BAB II
KAJIAN LITERATUR

2.1 *Artificial Intelligence*

Kecerdasan Buatan *(Artificial Intelligence)* adalah cabang ilmu komputer yang mengembangkan sistem yang mampu meniru kemampuan manusia dalam mengenali pola, belajar dari data sebelumnya, mengambil keputusan otomatis [18]. AI dirancang untuk menangani tugas – tugas yang memerlukan kecerdasan manusia, dari pengolahan data hingga *problem solving* yang kompleks [19]. Dengan kemampuan adaptasi tersebut yang semakin meningkat, AI menjadi teknologi yang sangat dipertimbangkan dalam berbagai bidang termasuk pengelolaan lingkungan.

Model AI memfasilitasi pemilahan sampah secara otomatis, yang sangat penting untuk pengelolaan sampah yang efisien dan mengurangi tenaga kerja manual. Hal ini khususnya penting di lokasi terpencil atau di dalam laut yang mana pembersihan manual menjadi tantangan [20] [21]. Model AI diintegrasikan dengan teknologi IoT untuk menciptakan sistem klasifikasi sampah yang komprehensif. AI dapat mendeteksi dan mengklasifikasikan sampah laut secara akurat, berkontribusi pada pengelolaan polusi lingkungan laut yang efektif [22].

Studi-studi terbaru menunjukkan bahwa model seperti YOLOv5 dan YOLOv8, serta CNN yang dioptimalkan, mampu mencapai presisi dan *recall* tinggi dalam mendeteksi sampah laut baik pada permukaan maupun di bawah air [4] [23] [24]. Selain itu, integrasi AI dengan sistem *monitoring real-time* memungkinkan pengambilan keputusan yang lebih cepat dan berbasis bukti, mendukung upaya mitigasi pencemaran laut secara efektif dan efisien [24] [25].

2.2 *Machine Learning*

Machine Learning (ML) adalah cabang dari *Artificial Intelligence* (AI) yang berfokus pada pengembangan algoritma dan model statistik yang memungkinkan komputer untuk belajar dari data dan meningkatkan kinerjanya secara otomatis tanpa perlu diprogram. Bidang ini erat kaitannya dengan pengenalan pola, statistika komputasi, dan kecerdasan buatan, serta banyak digunakan dalam berbagai bidang di mana pemrograman tradisional sulit diterapkan.

Salah satu algoritma ML yang populer dalam deteksi sampah laut adalah YOLO *(You Only Look Once)*, khususnya versi terbaru seperti YOLOv8. model YOLOv8 mampu

mendeteksi sampah mengapung di permukaan sungai dengan *precision* sebesar 84,02% dan *recall* 91,03%, membuktikan efektivitas algoritma ini dalam pengenalan objek secara *real-time* [26]. Implementasi teknologi ini juga dikembangkan pada drone dan robot bawah air untuk memantau distribusi sampah laut secara spasial dan waktu nyata [27]

Keunggulan *machine learning* dalam pengelolaan sampah laut tidak hanya terletak pada kemampuan identifikasi yang cepat dan akurat, tetapi juga pada kemampuannya menyediakan data berbasis bukti yang penting untuk pengambilan keputusan. Dengan integrasi teknologi IoT dan sensor, data yang dikumpulkan dapat dianalisis menggunakan model ML untuk memprediksi pola akumulasi sampah dan memberikan rekomendasi pengelolaan yang optimal [28]. Hal ini menjadikan *machine learning* sebagai solusi strategis dalam menghadapi tantangan pencemaran laut dan mendukung pelestarian ekosistem secara berkelanjutan.

2.3 *Deep Learning*

Algoritma *deep learning* mengenali berbagai jenis sampah di perairan kompleks dengan memanfaatkan kekuatan arsitektur jaringan saraf tiruan, terutama *Convolutional Neural Network* (CNN) dan model deteksi objek seperti YOLO dan Faster R-CNN. Model-model ini dilatih menggunakan dataset citra yang berisi beragam jenis sampah mulai dari plastik, botol, jaring ikan, dan kondisi lingkungan perairan [29]. Proses pelatihan memungkinkan model untuk mengekstrak fitur visual spesifik, seperti tekstur, warna, dan kontur, sehingga mampu membedakan antara satu jenis sampah dan lainnya, bahkan di tengah gangguan visual seperti air keruh atau refleksi cahaya

Dalam implementasinya, CNN dan YOLO secara otomatis melakukan ekstraksi fitur dari citra *input* melalui beberapa lapisan konvolusi dan *pooling*, yang memungkinkan identifikasi objek meskipun terdapat variasi pencahayaan, posisi, atau tumpang tindih antar objek [30]. Penelitian menunjukkan bahwa model seperti YOLOv8 dan Faster R-CNN dapat mendeteksi berbagai jenis sampah di perairan dengan nilai *precision* dan *recall* yang tinggi. Sebagai contoh, sistem berbasis *Faster* R-CNN mampu mengidentifikasi botol plastik dengan akurasi 96,3%, gelas plastik 100%, dan *styrofoam* 100% [29].

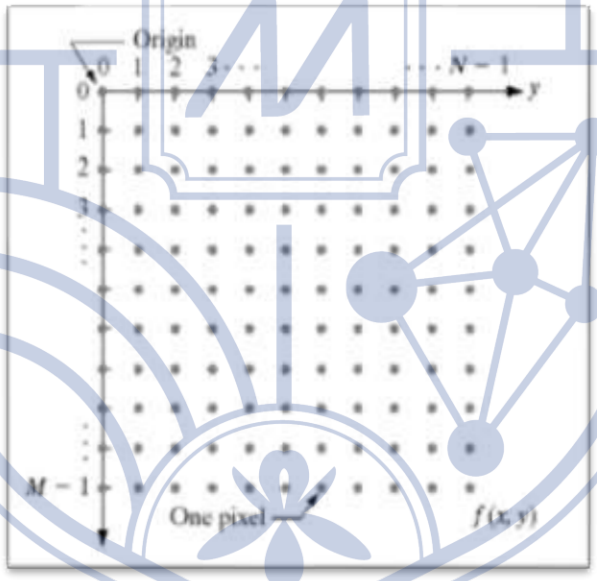
Selain itu, optimasi parameter model, pemilihan arsitektur yang tepat (misalnya ResNet50, YOLOv8), serta penggunaan teknik augmentasi data sangat berpengaruh terhadap tingkat akurasi dan kemampuan generalisasi model dalam kondisi perairan yang kompleks [31]. Dengan pendekatan ini, algoritma *deep learning* tidak hanya efektif dalam mengenali dan mengklasifikasikan berbagai jenis sampah, tetapi juga dapat diintegrasikan

ke dalam sistem *monitoring* otomatis untuk mendukung pengelolaan dan pelestarian lingkungan laut secara efisien.

2.4 Citra Digital

Citra digital merupakan gambar sebagai fungsi dua dimensi, di mana koordinat (x, y) mewakili lokasi spasial, dan nilai fungsi (f) menunjukkan intensitas atau tingkat abu-abu pada koordinat tersebut. Konsep komposisi gambar digital menekankan bahwa gambar digital terdiri dari sejumlah elemen terbatas yang dikenal sebagai piksel, yang masing-masing memiliki lokasi dan nilai tertentu. Konsep ini penting bagi algoritma AI yang memproses dan menganalisis gambar, karena mereka bergantung pada data piksel untuk [32] membuat keputusan.

Suatu citra dapat didefinisikan sebagai fungsi f(x, y) berukuran M baris dan N kolom, dengan x dan y adalah koordinat posisi, dan f(x, y) yang menyatakan besar intensitas citra atau tingkat keabuan atau warna dari piksel di titik tersebut.



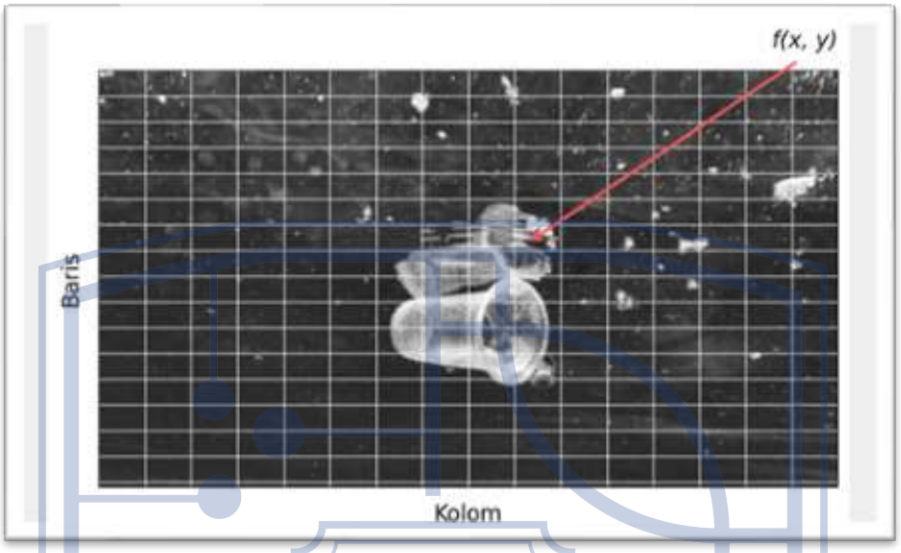
Gambar 2. 1 Titik Koordinat Citra Digital [32]

Citra digital dapat direpresentasikan sebagai matriks seperti berikut:

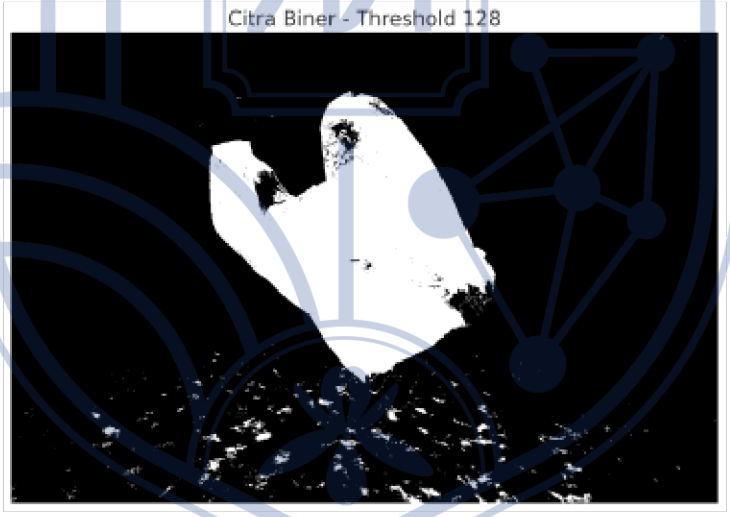
$$f(x,y) \approx \begin{bmatrix} f(0,0) & f(0,1) & \dots & f(0,M) \\ f(1,0) & f(1,1) & \dots & f(1,M) \\ \vdots & \vdots & \vdots & \vdots \\ f(N-1,0) & f(N-1) & \dots & f(N-1,M-1) \end{bmatrix}$$

Gambar 2. 2 Matriks Citra Digital [33]

Nilai pada suatu potongan antara baris dan kolom (pada posisi x, y) disebut *picture elements*, *image elements*, *pels (pixels)* yang sering digunakan pada citra digital. Gambar 2.3 menunjukkan ilustrasi digitalisasi citra dengan M= 16 baris dan N= 16 kolom.

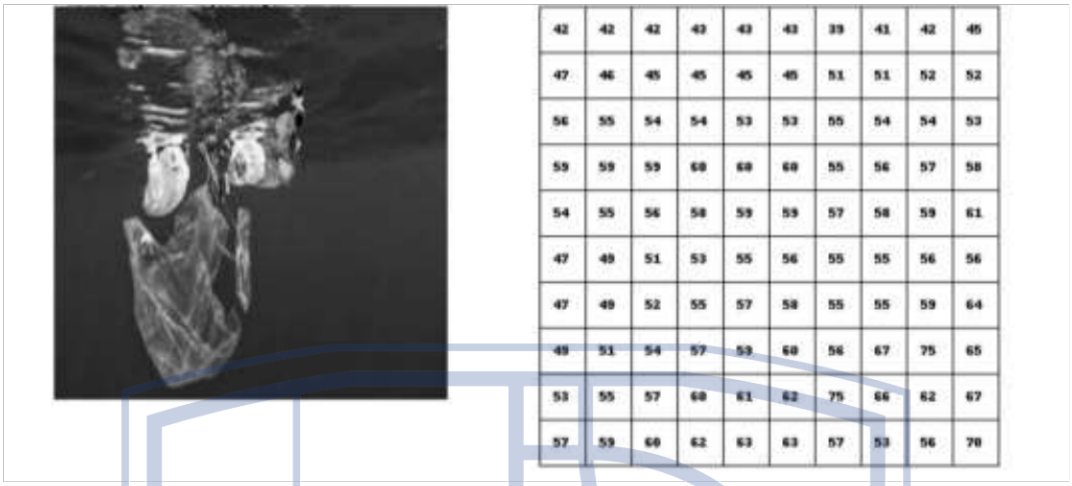


Gambar 2. 3 Illustrasi Citra Digital (Pixel Pada Koordinat x=7, y =9) [34]



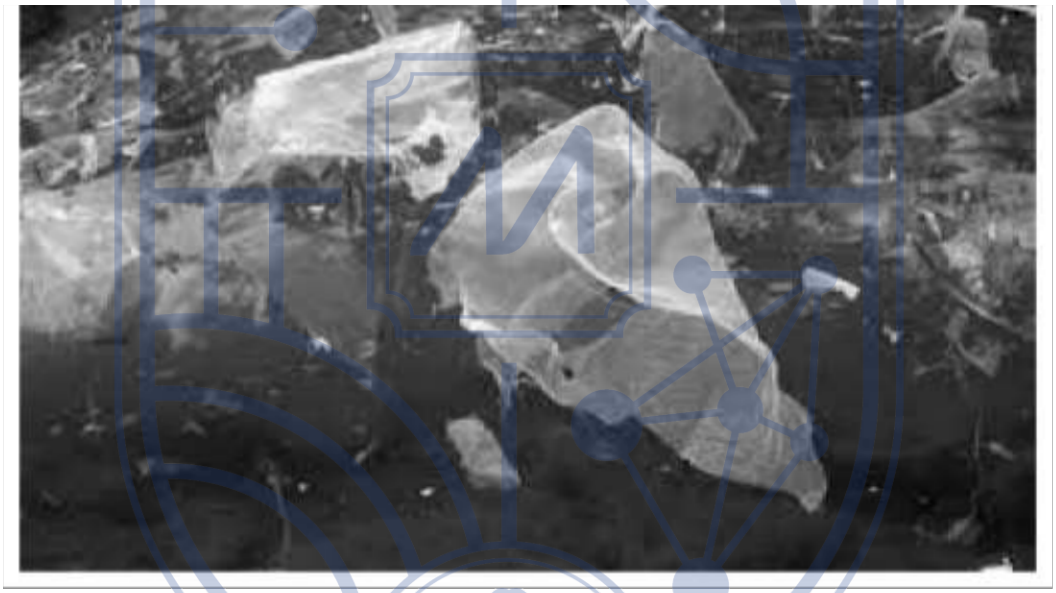
Gambar 2. 4 Citra *Biner*

Gambar 2.4 menunjukkan Citra Biner (Monokrom). Banyaknya dua warna, yaitu hitam dan putih. Dibutuhkan 1 bit di memori untuk menyimpan kedua warna ini



42	42	42	43	43	43	39	41	42	45
47	46	45	45	45	45	51	51	52	52
56	55	54	54	53	53	55	54	54	53
59	59	59	60	60	60	55	56	57	58
54	55	56	58	59	59	57	58	59	61
47	49	51	53	55	56	55	55	56	56
47	49	52	55	57	58	55	55	59	64
49	51	54	57	59	60	56	67	75	65
53	55	57	60	61	62	75	66	62	67
57	59	60	62	63	63	57	53	56	70

Gambar 2. 5 Citra *Grayscale* Dengan Tabel Intensitas



Gambar 2. 6 Citra *Grayscale*

Citra *Grayscale* (Skala Keabuan). Banyaknya warna tergantung pada jumlah bit yang disediakan di memori untuk menampung kebutuhan warna ini. Citra 2 bit mewakili 4 warna, citra 3 bit mewakili 8 warna, dan seterusnya. Semakin besar jumlah bit warna yang disediakan di memori, semakin halus gradasi warna yang terbentuk.

2.5 Pengolahan Citra Digital

Pengolahan citra digital merupakan proses penting dalam bidang teknologi informasi dan komputer yang bertujuan untuk menganalisis serta memodifikasi gambar digital guna mendapatkan informasi atau tampilan yang lebih baik. Salah satu metode utama dalam pengolahan citra adalah perbaikan kualitas citra *(image enhancement)*. Teknik ini digunakan

untuk meningkatkan kualitas visual suatu citra agar lebih mudah dikenali dan dianalisis, baik oleh manusia maupun sistem komputer. Perbaikan ini meliputi peningkatan kontras, penajaman detail, serta penegasan tepian objek agar objek dalam citra tampak lebih jelas dan informatif [35].

Selanjutnya, terdapat metode pemugaran citra *(image restoration)* yang berfokus pada pemulihan citra yang mengalami kerusakan atau gangguan selama proses pengambilan, penyimpanan, atau transmisi. Citra yang mengalami *noise*, blur, atau cacat visual lainnya dapat diperbaiki menggunakan teknik pemrosesan tertentu agar mendekati bentuk citra aslinya. Metode ini biasanya memanfaatkan model matematis untuk memperkirakan kondisi ideal dari citra sebelum terjadinya gangguan, sehingga hasil akhirnya menjadi lebih bersih dan akurat [36].

Metode berikutnya adalah kompresi citra *(image compression)* yang berfungsi untuk mengurangi ukuran file citra digital. Tujuannya adalah untuk menghemat ruang penyimpanan dan mempercepat proses transmisi data, khususnya pada sistem dengan keterbatasan *bandwidth* atau kapasitas penyimpanan. Kompresi dilakukan dengan cara menghilangkan informasi yang tidak terlalu penting tanpa mengurangi kualitas citra secara signifikan, sehingga citra tetap dap at digunakan sesuai kebutuhan. Ketiga metode ini saling melengkapi dalam rangka menghasilkan citra digital yang optimal baik dari segi kualitas maupun efisiensi [37].

2.6 Konversi RGB ke Grayscale

Metode yang paling umum digunakan untuk mengonversi gambar berwarna ke *grayscale* adalah dengan mempertahankan informasi kecerahan dan mengabaikan informasi tentang kroma dan *hue*. Salah satu contoh rumus konversi yang populer dapat ditemukan pada standar *PostScript*, yaitu:

$$GRAY = 0.30\ R + 0.59\ G + 0.11\ B \quad (1)$$

Rumus ini mengasumsikan bahwa nilai R (merah), G (hijau), dan B (biru) proporsional terhadap luminansi atau tingkat kecerahan yang dipersepsikan mata manusia. Dengan demikian, hasil konversi ini menghasilkan nilai *grayscale* yang cukup akurat dan sesuai dengan persepsi visual manusia terhadap kecerahan gambar [38] [39].

Selain rumus tersebut, terdapat beberapa metode lain untuk konversi RGB ke *grayscale*, seperti metode rata-rata *(average)* yang menghitung nilai rata-rata ketiga kanal warna, dan metode *lightness* yang menggunakan nilai maksimum dan minimum dari R, G,

B. Namun, metode *luminosity* dengan bobot berbeda pada masing-masing kanal (seperti rumus di atas) dianggap paling baik karena mempertimbangkan sensitivitas mata manusia terhadap warna hijau yang lebih besar dibanding merah dan biru [38]

2.7 CLAHE *(Contrast Limited Adaptive Histogram Equalization)*

Contrast Limited Adaptive Histogram Equalization (CLAHE) adalah teknik pengolahan citra yang digunakan untuk meningkatkan kontras gambar dengan cara membagi citra menjadi beberapa bagian kecil yang disebut *tiles*, kemudian melakukan histogram *equalization* secara adaptif pada setiap *tile* tersebut. Berbeda dengan metode *histogram equalization* biasa yang menerapkan transformasi seragam pada seluruh citra, CLAHE mengadaptasi transformasi berdasarkan distribusi intensitas lokal sehingga dapat memperbaiki kontras secara lebih merata dan detail pada setiap bagian gambar [40] [41].

Salah satu keunggulan CLAHE adalah kemampuannya untuk membatasi amplifikasi *noise* yang sering terjadi pada metode *adaptive histogram equalization* (AHE) tradisional. CLAHE menerapkan batasan *(clip limit)* pada nilai histogram yang diperbesar, sehingga bagian citra yang homogen tidak mengalami peningkatan *noise* berlebih. Proses ini melibatkan pembagian citra menjadi *tiles*, komputasi histogram tiap *tile*, pembatasan nilai histogram, dan interpolasi bilinear antar *tiles* untuk menghasilkan citra akhir dengan kontras yang lebih baik dan *noise* yang terkontrol [40] [41] [42].

Dalam penerapannya, ukuran *kernel* atau *tile* sangat berpengaruh terhadap hasil kualitas citra dan pengurangan *noise*. Menggunakan *kernel* berukuran 5x5 pada citra *X-ray* untuk mengurangi *noise* dan meningkatkan kualitas gambar secara signifikan menggunakan CLAHE. Metode ini juga banyak digunakan dalam berbagai aplikasi pengolahan citra digital seperti perbaikan gambar medis, pengolahan citra satelit, dan pengenalan objek di lingkungan dengan pencahayaan rendah [43] [44] [45].

2.8 Bilateral *Filtering*

Bilateral *Filtering* adalah teknik pengolahan citra non-linear yang digunakan untuk mereduksi *noise* sekaligus mempertahankan tepi *(edge)* pada gambar [46]. Berbeda dengan filter *smoothing* tradisional seperti *Gaussian blur* yang menghaluskan seluruh gambar tanpa membedakan tepi, Bilateral *Filtering* menggabungkan dua faktor bobot: jarak spasial antar piksel dan perbedaan intensitas warna antar piksel. Hal ini memungkinkan filter untuk menghaluskan area dengan intensitas serupa sambil menjaga ketajaman tepi yang penting agar detail gambar tetap terjaga [47].

Secara matematis, nilai piksel hasil filter dihitung sebagai rata-rata berbobot dari piksel-piksel tetangga, di mana bobotnya merupakan hasil perkalian antara fungsi *kernel* spasial (berdasarkan jarak piksel) dan *kernel range* (berdasarkan perbedaan intensitas). Fungsi *kernel* ini biasanya berupa *Gaussian* yang menurunkan bobot piksel yang jauh secara spasial maupun berbeda secara intensitas. Dengan demikian, Bilateral *Filtering* mampu menghilangkan *noise* tanpa mengaburkan tepi objek dalam citra [46] [48] [49].

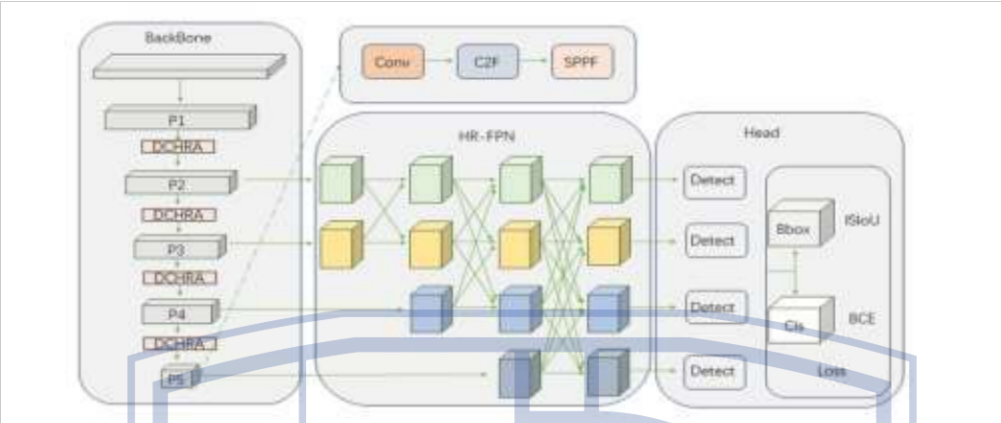
Bilateral Filtering banyak digunakan dalam berbagai aplikasi pengolahan citra, mulai dari pengurangan *noise* pada citra medis, pemrosesan citra satelit, hingga *computer vision*. Keunggulannya yang sederhana namun efektif membuatnya populer dalam *pipeline* pengolahan citra, terutama ketika detail tepi sangat penting.

2.9 YOLOv8

YOLO *(You Only Look Once)* merupakan salah satu algoritma deteksi objek berbasis *deep learning* yang dikenal dengan kecepatan dan akurasinya. YOLO yang awalnya diperkenalkan pada tahun 2015 telah berkembang melalui beberapa versi, yang masing-masing meningkatkan kecepatan dan akurasi sekaligus mengatasi berbagai tantangan dalam aplikasi *real-time*.

YOLOv1 (2015), Memperkenalkan pendekatan deteksi satu tahap, menggabungkan klasifikasi dan deteksi dalam satu jaringan saraf [50]. YOLOv2 (2017), Dikenal sebagai "YOLO9000," memiliki peningkatan dalam akurasi dan dapat mendeteksi lebih dari 9000 kelas objek [51]. YOLOv3 (2018), Menggunakan arsitektur yang lebih dalam dengan fitur *multi-scale prediction* untuk meningkatkan deteksi objek kecil [52]. YOLOv4 (2020), Dikembangkan oleh Alexey Bochkovskiy, meningkatkan efisiensi melalui optimalisasi arsitektur *backbone* [53]. YOLOv5 (2020), Dikembangkan oleh *Ultralytics*, memperkenalkan peningkatan efisiensi dalam pelatihan dan inferensi [53]. YOLOv6, YOLOv7 (2022), Memperkenalkan berbagai optimasi pada kecepatan dan akurasi [51]. YOLOv8 (2023), Versi terbaru yang dikembangkan oleh *Ultralytics* dengan peningkatan dalam deteksi, segmentasi, dan *tracking* objek [53].

2.9.1 Jenis-Jenis Model YOLOv8



Gambar 2. 7 Variasi dari Arsitektur FPN pada YOLOv8 [65]

a. YOLOv8-n (Nano):

Dikenal karena kecepatan pemrosesannya yang tinggi, sehingga cocok untuk aplikasi yang memerlukan deteksi cepat, seperti sistem pemantauan waktu nyata. Model ini mencapai kecepatan pemrosesan tertinggi sebesar 0,95 milidetik dalam tugas deteksi pesawat [54].

b. YOLOv8-s (Small)

Menawarkan keseimbangan antara kecepatan dan akurasi, sehingga cocok untuk aplikasi yang memiliki sumber daya komputasi sedang [55]

c. YOLOv8-m (Medium):

Memberikan jalan tengah dalam hal kompleksitas dan kinerja, sering digunakan dalam skenario di mana kecepatan dan akurasi sangat penting [55].

d. YOLOv8l (Large) :

Dirancang untuk akurasi yang lebih tinggi, cocok untuk tugas yang memerlukan deteksi objek terperinci, seperti penilaian kualitas dalam industri makanan [56]

e. YOLOv8 (Extra Large):

Model paling kompleks dalam seri ini, menawarkan akurasi tertinggi, seperti yang ditunjukkan dalam deteksi pesawat dengan presisi 0,94 dan *F1-score* 0,83 [54]

Performa model YOLOv8 merupakan hasil langsung dari peningkatan dan pengoptimalan arsitektur pada versi sebelumnya. Tabel berikut menyajikan ikhtisar komprehensif varian model YOLOv8, termasuk jumlah parameter, akurasi dalam hal Presisi

Rata-rata (mAP@0.5), dan waktu inferensi pada platform *CPU* dan *GPU* untuk ukuran gambar standar 640 piksel.

Tabel 2. 1 Metrik Performa Model YOLOv8

Model	Params (Million)	*Accuracy* (mAP@0.5)	*CPU Time* (*ms*)	*GPU Time* (*ms*)
YOLOv8n	2.0	47.2	42	5.8
YOLOv8s	9.0	58.5	90	6.0
YOLOv8m	25.0	66.3	210	7.8
YOLOv8l	55.0	69.8	400	9.8
YOLOv8x	90.0	71.5	720	11.5

Pada Tabel 2.1 di atas, mengilustrasikan kelebihan dan kekurangan yang melekat pada setiap model seri YOLOv8. Model terkecil, YOLOv8n, meskipun menawarkan waktu inferensi tercepat, memberikan akurasi yang lebih rendah dibandingkan dengan model yang lebih besar. Hal ini membuat YOLOv8n sangat cocok untuk aplikasi *edge computing* yang mengutamakan kecepatan dan sumber daya komputasi terbatas. Di sisi lain, YOLOv8x memberikan akurasi tertinggi, sehingga ideal untuk kasus penggunaan yang mengutamakan presisi, seperti dalam pencitraan medis atau aplikasi keamanan, tetapi memerlukan perangkat keras yang lebih kuat agar dapat berjalan secara efisien. Hasil ini menunjukkan fleksibilitas arsitektur YOLOv8 yang memungkinkan pengembang untuk memilih varian model yang paling sesuai dengan kebutuhan spesifik aplikasi yang dikembangkan, baik dari segi kecepatan, akurasi, maupun keseimbangan antara keduanya.

2.9.2 Komparasi Algoritma YOLOv8 dengan Algoritma lainnya

Dalam bidang visi komputer, khususnya pada tugas deteksi objek, berbagai algoritma telah dikembangkan dengan keunggulan dan kekurangan masing-masing. YOLOv8 sebagai generasi terbaru dari algoritma YOLO *(You Only Look Once)* menawarkan peningkatan signifikan dalam hal kecepatan, akurasi, dan efisiensi dibandingkan algoritma populer lainnya seperti Faster R-CNN, SSD, dan Mask R-CNN. Perbandingan kinerja antara algoritma-algoritma ini penting untuk menentukan pilihan terbaik sesuai kebutuhan aplikasi, terutama pada konteks yang menuntut pemrosesan real-time dan sumber daya terbatas. Sub bab ini akan membahas komparasi mendalam mengenai keunggulan dan kelemahan YOLOv8 dibandingkan algoritma lain dalam berbagai aspek seperti kecepatan inferensi,

akurasi deteksi, efisiensi komputasi, serta kemampuan adaptasi pada berbagai kondisi lingkungan.

Tabel 2. 2 Komparasi YOLOv8 dengan algoritma lain

Aspek	YOLOv8	Faster R-CNN	SSD	Mask R-CNN
Kecepatan	Sangat cepat, cocok untuk aplikasi *real-time* dengan latensi *GPU* sekitar 1.3 ms [57] [58]	Lebih lambat karena proses dua tahap dan jaringan proposal wilayah, latensi *GPU* sekitar 54 ms [57]	Cepat, mendeteksi objek dalam satu *forward pass*, efisien secara komputasi [59] [60]	Lebih lambat, arsitektur dua tahap yang kompleks, kurang cocok untuk *real-time* [61] [62] [63]
Akurasi (mAP@50)	Tinggi, mencapai sekitar 0.62 hingga 93,5% dalam berbagai pengujian [57] [62]	Cukup tinggi, sekitar 0.41, unggul dalam deteksi objek kecil tapi lebih lambat [59] [57]	Sedang, efisien namun akurasi di bawah YOLOv8 dan Faster R-CNN [59] [60]	Sangat tinggi, mencapai 0.77 pada COCO, unggul dalam segmentasi objek [61] [62] [64]
Efisiensi Komputasi	Dioptimalkan dengan pengurangan GFLOPs dan parameter, cocok untuk perangkat dengan sumber daya terbatas [59] [58] [62]	Memerlukan sumber daya besar karena proses dua tahap dan jaringan proposal wilayah [59] [58] [60]	Lebih ringan dari Faster R-CNN, seimbang antara kecepatan dan akurasi [59] [60]	Lebih berat dan kompleks, membutuhkan komputasi tinggi dan waktu inferensi lebih lama [61] [62]

Kemampuan Deteksi Objek Kecil	Meningkat dengan mekanisme perhatian dan konvolusi dinamis, lebih baik dari versi sebelumnya dan SSD [62]	Baik, unggul dalam deteksi objek kecil karena *region* proposal [59] [58]	Kurang optimal untuk objek kecil dibanding Faster R-CNN dan YOLOv8 [59] [60]	Sangat baik, dengan segmentasi instan yang presisi [61] [62]
Ketahanan dan Adaptabilitas	Fitur fusi multi-skala dan fungsi kerugian yang ditingkatkan, tahan terhadap kondisi kompleks [62]	Kurang adaptif dibanding YOLOv8 dalam kondisi *real-time* dan lingkungan dinamis [58] [60]	Adaptasi sedang, cocok untuk aplikasi dengan kebutuhan kecepatan sedang. [59]	Sangat adaptif untuk segmentasi dan klasifikasi objek kompleks [61] [62]
Desain Model	Ringan dengan FasterNet Block dan RFCAConv, ukuran model kecil, inferensi cepat [62] [64]	Kompleks, dua tahap dengan *region* proposal *network* (RPN) [57] [60]	Model satu tahap dengan arsitektur sederhana [59] [60]	Dua tahap dengan segmentasi piksel, kompleks dan berat [61] [62]
Kelebihan Utama	Kecepatan tinggi, akurasi baik, efisiensi komputasi, cocok untuk aplikasi *real-time* [57] [62]	Akurasi tinggi terutama untuk objek kecil, cocok untuk aplikasi presisi tinggi [59] [57]	Kecepatan dan efisiensi, cocok untuk perangkat dengan sumber daya terbatas [59] [60]	Presisi segmentasi objek terbaik, cocok untuk tugas yang membutuhkan detail tingkat piksel [61] [62]
Kekurangan	Presisi segmentasi	Lambat dan berat, kurang	Akurasi lebih rendah	Kompleks dan lambat, sulit

	kurang dibanding Mask R-CNN, sedikit kompromi pada presisi *bounding box* [62]	cocok untuk aplikasi *real-time* [57] [60]	dibanding YOLOv8 dan Faster R-CNN, kurang optimal untuk objek kecil [59]	diimplementasikan untuk aplikasi dengan batasan sumber daya [61] [62]

Pada Tabel 2.2 di atas, YOLOv8 unggul dalam kecepatan dan efisiensi komputasi, menjadikannya pilihan utama untuk aplikasi *real-time* dengan kebutuhan akurasi tinggi. Faster R-CNN dan Mask R-CNN menawarkan akurasi yang sangat baik, terutama untuk objek kecil dan segmentasi instan, namun dengan biaya komputasi yang jauh lebih tinggi dan kecepatan yang lebih rendah. SSD menempati posisi tengah dengan keseimbangan antara kecepatan dan akurasi, meskipun kurang optimal untuk objek kecil.

2.9.3 Fungsi dan Implikasi

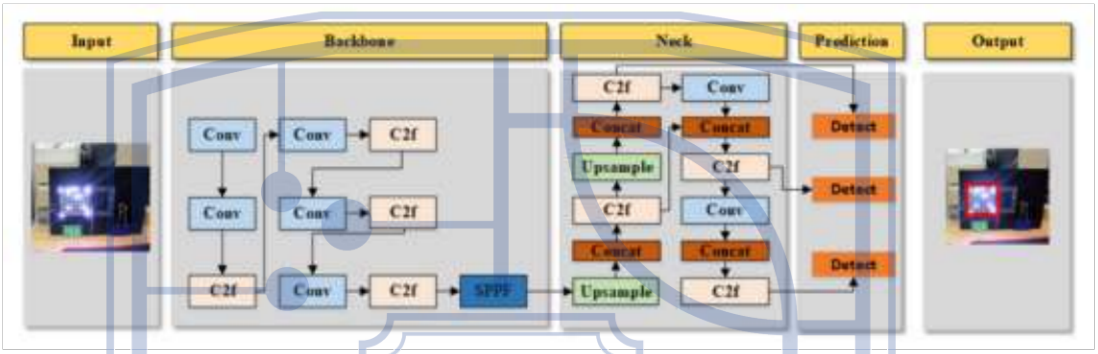
Pengaplikasian dan kontribusi setelah YOLO terus diperbarui ialah, YOLO telah diterapkan secara luas di berbagai bidang seperti mengemudi otonom, pengawasan, dan perawatan kesehatan, menunjukkan fleksibilitas dan efektivitasnya dalam skenario waktu nyata [16] [52], Pendekatan berbasis *grid* dari algoritma ini memungkinkan deteksi beberapa objek secara bersamaan, meningkatkan kegunaannya di lingkungan yang kompleks [16].

Meskipun YOLO telah menetapkan tolak ukur dalam pendeteksian objek, tantangan tetap ada, khususnya dalam mendeteksi objek kecil dan mempertahankan ketahanan dalam berbagai kondisi. Penelitian di masa mendatang dapat difokuskan pada penyempurnaan struktur jaringan dan peningkatan metodologi pelatihan untuk mengatasi keterbatasan ini [51].

2.9.4 Arsitektur YOLOv8

YOLOv8 dikembangkan dengan memanfaatkan fondasi kokoh dari versi-versi sebelumnya dalam keluarga YOLO, serta mengintegrasikan berbagai inovasi terkini dalam arsitektur jaringan saraf dan teknik pelatihan. Seperti pendahulunya, YOLOv8 menggabungkan proses pelokalan dan klasifikasi objek ke dalam satu kerangka kerja jaringan saraf *end-to-end* yang dapat dioptimalkan secara langsung, sambil tetap mempertahankan keseimbangan optimal antara kecepatan dan tingkat akurasi. Arsitektur

YOLOv8 terdiri dari tiga komponen utama : *Backbone* YOLOv8 menggunakan jaringan saraf konvolusional (CNN) canggih sebagai *backbone* yang dirancang untuk mengekstraksi fitur multi-skala dari gambar *input*. *Neck,* Modul *neck* bertugas menyempurnakan dan menggabungkan fitur multi-skala yang diekstraksi oleh *backbone*. *Head,* Modul *head* bertanggung jawab menghasilkan prediksi akhir, termasuk koordinat kotak pembatas *(bounding box)*, skor kepercayaan objek *(confidence score)*, dan label kelas dari fitur-fitur yang telah disempurnakan [65].



Gambar 2. 8 Proses Deteksi Objek *[66]*

Pendekatan anchor-free yang diterapkan dalam YOLOv8 menyederhanakan mekanisme prediksi, meminimalkan jumlah *hyperparameter* yang harus diatur, dan memperkuat adaptabilitas model terhadap objek dengan berbagai ukuran dan rasio aspek. Berkat penerapan inovasi arsitektural ini, YOLOv8 mampu memberikan peningkatan performa dalam deteksi objek, dengan hasil yang lebih akurat, proses yang lebih cepat, dan tingkat fleksibilitas yang lebih tinggi *[65]*.

2.9.5 Metode Training YOLOv8

Performa tinggi YOLOv8 dalam pendeteksian objek tidak hanya disebabkan oleh kemajuan arsitekturnya tetapi juga metodologi pelatihannya yang canggih:

a. Penambahan Augmentasi Data

   YOLOv8 menggabungkan serangkaian strategi augmentasi data baru yang meningkatkan generalisasi model.

b. Fungsi *Focal Loss*

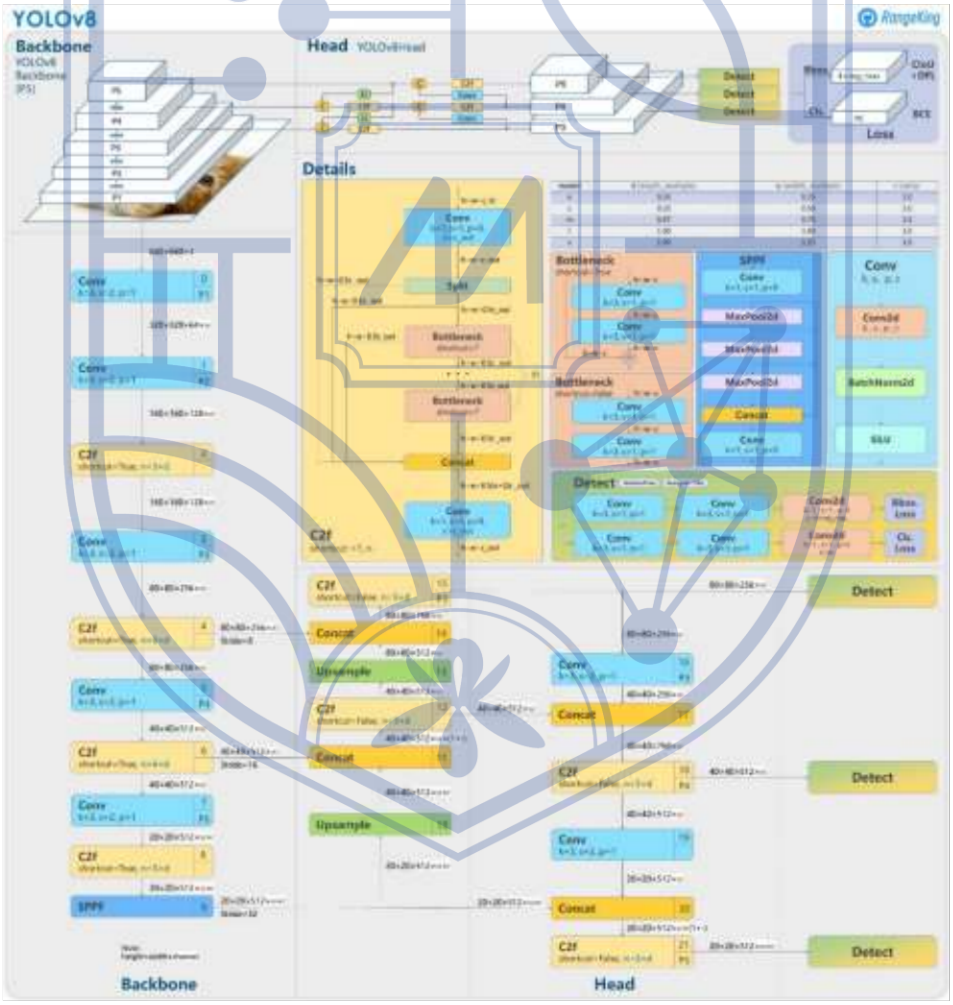
   Untuk meningkatkan performa klasifikasi, YOLOv8 menerapkan fungsi *focal loss* yang memprioritaskan contoh-contoh yang lebih sulit dikenali. Strategi ini secara efektif mengurangi dampak ketimpangan distribusi kelas dalam dataset dan memperkuat kemampuan model dalam mendeteksi objek yang kecil atau tertutup sebagian.

kemampuan model untuk mendeteksi objek kecil atau tertutup, yang sering kali kurang terwakili.

c. Transisi ke *PyTorch* dengan Optimasi

Sebagai bagian dari peralihan ke *platform PyTorch*, YOLOv8 mengoptimalkan desain jaringan dan prosedur pelatihannya agar lebih efisien di perangkat *GPU* masa kini. Dengan menggunakan teknik seperti *mixed-precision*, YOLOv8 mampu mengurangi waktu pelatihan dan prediksi, sekaligus tetap mempertahankan atau meningkatkan akurasi, menjadikannya ideal untuk implementasi pada sistem dengan sumber daya terbatas.

2.9.6 Struktur YOLOv8



Gambar 2. 9 Model Struktur Dari YOLOv8 *[67]*

2.9.7 Proses YOLOv8n-Cls (Pelatihan & Inferensi)

Gambar 2.8 menampilkan arsitektur dari algoritma YOLOv8n-cls. YOLOv8n-cls merupakan algoritma *Supervised* (terawasi) dan merupakan salah satu varian terbaru dari

keluarga YOLOv8 (*You Only Look Once* versi 8) yang dikhususkan untuk tugas klasifikasi gambar. Huruf n menandakan versi nano yaitu versi terkecil dan paling ringan dari model ini. cls adalah singkatan dari kata *classification* (klasifikasi).

Secara umum, arsitektur YOLOv8n-cls dibagi menjadi tiga komponen utama: (1) *Backbone*, (2) *Neck* (dalam varian klasifikasi ini dilewati), dan (3) *Head* klasifikasi.

1. *Backbone* (Ekstraktor Fitur)

   Backbone bertugas mengekstraksi fitur visual dari gambar masukan. Di dalam YOLOv8n-cls, *backbone* terdiri dari beberapa *layer* konvolusi (Conv), blok C2f (varian baru dari CSP yang mengatur aliran fitur agar efisien), serta modul SPPF (*Spatial Pyramid Pooling - Fast*) untuk memperkuat representasi spasial tanpa kehilangan resolusi.

2. *Neck* (Dilewati)

   Berbeda dari YOLOv8 untuk deteksi objek yang menggunakan Neck (biasanya PANet atau FPN), pada YOLOv8n-cls bagian *Neck* dilewati karena tidak diperlukan penggabungan fitur multi-skala dalam tugas klasifikasi.

3. *Head* (Klasifikasi)

   Setelah fitur diekstraksi oleh *backbone* dan diperkaya dengan SPPF, fitur spasial tersebut dirata-ratakan secara global menggunakan *Global Average Pooling* (GAP). Output vektor GAP kemudian diteruskan ke layer *Fully Connected* (FC) yang menghasilkan vektor prediksi dengan panjang sesuai jumlah kelas. Terakhir, fungsi aktivasi *Softmax* digunakan untuk mengubah nilai logit menjadi probabilitas kelas.

2.9.8 Evaluasi model YOLOv8n-cls

Model deteksi objek YOLOv8n-cls dievaluasi menggunakan sejumlah metrik kinerja utama, seperti presisi, *recall*, *F1-score*, dan *confusion matrix*. Metrik-metrik ini memberikan gambaran menyeluruh mengenai seberapa efektif model dalam menyelesaikan berbagai tugas deteksi objek, seperti pendeteksian kendaraan, drone, dan pemantauan lalu lintas. Setiap metrik memberikan informasi khusus mengenai aspek tertentu dari kinerja model, mulai dari kemampuannya dalam mengidentifikasi objek dengan benar hingga efisiensinya dalam memproses data visual yang kompleks. Bagian-bagian berikut akan menjelaskan secara rinci penggunaan metrik ini dalam mengevaluasi performa YOLOv8n-cls.

a. Presisi

   Presisi merupakan rasio antara deteksi positif yang benar (*true positive*) terhadap jumlah total deteksi positif, baik yang benar maupun yang salah (*true positive* + *false positive*).

Metrik ini menunjukkan seberapa baik model dalam menghindari deteksi yang keliru atau *false* alarm. Dalam tugas deteksi drone, YOLOv8 mencatat nilai presisi sebesar 0,946, melampaui model lain seperti *Mask* CNN dan YOLOv5. Hasil ini menegaskan kemampuannya dalam meminimalkan kesalahan deteksi positif palsu [68]. Sementara itu, dalam situasi lalu lintas yang kompleks, versi YOLOv8 yang telah disempurnakan berhasil mencapai presisi sebesar 83,8%, menunjukkan peningkatan dibandingkan dengan model YOLO sebelumnya [69]

$$Presisi = \frac{TP}{TP+FP} \quad (2)$$

Keterangan rumus (2) :

1. *TP (True Positive)* : jumlah objek yang benar – benar positif dan berhasil dideteksi sebagai positif
2. *FP (False Positive)* : jumlah objek yang seharusnya negatif tetapi terdeteksi sebagai positif (kesalahan deteksi).

b. Recall

Recall, atau sensitivitas, mengukur seberapa besar proporsi objek yang benar-benar ada (positif aktual) berhasil dikenali oleh model. Metrik ini mencerminkan kemampuan model dalam mendeteksi semua objek yang relevan. Dalam deteksi drone, YOLOv8 mencatat nilai *recall* sebesar 0,9605, yang menunjukkan efektivitasnya dalam menangkap sebagian besar objek yang penting [68]. Sementara itu, di lingkungan lalu lintas yang kompleks, versi YOLOv8 yang telah ditingkatkan mencapai *recall* sebesar 82,7%, memperlihatkan ketangguhannya dalam mengenali berbagai target secara akurat [69]

$$Recall = \frac{TP}{TP+FN} \quad (3)$$

Keterangan rumus (3) :

1. *FN (False Negative)* : jumlah objek yang seharusnya positif tetapi tidak terdeteksi.

c. F1-Score

F1-score adalah rata-rata harmonik dari presisi dan *recall*, yang memberikan keseimbangan antara kedua metrik tersebut. Dalam kondisi cahaya rendah, YOLOv8 mengungguli YOLOv5 dengan *F1-score* yang lebih tinggi sebesar 9%, menunjukkan performanya yang lebih baik dalam lingkungan yang menantang [70]. *F1-score* untuk YOLOv8 dalam skenario mobil *swakemudi* berkisar antara 0,75 hingga 0,87 pada siang hari, tetapi turun menjadi 0,27 hingga 0,46 pada malam hari, yang mencerminkan pengaruh kondisi pencahayaan terhadap kinerja deteksi [6].

$$F1 - Score = 2\ X\ \frac{Presisi\ X\ Recall}{Presisi+Recall} \quad (4)$$

Keterangan rumus (4) :

1. Presisi *(Precision)* : Mengukur berapa banyak dari semua hasil positif yang diprediksi model adalah benar.
2. Recall (Sensitivitas) : Mengukur seberapa banyak dari semua kasus positif yang sebenarnya berhasil ditemukan oleh model

d. Confusion Matrix

Confusion matrix menyajikan rincian yang mendalam mengenai jumlah deteksi benar positif (*true positive*), salah positif (*false positive*), benar negatif (*true negative*), dan salah negatif (*false negative*), sehingga memberikan pemahaman lebih spesifik tentang performa model pada berbagai aspek. Dari hasil analisis *confusion matrix*, diketahui bahwa YOLOv8 menunjukkan kinerja yang sedikit lebih baik dibandingkan YOLOv5 dalam tugas deteksi kendaraan, dengan nilai akurasi dan presisi yang lebih tinggi [71].

Tabel 2. 3 Komponen dasar *Confusion Matrix*

	Predicted Positive	*Predicted Negative*
Actual Positive	*True Positive* (TP)	*False Negative* (FN)
Actual Negative	*False Positive* (FP)	*True Negative* (TN)

Penjelasan setiap komponen :

1. TP *(True Positive)* : Kasus positif yang berhasil dikenali dengan benar.
2. FP *(False Positive)* : Kasus negatif yang salah dikenali sebagai positif (*false alarm*).
3. FN *(False Negative)* : Kasus positif yang gagal dikenali (*missed detection*).
4. TN *(True Negative)* : Kasus negatif yang dikenali dengan benar.

2.9.9 Penjabaran Matematis YOLOv8n-cls

1. Operasi Konvolusi *(Convolution Operation)*

Pada tahap awal, citra masukan diproses melalui *layer* konvolusi untuk mengekstraksi fitur spasial penting. Secara matematis, hasil konvolusi dinyatakan dengan persamaan :

$$Y(i,j) = \sum_{m=0}^{M-1} \sum_{n=0}^{M-1} X(i+m, j+n) \cdot K(m,n) \quad (5)$$

Keterangan rumus (5) :

1. Y(i, j) : Nilai hasil konvolusi pada posisi (i, j) pada citra keluaran (*output image*).
2. X(i + m, j + n) : Nilai piksel pada posisi (i + m, j + n) pada citra masukan (*input image*) X.
3. K(m, n) : Nilai *kernel*/filter pada posisi (m, n). *Kernel* adalah matriks kecil (misal 3x3, 5x5) yang "digeser" ke seluruh citra untuk melakukan operasi konvolusi.
4. M : Ukuran sisi *kernel* (misalnya, jika *kernel* 3x3 maka M = 3)
5. Σ (Sigma) : Simbol penjumlahan, yang berarti menjumlahkan seluruh hasil perkalian antara elemen *kernel* dengan elemen citra yang bersesuaian di setiap posisi.

2. Fungsi Aktivasi SiLU (*Activation Function (SiLU))*

Setelah konvolusi, hasilnya dilewatkan ke fungsi aktivasi untuk memperkenalkan sifat non-linear. YOLOv8n-cls menggunakan aktivasi SiLU (*Sigmoid Linear Unit*), yang dirumuskan sebagai:

$$f(x) = x \cdot \sigma(x) = \frac{x}{1 + e^{-x}} \quad (6)$$

Keterangan rumus (6) :

1. f(x) : Fungsi aktivasi *Swish*, yaitu output dari fungsi aktivasi untuk *input x*.
2. *X* : Nilai input (biasanya hasil dari proses linear di neuron pada jaringan saraf tiruan).
3. σ(x) : Fungsi sigmoid, didefinisikan sebagai:

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

Fungsi *sigmoid* mengubah nilai input menjadi rentang 0 hingga 1.

4. x · σ(x) : Perkalian antara nilai input dengan hasil fungsi *sigmoid* dari *input* tersebut.

Fungsi ini membantu jaringan dalam mengoptimalkan aliran gradien selama pelatihan, meningkatkan stabilitas dan konvergensi model. Digunakan sebagai fungsi aktivasi di YOLOv8

3. C2f Module *(Cross Stage Partial with feature fusion)*

C2f memproses input fitur dengan membagi *channel* menjadi dua jalur yaitu, Jalur *shortcut* langsung dan jalur *bottleneck* untuk transformasi fitur. Langkah matematis yang dijabarkan adalah dengan cara memisahkan $\boldsymbol{X}$ menjadi dua bagian : $\boldsymbol{X_{shortcut}}$ (langsung dilewatkan), $\boldsymbol{X_{shortcut}}$ (diproses *bottleneck*).

Setiap *bottleneck* memproses bagian berikut :

$$F_{bottleneck} = Bottleneck(X_{bottleneck}) \quad (7)$$

Keterangan rumus (7):

1. $F_{bottleneck}$: Merupakan output fitur *(feature output)* dari blok *bottleneck*.

Kemudian digabungkan kembali dengan *shortcut* menggunakan *Concatenation*.

2. $Bottleneck()$: Adalah fungsi atau blok *bottleneck*, yaitu sebuah modul atau *layer* khusus dalam arsitektur jaringan saraf tiruan *(neural network)*, terutama pada model seperti *ResNet* atau YOLO. *Bottleneck* digunakan untuk mengefisienkan proses ekstraksi fitur dengan mengurangi dimensi *(channel) input* sebelum diproses lebih lanjut, lalu mengembalikannya ke dimensi semula.
3. $X_{bottleneck}$: Merupakan input fitur *(feature input)* yang masuk ke blok *bottleneck*

$$F_{out} = Concat(X_{shortcut}, F_{bottleneck}) \quad (8)$$

Keterangan Rumus (8) :

1. F_{out} : Merupakan output fitur akhir dari blok yang sedang dianalisis.
2. $Concat()$: operasi *Concatenation* (penggabungan), yaitu proses menggabungkan dua tensor (fitur) secara bersamaan, biasanya pada dimensi channel. Operasi ini sering digunakan dalam arsitektur *deep learning* modern, seperti YOLO, *ResNet*, dan *DenseNet*, untuk memperkaya representasi fitur
3. $X_{shortcut}$: Merupakan fitur input yang berasal dari jalur *shortcut* atau *skip connection*. *Shortcut* ini memungkinkan informasi dari lapisan awal dilewatkan langsung ke lapisan yang lebih dalam, membantu mengatasi masalah *vanishing gradient* dan memperbaiki aliran informasi.
4. $F_{bottleneck}$: Merupakan fitur hasil dari blok *bottleneck*, yaitu blok konvolusi yang bertugas mengekstrak dan memadatkan informasi penting dari *input*.

Dengan pendekatan ini, C2f menjaga efisiensi komputasi sekaligus memperkaya representasi fitur.

4. Agregasi Konteks*: Spatial Pyramid Pooling – Fast* (SPPF)

Untuk memperkuat representasi spasial multi-skala, YOLOv8n-cls menerapkan SPPF:

$$\mathrm{Y} = \mathrm{Concat}(\mathrm{X}, \mathrm{MP}_1(\mathrm{X}), \mathrm{MP}_2(\mathrm{X}), \mathrm{MP}_3(\mathrm{X})) \quad (9)$$

Keterangan Rumus (9) :

1. Y : Merupakan output hasil penggabungan (*concatenation*) dari beberapa fitur.
2. Concat() : Adalah operasi *concatenation* (penggabungan), biasanya dilakukan pada dimensi *channel* (fitur), sehingga semua hasil operasi di dalamnya digabungkan menjadi satu tensor.
3. X : *Input* asli (fitur awal) yang akan diproses
4. $\mathrm{MP}_1(\mathrm{X}), \mathrm{MP}_2(\mathrm{X}), \mathrm{MP}_3(\mathrm{X})$: Merupakan hasil dari operasi *Max Pooling* (MP) dengan parameter atau ukuran *kernel* yang berbeda pada *input* X.

a. $\mathrm{MP}_1(\mathrm{X})$: *Max Pooling* pertama pada X (misal, dengan ukuran *kernel* tertentu).
b. $\mathrm{MP}_2(\mathrm{X})$: *Max Pooling* kedua pada X (dengan ukuran *kernel* berbeda).
c. $\mathrm{MP}_3(\mathrm{X})$: *Max Pooling* ketiga pada X (dengan ukuran *kernel* berbeda lagi).

Rumus ini sering ditemukan pada arsitektur *deep learning* modern, seperti pada modul *Spatial Pyramid Pooling* (SPP) di YOLO (*You Only Look Once*) dan model deteksi objek lainnya. Tujuannya adalah untuk menangkap informasi dari berbagai skala (*multi-scale*), sehingga model dapat mengenali objek dengan ukuran berbeda secara lebih efektif.

Dengan menggabungkan hasil *pooling* dari berbagai ukuran *kernel*, fitur yang dihasilkan menjadi lebih kaya dan representatif sebelum diproses ke tahap berikutnya di jaringan.

5. *Global Average Pooling (GAP)*

Setelah feature map diperkaya, diterapkan *Global Average Pooling* untuk meratakan dimensi spasial menjadi vektor 1D :

$$g_c = \frac{1}{H \times W} \sum_{h=1}^{H} \sum_{w=1}^{W} F_{c,h,w} \quad (10)$$

Dengan $\boldsymbol{g_c}$ mewakili nilai rata – rata fitur pada *channel* c.

Keterangan rumus (10) :

1. g_c : hasil *pooling* (skalar) untuk *channel* ke-c
2. H : tinggi dari *feature map*

3. W : lebar dari *feature map*
4. $F_{c,h,w}$: nilai *feature map* pada *channel* $\boldsymbol{c}$, baris $\boldsymbol{h}$, dan kolom $\boldsymbol{w}$

6. Fully Connected (Linear Classifier)

Hasil GAP kemudian diproses melalui operasi linear dan selanjutnya skor prediksi z diubah menjadi probabilitas dengan fungsi *Softmax* sehingga model menghasilkan distribusi probabilitas atas semua kelas target.

$$z = W^T x + b \tag{11}$$

Keterangan rumus (11) :

1. z : *output* linear sebelum fungsi aktivasi
2. W^T : transpos dari matriks bobot
3. x : vektor *input*
4. b : vektor bias

$$\hat{y} = Softmax(z) \tag{12}$$

Keterangan rumus (12) :

1. $\hat{y}$: vektor probabilitas prediksi, di mana setiap elemen mewakili kemungkinan input termasuk dalam suatu kelas.
2. $Softmax(z)$: fungsi aktivasi yang mengubah vektor skor menjadi probabilitas
3. z : vektor skor (logit) dari output *Fully Connected Layer*

7. Loss Function (Cross-Entropy)

Untuk klasifikasi biasanya menggunakan *Binary Cross Entropy* (jika *multi-label*) atau *Categorical Cross Entropy :*

$$L_{cls} = -\sum_{i=1}^{C} y_i log(\hat{y}_i) \tag{13}$$

Keterangan rumus (13) :

1. L_{cls} : nilai loss klasifikasi (semakin kecil, semakin baik prediksi model).
2. C : jumlah kelas
3. y_i : label sebenarnya (1 jika kelas benar, 0 jika bukan).
4. $\hat{y}_i$: probabilitas prediksi untuk kelas i.
5. $\sum_{i=1}^{C}$: artinya adalah penjumlahan dari indeks i = 1 hingga i = c.

6. i : merepresentasikan indeks kelas ke-1 sampai ke-c (total kelas ada c)
7. $log()$: adalah logaritma natural

2.9.10 Perhitungan Parameter dan *FLOPs* per *Layer*

Parameter merupakan bobot dan bias yang diperlajari model pada setiap layer jumlah parameter menentukan ukuran model (*memory footptint*) dn kapasistasnya untuk menyimpan pola dari data. Semakin banyak parameter, biasa model bisa menangkap pola lebih kompleks, tapi juga lebih besar dan rentan *overfitting*.

FLOPs (*Floating-Point Operations*) per *layer* menggambarkan beban komputasi berapa banyak operasi *floating point* yang perlu dilakukan saat memproses satu *input* lewat layer tersebut. Semakin tinggi *FLOPs*, berarti *layer* itu lebih berat secara komputasi memerlukan waktu dan energi lebih banyak. Dan berguna juga untuk memperkirakan kecepatan inferensi dan efisiensi *hardware* (misal *GPU/CPU*) yang dibutuhkan. Seperti contoh perhitungan dibawah ini :

1. *Input Tensor* - > [3, 224, 224] (RGB, 224x224 resolusi).
   a. *Backbone*

Tabel 2. 4 Perhitungan Backbone

Layer	*Type*	*Kernel Size*	*Input Shape*	*Output Shape*	*Parameters*	*FLOPs*
1	*Conv2d*	3x3	[3, 224, 224]	[16, 112, 112]	(3×3×3+1)×16=448	448×112×112=5.62M
2	*BatchNorm*	-	[16, 112, 112]	[16, 112, 112]	16×2=32	16×112×112=0.2M
3	*SiLU*	-	[16, 112, 112]	[16, 112, 112]	0	0
4	*Conv2d (Depthwise)*		[16, 112, 112]	[16, 112, 112]	(3×3×16+16)=160	160×112×112=2.01M
5	*BatchNorm*	-	[16, 112, 112]	[16, 112, 112]	32	0.2M
6	*SiLU*	-	[16, 112, 112]	[16, 112, 112]	0	0

7	*Conv2d (Pointwise)*	1x1	[16, 112, 112]	[32, 112, 112]	(1×1×16+1) ×32=544	544×112×112=6.83M
…..	……	…….	……..	…….	……..	……
N	*Bottleneck Blocks*	-	-	-	~100K	~50M

Backbone terdiri dari beberapa blok *bottleneck* yang dioptimalkan untuk komputasi efisien (*depthwise separable conv, skip connections*).

b. *Head* (Klasifikasi)

Tabel 2. 5 Perhitungan Head

Layer	*Type*	*Input Shape*	*Output Shape*	*Parameters*	*FLOPs*
1	*GAP*	[512, 7, 7]	[512, 1, 1]	0	512×7×7=25K
2	*FC*	512	1000 (Kelas)	512×1000+1000=513K	513K

Total *Parameters* dan *FLOPs*

a. Total *Parameters*:

Backbone (~1.5M) + *Head* (~513K) = ~2.0 juta.

b. Total *FLOPs*:

Backbone (~100M) + *Head* (~0.5M) = ~100.5 juta Flops

Contoh Perhitungan Detail

1. *Convolutional Layer* (Contoh: *Layer* 1):
   a. *Input*: 3 *Channels*, *Output*: 16 *Channels*.
   b. *Kernel*: 3x3, *stride*=2, *padding*=1.
   c. *Parameters*: (3x3x3)x16+16 = 448
   d. *FLOPs*: 448×112×112=5.62×10^6
2. *Depthwise Separable* Conv (Contoh: Layer 4 + 7):
   a. *Depthwise*:
      i. *Parameters*: 3×3×16 =144 *weights* + 16 *biases* = 160.

ii. *FLOPs*: 160×112×112=2.01M.

b. *Pointwise*:

i. *Parameters*: 1×1×16×32=512 + 32 *biases* = 544.

ii. *FLOPs*: 544 x 112 x 112 = 6.83M

2.9.11 Format Anotasi Data

YOLOv8 menggunakan format anotasi yang merupakan pengembangan dari format YOLOv5 *PyTorch* TXT [72]. Anotasi disimpan dalam sebuah *file* teks di mana setiap baris mewakili sebuah objek dalam gambar. Setiap baris mencakup label kelas diikuti oleh koordinat normalisasi dari *bounding box* (center_x, center_y, lebar, tinggi) yang dihitung relatif terhadap dimensi gambar. Formatnya adalah sebagai berikut:

<class> <center_x> <center_y> <width> <height> [72]

Sebagai contoh, anotasi dapat terlihat seperti ini:

001.txt (Nama File .txt) [72]

1 0.617 0.3594420600858369 0.114 0.17381974248927037

1 0.094 0.38626609442060084 0.156 0.23605150214592274

1 0.295 0.3959227467811159 0.13 0.19527896995708155

1 0.785 0.398068669527897 0.07 0.14377682403433475

1 0.886 0.40879828326180256 0.124 0.18240343347639484

1 0.723 0.398068669527897 0.102 0.1609442060085837

1 0.541 0.35085836909871243 0.094 0.16952789699570817

1 0.428 0.4334763948497854 0.068 0.1072961373390558

1 0.375 0.40236051502145925 0.054 0.1351931330472103

1 0.976 0.3927038626609442 0.044 0.17167381974248927

File `data.yaml` berisi nilai konfigurasi yang digunakan oleh model untuk menemukan gambar dan memetakan nama kelas ke class_id [72].

data.yaml

train: ../train/images

val: ../valid/images

nc: 3

names: ['head', 'helmet', 'person']

Gambar 2. 10 Contoh Visual Format Anotasi YOLOv8*[72]*

Format ini disertai dengan file konfigurasi YAML, yang menjelaskan arsitektur model dan label kelas. File ini memastikan bahwa YOLOv8 dapat dengan mudah disesuaikan dengan berbagai dataset dan tugas. Untuk kompatibilitas, anotasi dari alat seperti *Roboflow*, VOTT, *LabelImg*, dan CVAT mungkin perlu dikonversi agar sesuai dengan format YOLOv8. Alat-alat ini sering menyediakan opsi ekspor langsung atau utilitas konversi untuk mempermudah proses ini.

2.10 Dataset

Kaggle Datasets adalah kumpulan data terbuka yang diunggah dan dikelola oleh komunitas data *scientist* di *platform Kaggle*. Setiap dataset biasanya dilengkapi dengan deskripsi singkat, kolom–kolom *(feature)* yang tersedia, ukuran file, serta format (CSV, JSON, gambar, dan lain-lain). Dengan antarmuka intuitif, pengguna dapat menelusuri dataset berdasarkan popularitas, topik, atau ukuran, sehingga mempermudah dalam menemukan data yang relevan untuk analisis atau eksperimen *machine learning*.

Dengan menggabungkan *Kaggle* Datasets, pengguna dapat memperoleh dataset yang lengkap baik dalam bentuk data terstruktur maupun data gambar, sehingga mendukung pengembangan model *machine learning* yang lebih akurat dan efektif dalam berbagai aplikasi, termasuk konservasi laut dan klasifikasi sampah di dalam laut.

2.11 *Tools*

Tools pada penelitian ini menggunakan JavaScript sebagai bahasa pemrograman untuk pengembangan antarmuka *(front-end)* website. Python untuk pengembangan bagian belakang *(back-end)* dan perancangan algoritma YOLOv8n-cls dalam klasifikasi objek. Penggunaan *Visual Studio Code* sebagai code editor utama dalam proses pengembangan untuk menghubungkan tampilan website dengan algoritma klasifikasi. Kemudian *Google Colaboratory (Google Colab)* sebagai platform untuk melakukan pengujian dan pelatihan model YOLOv8n-cls secara efisien dengan memanfaatkan *GPU* yang tersedia.