0.7294	...	0.3216	0.3216	0.3373
639	0.6980	0.7373	0.7216	...	0.3412	0.3608	0.3882

4.2.3.1 Perhitungan Manual Lapisan ConvBNSiLU

Layer ConvBNSILU merupakan gabungan dari layer conv2d, *batch normalization*, dan fungsi aktivasi SILU. Pada lapisan pertama arsitektur YOLOv5 terdapat lapisan ConvBNSILU dengan keterangan kernel berjumlah 6, *stride* berjumlah 2, *padding* berjumlah 2, dan *output channel* berjumlah 64. *Stride* merupakan jumlah Langkah yang diambil dalam melakukan konvolusi. Jika Stride berjumlah 2. Maka diambil 2 langkah ke samping untuk melakukan konvolusi. Sedangkan *padding* merupakan penambahan 2 baris piksel 0 di setiap sisi dari setiap dimensi spasial (lebar dan tinggi) dari channel input sebelum melakukan konvolusi. Pada perhitungan manual layer ini dilakukan percobaan terhadap 1 kernel saja dengan bobot nilai random yang dapat dilihat pada Tabel 4.23, Tabel 4.24, dan Tabel 4.25.

Tabel 4.23 Bobot Kernel 1 Channel R

j\i	0	1	2	3	4	5
0	0.0736	0.0799	-0.0225	0.0884	-0.0211	0.0194
1	-0.0468	0.0565	0.0848	0.0706	0.0836	0.0180
2	0.0711	0.0130	0.0464	-0.0136	0.0742	0.0142
3	0.0449	0.0245	-0.0443	-0.0113	-0.0391	0.0638
4	-0.0760	-0.0444	-0.0272	-0.0579	0.0091	-0.0950
5	0.0869	-0.0817	0.0743	0.0160	0.0312	0.0595

Tabel 4.24 Bobot Kernel 1 Channel G

j\i	0	1	2	3	4	5
0	0.0150	0.0777	0.0105	-0.0303	0.0259	-0.0261
1	0.0405	0.0859	0.0556	-0.0421	0.0555	0.0172
2	0.0489	-0.0586	-0.0953	-0.0372	-0.0738	0.0790
3	0.0277	0.0399	0.0304	-0.0017	0.0753	-0.0684
4	0.0061	-0.0657	0.0297	-0.0331	0.0295	-0.0200
5	0.0798	-0.0570	-0.0574	-0.0574	0.0865	0.0321

Tabel 4.25 Bobot Kernel 1 Channel B

j\i	0	1	2	3	4	5
0	0.0926	-0.0794	-0.0954	-0.0753	-0.0647	0.0390
1	0.0345	0.0800	-0.0497	-0.0656	0.0511	-0.0389
2	0.0584	-0.0228	0.0550	-0.0748	-0.0486	0.0293
3	0.0203	-0.0245	0.0574	0.0654	-0.0698	-0.0514
4	0.0881	-0.0325	-0.0341	-0.0931	-0.0551	0.0240
5	-0.0127	-0.0698	0.0023	-0.0657	-0.0816	-0.0530

Formulasi yang digunakan pada perhitungan konvolusi dapat menggunakan persamaan (2.1). Formulasi tersebut digunakan untuk menghitung bobot dari konvolusi pada masing-masing kernel pertama dengan masing-masing channel. Nantinya hasilnya akan menjadi bobot pada indek (0,0) pada channel pertama. Sesuai dengan inisialisasi pada *layer* yang mana nantinya terdapat 64 *channel* keluaran dari hasil *layer* tersebut. Untuk nilai bias menggunakan nilai random. Berikut perhitungan konvolusi pada channel pertama:

$$h(1,1) = \left(\sum_{c=0}^{channel} \sum_{a=-n}^n \sum_{b=-n}^n f_c(a,b) \times g_c(x-a, y-b) \right) + Bias$$

$$h(1,1)_r = (0.1569 \times 0.0736 + 0.1529 \times 0.0799 + 0.1608 \\ \times (-0.025) + \dots + 1.922 \times 0.016 + 0.1804 \times 0.0312 + 0.1804 \\ \times 0.0595)$$

$$h(1,1)_r = 0.053$$

$$h(1,1) = h(1,1)_r + h(1,1)_g + h(1,1)_b + bias$$

$$h(1,1) = -0.0857$$

Hasil dari output channel pertama dari konvolusi antara bobot kernel dan input gambar dapat dilihat pada Tabel 4.26:

Tabel 4.26 Bobot Chanel Pertama pada Layer Konvolusi

j\i	0	1	2	...	317	318	319
0	-0.1475	-0.1595	-0.1599	...	-0.1197	-0.1171	-0.1262
1	-0.1519	-0.0857	-0.0791	...	-0.0203	-0.0221	-0.0615
2	-0.1453	-0.0741	-0.0709	...	-0.0181	-0.0215	-0.0597
...
317	-0.4587	-0.1595	-0.0988	...	-0.0854	-0.1005	-0.0941
318	-0.4653	-0.1558	-0.0886	...	-0.0899	-0.0915	-0.1299
319	-0.1593	0.2908	0.3288	...	0.0875	0.0908	0.0538

Setelah itu ada proses yang disebut *batch normalization*. Proses ini digunakan untuk mengnormalisasi input ke dalam sebuah lapisan (layer) agar memiliki rata-rata nol dan varians satu. Hal ini membantu dalam mempercepat pelatihan dan mengurangi masalah vanishing gradient atau exploding gradient pada jaringan saraf. Pada formulasi (2.4) memiliki variabel ϵ yang bernilai 0.001 yang mana nilai ini mengacu kepada *state of art* dari YOLOv5. Nilai bobot setelah melalui tahap batch normalization dapat dilihat pada Tabel 4.27.

Tabel 4.27 Bobot Chanel Pertama Setelah Batch Normalization

j\i	0	1	2	...	317	318	319
0	-0.9874	-1.146	-1.151	...	-0.6187	-0.5848	-0.7047
1	-1.045	-0.167	-0.08	...	0.6986	0.6742	0.1529
2	-0.9587	-0.01	0.028	...	0.7272	0.6822	0.177
...
317	-5.112	-1.145	-0.341	...	-0.164	-0.3643	-0.2793
318	-5.1992	-1.097	-0.2063	...	-0.224	-0.245	-0.7538
319	-1.1432	4.822	5.3259	...	2.127	2.17	1.68

Setelah melalui tahap *batch normalization*, bobot melewati proses fungsi aktivasi. Pada arsitektur YOLOv5, fungsi aktivasi yang digunakan adalah SiLU yang merupakan singkatan dari *Sigmoid Linear Unit*. Persamaan sigmoid dapat dilihat

pada persamaan (2.6). Untuk penjabaran perhitungan manual dapat dilihat pada penjabaran berikut:

$$\text{SiLU}(x) = x \cdot \frac{1}{1 + e^{-x}}$$

$$\text{SiLU}(-0.9874) = -0.9874 \cdot \frac{1}{1 + e^{0.9874}} = -0.2688$$

Hasil perhitungan bobot pada *chanel* pertama setelah melalui proses perhitungan fungsi aktivasi dapat dilihat pada Tabel 4.28.

Tabel 4.28 Bobot *Chanel* Pertama Setelah Fungsi Aktivasi SiLU

j\i	0	1	2	...	317	318	319
0	-0.268	-0.2764	-0.2766	...	-0.2166	-0.2093	-0.2331
1	-0.2719	-0.076	-0.03	...	0.4666	0.4467	0.0823
2	-0.2657	-0.007	0.014	...	0.4902	0.4532	0.0963
...
317	-0.0306	-0.2764	-0.1419	...	-0.075	-0.1493	-0.1203
318	-0.0285	-0.2746	0.0926	...	-0.09	-0.1075	-0.2412
319	-0.2763	4.7873	5.3001	...	1.9005	1.9486	1.4164

4.2.3.2 Perhitungan Manual Fungsi Loss

Algoritma YOLOv5 menggunakan 3 jenis loss function. Diantaranya adalah klasifikasi loss, box loss, dan objectness loss. Pada percobaan perhitungan manual kali ini melakukan perhitungan pada box loss dengan formulasi sebagai pada persamaan (4.7).

$$loss_{box} = \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^B I_{i,j}^{obj} b_j (2 - w_i \times h_i) \left[(x_i - \hat{x}_i^j)^2 + (y_i - \hat{y}_i^j)^2 + (w_i - \hat{w}_i^j)^2 + (h_i - \hat{h}_i^j)^2 \right] \quad (4.7)$$

Pada formula tersebut, S^2 merupakan variable dari jumlah *grid* yang terbentuk pada sebuah image yang sedang dideteksi. Sedangkan B merupakan variable dari jumlah bounding box yang terbentuk dari hasil prediksi. Sedangkan I merupakan variabel indeks yang menandakan apakah dari sebuah sel grid tersebut terdapat objek atau tidak. Terakhir λ_{coord} berupa nilai bobot yang menyesuaikan dampak dari loss koordinat. Pada perhitungan kali ini menggunakan data dummy berupa nilai berikut:

- Koordinat pusat prediksi: $x_i = 0.4, y_i = 0.3$
- Lebar-tinggi prediksi: $w_i = 0.07, h_i = 0.25$
- *Ground truth*: $x_{true} = 0.5, y_{true} = 0.4, w_i = 0.1, h_i = 0.3$

- $\lambda_{coord} = 5$

$$loss_{box} = 5 \sum_{i=0}^{S^2} \sum_{j=0}^B 0.242(2 - 0.1 \times 0.3) \left[(0.5 - 0.4)^2 + (0.3 - y_i^j)^2 + (0.1 - \hat{w}_i^j)^2 + (0.3 - \hat{h}_i^j)^2 \right] \quad (4.11)$$

$$loss_{box} = 0.202 \quad (4.12)$$

4.2.3.3 Perhitungan Manual Propagasi Mundur

Proses propagasi balik menggunakan turunan dari fungsi aktivasi setiap layer dan menerapkan chain rule yang bisa dilihat pada Persamaan Propagasi balik bertujuan mendapatkan nilai perubahan bobot yang sesuai berdasarkan propagasi maju yang telah dilakukan sebelumnya. Formulasi turunan pada fungsi aktivasi SiLU dan Batch Normalization dapat dilihat pada persamaan (4.13) dan (4.14).

$$SiLU\ derivative(x) = \frac{1 + e(-x) + x \cdot e(-x)}{(1 + e(-x))^2} \quad (4.13)$$

$$BatchNorm_d erivative = \frac{1}{\sqrt{\sigma^2 + \epsilon}} \quad (4.14)$$

Hasil bobot terbaru setelah melakukan perhitungan propagasi mundur dapat dilihat pada Tabel 4.29

Tabel 4.29 Chanel Indeks 0

j\i	0	1	2	...	317	318	319
0	-0.029	-0.119	0.395	...	0.034	0.436	0.209
1	0.546	-0.127	0.034	...	0.194	0.098	0.093
2	-0.001	0.016	-0.479	...	-0.107	0.655	-0.377
...
317	-0.336	-0.066	0.618	...	-0.350	-0.055	0.710
318	0.031	0.504	-0.309	...	0.194	-0.263	-0.297
319	0.250	0.466	0.199	...	0.527	0.447	-0.159

4.2.4 Perhitungan Manual Kalman Filter Predict

Kalman Filter (KF) adalah algoritma yang digunakan untuk memperkirakan keadaan (state) suatu sistem dinamis. Tahap prediksi (predict) dalam Kalman Filter bertanggung jawab untuk memperkirakan keadaan sistem pada langkah waktu berikutnya berdasarkan keadaan sebelumnya. Formulasi pada prediksi dengan

menggunakan kalman filter dapat dilihat pada persamaan (2.6) dan persamaan (2.7). Nilai matriks transisi atau F_k pada perhitungan manual ini menggunakan nilai berikut:

$$F_k = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Untuk nilai mean dalam percobaan perhitungan manual menggunakan salah satu dari tracks yang ada pada percobaan hasil deteksi sebelumnya dengan nilai seperti berikut ini:

$$\hat{x}_{k-1} = \{778, 156.5, 1.0728, 151, 0, 0, 0, 0\}$$

Sehingga hasil prediksi *mean* dapat dilakukan dengan melakukan perkalian matrik antara matriks transisi dengan nilai mean. Sehingga menghasilkan nilai berikut:

$$\hat{x}'_k = F_k \hat{x}_{k-1}$$

$$\hat{x}'_k = \{778, 156.5, 1.0728, 151, 0, 0, 0, 0\}$$

Kemudian untuk menghitung prediksi eror kovarian menggunakan persamaan (2.3). Pada percobaan perhitungan manual ini nilai matrik transisi atau F_k akan mengikuti nilai F_k sebelumnya. Kemudian pada nilai kovarian mengambil nilai sebuah nilai tracks dimana merupakan track yang sama yang diambil pada saat menghitung nilai mean sebelumnya. Nilai kovarian yang digunakan pada perhitungan manual ini adalah sebagai berikut:

$$P_{k-1} = \begin{pmatrix} 42.735 & 0 & 0 & 0 & 11.479 & 0 & 0 & 0 \\ 0 & 42.735 & 0 & 0 & 0 & 11.479 & 0 & 0 \\ 0 & 0 & 0.0004 & 0 & 0 & 0 & 9.2e-10 & 0 \\ 0 & 0 & 0 & 42.735 & 0 & 0 & 0 & 11.479 \\ 11.479 & 0 & 0 & 0 & 20.185 & 0 & 0 & 0 \\ 0 & 11.479 & 0 & 0 & 0 & 20.185 & 0 & 0 \\ 0 & 0 & 9.2e-10 & 0 & 0 & 0 & 5e-10 & 0 \\ 0 & 0 & 0 & 11.479 & 0 & 0 & 0 & 20.185 \end{pmatrix}$$

Ukuran dari matrik kovarian ini adalah 8x8, sehingga dapat digunakan untuk melakukan perkalian pada matrik transisi. Setelah itu, Matrik Q_k atau matrik Kovarian proses noise yang digunakan pada perhitungan manual ini adalah sebagai berikut:

$$Q_k = \begin{pmatrix} 57.003 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 57.003 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.0001 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 57.003 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 89066 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 89066 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1e-10 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.89066 \end{pmatrix}$$

Melaui perhitungan dengan menggunakan persamaan (2.3), maka didapatkan hasil covariansi error berikut ini:

$$F_k^T = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$P'_k = F_k P_{k-1} F_k^T + Q_K$$

$$P'_k = \begin{pmatrix} 279.93 & 0 & 0 & 0 & 75.01 & 0 & 0 & 0 \\ 0 & 279.93 & 0 & 0 & 0 & 75.01 & 0 & 0 \\ 0 & 0 & 0.0004 & 0 & 0 & 0 & 1.4e-09 & 0 \\ 0 & 0 & 0 & 279.93 & 0 & 0 & 0 & 75.01 \\ 75.01 & 0 & 0 & 0 & 30.38 & 0 & 0 & 0 \\ 0 & 75.01 & 0 & 0 & 0 & 30.38 & 0 & 0 \\ 0 & 0 & 1.4e-09 & 0 & 0 & 0 & 5e-10 & 0 \\ 0 & 0 & 0 & 75.01 & 0 & 0 & 0 & 30.38 \end{pmatrix} + Q_K$$

$$P'_k = \begin{pmatrix} 160.28 & 0 & 0 & 0 & 44.63 & 0 & 0 & 0 \\ 0 & 160.28 & 0 & 0 & 0 & 44.63 & 0 & 0 \\ 0 & 0 & 0.0004 & 0 & 0 & 0 & 9.6748e-10 & 0 \\ 0 & 0 & 0 & 160.28 & 0 & 0 & 0 & 44.63 \\ 44.63 & 0 & 0 & 0 & 30.38 & 0 & 0 & 0 \\ 0 & 44.63 & 0 & 0 & 0 & 30.38 & 0 & 0 \\ 0 & 0 & 9.6748e-10 & 0 & 0 & 0 & 5e-10 & 0 \\ 0 & 0 & 0 & 44.63 & 0 & 0 & 0 & 30.38 \end{pmatrix}$$

4.2.5 Perhitungan Manual Kalman Filter Project

Fungsi project ini melakukan proyeksi dari keadaan objek ke ruang pengukuran. Proses ini melibatkan perkalian matriks keadaan dan kovariansi dengan matriks proyeksi (self._update_mat) dan penambahan matriks kovariansi inovasi. Hasilnya adalah prediksi posisi objek di ruang pengukuran beserta ketidakpastian yang diperkirakan. Proyeksi digunakan pada tahap kalman filter update. Pada percobaan perhitungan, digunakan salah satu nilai mean dan nilai kovarian dari salah satu track yang telah dilakukan deteksi objek. Berikut nilai mean dan nilai kovarian yang digunakan.

$$mean = \{778, 156.5, 1.0728, 151, 0, 0, 0, 0\}$$

$$cov = \begin{pmatrix} 43.507 & 0 & 0 & 0 & 15.143 & 0 & 0 & 0 \\ 0 & 43.507 & 0 & 0 & 0 & 15.143 & 0 & 0 \\ 0 & 0 & 0.0003731 & 0 & 0 & 0 & 5.674e-10 & 0 \\ 0 & 0 & 0 & 43.507 & 0 & 0 & 0 & 15.143 \\ 15.143 & 0 & 0 & 0 & 29.49 & 0 & 0 & 0 \\ 0 & 15.143 & 0 & 0 & 0 & 29.49 & 0 & 0 \\ 0 & 0 & 5.674e-10 & 0 & 0 & 0 & 4e-10 & 0 \\ 0 & 0 & 0 & 15.143 & 0 & 0 & 0 & 29.49 \end{pmatrix}$$

Selain itu, ada juga matriks transformasi yang digunakan untuk memproyeksikan keadaan ke ruang pengukuran. Matrik ini disebut sebagai matrik update. Berikut nilai dari matrik update yang digunakan:

$$matrik\ update = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

Terakhir ada matrik kovariansi inovasi, yang mengukur ketidakpastian pengukuran atau pengamatan dari sensor. Matrik ini merupakan matrik diagonal dari vektor yang dikuadratkan dibawah ini:

$$std = [7.55, 7.55, 0.1, 7.55]$$

$$innovation_{cov} = \begin{pmatrix} 7.55 & 0 & 0 & 0 \\ 0 & 7.55 & 0 & 0 \\ 0 & 0 & 0.1 & 0 \\ 0 & 0 & 0 & 7.55 \end{pmatrix}$$

Nilai pada indek 0,1, dan 3 didapatkan dengan mengalikan nilai mean pada indek ke 3 dari *track* yang diproyeksi kemudian dikalikan dengan 0.05. Nilai dari 0.05 merupakan *state of art* dari implementasi kalman filter pada algoritma *DeepSORT*. Sedangkan indek *mean* ke-3 pada *track* yang digunakan mengacu kepada nilai skalar yang berkaitan dengan kecepatan atau perubahan kecepatan dari objek yang sedang diprediksi.

Formulasi yang digunakan untuk menghitung project dari masing-masing mean dan kovarian adalah sebagai berikut:

$$project_{mean} = matrik_{update} \cdot mean \quad (4.15)$$

$$project_{cov} = (matrik_{update} \cdot cov \cdot matrik_{update}^T) + innovation_{cov} \quad (4.16)$$

Sehingga hasil akhir yang didapatkan pada perhitungan masing-masing adalah sebagai berikut:

$$project_{mean} = [778, 156.5, 1.0728, 151]$$

$$project_{cov} = \begin{pmatrix} 100.51 & 0 & 0 & 0 \\ 0 & 100.51 & 0 & 0 \\ 0 & 0 & 0.010373 & 0 \\ 0 & 0 & 0 & 100.51 \end{pmatrix}$$

4.2.6 Perhitungan Manual Kalman Filter Update

Pada perhitungan kalman filter update, ada 3 nilai yang dilakukan perhitungan. Diantaranya adalah nilai matrik Kalman gain, Estimasi nilai mean, dan perbaruan matrik kovariansi. Pada perhitungan Kalman gain menggunakan faktorisasi Cholesky. Pada perhitungan manual kali ini nilai mean dan covariance yang digunakan masih sama dengan perhitungan Kalman filter project.

$$mean = \{778, 156.5, 1.0728, 151, 0, 0, 0, 0\}$$

$$cov = \begin{pmatrix} 43.507 & 0 & 0 & 0 & 15.143 & 0 & 0 & 0 \\ 0 & 43.507 & 0 & 0 & 0 & 15.143 & 0 & 0 \\ 0 & 0 & 0.0003731 & 0 & 0 & 0 & 5.674e-10 & 0 \\ 0 & 0 & 0 & 43.507 & 0 & 0 & 0 & 15.143 \\ 15.143 & 0 & 0 & 0 & 29.49 & 0 & 0 & 0 \\ 0 & 15.143 & 0 & 0 & 0 & 29.49 & 0 & 0 \\ 0 & 0 & 5.674e-10 & 0 & 0 & 0 & 4e-10 & 0 \\ 0 & 0 & 0 & 15.143 & 0 & 0 & 0 & 29.49 \end{pmatrix}$$

Selain itu, hasil perhitungan kalam filter project sebelumnya digunakan juga pada perhitungan untuk mendapatkan nilai Kalman gain. Lalu ada nilai *measurement* yang merupakan nilai dari sebuah object deteksi dari suatu track. Nilai *measurement* berupa sebuah vector yang berisi nilai titik tengah deteksi pada titik x, titik tengah deteksi pada titik y, aspek ratio, dan tinggi dari deteksi. *Measurement* digunakan untuk menghitung nilai update mean. Nilai *measurement* yang digunakan adalah sebagai berikut:

$$measurement = \{778, 156.5, 1.0728, 151\}$$

Hasil dari perhitungan Kalman gain dengan persamaan (2.4) adalah sebagai berikut:

$$K = \begin{pmatrix} 0.43287 & 0 & 0 & 0 \\ 0 & 0.43287 & 0 & 0 \\ 0 & 0 & 0.035969 & 0 \\ 0 & 0 & 0 & 0.43287 \\ 0.15066 & 0 & 0 & 0 \\ 0 & 0.15066 & 0 & 0 \\ 0 & 0 & 5.4706e-08 & 0 \\ 0 & 0 & 0 & 0.15066 \end{pmatrix}$$

Setelah mendapatkan nilai dari Kalman gain, maka dapat dihitung nilai dari *mean* dan *covarian* terbaru.

$$innovation = measurement - project_{mean}$$

$$innovation = \{0, 0, 0, 0\}$$

$$\begin{aligned}
new_{mean} &= mean + innovation \cdot K^T \\
new_{mean} &= \{778, \quad 156.5, \quad 1.0728, \quad 151, \quad 0, \quad 0, \quad 0, \quad 0\} \\
new_{cov} &= cov - (K \cdot project_{cov} \cdot K^T) \\
new_{cov} &= \begin{pmatrix} 57.003 & 0 & 0 & 0 \\ 0 & 57.003 & 0 & 0 \\ 0 & 0 & 0.01 & 0 \\ 0 & 0 & 0 & 57.003 \end{pmatrix}
\end{aligned}$$

4.2.7 Perhitungan Manual Matriks Biaya **Matching Cascade**

Pada perhitungan matrik biaya pada proses *matching cascade* pertama-tama menggunakan formulasi *cosine distance* untuk menghitung jarak dari fitur dan masing-masing sampel dari fitur. Pada *track* yang telah terbentuk memiliki ukuran matrik 7×1280 dan sample untuk masing-masingnya adalah 1×7 .

$$\begin{aligned}
cosine_{distance} &= 1 - \frac{a \cdot b^T}{\|a\| \|b\|} \tag{4.17} \\
features &= \begin{pmatrix} 0.075387 \ 0.001618 : 0.14495 \ 0.50864 \\ 0.022283 \ 0.043879 : 0.42603 \ 0.10204 \\ 0.2419 \ 0.33243 : 0.0665 \ 0.37673 \\ 0.62396 \ 0.025597 : 0 \ 0.33918 \\ 0.70059 \ 0.2212 : 0.11516 \ 0.45626 \\ 0.80901 \ 0.054984 : 0.0243560 \ 0.45082 \\ 0.42118 \ 0.036621 : 0.0214 \ 0.1951 \end{pmatrix} \\
sample[0] &= \{0.075387, \quad 0.001618, \quad \dots, \quad 0.14495, \quad 0.50864\}
\end{aligned}$$

Nilai fitur akan di-*transpose*-kan agar dapat melakukan proses perkalian matrik dengan nilai sample. Berikut hasil dari nilai cosine yang dipapatkan. Namun, sebelum melakukan perhitungan *cosine distance* perlu dilakukan normalisasi pada feature dan sample yang digunakan. Tujuan dari normalisasi ini adalah untuk memastikan konsistensi skala dan mengatasi outlier. Teknik normalisasi yang digunakan pada *features* dan *sample[0]* adalah normalisasi Euclidean. Sehingga hasil normalisasi menjadi seperti berikut ini:

$$\begin{aligned}
features &= \begin{pmatrix} 0.0036378 \ 7.8079e-05 : 0.0069948 \ 0.024545 \\ 0.00098161 \ 0.0019329 : 0.018767 \ 0.0044951 \\ 0.0089132 \ 0.012249 : 0.0024503 \ 0.013881 \\ 0.034024 \ 0.0013958 : 0 \ 0.018495 \\ 0.034403 \ 0.010862 : 0.0056552 \ 0.022405 \\ 0.038014 \ 0.0025836 : 0.0011444 \ 0.021183 \\ 0.017659 \ 0.0015354 : 0.00089724 \ 0.0081799 \end{pmatrix} \\
sample[0] &= \{0.0036378, \quad 7.8079e-05, \quad \dots, \quad 0.0069948, \quad 0.024545\}
\end{aligned}$$

$$\begin{aligned} cosine_distance[0] \\ = [1.1921e \\ - 07, 0.40843, 0.50448, 0.32701, 0.27255, 0.30469, 0.35642] \end{aligned}$$

Setelah melakukan perhitungan *cosine distance* untuk semua nilai sample atau jumlah objek yang terdeteksi dalam satu frame, maka tahap selanjutnya adalah menghitung *gate cost metric*. Nantinya nilai dari *gate cost metric* ini akan dilanjutkan kedalam perhitungan dengan menggunakan algoritma Hungarian atau Munkers. Perhitungan *gate cost metric* akan menggunakan nilai *mean* dan kovarian dari masing-masing *track* yang telah terbentuk sebelumnya dan kemudian dihitung nilai proyeksinya dengan menggunakan proses Kalman filter project.

$$\begin{aligned} cholesky_{factor(project_{cov})} &= \begin{pmatrix} \sqrt{100.51} & 0 & 0 & 0 \\ 0 & \sqrt{100.51} & 0 & 0 \\ 0 & 0 & \sqrt{0.010373} & 0 \\ 0 & 0 & 0 & \sqrt{100.51} \end{pmatrix} \\ cholesky_{factor(project_{cov})} &= \begin{pmatrix} 10.025 & 0 & 0 & 0 \\ 0 & 10.025 & 0 & 0 \\ 0 & 0 & 0.10185 & 0 \\ 0 & 0 & 0 & 10.025 \end{pmatrix} \\ cholesky_{factor(project_{cov})} \cdot \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{pmatrix} &= innovation \end{aligned}$$

Dikarenakan nilai dari matrik *innovation* adalah semuanya 0. Maka nilai dari matriks *y* dengan menggunakan formulasi triangular otomatis nilainya adalah 0 sama dengan matriks *innovation*. Sehingga Panjang vektornya adalah 0. Lalu nilai dari masing-masing dari *cosine distance* akan dibandingkan dengan nilai panjang vector yang didapatkan atau 0. Jika nilai kecil dari 0 maka akan tetap mempertahankan nilainya. Jika tidak maka akan diubah menjadi 1e5 yang mana menjadi angka yang menyamakan pada nilai lainnya jika ada angka yang melebihi dari nilai 0 tersebut. Sehingga hasil dari *gate cost metric* adalah berikut ini:

$$\begin{aligned} gatecostmetric[0] \\ = [1.1921e - 07, 1e + 5, 1e + 5] \end{aligned}$$

Setelah itu lakukan perhitungan algoritma Hungarian dengan hasil seluruh *gate cost metric* yang didapatkan dalam satu frame.

$$\begin{aligned} gate cost metric \\ = \begin{pmatrix} 1.19e - 7 & 1e + 05 \\ 1e + 05 & 5.96e - 8 & 1e + 05 \\ 1e + 05 & 1e + 05 & 5.96e - 8 & 1e + 05 & 1e + 05 & 1e + 05 & 1e + 05 \\ 1e + 05 & 1e + 05 & 1e + 05 & 1.19e - 7 & 1e + 05 & 1e + 05 & 1e + 05 \\ 1e + 05 & 1e + 05 & 1e + 05 & 1e + 05 & 1.78e - 7 & 1e + 05 & 1e + 05 \\ 1e + 05 & 1.19e - 7 & 1e + 05 \\ 1e + 05 & 1.78e - 0 \end{pmatrix} \end{aligned}$$

Gate cost metric masih perlu diperiksa lagi sesuai dengan konfigurasi yang diinginkan. Pemeriksaan berupa jika ada nilai dari *gate cost metric* melebihi dari nilai jarak maksimal yang ditentukan maka nilanya akan diubah menjadi nilai jarak yang ditentukan ditambah dengan 1e-5. Nilai jarak maksimal yang ditetapkan dalam perhitungan manual ini adalah 0.2. Sehingga *gate cost metric* akan berubah lagi menjadi seperti berikut:

$$\text{gate cost metric} = \begin{pmatrix} 1.19e-7 & 0.20001 & 0.20001 & 0.20001 & 0.20001 & 0.20001 & 0.20001 \\ 0.20001 & 5.96e-8 & 0.20001 & 0.20001 & 0.20001 & 0.20001 & 0.20001 \\ 0.20001 & 0.20001 & 5.96e-8 & 0.20001 & 0.20001 & 0.20001 & 0.20001 \\ 0.20001 & 0.20001 & 0.20001 & 1.19e-7 & 0.20001 & 0.20001 & 0.20001 \\ 0.20001 & 0.20001 & 0.20001 & 0.20001 & 1.78e-7 & 0.20001 & 0.20001 \\ 0.20001 & 0.20001 & 0.20001 & 0.20001 & 0.20001 & 1.19e-7 & 0.20001 \\ 0.20001 & 0.20001 & 0.20001 & 0.20001 & 0.20001 & 0.20001 & 1.78e-0 \end{pmatrix}$$

Setelah itu carilah pasangannya berdasarkan algoritma Hungarian. Yang mana menghasilkan berikut ini (0, 0), (1, 1), (2, 2), (3, 3), (4, 4), (5, 5), (6, 6). Dapat dilihat bahwa dalam pelacakan selanjutnya tidak ada pelacakan yang hilang yang mana hasil dari perhitungan ini adalah semua nilai dari objek yang dideteksi termasuk kedalam jenis *matches* dalam alur matching cascade.

4.2.8 Perhitungan Manual *IoU Match*

Perhitungan *IoU match* sangatlah mirip dengan proses matching cascade. Yang membedakannya adalah pada metrik yang digunakan. Jika *matching cascade* menggunakan metric perbandingan jarak kosinus. Maka pada matching cascade memiliki metric perhitungan jaraknya sendiri yang mana mempertimbangkan bounding box sebagai penentu jaraknya. IoU sendiri menggunakan parameter dari *unmatched detections* dan *unmatched track* dari hasil perhitungan pada *matching cascade*. Perhitungan manual pada bagian ini hanya berfokus pada *cost metric* yang digunakan pada perhitungan IoU Match. Formula yang digunakan untuk menghitung IoU adalah sebagai berikut:

$$IoU = \frac{\text{area intersection}}{\text{area bounding box} + \text{area candidate} - \text{area intersection}} \quad (4.18)$$

Nilai yang digunakan pada percobaan perhitungan manual adalah sebagai berikut:

$$\text{candidate} = \begin{pmatrix} 697 & 81 & 162 & 151 \\ 1246 & 774 & 358 & 306 \\ 1013 & 349 & 189 & 503 \\ 1029 & 249 & 84 & 204 \\ 946 & 112 & 161 & 141 \\ 1004 & 159 & 80 & 186 \\ 177 & 134 & 89 & 124 \end{pmatrix}$$

$$\text{track}[0]_{bbox} = (1246 \quad 774 \quad 358 \quad 306)$$

Candidate merupakan kumpulan dari seluruh *bounding box* yang ada dalam satu frame. Jadi nilai *track* yang digunakan akan dihitung satu persatu untuk menghitung nilai IoU. Dari nilai *bounding box* pada *track*, ada informasi lainnya yang dapat kita ketahui. Informasi yang tersimpan adalah titik teratas kiri x, titik teratas kiri y, lebar, dan tinggi. Peletakan informasi ini juga sama pada nilai matrik *candidate*. Sehingga dapat kita jabarkan lagi nilai pada *track* seperti ini:

$$top\ left = (1246 \quad 774)$$

$$bottom\ right = (1604 \quad 1080)$$

$$candidate_{top\ left} = \begin{pmatrix} 697 & 81 \\ 1246 & 774 \\ 1013 & 349 \\ 1029 & 249 \\ 946 & 112 \\ 1004 & 159 \\ 177 & 134 \end{pmatrix}$$

$$candidate_{bottom\ right} = \begin{pmatrix} 859 & 232 \\ 1604 & 1080 \\ 1202 & 852 \\ 1113 & 453 \\ 1007 & 253 \\ 1084 & 345 \\ 266 & 258 \end{pmatrix}$$

Untuk memastikan adanya timpaan pada sebuah deteksi yang sedang dihitung dengan deteksi lainnya yang tersimpan dalam matrik *candidate*, Maka lakukan perbandingan nilai maksimal antara *top left* dari masing-masing matrik *candidate* dengan nilai *top left* dari *track* dan perbandingan nilai minimum antara nilai *bottom right* dari masing-masing matrik *candidate* dengan nilai *bottom left* dari *track*. Sehingga didapatkan nilai berikut.

$$true\ top\ left = \begin{pmatrix} 1246 & 774 \\ 1246 & 774 \\ 1246 & 774 \\ 1246 & 774 \\ 1246 & 774 \\ 1246 & 774 \\ 1246 & 774 \end{pmatrix}$$

$$true\ bottom\ right = \begin{pmatrix} 859 & 232 \\ 1604 & 1080 \\ 1202 & 852 \\ 1113 & 453 \\ 1007 & 253 \\ 1084 & 345 \\ 266 & 258 \end{pmatrix}$$

Setelah itu lakukan perhitungan tinggi dan lebar dengan menggunakan rumus *true bottom right – true top left*. Jika nilainya kurang dari 0. Maka simpanlah nilai 0 bukan nilai selisihnya. Ini digunakan untuk mencari nilai yang overlap saja. Sehingga nilai dari width dan height adalah sebagai berikut:

$$true\ width\ height = \begin{pmatrix} 0 & 0 \\ 358 & 306 \\ 0 & 78 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix}$$

Dari nilai *true width height* kita bisa mendapatkan nilai dari *area intersection*. Nilai dari *area intersection* sendiri merupakan nilai perkalian *width* dan *height* dari masing-masing matrik tersebut. Sedangkan nilai dari *area bounding box* merupakan perkalian dari tinggi dan lebar dari *bounding box* pada track yang sedang diperhitungkan IoU-nya. Sedangkan *area candidate* adalah nilai perkalian dari tinggi dan lebar pada matrik *candidate* sebelumnya. Terjadinya perkalian panjang dan lebar ini dilakukan untuk menggambarkan area yang sedang terdeteksi dan melakukan perhitungan dengan perbandingan dari *bounding box* yang sedang diperhitungkan.

$$area\ intersection = (0 \ 1.0955e + 05 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0)$$

$$area\ bounding\ box = 1.0955e + 05$$

$$areaintersection = (24462, 1.0955e + 05, 95067, 17136, 8601, 14880, 11036)$$

$$IoU = (0, 1, 0, 0, 0, 0, 0)$$

Kemudian untuk mendapatkan nilai matrik *cost* pada IoU. Dilakukan perhitungan sebagai berikut:

$$cost = \sum_{i=0}^{tracking\ detection} \sum_{j=0}^{tracking\ detection} 1 - IoU_{i,j} \quad (4.19)$$

Perhitungan IoU dilakukan terhadap seluruh objek yang terdeteksi dan yang telah tertandai dengan *tracking*. Sehingga hasil perhitungan ini menghasilkan hasil akhir seperti ini:

$$cost = \begin{pmatrix} 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0.91557 & 1 & 1 & 1 \\ 1 & 1 & 0.91557 & 0 & 1 & 0.80251 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0.98784 & 1 \\ 1 & 1 & 1 & 0.80251 & 0.98784 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 \end{pmatrix}$$

4.3 Perancangan Pengujian

Ada beberapa hal yang diuji pada penelitian ini untuk menjawab dari rumusan masalah yang ditentukan. Diantaranya adalah evaluasi model objek deteksi terhadap penggunaan algoritma YOLOv5, evaluasi pelakan objek dengan penggunaan algoritma *DeepSORT*, dan terakhir akurasi perhitungan kendaraan masuk. Tabel yang digunakan dalam tahapan pengujian, evaluasi, dan analisis dapat dilihat pada Tabel 4.30, Tabel 4.31 dan Tabel 4.32:

Tabel 4.30 Tabel Evaluasi Deteksi Objek

No.	Preproses Gambar	Precision	Recall	F1-Score	Best confidence

Tabel 4.31 Tabel Evaluasi Pelacakan Objek

No.	Video	Preproses Gambar	MOTA	MOTP

Tabel 4.32 Tabel Evaluasi Perhitungan Objek

No.	Video	Ground Truth		Preproses Gambar	TP	FP	FN	Precision	Recall	Akurasi	F1 score
		Mobil	Motor								

BAB 5 IMPLEMENTASI

5.1 Implementasi Seleksi *Frame*

Frame diambil secara acak sebelum melakukan anotasi objek gambar. Berikut kode yang digunakan untuk mengambil frame yang akan dianotasikan dapat dilihat pada Kode Program 5.1.

Kode Program 5.1 Seleksi Frame

```
1. def get_frame_video(src_path, dst_path, list_name_video):
2.     os.makedirs(dst_path, exist_ok=True)
3.     name_path_video = [
4.         os.path.join(src_path, name)
5.         for name in list_name_video
6.         if name.endswith('.mkv')]
7.     frames_per_video = 50
8.     for video_file in name_path_video:
9.         cap = cv2.VideoCapture(video_file)
10.        frame_count = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
11.        for _ in range(frames_per_video):
12.            frame_number = random.randint(0, frame_count - 1)
13.            cap.set(cv2.CAP_PROP_POS_FRAMES, frame_number)
14.            ret, frame = cap.read()
15.            if ret:
16.                video_file_name = os.path.basename(video_file)
17.                frame_output_path = os.path.join(dst_path,
18.                    f'{video_file_name}_frame_{frame_number}.jpg')
19.                cv2.imwrite(frame_output_path, frame)
20.        cap.release()
```

Penjelasan masing-masing baris pada Kode Program 5.1 adalah sebagai berikut:

1. Baris 1 berfungsi sebagai deklarasi suatu fungsi bernama ‘get_frame_video’ dengan menerima 3 parameter berupa sumber direktori *path* video, tujuan *path* direktori untuk menyimpan video, dan *list* dari nama video yang akan diambil *frame*-nya.
2. Baris 2 berfungsi untuk membuat direktori tujuan penyimpanan jika misalnya direktori tersebut belum ada.
3. Baris 3 akan mendeklarasikan sebuah variable ‘name_path_video’ untuk menyimpan *path* lengkap dari sumber video pada variable

'list_name_video' dan memeriksa bahwa file yang diterima berformat '.mkv'.

4. Baris 7 akan menginisiasi jumlah *frame* yang akan diambil secara acak dari sebuah video.
5. Baris 8-10 berfungsi melakukan looping dari masing-masing video dari *list name_path_video* dan menangkap seluruh frame disetiap videonya
6. Baris 11-14 berfungsi mengambil jumlah *frame* video secara acak dengan jumlah 50 *frame* untuk masing-masing video.
7. Baris 15-19 berfungsi untuk melakukan penamaan dan penyimpanan frame menjadi gambar dengan memberikan 'frame_number' untuk mengetahui sebuah gambar diambil dari *frame* ke berapa dari sebuah video.
8. Baris 20 berfungsi untuk melepaskan sumber daya yang digunakan untuk menjalankan tugas pembacaan video dari fungsi 'cv2.VideoCapture'.

5.2 Implementasi Anotasi Gambar

Anotasi gambar menggunakan sebuah aplikasi label studio yang dijalankan dengan menggunakan *container* docker. Tata cara instalasi dapat dilakukan mengikuti petunjuk pada website labelstud.io. Objek yang akan dianotasikan ke pada penelitian ini hanyalah objek kendaraan motor dan mobil pada malam hari. Langkah-langkah penggunaan aplikasi studio dapat dilakukan dengan mengikuti tahapan berikut:

1. Unduh docker melalui *website* resmi pada docs.docker.com lalu jalankan program dan lakukan instalasi dan pembuatan akun.
2. Jalankan *script* pada Kode Program 5.2 pada *command prompt*.

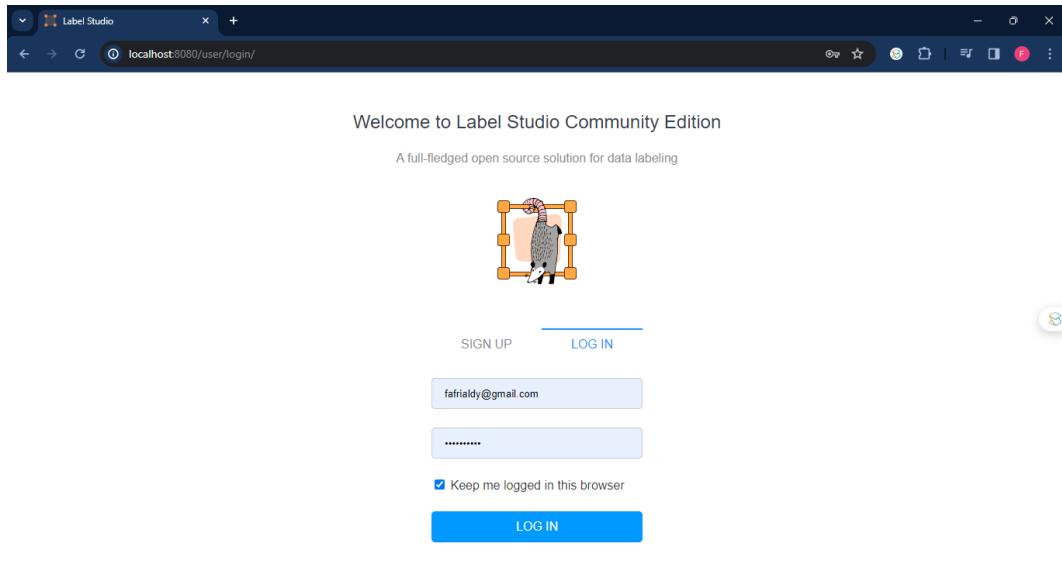
Kode Program 5.2 Script Instalasi dan Menjalankan Label Studio pada Docker

1.	docker pull heartexlabs/label-studio:latest
2.	docker run -it -p 8080:8080 -v `pwd`/mydata:/label-studio/data heartexlabs/label-studio:latest

Penjelasan masing-masing baris pada Kode Program 5.2 adalah sebagai berikut:

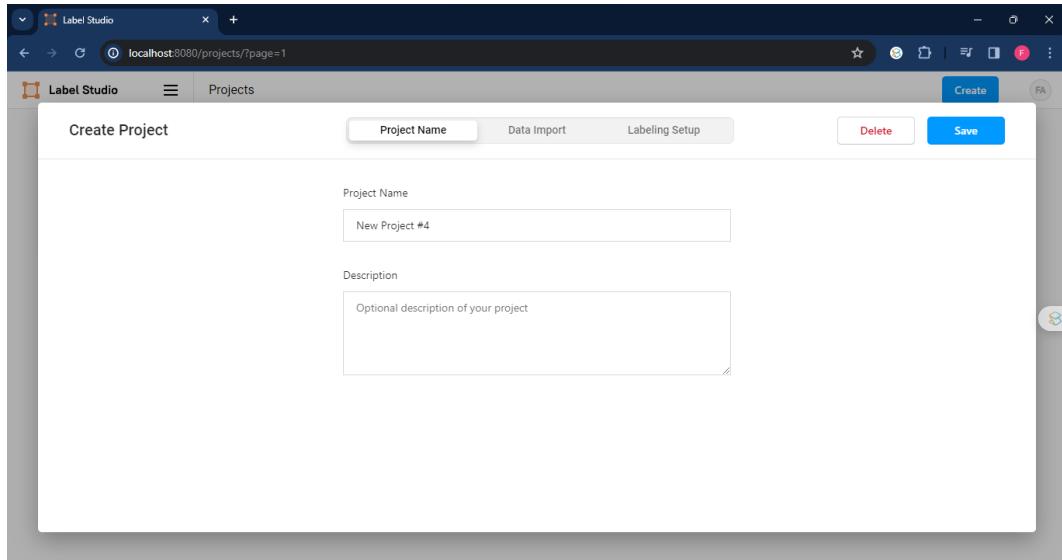
- 2.a. Baris 1 berfungsi untuk melakukan pengunduhan program label studio ke dalam *container* docker
- 2.b. Baris 2 berfungsi untuk menjalankan program label studio. Program dapat dijalankan dengan membuka aplikasi browser dan memasukkan *link* <http://localhost:8080/>

3. Langkah selanjutnya berupa pembuatan akun jika belum memiliki akun pada label studio atau melakukan *sign in* jika sudah memiliki akun pada label studio. Tampilan label studio dapat dilihat pada Gambar 5.1.



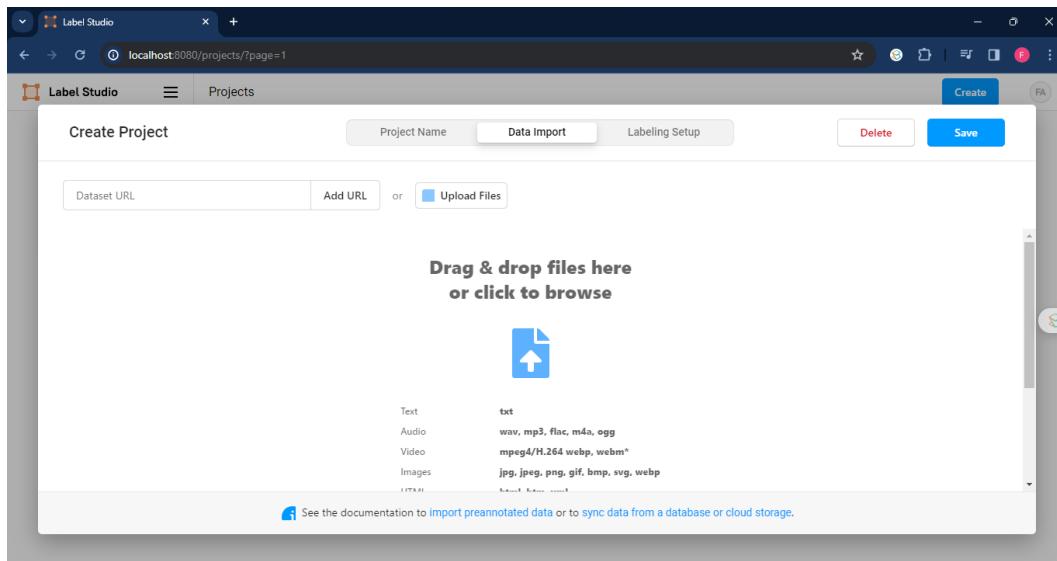
Gambar 5.1 Tampilan Awal dari Label Studio

4. Setelah itu klik tombol create untuk memulai *project* baru. Kemudian isilah bagian *Project Name*, *Data Import*, dan *Labeling Setup*. Tampilan dapat dilihat pada Gambar 5.2.



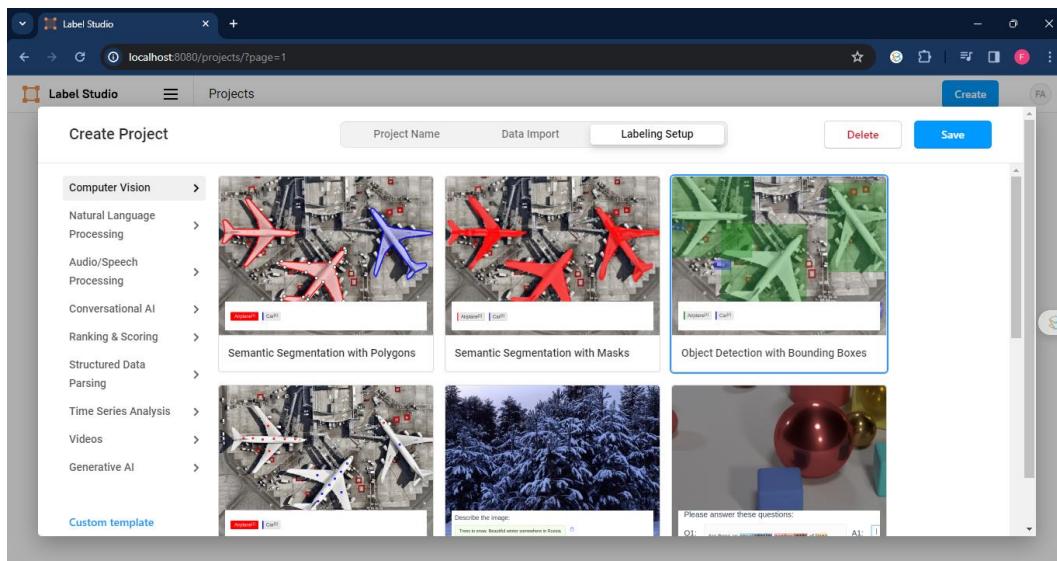
Gambar 5.2 Pembuatan *Project* baru

5. *Project Name* digunakan untuk menentukan nama project yang akan dibuat dan memberikan deskripsi projek. *Data Import* berupa memasukan data gambar yang nantinya akan dilakukan anotasi gambar. Tampilan dapat dilihat pada Gambar 5.3.



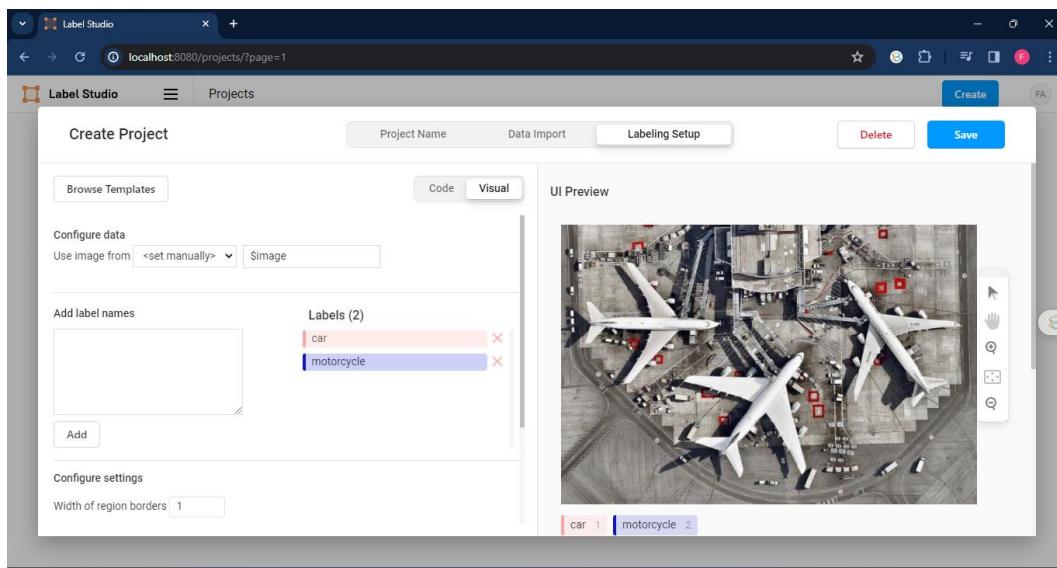
Gambar 5.3 Tampilan *Data Import*

6. Kemudian pada bagian *Labeling Setup* berupa jenis pelabelan yang akan dilaksanakan pada penggunaan label studio. Pada penelitian ini akan memilih *Object Detection with Bounding Boxes*. Tampilan dapat dilihat pada Gambar 5.4.



Gambar 5.4 Tampilan *Labeling Setup*

7. Setelah memilih jenis pelabelan yang akan dijalankan pada program label studio. Selanjutnya adalah menentukan kelas yang akan dilabelkan. Pada penelitian ini, nama kelas yang akan digunakan adalah mobil dan motor. Tampilan inisiasi kelas dapat dilihat pada Gambar 5.5. Setelah itu klik *save* dan klik pada projek yang telah dibuat.



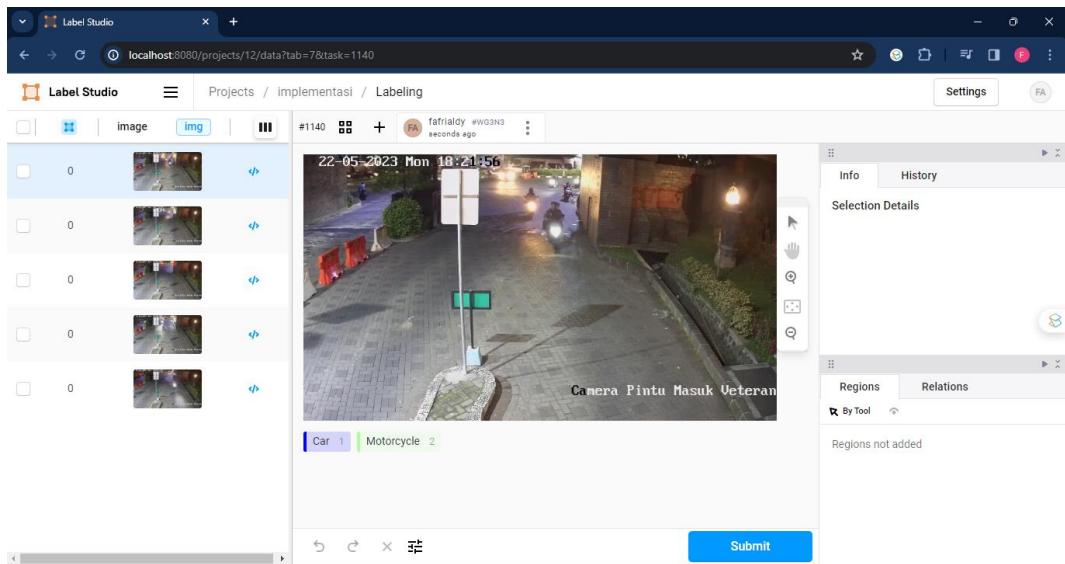
Gambar 5.5 Inisiasi Kelas Pelabelan

8. Tampilan yang akan diberikan setelah membuka projek tadi akan memberikan tampilan yang dapat dilihat pada Gambar 5.6.

ID	Completed	0	0	0	image	img
1140		0	0	0		
1141		0	0	0		
1142		0	0	0		
1143		0	0	0		
1144		0	0	0		

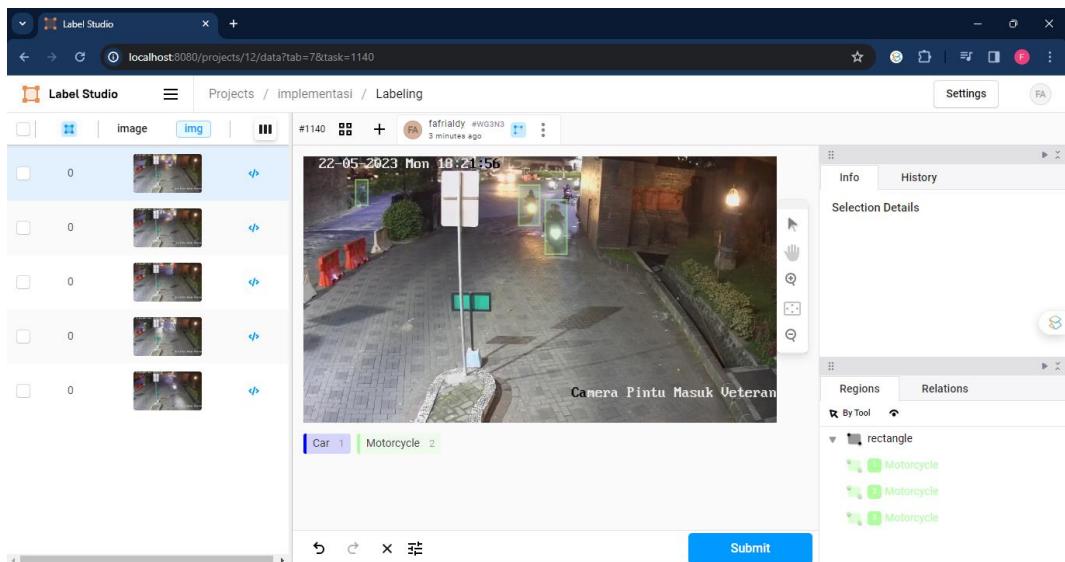
Gambar 5.6 Tamplan pada Projek yang Telah dibuat

9. Untuk memulai anotasi objek, kliklah salah satu gambar tersebut nantinya akan memunculkan tampilan seperti pada Gambar 5.7.



Gambar 5.7 Tampilan Tempat Anotasi Gambar

- Untuk melakukan anotasi sesuai dengan nama kelasnya, bisa menggunakan *shortcut* sesuai dengan id pada kelas. Untuk kelas ‘Car’ dapat klik 1 sedangkan kelas ‘Motorcycle’ klik tombol 2 pada keyboard. Setelah melakukan klik maka buatlah sebuah kotak atau *bounding box* pada setiap objek yang ada pada sebuah gambar. Hasil anotasi dapat dilihat pada Gambar 5.8.

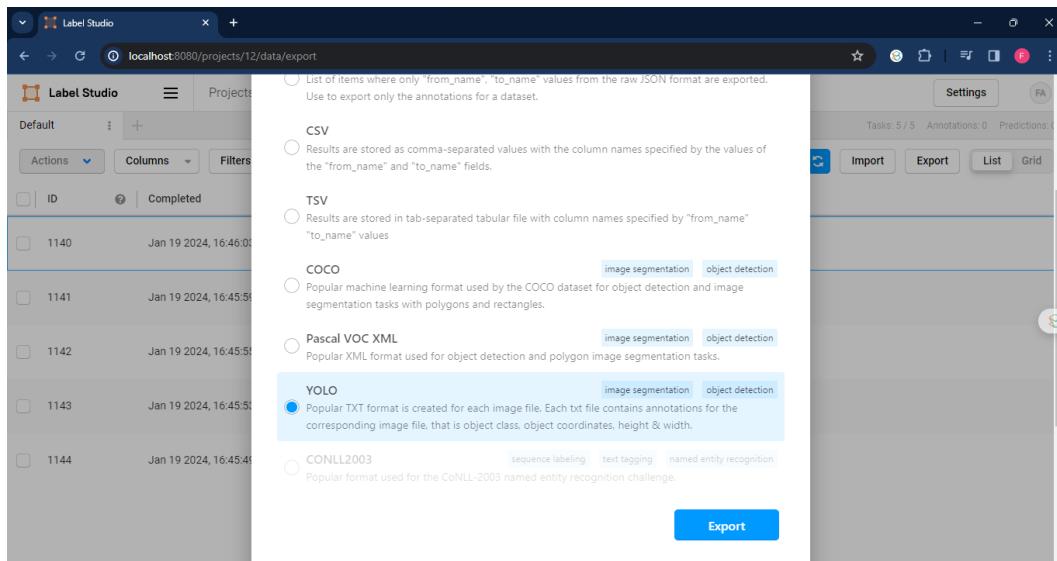


Gambar 5.8 Anotasi Objek

Setelah memberikan *bounding box* dari masing-masing objek yang ada, klik tombol ‘submit’ untuk menyimpan seluruh nilai anotasi dari masing-masing frame

- Setelah melakukan anotasi keseluruhan gambar, langkah selanjutnya melakukan *export*. Untuk melakukan tahap ini kembalilah ke tampilan project sebelumnya seperti pada tampilan Gambar 5.6. Lalu klik tombol

'export'. Nantinya akan muncul tampilan seperti pada Gambar 5.10. Dikarenakan penelitian ini akan menggunakan algoritma YOLOv5 untuk melakukan objek deteksi, maka format yang tepat adalah format YOLO. Kemudia klik tombol 'export' untuk melakukan pengunduhan dan akan menerima file dengan format zip.



Gambar 5.9 Pemilihan Format Export

12. Pada file zip tersebut akan berisi direktori *image* berupa gambar original sesuai dengan gambar dalam melakukan *import*, direktori *labels* berupa kordinat *bounding box* dengan masing-masing kelas pada setiap gambar dengan nama yang sama dengan nama file gambar original dengan tipe data berupa *file txt*, dan terakhir *file 'classes.txt'* yang berisikan nama kelas.

5.3 Implementasi Dataset Segmentasi *flare*

Pada penelitian ini dilakukan percobaan dengan menggunakan metode preprocessing gambar yang menggunakan metode *machine learning* berupa *image segmentasi*. Dataset yang digunakan untuk melakukan pelatihan model menggunakan dataset yang bersifat sekunder yang bersumber dari penelitian Esfahani dan Wang (2021) yang bertujuan untuk membuat dataset segmentasi *flare* cahaya. Berikut kode yang digunakan dalam melakukan pembagian dataset menjadi data latih dan data validasi yang akan digunakan dalam penelitian ini dapat dilihat pada Kode Program 5.3.

Kode Program 5.3 Kode Preprocess Dataset Flare

```

1. def get_learning_preprocess_dataset(src_path):
2.     image_paths = []
3.     mask_paths = []
4.     for folder in os.listdir(src_path):
5.         image_folder = os.path.join(

```

```

6.         src_path,
7.         folder,
8.         'images')
9.     mask_folder = os.path.join(
10.        src_path,
11.        folder,
12.        'masks')
13.     for image_file in os.listdir(image_folder):
14.         image_path = os.path.join(
15.             image_folder,
16.             image_file)
17.         mask_path = os.path.join(
18.             mask_folder,
19.             f"{folder}_GT.jpg")
20.         image_paths.append(image_path)
21.         mask_paths.append(mask_path)
22.     for i, mask_path in enumerate(mask_paths):
23.         img_temp = Image.open(mask_path).convert("L")
24.         threshold = 1
25.         img_binary = img_temp.point(lambda p:
26.             p > threshold and 255)
27.         binarized_mask_path = os.path.join(
28.             os.path.dirname(mask_path),
29.             f"binarized_{os.path.basename(mask_path)}")
30.         img_binary.save(binarized_mask_path)
31.         mask_paths[i] = binarized_mask_path
32.     image_train, image_val, \
33.     mask_train, mask_val = train_test_split(
34.         image_paths,
35.         mask_paths,
36.         test_size=0.3,
37.         random_state=42
38.     )
39.     print(f"Number of training samples: {len(image_train)}")
40.     print(f"Number of validation samples: {len(image_val)}")
41.     return image_train, mask_train, image_val, mask_val

```

Penjelasan masing-masing baris pada Kode Program 5.3 adalah sebagai berikut:

1. Baris 1 berfungsi untuk mendeklarasikan sebuah fungsi Bernama ‘get_learning_preprocess_dataset’ dengan menerima satu parameter berupa sumber *path* dari direktori gambar *flare* pada dataset Esfahani dan Wang (2021).
2. Baris 2 dan 3 akan mendeklarasikan variable untuk menyimpan *path* dari masing-masing *images* dan *mask* dari sumber direktori.
3. Baris 4-21 berfungsi untuk menyimpan *path* gambar original ke dalam variabel *list* ‘image_path’ dan menyimpan *path mask* dari masing-masing gambar original ke dalam variable *list* ‘mask_path’.
4. Baris 22-31 berfungsi untuk melakukan proses *theresholding* pada gambar *mask*. Ini dikarenakan sumber dataset memiliki 2 kelas dari *masking* gambar. Kelas itu berupa sumber cahay dan *flare* yang dihasilkan. Pada penelitian ini sumber cahaya dan *flare* yang dihasilkan akan dianggap sama.
5. Baris 32-38 berfungsi untuk mengacak gambar original dan gambar *mask* dengan menggunakan *library* dari scikit-learn. Perbandingan pembagian antara data latih dan data validasi berjumlah 70% dan 30 %. Inisiasi ‘random_state’ digunakan untuk memberikan nilai konsistensi dari keacakan jika misalnya sebuah kode tersebut akan dijalankan kembali.
6. Baris 39-40 akan memberikan output berupa informasi jumlah data latih dan data validasi.
7. Baris 41 akan memberikan kembalian atau *return value* berupa *path* dari masing-masing *image_train*, *mask_train*, *image_val*, *mask_val*.

5.4 Implementasi Struktur Direktori

Untuk menyimpan gambar setelah melalui tahapan pembagian dataset menjadi data latih dan data validasi. Selanjutnya dapat melakukan penyimpanan gambar dengan menggunakan *script* pada Kode Program 5.4.

Kode Program 5.4 Script Penyusunan Direktori

```
1. os.mkdir('dataset_flare')
2. os.mkdir('dataset_flare/train')
3. os.mkdir('dataset_flare/val')
4. os.mkdir('dataset_flare/train/image')
5. os.mkdir('dataset_flare/train/mask')
6. os.mkdir('dataset_flare/val/image')
7. os.mkdir('dataset_flare/val/mask')
8. for file_image, file_mask in zip(image_train, mask_train):
9.     shutil.copy(file_image, 'dataset_flare/train/image')
10.    shutil.copy(file_mask, 'dataset_flare/train/mask')
11.    for file_image, file_mask in zip(image_val, mask_val):
```

```

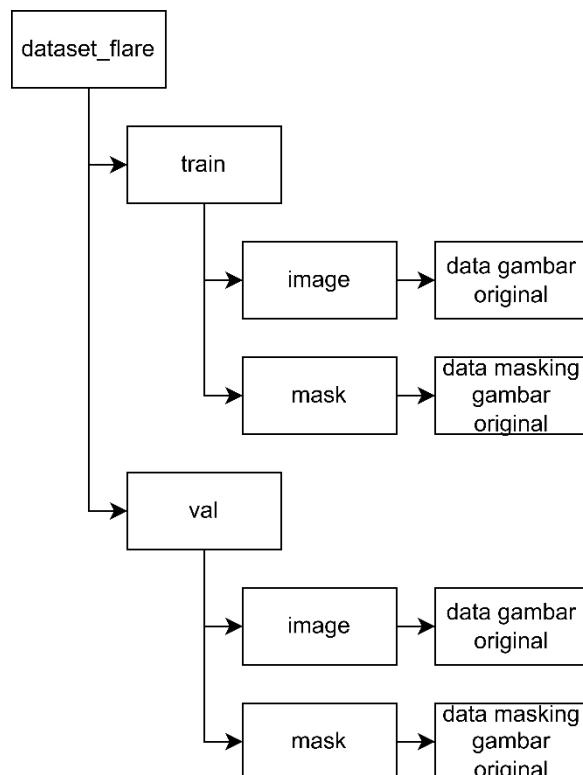
12.     shutil.copy(file_image, 'dataset_flare/val/image')
13.     shutil.copy(file_mask, 'dataset_flare/val/mask')

```

Penjelasan masing-masing baris pada Kode Program 5.4 adalah sebagai berikut:

1. Baris 1-7 akan membuat direktori bari sesuai dengan nama *path* yang dituju.
2. Baris 8-13 akan menyimpan variable dari keluaran pada fungsi ‘get_learning_preprocess_dataset’ ke dalam direktori yang sudah dibuat dan ditentukan.

Struktur direktori yang dihasilkan dengan menjalankan *script* dapat dilihat pada Gambar 5.10.



Gambar 5.10 Struktur Direktori Data *Preprocessing Learning*

5.5 Implementasi Model U-Net

Pembuatan model U-Net mengikuti arsitektur yang ada pada Gambar 2.4. Penulisan model U-Net dapat dilihat pada Kode Program 5.5.

Kode Program 5.5 Kode Model U-NET

```

1. class UNET(nn.Module):
2.     def __init__(self, n_class):
3.         super().__init__()
4.

```

```

5.          self.e11 = nn.Conv2d(3, 64,
6.                          kernel_size=3, padding=1)
7.          self.e12 = nn.Conv2d(64, 64,
8.                          kernel_size=3, padding=1)
9.          self.pool1 = nn.MaxPool2d(
10.                         kernel_size=2, stride=2)
11.
12.          # input: 284x284x64
13.          self.e21 = nn.Conv2d(64, 128,
14.                          kernel_size=3, padding=1)
15.          self.e22 = nn.Conv2d(128, 128,
16.                          kernel_size=3, padding=1)
17.          self.pool2 = nn.MaxPool2d(
18.                         kernel_size=2, stride=2)
19.
20.          # input: 140x140x128
21.          self.e31 = nn.Conv2d(128, 256,
22.                          kernel_size=3, padding=1)
23.          self.e32 = nn.Conv2d(256, 256,
24.                          kernel_size=3, padding=1)
25.          self.pool3 = nn.MaxPool2d(
26.                         kernel_size=2, stride=2)
27.
28.          # input: 68x68x256
29.          self.e41 = nn.Conv2d(256, 512,
30.                          kernel_size=3, padding=1)
31.          self.e42 = nn.Conv2d(512, 512,
32.                          kernel_size=3, padding=1)
33.          self.pool4 = nn.MaxPool2d(
34.                         kernel_size=2, stride=2)
35.
36.          # input: 32x32x512
37.          self.e51 = nn.Conv2d(512, 1024,
38.                          kernel_size=3, padding=1)
39.          self.e52 = nn.Conv2d(1024, 1024,
40.                          kernel_size=3, padding=1)
41.
42.          # Decoder
43.          self.upconv1 = nn.ConvTranspose2d(1024, 512,
44.                                         kernel_size=2, stride=2)
45.          self.d11 = nn.Conv2d(1024, 512,
46.                          kernel_size=3, padding=1)
47.          self.d12 = nn.Conv2d(512, 512,
48.                          kernel_size=3, padding=1)
49.
50.          self.upconv2 = nn.ConvTranspose2d(512, 256,
51.                                         kernel_size=2, stride=2)
52.          self.d21 = nn.Conv2d(512, 256,
53.                          kernel_size=3, padding=1)
54.          self.d22 = nn.Conv2d(256, 256,
55.                          kernel_size=3, padding=1)
56.
57.          self.upconv3 = nn.ConvTranspose2d(256, 128,
58.                                         kernel_size=2, stride=2)
59.          self.d31 = nn.Conv2d(256, 128,
60.                          kernel_size=3, padding=1)
61.          self.d32 = nn.Conv2d(128, 128,

```

```

62.                 kernel_size=3, padding=1)
63.
64.                 self.upconv4 = nn.ConvTranspose2d(128, 64,
65.                                         kernel_size=2, stride=2)
66.                 self.d41 = nn.Conv2d(128, 64,
67.                                         kernel_size=3, padding=1)
68.                 self.d42 = nn.Conv2d(64, 64,
69.                                         kernel_size=3, padding=1)
70.
71.             # Output layer
72.             self.outconv = nn.Conv2d(64, n_class,
73.                                     kernel_size=1)
74.         def forward(self, x):
75.             # Encoder
76.             xe11 = relu(self.e11(x))
77.             xe12 = relu(self.e12(xe11))
78.             xp1 = self.pool1(xe12)
79.
80.             xe21 = relu(self.e21(xp1))
81.             xe22 = relu(self.e22(xe21))
82.             xp2 = self.pool2(xe22)
83.
84.             xe31 = relu(self.e31(xp2))
85.             xe32 = relu(self.e32(xe31))
86.             xp3 = self.pool3(xe32)
87.
88.             xe41 = relu(self.e41(xp3))
89.             xe42 = relu(self.e42(xe41))
90.             xp4 = self.pool4(xe42)
91.
92.             xe51 = relu(self.e51(xp4))
93.             xe52 = relu(self.e52(xe51))
94.
95.             # Decoder
96.             xu1 = self.upconv1(xe52)
97.             xu11 = torch.cat([xu1, xe42], dim=1)
98.             xd11 = relu(self.d11(xu11))
99.             xd12 = relu(self.d12(xd11))
100.
101.            xu2 = self.upconv2(xd12)
102.            xu22 = torch.cat([xu2, xe32], dim=1)
103.            xd21 = relu(self.d21(xu22))
104.            xd22 = relu(self.d22(xd21))
105.
106.            xu3 = self.upconv3(xd22)
107.            xu33 = torch.cat([xu3, xe22], dim=1)
108.            xd31 = relu(self.d31(xu33))
109.            xd32 = relu(self.d32(xd31))
110.
111.            xu4 = self.upconv4(xd32)
112.            xu44 = torch.cat([xu4, xe12], dim=1)
113.            xd41 = relu(self.d41(xu44))
114.            xd42 = relu(self.d42(xd41))
115.
116.            # Output layer
117.            out = self.outconv(xd42)
118.

```

```
119.     return out
```

Penjelasan masing-masing baris pada Kode Program 5.5 adalah sebagai berikut:

1. Baris 1 berfungsi untuk mendeklarasi kan *class* UNET yang akan mewariskan *nn.Module* dari *library* PyTorch. Kegunaan utama dari *nn.Module* adalah menyediakan struktur dasar untuk mendefinisikan, mengelola, dan melacak parameter-parameter dalam model neural network.
2. Baris 2-73 berfungsi untuk menginisiasi seluruh variable yang ada pada kelas UNET. Variabel yang disimpan berupa seluruh dari masing-masing layer pada arsitektur U-Net sesuai dengan Gambar 2.
3. Baris 74-117 berfungsi sebagai propagasi maju dari layer-layer inisiasi yang disambungkan.
4. Baris 119 berfungsi mengembalikan nilai dari keluaran pada lapisan terakhir yaitu layer konvolusi dari inisiasi pada variabel ‘outvony’.

5.6 Implementasi Dataloader untuk Pelatihan Model U-Net

Pemodelan pada tahapan implementasi ini memanfaatkan framework PyTorch. Data yang akan digunakan untuk melatih dan menguji harus disesuaikan formatnya agar bisa diterima model PyTorch. Sebelum melakukan pelatihan model, data harus dikonversi ke PyTorch Dataset dan disimpan ke dalam Dataloader. Hal ini dilakukan ke kedua himpunan, pelatihan dan validasi. Berikut ini kode untuk membaca citra dari himpunan pelatihan dapat dilihat pada Kode Program 5.6.

Kode Program 5.6 Kode Penyusunan Dataloader untuk Pelatihan Model U-Net

```
1. class ImageDataset(Dataset):
2.     def __init__(self,
3.                  image_path_list,
4.                  mask_path_list,
5.                  test=False
6.                 ):
7.         self.images = image_path_list
8.         self.masks = mask_path_list
9.         self.transform = transforms.Compose([
10.             transforms.Resize((640, 640)),
11.             transforms.ToTensor()])
12.
13.     def __getitem__(self, index):
14.         img = Image.open(self.images[index]).convert("RGB")
15.         mask = Image.open(self.masks[index]).convert("L")
16.
17.         return self.transform(img), self.transform(mask)
18.
19.     def len_(self):
```

```

20.         return len(self.images)
21.     train_dataset = ImageDataset(image_train, mask_train)
22.     val_dataset = ImageDataset(image_val, mask_val)
23.     train_dataloader = DataLoader(
24.         dataset=train_dataset,
25.         batch_size=BATCH_SIZE,
26.         shuffle=True)
27.     val_dataloader = DataLoader(
28.         dataset=val_dataset,
29.         batch_size=BATCH_SIZE,
30.         shuffle=True)

```

Penjelasan masing-masing baris pada Kode Program 5.6 adalah sebagai berikut:

1. Baris 1 berfungsi untuk mendeklarasi kan *class* UNET yang akan mewariskan Dataset dari *library* PyTorch. Kegunaan utama dari ‘Dataset’ dalam PyTorch digunakan untuk membuat dataset khusus untuk bekerja dengan data gambar.
2. Baris 2-11 berfungsi untuk melakukan inisisasi varabel di dalam kelas ‘ImageDataset’. Variabile ‘images’ dan ‘masks’ akan menyimpan seluruh *list path* dari gambar dan *mask*-nya. Sedangkan ‘transform’ digunakan untuk manipulasi data gambar untuk melakukan perubahan ukuran menjadi 640, 640 dan mengubah format data menjadi tensor.
3. Baris 13-17 merupakan fungsi untuk memberikan keluaran berupa gambar original dan gambar *mask* yang telah di manipulasi dengan variable ‘transform’.
4. Baris 19-20 berfungsi untuk mengembalikan jumlah gambar yang tersimpan dari kelas ImageDataset.
5. Baris 21 dan 22 berfungsi untuk menginisiasikan data gambar dari data latih dan data validasi dari masing-masing variable ke dalam kelas ‘ImageDataset’ yang telah dibuat sebelumnya pada baris 1-16.
6. Baris 23-30 menggunakan bantuan dari *library* PyTorch pada kelas ‘DataLoader’. DataLoader digunakan untuk mengatur data pelatihan ke dalam setiap *batch* yang jumlahnya ditentukan selama proses pelatihan.

5.7 Implementasi Pelatihan Model, Validasi Model dan Penyimpanan Bobot Pelatihan U-Net

Model yang telah dibuat selanjutnya dapat dilatih dengan menggunakan data yang telah disiapkan. Proses pelatihan ini dilakukan dengan memperbarui parameter-parameter yang ada dalam model. Parameter diperbarui untuk meminimalisir nilai *loss* yang didapatkan setelah model belajar dari data. Pelatihan ini dilakukan secara berulang (*epochs*). Tahapan pelatihan ini akan dilakukan beberapa kali dengan mengganti parameter pelatihan untuk mengetahui konfigurasi pelatihan yang memiliki kinerja paling baik. Penyimpanan model

dilakukan dengan menggunakan kondisional jika nilai loss pada validasi lebih rendah dari nilai loss validasi yang pernah ada tercatat pada epoch sebelumnya. Pelatihan model ini juga menggunakan *callback* manual berupa pemberhentian pembelajaran model jika selisih dari nilai loss pada data latih terlampaui jauh. Implementasi kode untuk melakukan pelatihan dapat dilihat pada Kode Program 5.7.

Kode Program 5.7 Kode Pelatihan, Validasi, dan Penyimpanan Model U-Net

```

1.    LEARNING_RATE = 1e-4
2.    BATCH_SIZE = 4
3.    EPOCHS = 100
4.    MODEL_SAVE_PATH = " /model640.pth"
5.    val_loss_min = float("inf")
6.    train_loss_history = []
7.    val_loss_history = []
8.    for epoch in tqdm(range(EPOCHS)):
9.        model.train()
10.       train_running_loss = 0
11.       for idx, img_mask in
12.           enumerate(tqdm(train_dataloader)):
13.               img = img_mask[0].float().to(device)
14.               mask = img_mask[1].float().to(device)
15.
16.               y_pred = model(img)
17.               optimizer.zero_grad()
18.
19.               loss = criterion(y_pred, mask)
20.               train_running_loss += loss.item()
21.
22.               loss.backward()
23.               optimizer.step()
24.
25.               train_loss = train_running_loss / (idx + 1)
26.
27.               model.eval()
28.               val_running_loss = 0
29.               with torch.no_grad():
30.                   for idx, img_mask in
31.                       enumerate(tqdm(val_dataloader)):
32.                           img = img_mask[0].float().to(device)
33.                           mask = img_mask[1].float().to(device)
34.
35.                           y_pred = model(img)
36.                           loss = criterion(y_pred, mask)
37.
38.                           val_running_loss += loss.item()
39.                           val_loss = val_running_loss / (idx + 1)
40.                           print(f'\n')
41.                           print("-"*30)
42.                           print(f"Train Loss EPOCH {epoch+1}:{train_loss:.4f}")
43.                           print(f"Valid Loss EPOCH {epoch+1}:{val_loss:.4f}")
44.                           print("-"*30)
45.                           if val_loss < val_loss_min:
46.                               val_loss_min = val_loss
47.                               torch.save(model.state_dict(), MODEL_SAVE_PATH)

```

```

46.         print(f'\n')
47.         print(f'model saved in epoch : {epoch+1}')
48.         if abs(val_loss - train_loss) > 0.15 and epoch > 10:
49.             print(f'\n')
50.             print(f'overfitting in {epoch+1}')
51.             break
52.         train_loss_history.append(train_loss)
53.         val_loss_history.append(val_loss)

```

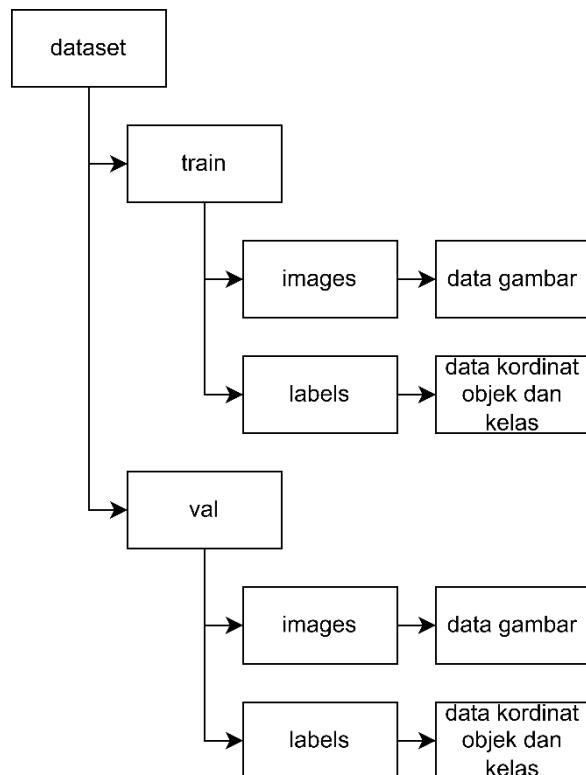
Penjelasan masing-masing baris pada Kode Program 5.7 adalah sebagai berikut:

1. Baris 1-7 merupakan inisiasi dari variable yang dibutuhkan dalam pelatihan model. Variabel yang diinisiasi berupa *learning rate*, ukuran *batch*, maksimal *epoch* yang ditempuh, *path* untuk menyimpan model, nilai inisiasi awal *val loss* sebagai pembatas untuk menyimpan model setiap mendapatkan nilai *val loss* terkecil yang terbaru, dan variable ‘*train_loss_history*’ dan ‘*val_loss_history*’ yang nantinya menyimpan setiap nilai loss dari masing-masing *train* dan *val* yang nantinya digunakan untuk visualisasi perubahan nilai loss.
2. Baris 8-10 berfungsi melakukan perulangan dari setiap *epoch*. Kemudian dilanjutkan dengan menetapkan model ke mode pelatihan. ‘*train_running_loss*’ diinisiasikan bernilai 0 untuk menghitung total loss pada setiap epochnya.
3. Baris 11-23 merupakan perulangan untuk setiap ‘*train_dataloader*’. Setiap iterasi akan mengambil satu-persatu gambar yang telah disimpan dari *train_dataloader* yang nantinya akan dilakukan prediksi terhadap model pada baris 15. Penggunaan ‘*optimizer.zero_grad()*’ berfungsi untuk mengatur gradien parameter model menjadi 0. Ini dikarenakan sebelum melakukan *backward pass* untuk menghindari akumulasi gradien dari iterasi sebelumnya. Selanjutnya dilakukan perhitungan loss antara nilai prediksi dari gambar original terhadap *ground truth*-nya yaitu *mask*. Nantinya nilai loss tersebut akan disimpan dan dijumlah dengan nilai loss dari masing-masing epoch sebelumnya. Setelah itu dilakukan propagasi mundur dan memperbarui bobot dengan ‘*optimizer.step()*’.
4. Baris 24 digunakan untuk menghitung rata-rata nilai loss dari 1 epoch terhadap data latih.
5. Baris 26-37 berfungsi mengubah mode dari model menjadi mode evaluasi sehingga pada baris ini model tidak akan melakukan update bobot yang sudah dilatih dari baris 11-23. Sehingga akan menggunakan ‘*model.eval()*’ dan ‘*with torch.no_grad()*’ agar memastikan bahwa tidak diperlukan perhitungan gradien selama iterasi berikutnya. Nilai rata-rata loss dari data validasi akan disimpan pada baris ke-37.

6. Baris 38-42 akan memberikan keluaran berupa informasi dari nilai *loss* pada data latih dan data validasi.
7. Baris 43-47 berfungsi untuk melakukan penyimpanan nilai minimum dari nilai *loss* validasi terbaru disetiap epochnya dan melakukan penyimpanan model setiap perubahan nilai *loss* validasi.
8. Baris 48-51 berfungsi sebagai *callback* yang mana pelatihan akan berhenti jika jarak antara nilai *loss* dari validasi dan latih terlampaui jauh dengan minimal epoch yang sudah ditempuh berjumlah 10.
9. Baris 52-53 akan menyimpan nilai *loss* dari data latih dan data validasi pada variable *list* yang sudah diinisiasi sebelumnya.

5.8 Implementasi Persiapan Data Pelatihan Deteksi Objek

Sebelum melakukan pelatihan model dengan algoritma YOLOv5, pengaturan peletakan data yang akan dilatih harus diatur sesuai dengan ketentuan pada prosedur. Arsitektur data untuk pelatihan YOLOv5 dapat dilihat pada Gambar 5.11.



Gambar 5.11 Arsitektur Data Pelatihan Model YOLOv5

Setelah itu buatlah sebuah file dengan format file .yaml yang berfungsi sebagai konfigurasi untuk menetapkan *path* atau alamat dari sebuah direktori yang akan dilatih dan juga nama kelas yang akan disimpan pada pelatihan. Penulisan *script* data dilihat pada Kode Program 5.8.

Kode Program 5.8 Penulisan Script File YAML

```
1. path: dataset
2. train: dataset/train
3. val: dataset/val
4. test:
5. names:
6.   0: Car
7.   1: Motorbike
```

Penjelasan masing-masing baris pada Kode Program 5.8 adalah sebagai berikut:

1. Baris 1-4 menginisiasi *path* dari direktori masing-masing.
2. Baris 5-7 menginformasikan id dan nama masing-masing kelas dari label.

Dikarenakan tidak memiliki data *testing* dengan rencana implementasi testing akan langsung bersamaan dengan tracking. Maka *path* testing dapat dikosongkan. Persiapan data dilakukan terhadap 2 dataset. Dataset pertama berisikan gambar tanpa metode *preprocessing learning* dan yang kedua berisikan data gambar dengan teknik *preprocessing learning*. Gambar yang ada pada masing-masing dataset akan sama sehingga kordinat label dari masing-masing gambar juga sama. Yang membedakan hanya dari gambarnya saja.

5.9 Implementasi Pelatihan dan Validasi Model Deteksi Objek YOLOv5

Untuk menjalankan pelatihan dengan algoritma YOLOv5 yang dikembangkan oleh tim Ultralytics. Maka diperlukan melakukan *install* dari sumber resmi dengan menggunakan *script* seperti pada Tabel 5.9.

Kode Program 5.9 Script Instalasi Algoritma YOLOv5

```
1. !git clone https://github.com/ultralytics/yolov5
2. %cd yolov5
3. !pip install -qr requirements.txt comet_ml
```

Penjelasan masing-masing baris pada Kode Program 5.9 adalah sebagai berikut:

1. Baris 1 berfungsi melakukan instalasi YOLOv5 dari situs resmi.
2. Baris 2 mengubah direktori kerja saat ini ke direktori YOLOv5 hasil dari instalasi pada baris ke-1.
3. Baris 3 melakukan instalasi kebutuhan dari YOLOv5 dari direktori yang sudah tersedia pada YOLOv5 tersebut.

Setelah melakukan instalasi program dari YOLOv5, maka bisa dilanjutkan dengan pelatihan model dari YOLOv5 dengan menjalankan *script* pada Kode Program 5.10.

Kode Program 5.10 Script Pelatihan Model YOLOv5

```
1. !python train.py \
2.     --img 640 \
3.     --batch 4 \
4.     --epochs 200 \
5.     --data dataset.yaml \
6.     --weights yolov5s.pt \
7.     --cfg yolov5s.yaml \
8.     --patience 10
```

Penjelasan masing-masing baris pada Kode Program 5.10 adalah sebagai berikut:

1. Baris 1 berfungsi menjalankan kode file *train.py* dari direktori YOLOv5 yang sudah *di-install*.
2. Baris 2 berfungsi melakukan resize image dari gambar masukan dalam melakukan pelatihan.
3. Baris 3 menentukan jumlah *batch* dalam dataset pelatihan.
4. Baris 4 menentukan jumlah maksimal *epoch* pelatihan.
5. Baris 5 menentukan *path* dari file ‘.yaml’ yang berisikan *path* direktori data pelatihan.
6. Baris 6 menentukan bobot yang sudah dilatih (*pre-trained weights*) yang tersedia dari YOLOv5.
7. Baris 7 merujuk dari configurasi pada *pre-trained weights* yang sesuai dengan bobot yang ditentukan.
8. Baris 8 merupakan sebuah *callback* untuk memberhentikan pelatihan model jika nilai *loss* validasi tidak ada improvisasi atau nilai lebih rendah sebanyak 10 *epoch*.

Sedangkan untuk melakukan validasi model dengan YOLOv5 dapat menjalankan script berikut pada Kode Program 5.11:

Kode Program 5.11 Script Pelatihan Model YOLOv5

```
1. !python val.py \
2.     --weights yolov5/runs/train/exp/weights/best.pt \
3.     --data dataset.yaml \
4.     --img 640
```

Penjelasan masing-masing baris pada Kode Program 5.11 adalah sebagai berikut:

1. Baris 1 berfungsi menjalankan kode file val.py dari direktori YOLOv5 yang sudah di-*install*.
2. Baris 2 berfungsi memuat bobot dari hasil latih pada *script* sebelumnya. Bobot akan tersimpan dari direktori YOLOv5.
3. Baris 3 menentukan *path* dari file ‘.yaml’ yang berisikan *path* direktori data pelatihan. Path yang akan digunakan adalah path validasi dikarenakan akan menjalankan file val.py.
4. Baris 4 berfungsi melakukan resize image dari gambar masukan dalam melakukan validasi model menjadi ukuran 640 x 640.

5.10 Implementasi Persiapan Pelacakan Objek

Sebelum melakukan tracking, ada beberapa hal yang perlu dipersiapkan berupa instalasi *library DeepSORT*, kelas detector, dan kelas untuk melakukan preprocessing jika ingin menggunakan metode preprocessing. Untuk melakukan instalasi *DeepSORT* dapat menjalankan *script* pada Kode Program 5.12.

Kode Program 5.12 Script Instalasi *Library DeepSORT*

```
1. | !pip install deep-sort-realtime
```

Penjelasan masing-masing baris pada Kode Program 5.12 adalah sebagai berikut:

1. Baris 1 berfungsi melakukan instalasi *library DeepSORT*

Untuk memanggil atau menggunakan *library* tersebut dapat melakukan inisiasi yang dapat dilihat pada Tabel 5.13.

Kode Program 5.13 Kode Memanggil Kelas *DeepSORT* dari *Library ‘deep_sort_realtime’*

```
1. | from deep_sort_realtime.DeepSORT_tracker import DeepSORT
2. | object_tracker = DeepSORT()
```

Penjelasan masing-masing baris pada Kode Program 5.13 adalah sebagai berikut:

1. Baris 1 berfungsi memanggil modul deep_tracker yang digunakan untuk melakukan pelacakan dengan memanfaatkan kelas ‘*DeepSORT*’.
2. Baris 2 berfungsi sebagai *instance* menciptakan suatu objek yang mewakili sistem pelacakan objek yang diimplementasikan dalam kelas ‘*DeepSORT*’.

Setelah itu membuat kelas Bernama YoloDetector. Kelas ini nantinya akan berfungsi untuk melakukan objek deteksi dari setiap *frame* video masukan yang akan dilakukan pelacakan objek. Kode program dapat dilihat pada Tabel 5.14.

Kode Program 5.14 Kode Pembuatan Kelas untuk Deteksi Objek dengan YOLO

```
1. class YoloDetector():
2.     def __init__(self, model):
3.         self.model = torch.hub.load(
4.             'ultralytics/yolov5',
5.             'custom',
6.             path=model,
7.             force_reload= True)
8.         self.classes = self.model.names
9.         self.device = (
10.             'cuda' if torch.cuda.is_available()
11.             else 'cpu')
12.         print('Using Device: ', self.device)
13.     def score_frame(self, frame):
14.         self.model.to(self.device)
15.         downscale_factor = 2
16.         width = int(frame.shape[1] / downscale_factor)
17.         height = int(frame.shape[0] / downscale_factor)
18.         frame = cv2.resize(frame, (width, height))
19.         results = self.model(frame)
20.         labels = results.xyxy[0][:, :-1],
21.         cord = results.xyxy[0][:, :-1]
22.         return labels, cord
23.     def class_to_label(self, x):
24.         return self.classes[int(x)]
25.     def plot_boxes(self,
26.                   results,
27.                   frame,
28.                   height,
29.                   width,
30.                   confidence=0.15):
31.         labels, cord = results
32.         detections = []
33.         n = len(labels)
34.         x_shape, y_shape = width, height
35.         for i in range(n):
36.             row = cord[i]
37.             x1 = int(row[0]*x_shape)
38.             y1 = int(row[1]*y_shape)
39.             x2 = int(row[2]*x_shape)
40.             y2 = int(row[3]*y_shape)
41.             detections.append(([x1, y1,
42.                                 int(x2-x1),
43.                                 int(y2-y1)],
44.                                 row[4].item(),
45.                                 str(self.class_to_label(labels[i]))))
46.         return frame, detections
```

Penjelasan masing-masing baris pada Kode Program 5.14 adalah sebagai berikut:

1. Baris 1 berfungsi mendeklarasikan kelas Bernama ‘YoloDetector’
2. Baris 2-12 berfungsi untuk mendeklarasikan variable yang akan ada setiap kelas ‘YoloDetector’ dipanggil dan menerima parameter *path* dari model *yolo* yang telah dilatih.
3. Baris 13-22 akan melakukan mengembalikan nilai berupa *kelas* dan kordinat dari masing-masing objek yang terdeteksi disetiap *frame* masukan dengan menggunakan model yang telah dilatih. Kemudian akan mengembalikan nilai *return* berupa *list* dari variable ‘labels’ dan ‘cords’.
4. Baris 23-24 berfungsi untuk mengembalikan nilai string berupa nama asli dari ‘class’ yang bertipe *integer* sebagai nilai identitasnya pada hasil objek deteksi.
5. Baris 25-46 berfungsi sebagai pengubah kordinat sesuai dengan ukuran panjang dan lebar untuk membuat video hasil pelacakan pada kelas inferensi.

Selanjut kelas yang akan dibuat adalah kelas *preprocessing* gambar jika misalnya ingin menggunakan untuk melakukan uji *test* terhadap pengaruh *preprocessing* terhadap pelacakan objek. Kode program dapat dilihat pada Tabel 5.15.

Kode Program 5.15 Kode Pembuatan Kelas untuk *Preprocessing Frame*

```

1. class PreprocessingTracker():
2.     def __init__(self):
3.         self.onnx_model_path = 'model640.onnx'
4.         self.onnx_session =
5.             onnxruntime.InferenceSession(self.onnx_model_path)
6.                 self.device = 'cuda' if torch.cuda.is_available()
7.             else 'cpu'
8.                 print('Using Device: ', self.device)
9.                 def learning_preprocessing(self, img):
10.                     img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)    #
11.                         Convert to RGB format
12.
13.                         resized_image = cv2.resize(img, (640,
14.                                         640)).astype(np.float32) /255.0
15.                                         input_data = np.transpose(resized_image, (2, 0, 1))
16.                                         input_data = np.expand_dims(input_data, axis=0)
17.                                         output = self.onnx_session.run(None, {'input':
18.                                         input_data})
19.                                         prediction = np.squeeze(output[0])
20.
21.                                         pred_mask = prediction
22.                                         pred_mask[pred_mask < 0] = 0
23.                                         pred_mask[pred_mask > 0] = 1
24.                                         pred_mask = np.uint8(pred_mask * 255)
25.                                         resized_image = cv2.resize(img, (1920, 1080))
26.                                         resized mask = cv2.resize(pred mask, (1920, 1080))

```

```

22.     inpainted_image = cv2.inpaint(resized_image,
23.         resized_mask, 3, cv2.INPAINT_TELEA)
24.     return inpainted_image
25.
26.     def CV2_to_PIL_img(self, cv2_im):
27.         cv2_im = cv2.cvtColor(cv2_im, cv2.COLOR_BGR2RGB)
28.         pil_im = Image.fromarray(cv2_im)
29.         return pil_im
30.
31.     def PIL_to_CV2_img(self, img):
32.         cv_image = np.array(img.convert('RGB'))
33.         cv_image = cv_image[:, :, ::-1].copy()
34.         return cv_image
35.
36.     def first_polynomial_function(self, image):
37.         table = np.array([1.657766*i-0.009157128*(i**2) +
38.             0.00002579473*(i**3)
39.                 for i in np.arange(0, 256)]).astype("uint8")
40.         return cv2.LUT(image, table)
41.
42.     def second_polynomial_function(self, image):
43.         table = np.array([
44.             -4.263256 * math.exp(-14)+1.546429*i-
45.             0.005558036*(i**2)+0.00001339286*(i**3)
46.                 for i in np.arange(0, 256)]).astype("uint8")
47.         return cv2.LUT(image, table)
48.
49.     def adjust_gamma(self, image, gamma=1.0):
50.         invGamma = 1.0 / gamma
51.         table = np.array([(i / 255.0) ** invGamma) * 255
52.                         for i in np.arange(0, 256)]).astype("uint8")
53.
54.         return cv2.LUT(image, table)
55.
56.     def enhance_contrast(self, image, factor=1.4):
57.         _image = ImageEnhance.Contrast(
58.             self.CV2_to_PIL_img(image)
59.         ).enhance(factor)
60.
61.         return self.PIL_to_CV2_img(_image)
62.
63.     def reduce_glare(self, image):
64.         _image = self.adjust_gamma(
65.             self.second_polynomial_function(
66.                 self.adjust_gamma(
67.                     self.first_polynomial_function(image),
68.                     0.75
69.                 )
70.             ),
71.             0.8
72.         )
73.         return _image
74.
75.     def mix_filter(self, image):
76.         _image = self.enhance_contrast(
77.             self.reduce_glare(
78.                 self.enhance_contrast(

```

```

76.                     self.reduce_glare(image),
77.                     factor=1.6
78.                 )
79.             ),
80.             factor=1.4
81.         )
82.     return image

```

Penjelasan masing-masing baris pada Kode Program 5.15 adalah sebagai berikut:

1. Baris 1 berfungsi mendeklarasikan kelas Bernama ‘PreprocessingTracker’.
2. Baris 2-6 berfungsi untuk melakukan inisiasi awal setiap kelas dipanggil. Variabel yang disimpan di kelas ini berupa pemuatan model dari metode preprocessing sebelumnya yang telah dilatih dan penggunaan devide. Pada baris ke-6 akan mengeluarkan *output* berupa *device* yang digunakan dalam melakukan inferensi.
3. Baris 7-23 merupakan inferensi dari setiap *frame* masukan yang nantinya ingin dilakukan *preprocessin* dengan metode *learning*. Keluaran akan berupa gambar bertipe data numpy array hasil dari *inpaiting* antara gambar original dengan hasil deteksi masking.
4. Baris 25-28 merupakan fungsi yang berfungsi untuk mengubah format pembacaan gambar dari *library cv2* menjadi *library PIL*.
5. Baris 30-33 merupakan fungsi yang berfungsi untuk mengubah format pembacaan gambar dari *library PIL* menjadi *library cv2*.
6. Baris 35-38 merupakan fungsi yang berfungsi untuk melakukan preprocessing gambar dengan menggunakan metode fungsi *polynomial* dengan menggunakan persamaan $f(x) = 1.657766x - 0.009157128x^2 + 0.00002579473x^3$.
7. Baris 40-44 merupakan fungsi yang berfungsi untuk melakukan preprocessing gambar dengan menggunakan metode fungsi *polynomial* dengan menggunakan persamaan $f(x) = -4.263256e^{14} + 1.546429x - 0.005558036x^2 + 0.00001339286x^3$.
8. Baris 46-51 merupakan fungsi yang berfungsi melakukan *preprocessing* gambar dengan menggunakan metode *reduce gamma*.
9. Baris 53-58 merupakan fungsi yang berfungsi melakukan *preprocessing* gambar dengan menggunakan metode *enhance contrast* yang dibantu oleh *library PIL*.

10. Baris 60-71 merupakan fungsi yang menginisiasi fungsi ‘reduce_glare’ yang berikan kombinasi dari fungsi *preprocessing* gambar. Fungsi yang dimasukan berupa ‘first_polynomial_function’, ‘second_polynomial_function’, dan ‘adjust_gamma’.
11. Baris 72-82 merupakan fungsi yang menginisiasi fungsi ‘mix_filter’ dengan mengabungkan fungsi ‘reduce_glare’ dan fungsi ‘enhance_contrast’.

5.11 Implementasi Fungsi Inferensi

Fungsi inferensi akan digunakan untuk melakukan pengujian dengan menggunakan data masukan berupa sebuah video. Inferensi berisi kode untuk melakukan deteksi objek, pelacakan objek, dan *preprocessing inpainting* jika misalnya ingin menggunakan metode *preprocessing* pada *frame*. Selain itu pada inferensi ini juga bertujuan untuk memberikan jumlah kendaraan motor dan mobil yang masuk ke dalam gerbang veteran Universitas Brawijaya pada malam hari. Implementasi kode dapat dilihat pada Tabel 5.16.

Kode Program 5.16 Implementasi Kode Fungsi Inferensi

```

1. def make_track_video(src_video_path,
2.                      dest_path, model,
3.                      preprocessing_type):
4.     detector = YoloDetector(model)
5.     preprocessing_frame = PreprocessingTracker()
6.
7.     cap = cv2.VideoCapture(src_video_path)
8.
9.     cap.set(cv2.CAP_PROP_FRAME_WIDTH, 640)
10.    cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 640)
11.    output_path = dest_path
12.    fourcc = cv2.VideoWriter_fourcc(*'mp4v')
13.    fps = 30
14.    video_writer = cv2.VideoWriter(output_path,
15.                                    fourcc,
16.                                    fps,
17.                                    (1920, 1080))
18.
19.    # Define the counting line
20.    line_start = (0, 600)
21.    line_end = (1700, 600)
22.
23.    # Initialize counters
24.    counter_car = 0
25.    counter_motorbike = 0
26.
27.    set_id_datang = set()
28.    set_id_lewat = set()
29.    while cap.isOpened():
30.        success, img = cap.read()
31.        if not success:
32.            break
33.        start = time.perf_counter()

```

```

34.         if preprocessing_type == 'machine_learning':
35.             preprocessing_img = preprocessing_frame.learning_preprocessing(img)
36.         elif preprocessing_type == 'reduce_glare':
37.             preprocessing_img = preprocessing_frame.mix_filter(img)
38.         else:
39.             preprocessing_img = img
40.
41.         results = detector.score_frame(preprocessing_img)
42.         img, detections = detector.plot_boxes(results,
43.                                                 preprocessing_img,
44.                                                 height=preprocessing_img.shape[0],
45.                                                 width=preprocessing_img.shape[1],
46.                                                 confidence=0.15)
47.         tracks = object_tracker.update_tracks(detections,
48.                                                 frame=img)
49.
50.         for track in tracks:
51.             if not track.is_confirmed():
52.                 continue
53.             track_id = track.track_id
54.             ltrb = track.to_ltrb()
55.
56.             bbox = ltrb
57.
58.             cv2.line(img,
59.                      line_start,
60.                      line_end,
61.                      (255, 250, 250),
62.                      2)
63.
64.             cv2.rectangle(img,
65.                           (int(bbox[0]), int(bbox[1])),
66.                           (int(bbox[2]), int(bbox[3])),
67.                           (0, 0, 255),
68.                           2)
69.
70.             cv2.putText(img,
71.                         f"ID: {str(track_id)} {track.get_det_class()}", 
72.                         (int(bbox[0]), int(bbox[1] - 10)),
73.                         cv2.FONT_HERSHEY_SIMPLEX, 1.5,
74.                         (0, 255, 0),
75.                         2)
76.
77.             if (bbox[1] < line_start[1]
78.                 and bbox[3] > line_start[1]):
79.                 if (track.get_det_class() == 'Car'
80.                     and track_id not in set_id_lewat
81.                     and track_id in set_id_datang):
82.                     counter_car += 1
83.                     set_id_lewat.add(track_id)
84.                 elif (track.get_det_class() == 'Motorbike'
85.                     and track_id not in set_id_lewat
86.                     and track_id in set_id_datang):
87.                     counter_motorbike += 1
88.                     set_id_lewat.add(track_id)

```

```

88.         else:
89.             if track_id not in set_id_datang:
90.                 set_id_datang.add(track_id)
91.             else:
92.                 pass
93.
94.             end = time.perf_counter()
95.             totalTime = end - start
96.             fps = 1 / totalTime
97.
98.             cv2.putText(img,
99.                         f'FPS: {int(fps)}',
100.                         (20, 70),
101.                         cv2.FONT_HERSHEY_SIMPLEX,
102.                         1.5,
103.                         (0, 255, 0),
104.                         2)
105.
106.             cv2.putText(img,
107.                         f'Car Count: {counter_car}',
108.                         (20, 120),
109.                         cv2.FONT_HERSHEY_SIMPLEX,
110.                         1.5,
111.                         (0, 255, 0),
112.                         2)
113.             cv2.putText(img,
114.                         f'Motorbike Count: {counter_motorbike}',
115.                         (20, 170),
116.                         cv2.FONT_HERSHEY_SIMPLEX,
117.                         1.5,
118.                         (0, 255, 0),
119.                         2)
120.
121.             video_writer.write(img)
122.
123.             if cv2.waitKey(1) & 0xFF == 27:
124.                 break
125.
126.             cap.release()
127.             video_writer.release()
128.             cv2.destroyAllWindows()
129.

```

Penjelasan masing-masing baris pada Kode Program 5.16 adalah sebagai berikut:

1. Baris 1-3 berfungsi mendeklarasikan fungsi Bernama ‘make_track_video’ yang akan menerima parameter berupa *path* sumber video, *path* tujuan untuk menyimpan video hasil inferensi, model objek deteksi dari yolo, dan tipe *preprocessing* gambar yang akan digunakan.
2. Baris 4 dan 5 akan mendeklarasikan kelas yang sebelumnya telah dibuat. Kelas tersebut berupa ‘YoloDetector’ dengan menerima inputan model

sesuai yang digunakan dana kelas ‘PreprocesingTracker’ jika akan menggunakan metode *preprocessing*.

3. Baris 7 berfungsi membaca video per *frame* dengan bantuan *library* OpenCV
4. Baris 9 dan 10 akan melakukan inisiasi perubahan ukuran panjang dan lebar dari *frame* sebesar 640 x 640 dikarenakan semua model dilatih pada ukuran 640 pada algoritma YOLOv5 sebelumnya.
5. Baris 11-17 berupa inisiasi awal untuk menimpan hasil video dengan melakukan inisiasi formati video, fps, dan persiapan video *writer* dengan bantuan *library* OpenCV.
6. Baris 20 dan 21 berfungsi sebagai penanda atau garis yang dibuat di video untuk menghitung kendaraan berdasarkan objek yang melewati garis.
7. Baris 24 dan 25 sebagai variable penghitung jumlah kendaraan untuk masing-masing motor dan mobil.
8. Baris 27 dan 28 sebagai penyimpan id dengan tipe set. Ini digunakan untuk menentukan kendaraan masuk dengan melihat sejauh objek melewati garis dari daerah datang menuju area masuk berdasarkan garis dari kode baris 16 and 17.
9. Baris 29 -32 inisiasi perulangan berdasarkan *frame* dari video masukan dan akan berhenti jika video tidak dapat dibuka atau sudah tidak ada *frame* lagi yang dapat dilihat dari variable *cap* pada baris kode ke-5.
10. Baris 33 digunakan sebagai penanda untuk memberikan informasi jumlah *frame* yang diproses setiap detiknya.
11. Baris 34-39 merupakan *conditional* jika ingin melakukan *preprocessing* gambar terhadap *frame* masukan dengan bantuan kelas ‘*preprocessing-frame*’.
12. Baris 41-46 berfungsi untuk mendapatkan hasil deteksi dengan menggunakan model YOLO pada kelas ‘YoloDetector’ sesuai dengan model yang digunakan.
13. Baris 47 berfungsi melakukan tracking dengan bantuan *library* ‘*deep_sort_realtime*’ dengan menggunakan kelas ‘*update_track*’. Ini akan memungkinkan untuk memberikan identitas objek dari setiap objek yang terdeteksi sehingga memungkinkan melacaknya pada *frame* selanjutnya.
14. Baris 50-52 akan melakukan perulangan dari setiap lacakan. Lacakan merupakan setiap objek yang berhasil dilacak pada algoritma *DeepSORT*. Perulangan akan melakukan *skip* jika *track* yang ada belum terkonfirmasi.

15. Baris 53-56 akan mengambil nilai id sebagai identitas objek yang ditracking berupa nilai *integer* dan nilai kordinat dari objek yang dilacak. ‘to_ltrb()’ mengebalikan nilai format kordinat berupa nilai *min x*, *min y*, *max x*, dan *max y*.
16. Baris 58 berfungsi sebagai visual garis yang akan ditambahkan disetiap *frame*.
17. Baris 64 berfungsi untuk menggambarkan sebuah *bounding box* sesuai dengan keluaran dari fungsi ‘to_ltrb()’ yang nantinya akan merujuk ke masing-masing objek yang terlacak.
18. Baris 70 berfungsi untuk memberikan id dan nama kelas dari masing-masing objek yang telah terlacak.
19. Baris 77-88 merupakan sebuah kondisional untuk menghitung jumlah kendaraan yang melewati garis dari arah datang menuju pintu masuk akan dihitung sebagai objek yang akan dihitung sesuai dengan nama kelas dan menjadi penghitung pada variable *counter* untuk masing-masing objek kendaraan.
20. Baris 89-93 berfungsi sebagai penanda id baru terlacak yang akan melewati arah datang.
21. Baris 95-97 berfungsi untuk menghitung berapa lama sebuah *frame* terproses dalam melakukan inferensi.
22. Baris 99 berfungsi sebagai penulis jumlah fps pada *frame* yang nantinya akan dijadikan video.
23. Baris 107 dan 114 berfungsi untuk memberikan jumlah informasi motor dan mobil yang telah terhitung pada setiap *frame*.
24. Baris 122 berfungsi untuk mengumpulkan seluruh *frame* terbaru untuk membuat video hasil inferensi.
25. Baris 124 berfungsi sebagai pemberhentian perulangan jika tombol ‘esc’ ditekan.
26. Baris 127-129 berfungsi untuk memberhentikan sumber daya yang digunakan pada *library* OpenCV dalam melakukan pengolahan video.

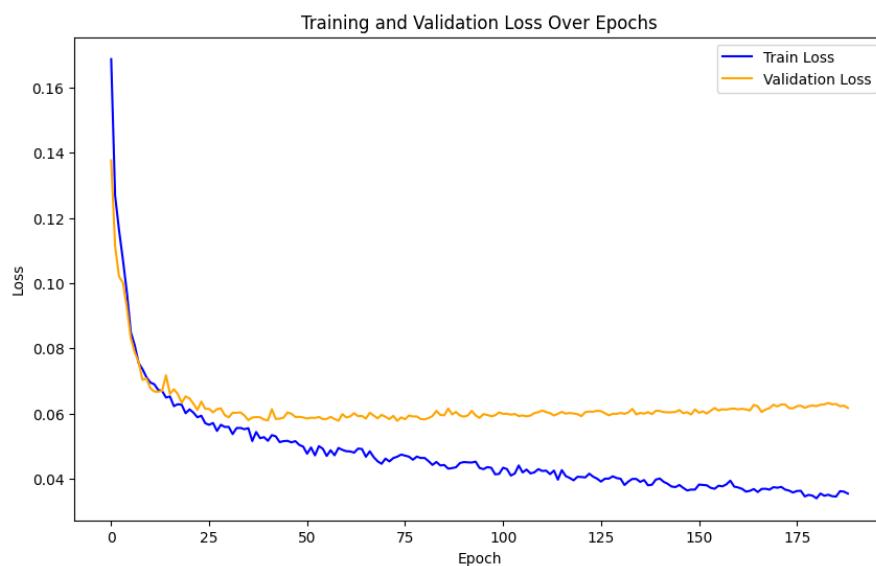
BAB 6 PENGUJIAN DAN ANALISIS

6.1 Pengujian Deteksi Objek

Pada pengujian deteksi objek, dilakukan dengan melakukan perbandingan antara gambar yang menggunakan preproses gambar dan tanpa preproses. Teknik preproses gambar yang digunakan berupa segmentasi kesilauan atau *flare* dari suatu gambar kemudian menerapkan metode *inpaint* untuk melakukan restorasi pada bagian silau yang didapatkan setelah melakukan segmentasi. Masing-masing dataset memiliki gambar yang sama dengan perbedaan terletak dari dataset gambar yang dipreproses dengan gambar dataset yang tidak dipreproses sebelumnya. Masing-masing data gambar akan dilakukan pelatihan model YOLOv5 dengan menggunakan bobot yang telah dilatih pada model yolov5s (*small*) dengan dilatih sebanyak 200 *epoch* maksimal dan masing-masing data latih menggunakan nilai *batch* sebesar 4. Kemudian semua gambar akan dilakukan perseragaman ukuran gambar menjadi 640 x 640.

6.1.1 Deteksi Objek dengan Gambar tanpa Preproses

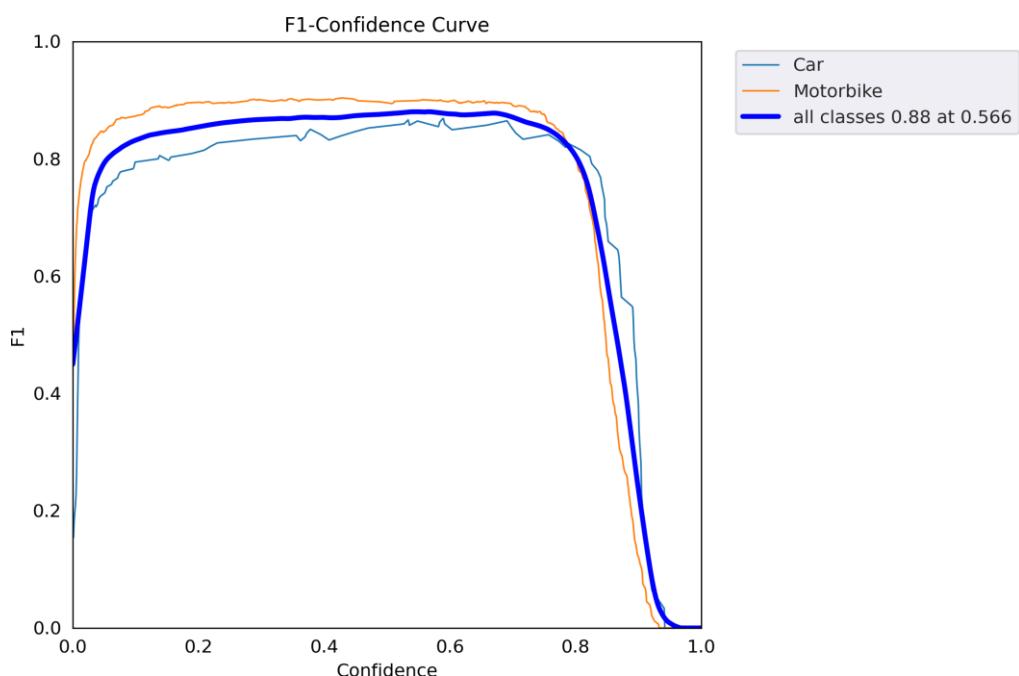
Hasil pelatihan pada dataset gambar yang tidak menggunakan preproses sebelumnya dengan menggunakan model YOLOv5 mendapatkan nilai *loss* terendah pada *epoch* ke-113 dengan nilai total *loss* validasi terkecil sebesar 0.04092725 dengan menggunakan nilai *patience* sebesar 75 untuk mencegah *overfitting*. Pergerakan nilai *loss* dengan seiring berjalananya *epoch* pelatihan dapat dilihat pada Gambar 6.1.



Gambar 6.1 Nilai *Loss* pada Pelatihan Model YOLOv5 pada Dataset tanpa Dipreproses

Nilai *F1 score* pada hasil validasi dataset, didapatkan nilai sebesar 0.88 pada dengan nilai *confidence* (kepercayaan) sebesar 0.566. Maksud dari nilai kepercayaan yang digunakan adalah melakukan penyaringan deteksi terhadap pengaturan nilai kepercayaan. Jika menggunakan nilai kepercayaan 0.566, maka akan diharapkan mendapatkan nilai *F1 score* sebesar 0.88. Kurva *F1 score* dilihat pada Gambar 6.2.

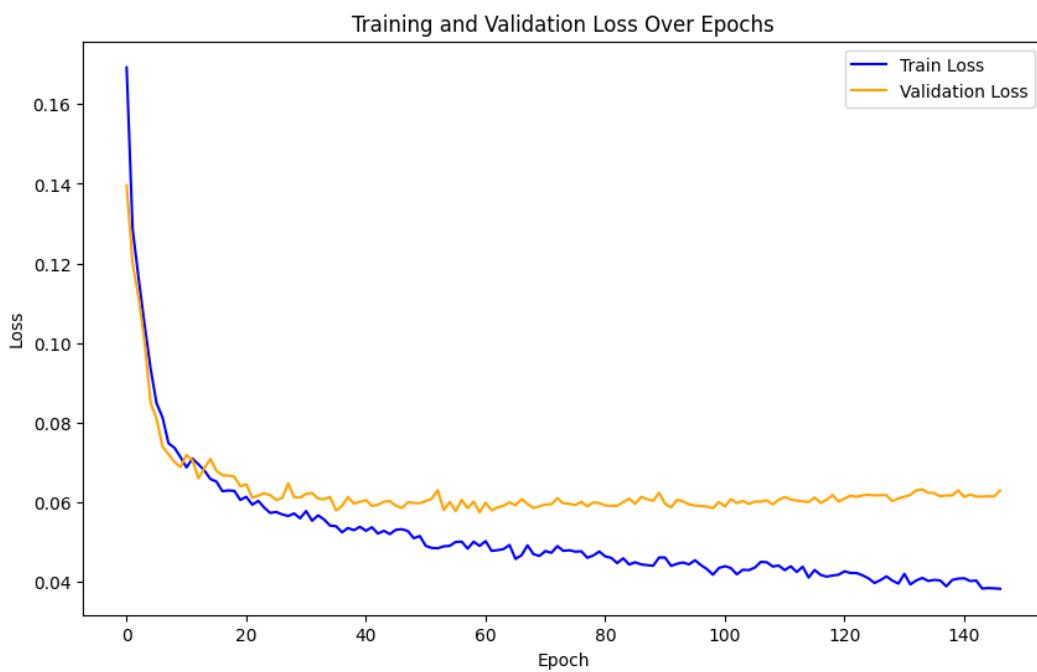
Metrik pengujian yang digunakan untuk analisis selanjutnya adalah *precision* dan *recall*. Nilai masing-masing *precision* dan *recall* yang didapatkan pada data validasi terhadap dataset gambar yang tidak di preproses adalah 0.91 dan 0.855.



Gambar 6.2 Kurva *F1 score* Validasi Data Gambar tanpa Preproses

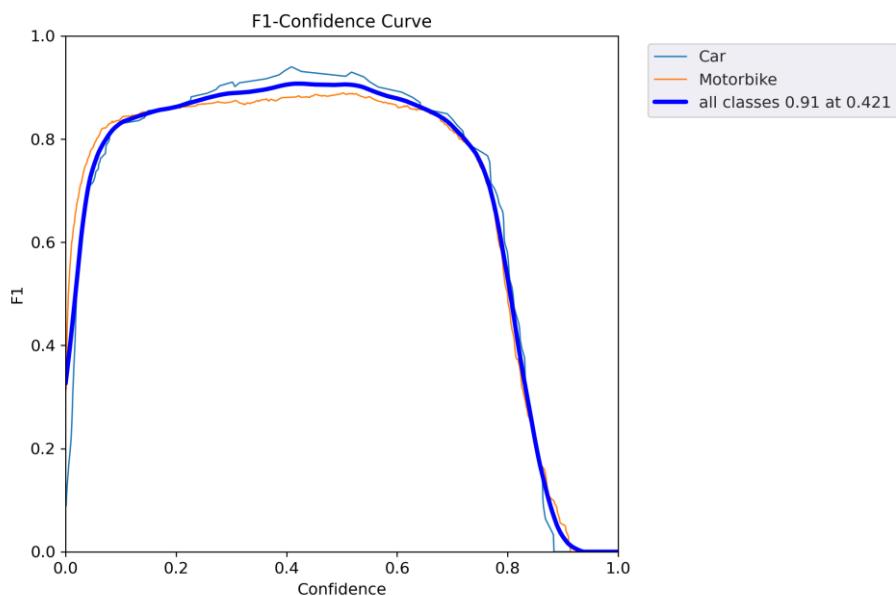
6.1.2 Deteksi Objek dengan Gambar dengan Preproses

Hasil pelatihan pada dataset gambar yang menggunakan metode preproses sebelumnya berupa segmentasi U-Net dan *inpaint*. Setelah melakukna preproses dilanjutkan dengan pelatihan model YOLOv5. Hasil pelatihan model mendapatkan nilai *loss* terendah pada *epoch* ke-46 dengan nilai total *loss* validasi terkecil sebesar 0.0531787. Pergerakan nilai *loss* dengan seiring berjalannya *epoch* pelatihan dapat dilihat pada Gambar 6.3.



Gambar 6.3 Nilai *Loss* pada Pelatihan Model YOLOv5 pada Dataset yang Diprepreses

Nilai *F1 score* yang didapatkan dari hasil validasi model pada dataset gambar dengan melakukan preproses sebelumnya sebesar 0.91 dengan skor kepercayaan 0.421. Kurva *F1 score* dapat dilihat pada Gambar 6.4. Sedangkan pada nilai *precision* dan *recall* yang didapatkan pada hasil validasi dataset gambar yang dilakukan preproses bernilai 0.942 dan 0.873.



Gambar 6.4 Kurva *F1 score* Validasi Data Gambar dengan Preproses

Dari hasil perbandingan kualitas objek deteksi dengan perbandingan penggunaan metode preproses dengan yang tidak menggunakan metode preproses dapat disimpulkan bahwa deteksi objek yang menggunakan preproses gambar terlebih dahulu dengan menggunakan segmentasi U-Net dan metode *inpaint* sedikit lebih baik. Hal ini dapat dilihat pada nilai *precision* dan *recall* yang dihasilkan mendapatkan nilai sedikit lebih unggul pada model yang dilatih tanpa proses *preprocessing* gambar sebelumnya. Namun nilai *confidence* yang didapatkan pada model tanpa menggunakan metode preproses gambar sedikit lebih tinggi namun dengan nilai *F1 score* yang lebih rendah dibanding pada model yang dilatih pada gambar menggunakan metode preproses sebelumnya. Ringkasan dari hasil pengujian dapat dilihat pada Tabel 6.1.

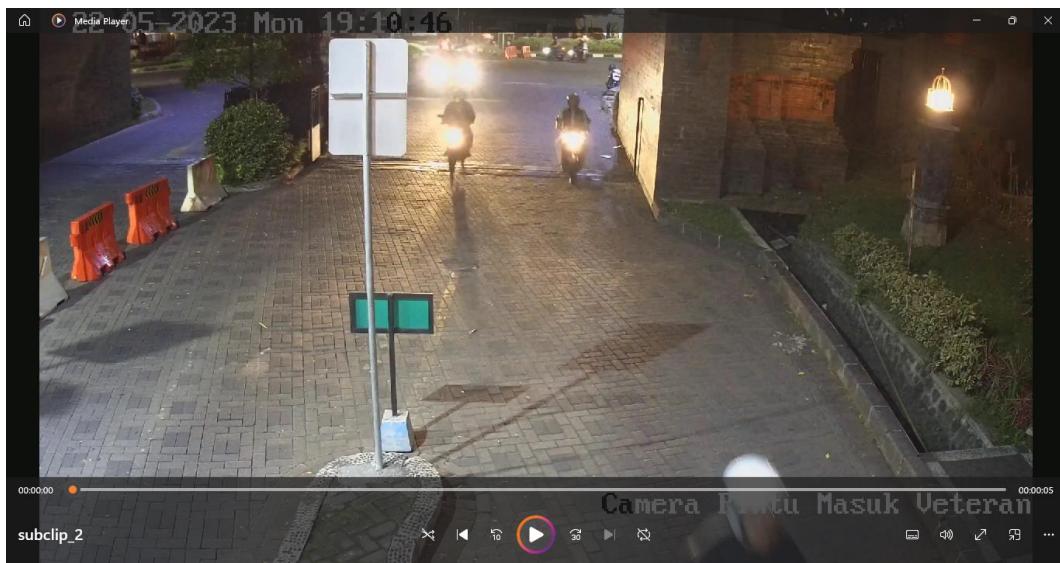
Tabel 6.1 Tabel Pengujian Deteksi Objek

No.	Preproses Gambar	Precision	Recall	F1-Score	Best confidence
1	Tanpa Preproses Gambar	0.91	0.855	0.88	0.566
2	Segmentasi U-Net dan <i>inpaint</i>	0.942	0.873	0.91	0.421

Penelitian ini melakukan perbandingan pada penggunaan metode preproses gambar dalam melakukan pelatihan model deteksi objek dan pelacakan objek untuk melihat apakah dapat meningkatkan performa model *deep learning*. Melihat dari hasil pengujian pada deteksi objek. Penggunaan metode preproses gambar dengan menggunakan metode segmentasi U-Net dan *inpaint* memiliki hasil yang lebih baik dibandingkan yang tidak menggunakan metode preproses gambar dari metrik *precision*, *recall*, dan *F1 score* dengan masing-masing nilai sebesar 0.942, 0.873, dan 0.91. Namun dalam penilaian *best confidence* dimana model memberikan batas nilai kepercayaan dalam melakukan deteksi objek, model tanpa penerapan metode *preprocessing* sedikit lebih tinggi dengan nilai *best confidence* sebesar 0.566. Namun masih ada beberapa kekurangan dari penggunaan metode segmentasi ini. Hal ini dikarenakan pelatihan model segmentasi U-Net memiliki mayoritas kesilauan yang disebabkan oleh cahaya yang berwarna kekuningan sehingga segmentasi pada cahaya silau pada cahaya yang berwarna putih sangat sedikit terdeteksi. Selain itu jumlah dataset yang dimiliki berjumlah sangat sedikit dengan total 200 yang dipisah menjadi 80% dan 20% untuk masing-masing data pelatihan dan validasi data. Sehingga metode ini memungkinkan dapat bekerja lebih baik dalam hal deteksi objek jika memiliki lebih banyak dataset *masking flare* atau kesilauan terkhususnya pada cahaya yang berwarna putih.

6.2 Pengujian Pelacakan Objek

Pengujian pelacakan objek akan dilakukan terhadap 3 video yang berbeda dengan menggunakan metrik MOTA dan MOTP sebagai pengukur kinerja pelacakan objek. Hal yang menjadi perbedaan dari pengujian terhadap video berupa dari model YOLOv5 yang sudah dilatih sebelumnya. Video yang digunakan untuk pengujian memiliki rentang waktu sekitar 4 sampai 6 detik dengan persyaratan pada video tersebut memiliki objek yang lebih dari 1. Hal ini dikarenakan MOTA dan MOTP merupakan metrik yang menghitung akurasi dan presisi pelacakan objek yang berjumlah lebih dari satu. Sedangkan kriteria pemilihan video lainnya adalah video yang digunakan adalah video dengan kendaraan arah masuk saja dan kendaraan yang menunjukkan *glare* atau kesilauan pada video tersebut. Contoh potongan gambar video pengujian pelacakan dapat dilihat pada Gambar 6.5.



Gambar 6.5 Contoh Gambar dari Video Pengujian Pelacakan

Model yang diuji berjumlah 2 dengan masing-masing model dilatih tanpa preproses gambar dan model yang dilatih dengan preproses gambar dengan menggunakan metode segmentasi dan *inpaint*. Selain itu, nilai *confidence* yang digunakan apabila sebuah objek terdeteksi pada model yang digunakan menggunakan nilai *best confidence* dari penilaian *F1 score* yang dapat dilihat pada Tabel 6.1 pada masing-masing model. Nilai MOTA dan MOTP yang didapatkan dari masing-masing model dapat dilihat pada Tabel 6.2.

Tabel 6.2 Hasil Pengujian Pelacakan Objek

No.	Video	Preproses Gambar	MOTA	MOTP
1.	Sub_clip1.mp4	Tanpa Preproses Gambar	0.787634	0.234077
		Segmentasi U-Net dan <i>inpaint</i>	0.704301	0.244867
2.	Sub_clip2.mp4	Tanpa Preproses Gambar	0.716418	0.207769
		Segmentasi U-Net dan <i>inpaint</i>	0.636816	0.224216
3.	Sub_clip3.mp4	Tanpa Preproses Gambar	0.538306	0.217443
		Segmentasi U-Net dan <i>inpaint</i>	0.377016	0.224447
4.	Sub_clip4.mp4	Tanpa Preproses Gambar	0.747368	0.198892
		Segmentasi U-Net dan <i>inpaint</i>	0.631579	0.221444

Pada pengujian kualitas pelacakan dengan metrik MOTA dan MOTP. Berbanding terbalik dengan hasil deteksi objek, penggunaan model deteksi objek yang tidak menggunakan metode preproses gambar jauh lebih baik dalam melakukan pelacakan objek dibandingkan dengan model yang dilatih dengan gambar yang telah dilakukan preproses sebelumnya dengan metode segmentasi U-Net dan *inpaint*. Penggunaan metrik MOTA bertujuan untuk menghitung akurasi sedangkan MOTP bertujuan untuk menghitung presisi. Nilai rata-rata MOTA dan MOTP pada model tanpa preproses gambar masing-masing berjumlah 0.6974315 dan 0.21454525. Sedangkan nilai rata-rata MOTA dan MOTP pada model yang menggunakan preproses gambar masing-masing bernilai 0.5874285 dan 0.2287435.

Pada pengujian nilai MOTA pada kedua metode pada pengujian video sub_clip3.mp4 rendah dikarenakan masing-masing hasil inferensi metode melacak sesuatu yang bukan objeknya. Dari hasil pengamatan, kesalahan pelacakan pada

metode tanpa preproses terjadi pada area arah keluar gerbang yang mana terdeteksi kelas mobil pada model pembelajaran. Sehingga nilai akurasi MOTA terganggu dikarenakan objek tersebut tidak termasuk kedalam ground truth atau nilai nyata. Sedangkan pada metode dengan segmentasi U-Net dan inpaint kesalahan terjadi adanya deteksi ganda pada sebuah objek motor. Ini kemungkinan besar dikarenakan penggunaan metode inpaint yang kurang sempurna yang mulanya bertujuan untuk mengurangi kesilauan justru menghilangkan bentuk objek dari kendaraan motor. Namun tidak dengan penilaian precision yang mana masih stabil jika dibandingkan dengan hasil inferensi pada pengujian pelacakan pada video lainnya. Ini dikarenakan MOTP hanya menguji peletakan bounding box yang sesuai antara bounding box pada ground truth dan bounding box pada hasil pelacakan.

6.3 Pengujian Perhitungan Jumlah Objek

Pengujian selanjutnya adalah melakukan perhitungan jumlah objek untuk melihat apakah ada pengaruh dari jumlah perhitungan dari sebuah video dengan menerapkan metode YOLOv5 untuk melakukan deteksi objek dan algoritma *DeepSORT* pada pelacakan objek. Hal yang membedakan dari penggunaan algoritma tersebut untuk menilai kualitas dalam melakukan perhitungan pada keadaan gelap berupa penggunaan metode preproses gambar atau tidak. Pengujian menggunakan 4 video dengan masing-masing video berdurasi 2 hingga 3 menit dengan mempertimbangkan jumlah objek yang banyak dan kesilauan yang ada pada video yang diuji. Contoh dari Hasil inferensi dapat dilihat pada Gambar 6.6.



Gambar 6.6 Hasil Video Inferensi Pengujian Tanpa Menggunakan Metode Preproses Gambar

Objek yang akan dihitung hanyalah objek yang masuk ke dalam lingkungan universitas Brawijaya dan tidak mempertimbangkan dan menghitung jumlah objek yang keluar dari Universitas Brawijaya melalui gerbang veteran baik untuk objek

mobil ataupun motor. Sebuah kendaraan akan terhitung masuk jika kordinat dari nilai titik y bawah suatu objek sudah melewati garis pembatas berwarna putih dan nilai titik y atas masih berada di atas garis pembatas berwarna putih.

Perhitungan dilakukan dengan menghitung nilai TP (*True positive*), FN (*False Negative*), dan FP (*False Positive*). Nilai TP dihitung jika objek mobil dan motor yang terhitung benar merupakan objek mobil dan motor yang sesuai. Nilai FP dihitung jika misalnya tidak ada objek mobil dan motor pada kenyataan tapi terdeteksi ada mobil atau motor yang terdeteksi dan terhitung. FN terhitung jika secara kenyataan ada mobil dan motor namun tidak terdeteksi oleh sistem inferensi dan tidak terhitung ke dalam perhitungan objek. TN (*True Negative*) tidak akan dihitung pada pengamatan dikarenakan kelas kosong atau objek yang tak terdeteksi tidak dianggap. Ringkasan hasil dari pengujian terhadap perhitungan jumlah kendaraan mobil dan motor yang masuk ke dalam Universitas Brawijaya pada gerbang veteran dapat dilihat pada Tabel 6.3.

Pada pengujian perhitungan kendaraan, nilai rata-rata *F1 score* dan akurasi tertinggi masih diungguli oleh penggunaan model yang tidak menggunakan metode preproses gambar dengan rata-rata *F1 score* 0.9935 dan rata-rata akurasi sebesar 0.987. Namun nilai rata-rata *F1 score* dan akurasi pada model yang menggunakan metode preproses tidak terlalu jauh dengan rata-rata *F1 score* 0.984 dan rata-rata akurasi sebesar 0.97 jika dibandingkan dengan metode preproses.

Tabel 6.3 Hasil Pengujian Perhitungan Objek

No.	Video	Nilai Asli		<i>Preprocess Image</i>	TP	FP	FN	<i>Precision</i>	<i>Recall</i>	Akurasi	<i>F1 score</i>
		Mobil	Motor								
1.	test_c1.mp4	2	22	Tidak	24	0	0	1	1	1	1
				Segmentasi U-Net dan <i>inpaint</i>	24	0	0	1	1	1	1
2.	test_c2.mp4	4	42	Tidak	45	0	1 Motor	1	0.978	0.978	0.988
				Segmentasi U-Net dan <i>inpaint</i>	44	0	2 Motor	1	0.956	0.956	0.977
3.	test_c3.mp4	1	35	Tidak	36	0	0	1	1	1	1
				Segmentasi U-Net dan <i>inpaint</i>	36	0	0	1	1	1	1
4.	test_c4.mp4	4	34	Tidak	37	0	1 Motor	1	0.973	0.973	0.986
				Segmentasi U-Net dan <i>inpaint</i>	37	1 Mobil 1 Motor	1 Motor	0.948	0.973	0.925	0.9603

BAB 7 PENUTUP

7.1 Kesimpulan

Berdasarkan analisis dan hasil pengujian yang telah dilakukan, maka dapat disimpulkan sebagai berikut:

1. Penggunaan preproses gambar dengan metode segmentasi gambar U-Net dan *inpaint* dapat meningkatkan nilai *precision* sebesar 0.032, *recall* 0.018, dan *F1-Score* sebesar 0.3 pada model deteksi objek dibandingkan model yang dilatih pada data set tanpa melakukan preproses gambar sebelumnya. Namun nilai *best confidence* masih diungguli oleh model yang dilatih tanpa preproses gambar dengan selisih 0.145.
2. Penggunaan model yang dilatih pada dataset tanpa melakukan preproses gambar sebelumnya lebih unggul pada evaluasi pelacakan dengan metrik MOTA dan MOTP dengan nilai selisih rata-rata MOTA 0.11 lebih tinggi dan selisih rata-rata MOTP -0.014 lebih rendah dibandingkan evaluasi pelacakan pada model deteksi objek yang dilatih dengan metode preproses segmentasi U-Net dan *inpaint*.
3. Hasil evaluasi perhitungan kendaraan yang masuk pada gerbang veteran pada Universitas Brawijaya didapatkan pelacakan dengan model deteksi pada dataset gambar tanpa preproses sebelumnya lebih unggul dibanding model yang dilatih dengan metode preproses gambar dengan nilai rata-rata *F1-Score* 0.0095 dan akurasi 0.017 lebih tinggi dibandingkan model pada dataset gambar yang dilatih dengan metode preproses segmentasi U-Net dan *inpaint*.

7.2 Saran

Berdasarkan Batasan masalah dan hasil yang didapatkan dari penelitian ini, terdapat beberapa saran untuk keberlanjutan penelitian. Terkhususnya penelitian pada bidang visi computer dalam mengatasi pemasalahan deteksi, pelacakan, dan perhitungan objek yang terkendala dengan kesilauan. Berikut saran-saran yang dapat dibertimbangkan diantara lain sebagai berikut:

1. Penambahan dataset gambar pelatihan dan validasi deteksi objek kemungkinan dapat meningkatkan kinerja dan fungsionalitas dari model YOLOv5. Model bisa menghasilkan nilai *F1 score* pada nilai kepercayaan atau *confidence* yang lebih tinggi dari penelitian ini.
2. Penggunaan metode *inpaint* untuk resotorasi gambar yang mengalami kesilauan mungkin kurang tepat guna. Penggunaan GAN atau singkatan dari *Generative Adversarial Network* mungkin lebih cocok dalam melakukan restorasi gambar dikarenakan metode ini merupakan metode terbaru dalam tugas restorasi gambar.
3. Penggunaan metode segmentasi U-Net dan *inpaint* untuk melakukan preproses gambar dalam mengatasi kesilauan atau *flare* pada gambar kurang efektif

terhadap data *real time*. Ini dikarenakan dalam tiap 1 detik, hanya 8 *frame* yang selesai terpreproses. Sehingga disarankan untuk melakukan pengoptimalan dalam penggunaan implementasi metode segmentasi U-Net dan *inpaint* pada tugas preproses gambar.

DAFTAR REFERENSI

- Bewley, A., Ge, Z., Ott, L., Ramos, F. and Upcroft, B., 2016. Simple Online and Realtime Tracking. In: *IEEE*. [online] pp.3464–3468. Tersedia di: <<https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7533003>> [Diakses 23 Agustus 2023].
- Du, Y., Zhao, Z., Song, Y., Zhao, Y., Su, F., Gong, T. and Meng, H., 2023. StrongSORT: Make *DeepSORT* Great Again. *IEEE Transactions on Multimedia*, [online] pp.1–14. Tersedia di: <<https://doi.org/10.1109/TMM.2023.3240881>> [Diakses 26 Oktober 2023].
- Esfahani, M. and Wang, H., 2021. *Robust Glare Detection: Review, Analysis, and Dataset Release*.
- Fiaz, M., Mahmood, A. and Jung, S.K., 2018. Tracking Noisy targets: a Review of Recent Object Tracking Approaches. *CoRR*, [online] abs/1802.03098. Available at: <<http://arxiv.org/abs/1802.03098>>.
- Guillemot, C. and Meur, L., 2014. Image Inpainting : Overview and Recent Advances. *Signal Processing Magazine, IEEE*, 31, pp.127–144. <https://doi.org/10.1109/MSP.2013.2273004>. [Diakses 18 Januari 2024]
- Guo, F. and Xu, Y., 2022. Vehicle Analysis System Based on *DeepSORT* and YOLOv5. In: *2022 3rd International Conference on Computer Vision, Image and Deep Learning & International Conference on Computer Engineering and Applications (CVIDL & ICCEA)*. [online] pp.175–179. Tersedia di: <<https://doi.org/10.1109/CVIDLICCEA56201.2022.9824363>> [Diakses 23 Agustus 2023]
- Jocher, G., Chaurasia, A., Stoken, A., Borovec, J., NanoCode012, Kwon, Y., Michael, K., TaoXie, Fang, J., imyhxy, Lorna, 曾逸夫(Zeng Yifu, Wong, C., Abhiram V, Montes, D., Wang, Z., Fati, C., Nadar, J., Laughing and UglvKitDe, 2022. ultralytics/yolov5: v7.0 - YOLOv5 SOTA Realtime Instance Segmentation.

[online] Tersedia di: <<https://doi.org/10.5281/zenodo.7347926>> [Diakses 23 Agustus 2023].

Kutlimuratov, A., Khamzaev, J., Kuchkorov, T., Anwar, M.S. and Choi, A., 2023. Applying Enhanced Real-Time Monitoring and Counting Method for Effective Traffic Management in Tashkent. *Sensors*, 23(11), [online] p.5007. Tersedia di: <<https://doi.org/10.3390/s23115007>> [Diakses 25 Oktober 2023].

Lin, L., He, H., Xu, Z. and Wu, D., 2023. Realtime Vehicle Tracking Method Based on YOLOv5 + DeepSORT. *Computational Intelligence and Neuroscience*, 2023, [Online] pp.1–11. Tersedia di: <<https://doi.org/10.1155/2023/7974201>> [Diakses 27 Oktober 2023].

Milan, A., Leal-Taixé, L., Reid, I.D., Roth, S. and Schindler, K., 2016. MOT16: A benchmark for multi-object tracking. *CoRR*, [Online] abs/1603.00831. Tersedia di: <<http://arxiv.org/abs/1603.00831>> [Diakses 06 Februari 2024].

Olaf Ronneberger, Fischer, P. and Brox, T., 2015. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, [online] abs/1505.04597. Available at: <<http://arxiv.org/abs/1505.04597>>.

Pangestu, I., 2022. *Mengenal Pengertian CCTV: Fungsi, Jenis dan Cara Kerjanya, Jasa Pembuatan Website - Metafora Indonesia Technology*. [online] idmetafora.com. Tersedia di: <<https://idmetafora.com/news/read/1411/Mengenal-Pengertian-CCTV-Fungsi-Jenis-dan-Cara-Kerjanya.html>> [Diakses 23 Agustus 2023].

Taye, M.M., 2023. Theoretical Understanding of Convolutional Neural Network: Concepts, Architectures, Applications, Future Directions. *Computation*, 11(3), [Online] p.52. Tersedia di: <<https://doi.org/10.3390/computation11030052>> [Diakses 27 Oktober 2023].

- Telea, A., 2004. An Image Inpainting Technique Based on the Fast Marching Method. *Journal of Graphics Tools*, 9.
<https://doi.org/10.1080/10867651.2004.10487596>.
- Thakur, N., Nagrath, P., Jain, R., Saini, D., Sharma, N. and Hemanth, J., 2021. Object Detection in Deep Surveillance. *Research Square (Research Square)*. [online] Tersedia di: <<https://doi.org/10.21203/rs.3.rs-901583/v1>> [Diakses 23 Agustus 2023]
- Wei, C., 2023. Vehicle Detecting and Tracking Application Based on YOLOv5 and DeepSORT for Bayer Data. In: *2022 17th International Conference on Control, Automation, Robotics and Vision (ICARCV)*. [online] pp.843–849. Tersedia di: <<https://doi.org/10.1109/ICARCV57592.2022.10004379>> [Diakses 23 Agustus 2023]
- Wu, S., Ge, F. and Zhang, Y., 2023. A Vehicle LinePressing Detection Approach Based on YOLOv5 and DeepSORT. In: *2022 IEEE 22nd International Conference on Communication Technology (ICCT)*. [online] pp.1745–1749. Tersedia di: <<https://doi.org/10.1109/ICCT56141.2022.10072680>> [Diakses 23 Agustus 2023]
- Yang, Y., Cheng, Z., Yu, H., Zhang, Y., Cheng, X., Zhang, Z. and Xie, G., 2022. MSENet: generative image inpainting with multiscale encoder. *The Visual Computer*, [online] 38(8), pp.2647–2659. Tersedia di: <<https://doi.org/10.1007/s00371021021430>> [Diakses 18 Januari 2024].
- Zhao, Z.Q., Zheng, P., Xu, S.T. and Wu, X., 2019. Object Detection with Deep Learning: a Review. *IEEE Transactions on Neural Networks and Learning Systems*, [online] 30(11), pp.3212–3232. Tersedia di: <<https://doi.org/10.1109/TNNLS.2018.2876865>> [Diakses 23 Agustus 2023]