BAB I
PENDAHULUAN

1.1 Latar Belakang

Perkembangan teknologi kendaraan modern dan sistem transportasi cerdas telah menempatkan deteksi dan klasifikasi rambu lalulintas sebagai salah satu komponen dalam upaya meningkatkan keselamatan berkendara. Rambu lalu lintas berfungsi sebagai media komunikasi visual yang memberikan informasi struktural mengenai batas kecepatan, peringatan bahaya, kondisi jalan, berbagai larangan dan instruksi bernavigasi yang penting untuk menjaga keselamatan pengemudi dan pengguna jalan lainnya. Ketergantungan manusia terhadap persepsi visual menjadikan rambu sebagai salah satu sumber informasi di jalan raya dan ketika persepsi tersebut gagal, risiko kecelakaan meningkat secara signifikan. Wali et al. menegaskan bahwa rambu lalu lintas adalah komponen kunci pengambilan keputusan bagi pengemudi dan ketidakmampuan dalam mengenali rambu dapat menyebabkan kesalahan dalam mengontrol kendaraan (Wali et al., 2019).

Dalam konteks kendaraan berbasis ADAS (*Advanced Driver Assistance Systems*), kebutuhan untuk mengidentifikasi rambu secara otomatis menjadi semakin vital. Dhawan et al. menjelaskan bahwa kendaraan masa kini beberapa telah kamera yang dapat mendeteksi rambu dan menginformasikan perubahan kondisi lalu lintas seperti batas kecepatan maupun informasi lain. Mereka menekankan bahwa kesalahan kecil dalam membaca rambu dapat menimbulkan konsekuensi serius, sehingga sistem deteksi otomatis harus mampu bekerja dengan cepat dan akurat di lingkungan nyata yang kompleks (Dhawan et al., 2023).

Deteksi rambu rambu lalulintas merupakan tugas yang sangat menantang. Berbagai penelitian menunjukkan bahwa objek rambu memiliki karakteristik unik: ukurannya kecil, bentuknya beragam, warnanya bisa pudar, posisinya miring, dan sering dipengaruhi kondisi lingkungan yang tidak menentu seperti kabut, hujan, bayangan, dan pencahayaan rendah (Rani et al., 2024). Ningsi et al. menunjukkan bahwa blur akibat kecepatan kendaraan atau getaran kamera merupakan penyebab utama kegagalan deteksi rambu dan harus ditangani dengan metode khusus untuk mempertahankan ketajaman fitur visual rambu dalam berbagai kondisi cuaca dan pencahayaan (Ningsi et al., 2024). Pada kondisi malam hari, masalah bertambah rumit karena noise sensor kamera meningkat, kontras menurun, dan motion blur lebih mudah terjadi.

Model *deep learning* terbaru seperti YOLOv5, YOLOv7, YOLOv8, YOLOv9 dan YOLOv10 memang telah menunjukkan peningkatan signifikan, namun berbagai varian YOLO masih memiliki kelemahan besar dalam mendeteksi objek kecil, terutama rambu yang direkam dari jarak jauh. Pada penelitian yang dilakukan oleh Zhang et al. dan penelitian Henlin et al. menunjukkan bahwa beberapa versi YOLO standar masih gagal mendeteksi banyak rambu kecil, sehingga perlu penambahan *branch* resolusi tinggi dan penguatan fitur skala kecil untuk meningkatkan sensitivitas deteksi objek kecil di jalan raya (H. Zhang et al., 2025) (Henlin et al., 2024).

Selain ukuran objek, distorsi perspektif dan deformasi visual menjadi masalah besar dalam sistem deteksi berbasis konvolusi standar. Penelitian yang dilakukan Burgos

Madrigal et al. menunjukkan bahwa kernel konvolusi biasa memiliki sampling grid statis yang tidak mampu mengikuti deformasi objek, terutama ketika objek mengalami rotasi, kemiringan, atau blur. *Deformable Convolutional Network* (DCN) mengatasi hal ini dengan memungkinkan *sampling point* beradaptasi terhadap bentuk objek, sehingga fitur dari rambu yang terdistorsi dapat ditangkap lebih efektif. Penelitian tersebut menegaskan bahwa DCN mampu memberikan peningkatan akurasi signifikan pada objek dengan bentuk non-rigid atau yang muncul dalam pose kompleks, menjadikannya kandidat ideal untuk meningkatkan performa sistem deteksi rambu dalam kondisi dunia nyata yang penuh variasi geometris (Burgos Madrigal et al., 2024).

Keterbatasan deteksi rambu lalulintas semakin menegaskan perlunya arsitektur yang lebih adaptif dalam memahami fitur visual yang tidak stabil. Terkhusus pada malam hari, rambu lalulintas terlihat dalam bentuk yang jauh dari ideal, kecil karena jarak pandang yang sempit, mengalami *noise* karena sensitivitas sensor kamera, serta sering diperburuk oleh motion blur akibat pergerakan kendaraan itu sendiri dan rendahnya pencahayaan yang memaksa kamera menggunakan shutter lebih lambat. Tekstur, kontur, dan tepi rambu yang semula menjadi dasar pengenalan oleh model konvolusional standar menjadi kabur dan terdistorsi. Konvolusi biasa bekerja dengan sampling grid statis, sehingga ketika rambu mengalami pergeseran bentuk akibat blur, rotasi sudut pandang kendaraan, pantulan cahaya, atau hilangnya detail akibat noise malam hari, kernel tidak dapat menyesuaikan diri untuk menangkap pola visual yang berubah tersebut. Di sinilah peran DCN menjadi sangat relevan, mekanisme *offset* learning DCN memungkinkan titik sampling bergeser mengikuti kontur nyata objek, meskipun objek tersebut berada dalam kondisi non-ideal. Fleksibilitas sampling ini membuat DCN mampu menangkap fitur rambu yang terdistorsi secara spasial, mempertahankan representasi bentuk meskipun citra mengalami *noise* atau blur, dan mengekstraksi informasi penting dari rambu yang tampak sangat kecil dalam *frame*.

Berdasarkan kompleksitas permasalahan yang telah diuraikan, penelitian ini diarahkan untuk merancang dan mengimplementasikan model deteksi rambu lalu lintas pada malam hari yang lebih andal melalui optimasi pada algoritma YOLOv11 melalui integrasi DCN. Tidak seperti varian YOLO standar yang masih menggunakan *grid sampling* statis, DCN memberikan fleksibilitas bagi model untuk menyesuaikan titik ekstraksi fitur mengikuti bentuk rambu meskipun tampilan rambu mengalami perubahan akibat kondisi malam hari. Dengan kemampuan tersebut, model diharapkan dapat mengenali rambu yang tampak kecil, kabur, atau mengalami penurunan kualitas visual akibat *noise* dan pencahayaan rendah. Oleh karena itu, fokus penelitian ini adalah meningkatkan performa model terhadap berbagai gangguan visual, khususnya noise, blur, dan variasi Cahaya yang selama ini menjadi penyebab utama kegagalan deteksi rambu pada penelitian-penelitian sebelumnya.

1.2 Landasan Teori

1.2.1 Sistem Deteksi dan Klasifikasi Rambu Lalu Lintas

Sistem deteksi dan klasifikasi rambu lalu lintas merupakan komponen krusial dalam pengembangan teknologi *Advanced Driver Assistance Systems* (ADAS). Sistem ini

bertujuan mendeteksi keberadaan rambu lalu lintas pada citra atau video, kemudian mengklasifikasikan jenis rambu tersebut agar dapat memberikan informasi kepada pengemudi secara otomatis. Dengan meningkatnya kompleksitas lingkungan jalan dan kebutuhan keselamatan berkendara, Sistem deteksi dan klasifikasi rambu lalu lintas menjadi elemen fundamental dalam menjaga kepatuhan terhadap regulasi lalu lintas serta meminimalkan risiko kecelakaan.

Pada sistem ADAS modern, sistem deteksi dan klasifikasi rambu lalu lintas bekerja dengan memanfaatkan kamera yang dipasang pada kendaraan untuk menangkap citra rambu. Kamera tersebut berperan sebagai sensor visual utama yang menyediakan informasi lingkungan secara langsung kepada sistem komputasi kendaraan. Sebagaimana dijelaskan dalam penelitian mengenai identifikasi rambu berbasis *deep learning*, kamera berfungsi menangkap kondisi jalan, sinyal visual, dan rambu yang kemudian diproses oleh modul pengenalan objek untuk menginformasikan pengemudi mengenai kondisi atau perintah di jalan yang sedang dilalui (Drobiniak et al., 2025). Proses ini memungkinkan ADAS memberi peringatan dini kepada pengemudi, membantu pengambilan keputusan cepat, serta mengurangi potensi pelanggaran akibat kelalaian.

Pada implementasinya, sistem deteksi dan klasifikasi rambu lalu lintas terdiri dari dua proses utama, yaitu deteksi rambu dan klasifikasi rambu. Tahap deteksi bertugas menemukan lokasi rambu pada citra menggunakan *bounding box*, sedangkan tahap klasifikasi menentukan jenis rambu berdasarkan fitur visual seperti bentuk, warna, dan piktogram. Pendekatan tradisional berbasis segmentasi warna dan deteksi bentuk pernah menjadi metode utama, namun teknik ini kurang stabil ketika menghadapi kondisi nyata seperti pencahayaan rendah, *occlusion*, dan kompleksitas latar belakang (Drobiniak et al., 2025). Oleh karena itu, penelitian terbaru telah banyak beralih ke metode *deep learning*, khususnya CNN dan model deteksi modern seperti YOLO, yang terbukti lebih robust dalam menangani variasi visual dan noise pada lingkungan jalan yang dinamis (Hamza & Nawal, 2024).

Peran sistem deteksi dan klasifikasi rambu lalu lintas menjadi semakin penting dalam sistem ADAS karena informasi rambu dibutuhkan secara *real time* ketika kendaraan melaju. Algoritma yang digunakan harus mampu mempertahankan keseimbangan antara akurasi dan kecepatan pemrosesan agar sistem dapat memberikan peringatan atau tindakan korektif secara tepat waktu. Berbagai penelitian menunjukkan bahwa performa sistem deteksi dan klasifikasi rambu lalu lintas sangat dipengaruhi oleh kualitas citra, perubahan cuaca, serta kondisi pencahayaan, sehingga model yang digunakan harus adaptif terhadap berbagai skenario kompleks di jalan raya (Ghaffar et al., 2023). Dalam konteks Indonesia, sistem TSR juga harus mampu mengenali variasi rambu sesuai standar nasional yang diatur dalam Permenhub No. 13 Tahun 2014, yang membagi rambu menjadi empat kategori utama: peringatan, larangan, perintah, dan petunjuk (Kementerian Perhubungan RI, 2014).

1.2.2 Jenis-Jenis Rambu Lalu Lintas di Indonesia

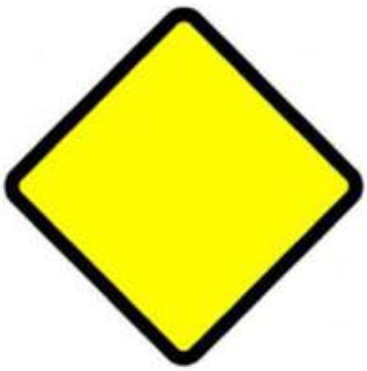
Menurut Peraturan Menteri Perhubungan Republik Indonesia Nomor PM 13 Tahun 2014, rambu lalu lintas di Indonesia diklasifikasikan menjadi empat kategori utama, yaitu rambu

peringatan, rambu larangan, rambu perintah, dan rambu petunjuk. Setiap kategori memiliki karakteristik visual yang berbeda, mencakup bentuk geometris, kombinasi warna, serta penggunaan simbol atau piktogram tertentu. Pemahaman terhadap ciri visual ini penting dalam pengembangan sistem deteksi rambu berbasis visi komputer karena fitur-fitur tersebut menjadi dasar dalam proses ekstraksi dan klasifikasi objek oleh model deteksi.

a. Rambu Peringatan

Rambu peringatan digunakan untuk memberikan informasi tentang kemungkinan adanya bahaya atau kondisi jalan yang membutuhkan kewaspadaan pengemudi. Rambu ini umumnya dipasang sebelum lokasi bahaya sehingga pengemudi dapat melakukan persiapan, seperti mengurangi kecepatan atau meningkatkan perhatian. Ciri visual rambu peringatan sebagaimana ditunjukan pada Gambar 1 adalah sebagai berikut:

Bentuk : Berbentuk belah ketupat.
Warna dasar : Kuning dengan garis tepi hitam.
Isi : Simbol peringatan berwarna hitam yang menggambarkan potensi bahaya (misal: tikungan, persimpangan, turunan, jalan licin).

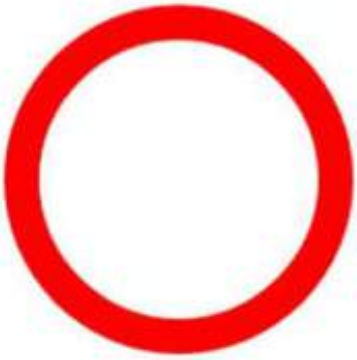


Gambar 1. Ciri Visual Rambu Peringatan

b. Rambu Larangan

Rambu larangan berfungsi memberikan perintah kepada pengguna jalan untuk tidak melakukan suatu tindakan tertentu, seperti larangan masuk, larangan mendahului, larangan berhenti, atau pembatasan kecepatan. Rambu ini bersifat mengikat dan pelanggarannya dapat berakibat sanksi hukum. Ciri visual rambu larangan sebagaimana ditunjukan pada Gambar 2 adalah sebagai berikut:

Bentuk : Lingkaran.
Warna dasar : Putih dengan garis tepi merah.
Isi : Simbol larangan berwarna hitam; beberapa rambu memiliki garis miring merah di tengah sebagai penanda "terlarang".

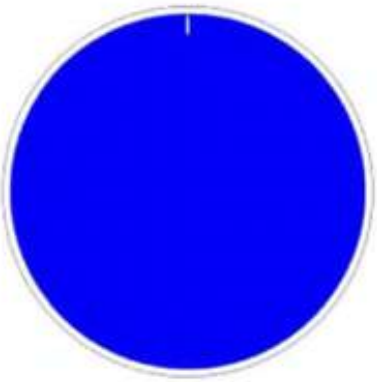


Gambar 2. Ciri Visual Rambu Larangan

c. Rambu Perintah

Rambu perintah digunakan untuk memberikan instruksi yang wajib dipatuhi oleh pengguna jalan, seperti perintah mengikuti arah tertentu, kewajiban menggunakan jalur tertentu, atau perintah pembatasan kecepatan minimum. Ciri visual rambu perintah sebagaimana ditunjukan pada Gambar 3 adalah sebagai berikut:

Bentuk : Lingkaran.
Warna dasar : Biru dengan piktogram putih.
Isi : Simbol yang menunjukkan tindakan yang wajib dilakukan (misal: wajib ke kiri, wajib lurus, wajib mengikuti jalur sepeda).



Gambar 3. Ciri Visual Rambu Perintah

d. Rambu Petunjuk

Rambu petunjuk berfungsi memberikan informasi atau penegasan kepada pengguna jalan mengenai arah, fasilitas umum, batas wilayah, kondisi lalu lintas, atau jalur alternatif. Rambu kategori ini lebih bersifat informatif daripada instruktif. Ciri visual rambu petunjuk sebagaimana ditunjukan pada Gambar 4 adalah sebagai berikut:

Bentuk : Persegi atau persegi panjang.
Warna dasar : Biru dengan piktogram putih.

Isi : Teks atau ikon berwarna putih yang memberikan arah, lokasi, fasilitas, atau informasi daerah.

Gambar 4. Ciri Visual Rambu Petunjuk

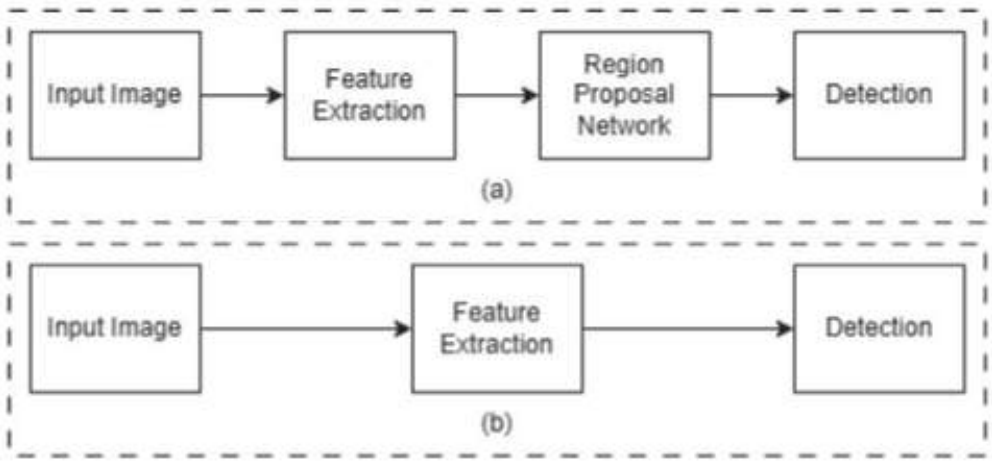
1.2.3 Deep Learning dan Deteksi Objek

Deep learning merupakan cabang dari kecerdasan buatan yang memanfaatkan jaringan saraf tiruan dengan banyak lapisan (*deep neural networks*) untuk mempelajari pola atau representasi data secara otomatis. Berbeda dengan metode pembelajaran tradisional yang membutuhkan rekayasa fitur secara manual, *deep learning* mampu mengekstraksi fitur penting langsung dari data melalui proses pembelajaran berulang. Secara umum, *deep learning* bekerja dengan menirukan cara kerja otak manusia dalam mengenali pola, di mana setiap lapisan dalam jaringan saraf mempelajari tingkat kompleksitas yang semakin tinggi, mulai dari pola sederhana hingga konsep visual yang lebih abstrak. Pendekatan berlapis ini menjadikan *deep learning* sangat efektif digunakan pada berbagai tugas visual seperti pengenalan objek, analisis citra, dan pemrosesan video karena kemampuannya mempelajari representasi fitur secara hierarkis dan mendalam (Taye, 2023).

Dalam bidang visi komputer, deteksi objek adalah proses yang bertujuan mengidentifikasi keberadaan suatu objek sekaligus menentukan lokasinya dalam citra menggunakan *bounding box*. Tidak seperti klasifikasi citra yang hanya memberikan satu label untuk seluruh gambar, deteksi objek memerlukan dua keluaran utama: apa objek tersebut dan di mana letaknya. Pendekatan berbasis *deep learning* telah menjadi standar dalam deteksi objek karena mampu menangani variasi skala, perubahan orientasi, occlusion, dan kondisi lingkungan yang kompleks, sebagaimana dijelaskan dalam tinjauan literatur yang menekankan bahwa model modern berbasis *deep neural networks* memiliki keunggulan dalam stabilitas dan akurasi pada berbagai situasi visual (Aziz et al., 2020).

Secara umum, metode deteksi objek berbasis *deep learning* dapat dibagi menjadi dua kategori utama, yaitu *two-stage detectors* dan *one-stage detectors*. *two-stage detectors* bekerja dengan menghasilkan kandidat wilayah objek terlebih dahulu melalui region proposal, kemudian melakukan klasifikasi dan regresi *bounding box* pada tahap

berikutnya. Contoh utama pendekatan ini adalah R-CNN dan turunannya. Sementara itu, *one-stage detectors* seperti SSD dan YOLO melakukan deteksi dan klasifikasi secara langsung dalam satu proses, sehingga memiliki kecepatan inferensi yang lebih tinggi. Perbedaan mendasar antara keduanya terletak pada alur kerja*: two-stage* menekankan ketelitian melalui proses dua langkah, sedangkan *one-stage* menekankan efisiensi dengan menggabungkan seluruh proses menjadi satu tahap. Pembagian ini menjadi dasar dalam memahami perkembangan berbagai metode deteksi objek modern di literatur (Aziz et al., 2020; He & Law, 2024). Untuk memberikan gambaran visual, Gambar 5 berikut menunjukkan perbedaan paradigma arsitektur *two-stage* dan *one-stage* pada tugas deteksi objek.



Gambar 5. Struktur Model (a) Two-Stage Object Detection dan (b) One-Stage Object Detection

1.2.4 You Only Look Once (YOLO)

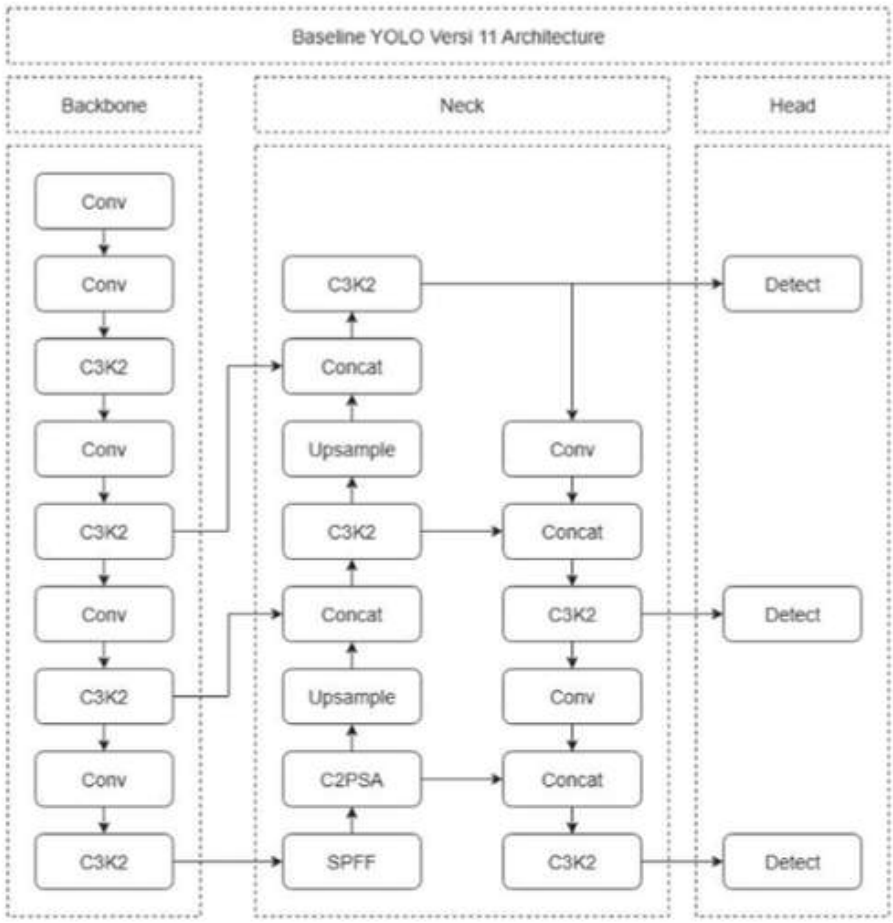
YOLO (You Only Look Once) merupakan salah satu algoritma deteksi objek satu tahap (*one-stage detector*) yang berpengaruh dalam bidang visi komputer karena menggabungkan proses prediksi lokasi dan klasifikasi objek dalam satu kali inferensi. Berbeda dengan pendekatan dua tahap seperti R-CNN atau Faster R-CNN yang memisahkan proses *region proposal* dan klasifikasi, YOLO memperlakukan deteksi objek sebagai masalah regresi langsung dari citra ke *bounding box* dan kelas objek, sehingga mencapai kecepatan yang sangat tinggi dan cocok untuk aplikasi *real time* seperti sistem bantuan pengemudi (ADAS) (Flores-Calero et al., 2024). Menurut beberapa tinjauan sistematis, keunggulan YOLO terletak pada efisiensi arsitekturnya dan kemampuannya untuk mempertahankan kinerja yang stabil pada lingkungan visual yang kompleks, seperti kondisi jalan raya, variasi iluminasi, dan skala objek yang beragam (Hussain, 2024).

Sejak diperkenalkan pada tahun 2016, YOLO telah mengalami perkembangan yang signifikan melalui berbagai versi mulai dari YOLOv1 hingga YOLOv11. YOLOv1 memperkenalkan konsep deteksi sebagai regresi tunggal dengan membagi citra menjadi grid, namun memiliki keterbatasan dalam mendeteksi objek kecil. YOLOv2 memperbaiki

beberapa kelemahan tersebut dengan menambahkan *batch normalization*, *anchor boxes*, dan *multi-scale training*, yang meningkatkan stabilitas model. YOLOv3 memanfaatkan *backbone* Darknet-53 dan *feature pyramid network* (FPN) sehingga lebih mampu menangani objek multiskala. Berikutnya, YOLOv4 dan YOLOv5 memperkenalkan berbagai inovasi seperti CSPDarknet, PANet, dan beragam *training tricks* seperti mosaic augmentation yang semakin meningkatkan akurasi dan efisiensi. Tinjauan mendalam mengenai evolusi YOLO dari YOLOv1 hingga YOLOv8 menunjukkan bahwa setiap versi membawa perbaikan signifikan pada struktur *backbone*, metode penggabungan fitur, serta mekanisme prediksi deteksi sehingga menjadikan keluarga YOLO sebagai standar industri dalam deteksi objek *real time* (Aziz et al., 2020). Versi terbaru seperti YOLOv9 dan YOLOv10 menambahkan konsep *programmable gradient information* serta optimisasi struktural untuk efisiensi yang lebih tinggi, sementara YOLOv11 menghadirkan penyempurnaan pada representasi fitur dan prediksi anchor-free yang lebih stabil, menjadikannya salah satu versi paling kuat dalam keluarga YOLO.

Secara umum, arsitektur YOLO modern, termasuk YOLOv11, terdiri dari tiga komponen utama: *backbone*, *neck*, dan detection head. *Backbone* berfungsi mengekstraksi fitur dari citra masukan melalui serangkaian operasi konvolusi, dan dalam konteks YOLOv11 digunakan blok C3k2 yang meningkatkan efisiensi ekstraksi fitur khususnya untuk objek kecil seperti rambu lalu lintas. Komponen *neck* bertugas menggabungkan fitur dari beberapa level resolusi menggunakan modul seperti SPPF (*Spatial Pyramid Pooling - Fast*) dan C2PSA (*Partial Self-Attention*). Struktur ini memungkinkan model menangkap representasi multiskala sehingga mampu mendeteksi objek dengan ukuran dan jarak berbeda. Sementara itu, *detection head* YOLOv11 memanfaatkan pendekatan *anchor-free*, yang lebih fleksibel dan efisien dibandingkan head anchor-based sebelumnya, dengan secara langsung memprediksi koordinat *bounding box*, skor keyakinan, dan kelas objek. Pendekatan *anchor-free* ini juga terbukti lebih stabil pada kondisi *low-light*, blur, maupun ketika objek memiliki bentuk kompleks atau tidak rigid, sebagaimana ditunjukkan dalam studi deteksi objek dan rambu lalu lintas berbasis YOLO di lingkungan nyata (Flores-Calero et al., 2024; Hussain, 2024).

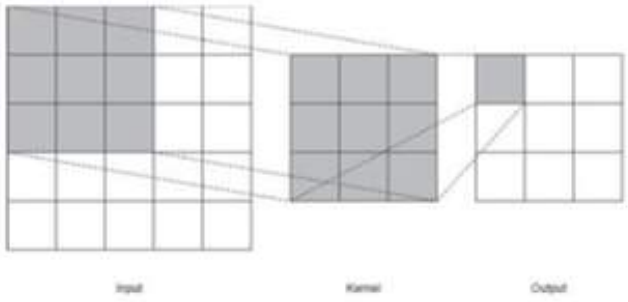
Arsitektur YOLOv11 yang digunakan dalam penelitian ini divisualisasikan pada Gambar 6. Gambar tersebut menunjukkan alur pemrosesan mulai dari citra masukan, proses ekstraksi fitur pada *backbone* C3k2, penggabungan fitur pada *neck* menggunakan SPPF dan C2PSA, hingga proses prediksi objek pada detection head anchor-free. Visualisasi ini memberikan gambaran menyeluruh mengenai bagaimana YOLOv11 bekerja dalam mendeteksi dan mengklasifikasikan rambu lalu lintas pada skenario nyata.



Gambar 6. Arsitektur YOLOv11

1.2.5 Convolution

Convolution merupakan operasi inti dalam *Convolutional Neural Network* (CNN) yang berfungsi mengekstraksi fitur visual dari citra melalui penerapan sebuah kernel atau filter berukuran kecil yang digeser pada seluruh area citra. Setiap kernel, misalnya berukuran 3×3, melakukan perkalian elemen-demi-elemen (*dot product*) dengan patch citra yang berada di bawahnya, kemudian hasil perkalian tersebut dijumlahkan untuk menghasilkan satu nilai pada *feature map*. Proses pergeseran kernel atau *sliding* dilakukan berdasarkan nilai *stride*, sementara *padding* dapat digunakan untuk mempertahankan ukuran keluaran (Taye, 2023).



Gambar 7. Ilustrasi Proses Convolusi

Ilustrasi proses convolution ditunjukkan pada Gambar 7 terlihat bagaimana kernel 3×3 ditempatkan pada sebuah *receptive field* citra, kemudian digeser ke seluruh area input untuk menghasilkan *feature map* yang merepresentasikan pola tertentu pada citra (Burgos Madrigal et al., 2024). Proses ini memungkinkan jaringan mengenali fitur dasar seperti tepi, sudut, dan tekstur pada lapisan awal, kemudian membangun representasi visual yang lebih kompleks pada lapisan-lapisan berikutnya sebagaimana dijelaskan dalam studi arsitektur CNN dan perkembangannya (Li et al., 2022). Secara matematis, operasi konvolusi dua dimensi dapat dituliskan sebagai:

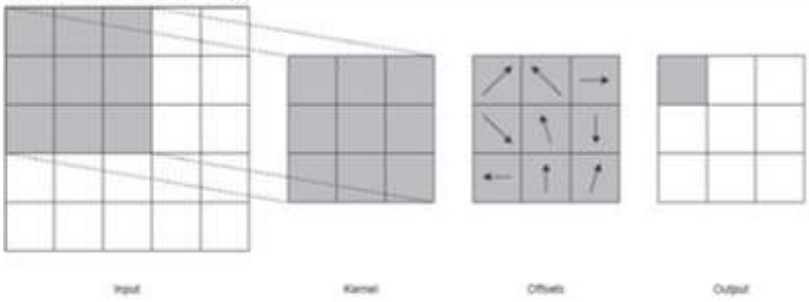
$$x_{i,j} = \sum_{m} \sum_{n} I_{(i+m),(j+n)} \cdot W_{m,n} \quad (1)$$

di mana x adalah citra masukan, w merupakan kernel, dan $I(i,j)$ adalah nilai pada *feature map* di lokasi (i,j). Indeks (m,n) berfungsi sebagai *offset* untuk menentukan posisi elemen kernel relatif terhadap pusat *receptive field*. Pendekatan matematis ini menggambarkan bahwa setiap nilai keluaran merupakan hasil kombinasi linear dari piksel-piksel di area lokal citra, sebagaimana dijelaskan dalam analisis dasar CNN dan implementasi konvolusi pada berbagai arsitektur modern (Burgos Madrigal et al., 2024). Operasi konvolusi inilah yang memungkinkan CNN membangun representasi fitur secara bertingkat, mulai dari pola sederhana pada lapisan awal hingga pola visual kompleks pada lapisan lebih dalam. Dengan demikian, convolution menjadi blok bangunan utama pada berbagai model *deep learning* berbasis CNN yang digunakan dalam klasifikasi citra, segmentasi, maupun deteksi objek (Li et al., 2022).

1.2.6 Deformable Convolutional Network (DCN)

Deformable Convolutional Network (DCN) merupakan pengembangan dari operasi convolution standar yang dirancang untuk meningkatkan kemampuan model dalam menangani variasi geometris suatu objek, seperti perubahan bentuk, rotasi, skala, dan distorsi perspektif. Konvolusi biasa melakukan sampling pada grid yang kaku dan teratur, sehingga kurang fleksibel ketika objek mengalami deformasi kompleks, terutama pada tugas visi komputer seperti deteksi objek di lingkungan nyata. Untuk mengatasi keterbatasan ini, DCN memperkenalkan *learnable offsets* yang memodifikasi posisi *sampling* kernel sehingga filter dapat menyesuaikan diri dengan bentuk dan struktur objek pada citra. Dengan kata lain, posisi tiap elemen kernel tidak lagi tetap, tetapi

bergeser secara adaptif berdasarkan fitur lokal yang dipelajari selama proses pelatihan, sehingga menghasilkan representasi fitur yang lebih kaya dan lebih sensitif terhadap variasi spasial (Li et al., 2022).



Gambar 8. Ilustrasi Proses DCN

Ilustrasi proses DCN dapat dilihat pada Gambar 8, yang menunjukkan perbedaan antara grid konvolusi standar (rigid sampling grid) dan grid DCN yang telah bergeser secara adaptif melalui *offset*. Gambar tersebut memperlihatkan bagaimana setiap titik sampling kernel berpindah dari posisi aslinya untuk menyesuaikan pola bentuk objek, sehingga memberikan fleksibilitas spasial yang tidak dimiliki konvolusi biasa. Secara umum, DCN bekerja dengan menambahkan *offset* Δpke posisi sampling konvolusi standar, sehingga formula dasar konvolusi berubah menjadi:

$$y(P_0) = \sum x(P_0 + P_n + \Delta P_n) \cdot w(P_n) \qquad (2)$$

di mana R adalah grid kernel (misalnya 3×3), x adalah *feature map* masukan, Δp_n adalah *offset* yang diprediksi oleh layer paralel, dan $w(p_n)$ adalah bobot kernel. Pendekatan ini memungkinkan jaringan menyesuaikan *receptive field* secara dinamis terhadap bentuk objek yang kompleks, mengungguli konvolusi biasa dalam situasi yang membutuhkan fleksibilitas spasial tinggi, seperti pengenalan objek dengan pose tidak teratur atau kondisi pencahayaan ekstrem (Burgos Madrigal et al., 2024). Sementara itu, peninjauan literatur dalam deteksi objek modern menempatkan DCN sebagai salah satu inovasi penting dalam peningkatan kinerja model, terutama pada arsitektur *deep learning* yang digunakan untuk tugas pendeteksian objek di lingkungan kompleks seperti lalu lintas dan kondisi pencahayaan sulit (Aziz et al., 2020).

Secara keseluruhan, DCN memberikan fleksibilitas spasial yang tidak dimiliki konvolusi standar. Kemampuannya dalam menyesuaikan area sampling secara adaptif membuatnya sangat relevan untuk peningkatan sistem deteksi objek modern, terutama pada domain yang membutuhkan ketahanan tinggi terhadap variasi bentuk, kemiringan, dan distorsi geometris.

1.2.7 Metrik Evaluasi

a. Confusion Matriks

Confusion matrix digunakan untuk menggambarkan hubungan antara kelas sebenarnya dan prediksi model dalam sistem deteksi objek. Pada YOLO, setiap prediksi dievaluasi

sebagai *true positive* ketika objek terdeteksi dengan kelas dan lokasi yang benar, *false positive* ketika model mendeteksi objek pada area yang sebenarnya merupakan background atau salah kelas, serta *false negative* ketika objek yang seharusnya muncul tidak terdeteksi. Area tanpa objek yang tidak menghasilkan prediksi sebenarnya merupakan *true negative*, nilai ini tidak ditampilkan secara eksplisit dalam *confusion matrix* YOLO karena jumlahnya sangat besar dan tidak memberikan informasi yang bermakna untuk analisis kinerja. Relasi ini menjadi dasar seluruh metrik evaluasi seperti precision, *recall*, F1-score, dan mAP (Wei et al., 2025).

Dalam implementasi evaluasi YOLO, *confusion matrix* ditampilkan dalam format *(K+1) × (K+1)* (*k* adalah baris dan kolom), di mana satu baris dan kolom tambahan digunakan untuk mewakili *background*. *Background* bukan merupakan kelas yang diprediksi oleh model, melainkan bagian dari mekanisme evaluasi. Ketika suatu prediksi tidak memiliki pasangan *ground truth*, kesalahan tersebut dicatat sebagai *false positive* pada baris kelas prediksi dan kolom *background* keadaan tersebut juga disebut sebagai *No True Label* (NTL). Sebaliknya, ketika suatu objek *ground truth* tidak memiliki pasangan prediksi, kesalahan tersebut dicatat sebagai false negative pada baris background dan kolom kelas *ground truth* keadaan tersebut juga disebut sebagai *No Predicted Label* (NPL). Dengan demikian NTL dan NPL direpresentasikan melalui baris dan kolom *background* dalam *confusion matrix* untuk mencatat kesalahan deteksi secara sistematis (Heydarian et al., 2022).

b. Precision

Precision digunakan untuk menggambarkan tingkat ketepatan model dalam menghasilkan prediksi positif. Metrik ini menunjukkan seberapa besar proporsi prediksi yang benar-benar sesuai dengan kondisi sebenarnya, sehingga memberikan penilaian tentang kemampuan model dalam menghindari kesalahan ketika menyatakan keberadaan suatu objek atau kelas. *Precision* yang tinggi berarti sebagian besar prediksi positif model adalah benar, sedangkan nilai yang lebih rendah menunjukkan bahwa model sering menghasilkan prediksi keliru yang tercatat sebagai *false positive*. Secara konseptual, *precision* didefinisikan melalui perbandingan antara prediksi benar dan seluruh prediksi positif yang dihasilkan, sehingga menekankan kualitas dan akurasi dari setiap keputusan positif yang dibuat model (Wei et al., 2025).

$$\text{Precision} = \frac{TP}{TP + FP} \tag{3}$$

Dimana:
TP = True Positive
FP = False Positive

c. Recall

Recall digunakan untuk menggambarkan sejauh mana model mampu mengenali seluruh *instance* positif yang terdapat pada data. Metrik ini menunjukkan proporsi objek atau label positif yang berhasil teridentifikasi dengan benar, sehingga mencerminkan kemampuan model dalam meminimalkan kesalahan berupa *instance* positif yang tidak terdeteksi. *Recall* yang tinggi menunjukkan bahwa model hanya melewatkan sedikit *instance* positif, sedangkan *recall* yang rendah menandakan banyaknya *false negative*

yang terjadi. Dengan demikian, *recall* menekankan tingkat kelengkapan deteksi dan didefinisikan melalui perbandingan antara prediksi positif yang benar dengan seluruh *instance* positif sebenarnya (Wei et al., 2025).

$$\text{Recall} = \frac{TP}{TP + FN} \tag{4}$$

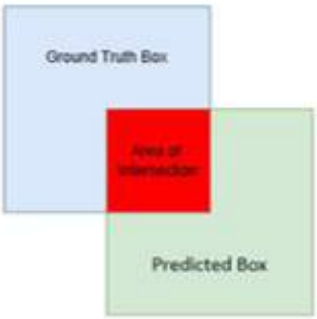
Dimana:
TP = True Positive
FN = False Negatif

d. Intersection over Union (IoU)

Intersection over Union (IoU) merupakan metrik yang digunakan untuk menilai tingkat kesesuaian antara *bounding box* hasil prediksi dengan *bounding box* sebenarnya. Metrik ini bekerja dengan membandingkan luas area yang menjadi tumpang tindih antara kedua kotak (*Area of Intersection*) dengan luas gabungan keduanya (*Area of Union*). Semakin besar area yang saling menutupi, semakin akurat prediksi posisi objek yang dilakukan model. IoU digunakan sebagai dasar untuk menentukan apakah suatu prediksi dianggap benar atau tidak berdasarkan ambang batas tertentu.

$$IoU = \frac{\text{Area of Intersection}}{\text{Area of Union}} \tag{5}$$

Untuk memberikan gambaran yang mengenai cara perhitungan IoU, ilustrasi pada gambar 9 menunjukkan posisi *ground truth box* dan *predicted box* beserta area tumpang tindih (*intersection*) di antara keduanya. Area yang ditandai dengan warna merah merupakan bagian yang menjadi *intersection*, sedangkan keseluruhan area yang dibentuk oleh kedua kotak tersebut merupakan *union*. Nilai IoU akan meningkat ketika area *intersection* semakin besar dibandingkan luas union, yang berarti prediksi model semakin mendekati anotasi sebenarnya.



Gambar 9. Ilustrasi Intersection over Union (IoU)

e. mAP@50 dan mAP@50-95

Average Precision (AP) digunakan untuk menilai performa deteksi suatu kategori objek dengan melihat hubungan antara *precision* dan *recall* pada berbagai tingkat *recall*. AP dihitung sebagai luas area di bawah kurva *precision* dan *recall*, sehingga

menggambarkan akurasi deteksi model di seluruh rentang *recall*. Nilai ini memberikan gambaran mengenai seberapa baik model menjaga *precision* ketika *recall* meningkat. Secara matematis, AP dinyatakan sebagai integral dari *precision* terhadap *recall* pada interval 0 hingga 1, sebagaimana ditunjukkan pada persamaan berikut (Wei et al., 2025):

$$AP = \int_0^1 Precision(R)\, dR \tag{6}$$

Mean Average Precision (mAP) merupakan nilai rata-rata dari AP seluruh kategori objek dan digunakan sebagai indikator umum untuk menilai kinerja model secara keseluruhan. Semakin tinggi nilai mAP, semakin baik kemampuan model dalam mendeteksi berbagai kelas objek secara konsisten. Rumus mAP dapat dinyatakan sebagai berikut:

$$mAP = \frac{1}{N}\sum_{i=1}^{N} AP_i \tag{7}$$

dimana

N : jumlah kelas

AP_i : Average Precision untuk kelas ke-i.

Dalam evaluasi berbasis YOLO, terdapat dua mAP yang digunakan mAP@50 dan mAP@50-95. Pada mAP@50, perhitungan AP dilakukan dengan ambang *Intersection over Union* (IoU) tunggal sebesar 0.50, sehingga prediksi dianggap benar apabila IoU antara *bounding box* prediksi dan *ground truth* mencapai nilai lebih besar atau sama dengan 50%. Sementara itu, mAP@50-95 merupakan metrik di mana nilai AP dihitung pada ambang IoU berbeda, yaitu 0.50, 0.55, 0.60, hingga 0.95, kemudian seluruh hasilnya dirata-ratakan. Dengan penggunaan banyak ambang IoU, mAP@50-95 memberikan gambaran performa yang lebih komprehensif, karena model harus menunjukkan akurasi deteksi sekaligus presisi pada berbagai tingkat IoU yang berbeda.

f. PIQE

Perception-based Image Quality Evaluator (PIQE) merupakan metrik *no-reference image quality assessment* yang digunakan untuk mengevaluasi kualitas citra tanpa memerlukan citra referensi. PIQE merupakan pengujian kualitas citra berbasis *no-reference* yang tidak memerlukan *reference image*. Penilaian dilakukan dengan mengestimasi noise pada area penting citra, menghitung tingkat noise pada blok lokal berukuran 16 × 16, kemudian menghasilkan skor rendah untuk kualitas perseptual yang lebih baik dan skor tinggi untuk kualitas yang lebih buruk. PIQE digunakan karena citra rambu lalu lintas malam hari tidak memiliki pasangan citra referensi, sehingga evaluasi kualitas citra lebih tepat dilakukan dengan pendekatan *no-reference* (Yoon & Cho, 2023) (Venkatanath N et al., 2015).

g. Kompleksitas Komputasi

Kompleksitas komputasi digunakan untuk mengevaluasi efisiensi arsitektur model deteksi objek dari sisi kebutuhan sumber daya dan kecepatan pemrosesan. Evaluasi ini penting untuk memahami *trade-off* antara performa deteksi dan beban komputasi yang dihasilkan oleh suatu model (Chen et al., 2025). Kompleksitas komputasi dievaluasi

menggunakan empat metrik utama, yaitu *parameter count*, *floating point operations* (FLOPs), *frames per second* (FPS), dan *inference time*.

Jumlah parameter (*parameter count*) merepresentasikan total bobot yang disimpan oleh model. Metrik ini berkaitan langsung dengan ukuran model dan kebutuhan memori. Model dengan jumlah parameter yang lebih besar memiliki kapasitas representasi yang lebih tinggi, namun berpotensi meningkatkan beban komputasi (Chen et al., 2025). FLOPs digunakan untuk mengukur jumlah operasi aritmatika yang dilakukan model dalam satu kali proses inferensi. Metrik ini menggambarkan kompleksitas komputasi teoretis dari suatu arsitektur jaringan. Nilai FLOPs yang lebih rendah menunjukkan bahwa model memerlukan lebih sedikit operasi komputasi untuk menghasilkan prediksi (Z. Zhang et al., 2024).

Frames per second (FPS) mengukur jumlah citra atau *frame* yang dapat diproses oleh model dalam satu detik. Metrik ini mencerminkan kemampuan pemrosesan model secara keseluruhan dan sangat dipengaruhi oleh jumlah parameter, FLOPs, serta arsitektur jaringan. Priadana *et al.*, menggunakan FPS sebagai indikator utama untuk menilai efisiensi pemrosesan model YOLO yang dioptimalkan, di mana FPS yang lebih tinggi menunjukkan kemampuan pemrosesan yang lebih cepat (Priadana et al., 2025). *Inference time* mengukur waktu yang dibutuhkan model untuk memproses satu citra atau satu frame, mulai dari input hingga menghasilkan keluaran prediksi. Berbeda dengan FPS yang bersifat agregatif, *inference time* memberikan gambaran latensi pemrosesan per citra. Nilai *inference time* yang lebih kecil menunjukkan bahwa model mampu menghasilkan prediksi dengan lebih cepat pada setiap input (Z. Zhang et al., 2024).

1.3 Rumusan Masalah

Berdasarkan latar belakang yang telah diuraikan, maka rumusan masalah dalam penelitian ini adalah sebagai berikut:

1. Bagaimana mengoptimalkan performa algoritma YOLOv11 melalui integrasi Deformable Convolutional Network (DCN) dalam melakukan deteksi dan klasifikasi rambu lalulintas pada malam hari?
2. Bagaimana mengevaluasi pengaruh integrasi modul Deformable Convolutional Network (DCN) dengan YOLOv11 dalam melakukan deteksi dan klasifikasi rambu lalulintas pada malam hari?

1.4 Tujuan dan Manfaat

1.4.1 Tujuan

Tujuan yang ingin dicapai dalam penelitian ini adalah:

1. Mengoptimalkan performa algoritma YOLOv11 melalui integrasi *Deformable Convolutional Network* (DCN) dalam melakukan deteksi dan klasifikasi rambu lalulintas pada malam hari.
2. Mengevaluasi pengaruh integrasi modul *Deformable Convolutional Network* (DCN) dengan YOLOv11 dalam melakukan deteksi dan klasifikasi rambu lalulintas pada malam hari.

1.4.2 Manfaat

Penelitian ini diharapkan memberikan kontribusi secara akademis dan praktis sebagai berikut:

1. Memberikan kontribusi ilmiah dalam pengembangan metode deteksi objek berbasis *deep learning*, khususnya melalui integrasi *Deformable Convolutional Network* (DCN) pada arsitektur YOLOv11 dalam deteksi dan klasifikasi rambu lalu lintas malam hari.
2. Menjadi referensi akademik bagi penelitian selanjutnya yang membahas deteksi objek pada kondisi visual yang sulit, seperti pencahayaan rendah, blur akibat pergerakan, dan distorsi bentuk.
3. Mendukung kinerja ADAS dalam proses navigasi, dengan menyediakan model deteksi rambu malam hari yang lebih akurat sehingga sistem dapat membaca batas kecepatan, arah, dan aturan jalan untuk membantu pengendara mengambil keputusan yang lebih aman dan sesuai kondisi jalan.

1.5 Batasan Masalah

Agar penelitian tetap fokus dan terarah, ruang lingkup penelitian ini dibatasi oleh beberapa hal berikut:

1. Penelitian hanya berfokus pada deteksi dan klasifikasi rambu lalu lintas pada kondisi malam hari dengan menggunakan video yang diambil oleh kamera *dashcam* DDPAI Mola N3 pada kecepatan kendaraan relatif antara 30 km/jam hingga 50 km/jam.
2. Penelitian hanya menggunakan model YOLOv11 sebagai dasar dan mengintegrasikan modul *Deformable Convolutional Network* (DCN) untuk meningkatkan performa.
3. Evaluasi performa model dilakukan menggunakan metrik precision, recall, mAP, inference time, FLOPs, Parameters Count.
4. Penelitian tidak membahas sistem *end to end* kendaraan otonom, melainkan hanya fokus pada aspek deteksi objek berbasis visi komputer.
5. Jenis rambu yang digunakan dibatasi pada rambu lalu lintas berikut, Larangan Parkir, Larangan Berkendara dengan Kecepatan Lebih dari 40km/jam, Petunjuk Lokasi Putar Balik, Petunjuk Lokasi Masjid, Peringatan Hati-hati, Peringatan Tikungan Ke Kanan, Peringatan Tikungan Ke Kiri, Peringatan Jembatan atau Penyempitan Badan Jalan, Peringatan Banyak Lalu Lintas Pejalan Kaki, Peringatan Alat Pemberi Isyarat Lalu Lintas.

BAB II
METODE PENELITIAN

2.1 Tahapan Penelitian

Penelitian ini dilakukan melalui beberapa tahapan yang disusun secara sistematis untuk mencapai tujuan penelitian sebagaimana dapat dilihat pada Gambar 10. Proses diawali dengan identifikasi permasalahan utama terkait rendahnya performa deteksi dan klasifikasi rambu lalu lintas pada kondisi malam hari yang kemudian dilanjutkan dengan studi literatur guna memperoleh dasar teori dan pendekatan yang relevan, khususnya terkait algoritma YOLO dan *Deformable Convolutional Network* (DCN). Selanjutnya, dilakukan pengumpulan data berupa citra rambu lalu lintas malam hari menggunakan kamera dashcam, yang kemudian dipersiapkan melalui proses pengolahan data agar sesuai dengan format pelatihan model. Tahap berikutnya adalah perancangan sistem dengan memodifikasi arsitektur YOLOv11 melalui integrasi modul DCN. Model hasil perancangan kemudian dilatih dan diuji menggunakan dataset yang telah disiapkan untuk mengukur kinerjanya. Evaluasi dilakukan dengan membandingkan performa model yang diusulkan terhadap model YOLOv11 menggunakan metrik evaluasi kuantitatif serta analisis kualitatif melalui visualisasi hasil deteksi. Seluruh hasil eksperimen dan analisis yang diperoleh selanjutnya disusun secara sistematis dalam bentuk laporan tesis.

Gambar 10. Tahapan Penelitian

2.2 Tempat dan Waktu

Penelitian ini dilaksanakan sejak disetuinya proposal penelitian pada bulan Januari 2025 hingga penyusunan laporan hasil penelitian pada bulan Desember 2025. Seluruh kegiatan penelitian dilakukan di Laboratorium Kecerdasan Buatan, Departemen Teknik Informatika, Universitas Hasanuddin. Proses pengambilan data dilakukan secara langsung di lapangan menggunakan kamera *dashcam* yang dipasang pada kendaraan. Lokasi pengambilan data mencakup sejumlah ruas jalan di Kota Makassar, Kabupaten Gowa, Kabupaten Maros dan Kabupaten Pangkep, Provinsi Sulawesi Selatan.

2.3 Perangkat dan Lingkungan Implementasi Penelitian

Perangkat yang digunakan dalam penelitian ini meliputi perangkat lunak (*software*) dan keras (*hardware*) sebagai penunjang dalam proses pengumpulan, pengolahan, serta pengujian data.

2.3.1 Perangkat Lunak

a. *Operating System* Windows 11 64-bit
b. Visual Studio Code
c. Roboflow
d. Kaggle (GPU P100)
e. Zetero

2.3.2 Perangkat Keras

a. Dashcam Mola N3.
b. Personal komputer dengan prosesor Intel® Core™ i7 Gen 13, RAM 16 GB, dan Intel UHD Graphics 770 (8 GB).
c. Lenovo, Prosesor AMD Athlon Silver 7120U, RAM 8GB

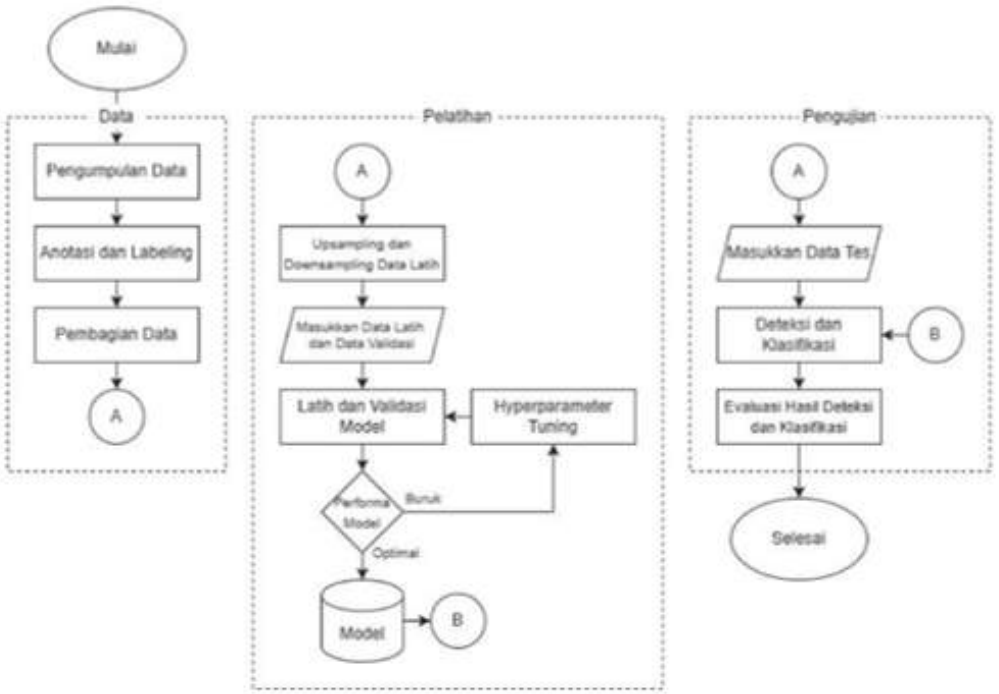
2.3.3 Lingkungan Implementasi

Penelitian ini dilaksanakan menggunakan dua lingkungan komputasi, yaitu platform kaggle sebagai lingkungan utama dan perangkat komputer sebagai pendukung. Seluruh proses inti penelitian meliputi pelatihan, validasi, dan pengujian model YOLOv11 *baseline* serta model YOLOv11 yang ditingkatkan dengan modul *Deformable Convolutional Network* (DCN) dilakukan sepenuhnya pada platform Kaggle. Penggunaan satu lingkungan yang sama memastikan bahwa kedua model dievaluasi secara adil (*fair comparison*) tanpa adanya pengaruh perbedaan perangkat keras. Kaggle dipilih karena menyediakan lingkungan komputasi berbasis cloud dengan dukungan GPU seperti NVIDIA Tesla P100 atau T4 yang mampu memproses pelatihan model deteksi objek. Selain itu, Kaggle telah terintegrasi dengan pustaka penting seperti PyTorch dan Ultralytics, sehingga memudahkan proses implementasi, pemanggilan dataset, serta pemantauan metrik pelatihan dan evaluasi.

Perangkat komputer pribadi digunakan untuk aktivitas pendukung yang tidak memerlukan komputasi intensif, seperti pengelolaan dataset, anotasi, pelabelan, serta penyusunan laporan penelitian. Meski tidak digunakan dalam proses pelatihan, perangkat ini berperan dalam memastikan alur kerja penelitian dapat dilakukan secara teratur dan efisien. Dengan pemanfaatan Kaggle sebagai lingkungan pelatihan utama dan komputer sebagai perangkat pendukung, penelitian ini memperoleh konsistensi komputasi yang diperlukan serta fleksibilitas dalam pengolahan data dan dokumentasi.

2.4 Alur Sistem

Alur sistem pada penelitian ini dirancang untuk menggambarkan proses lengkap mulai dari persiapan data, pelatihan model, hingga pengujian model deteksi dan klasifikasi rambu lalu lintas malam hari. Setiap tahap saling terintegrasi untuk memastikan bahwa model yang dihasilkan memiliki kemampuan generalisasi yang baik serta mampu menangani variasi kondisi yang muncul pada data sesungguhnya. Gambar 11 berikut menyajikan alur sistem secara menyeluruh.



Gambar 11. Alur Sistem

Tahap pertama adalah proses pengelolaan data, dimulai dari pengumpulan citra rambu lalu lintas malam hari kemudian dilanjutkan dengan proses anotasi dan labeling. Pada tahap ini setiap objek rambu ditandai sesuai kelasnya untuk menghasilkan *ground truth* yang akan digunakan dalam pelatihan. Setelah itu, data dibagi menjadi subset data latih, validasi, dan tes. Pemisahan ini penting agar data yang digunakan pada proses pelatihan dan validasi berbeda dengan data yang digunakan pada tahap pengujian, sehingga model dapat dievaluasi secara objektif terhadap data yang tidak masuk kedalam data yang digunakan dalam proses pelatihan.

Tahap kedua merupakan proses pelatihan model, yang mencakup *upsampling* dan atau *downsampling* untuk menyeimbangkan distribusi kelas pada data latih. Selanjutnya, data latih dan validasi dimasukkan ke dalam model untuk melalui proses *training* dan evaluasi performa secara iteratif. Jika performa yang dihasilkan belum optimal, dilakukan penyesuaian hiperparameter (*hyperparameter tuning*) hingga diperoleh konfigurasi terbaik. Ketika model mencapai performa optimal, bobot model disimpan sebagai model akhir yang siap digunakan pada tahap pengujian.

Tahap ketiga adalah proses pengujian model, dimulai dengan memasukkan data uji ke dalam sistem. Model kemudian melakukan proses deteksi dan klasifikasi objek rambu berdasarkan pola dan representasi yang telah dipelajari selama proses pelatihan. Hasil deteksi dibandingkan dengan *ground truth* untuk menghasilkan metrik evaluasi, *precision*, *recall*, dan mAP. Tahap ini memberikan gambaran mengenai kemampuan model ketika dihadapkan pada kondisi nyata, sekaligus menjadi dasar untuk menilai efektivitas pendekatan yang digunakan dalam penelitian ini.

2.5 Dataset dan Sumber Data

2.5.1 Sumber Data dan Proses Akuisisi

Dataset yang digunakan dalam penelitian ini merupakan data primer yang diperoleh hanya dari proses perekaman langsung kondisi jalan raya pada malam hari. Penggunaan data primer tanpa data sekunder dari misalnya Kaggle untuk memastikan bahwa dataset yang digunakan merepresentasikan kondisi nyata yang dihadapi sistem deteksi rambu lalu lintas pada lingkungan sesungguhnya, khususnya pada area atau Lokasi pengambilan data. Seluruh data direkam menggunakan kamera *dashcam* yang dipasang pada kendaraan dan diarahkan ke depan mengikuti arah jalannya kendaraan, sehingga sudut pandang citra yang diperoleh menyerupai kondisi operasional sistem deteksi rambu pada kendaraan nyata.

Gambar 12. DDPAI Mola N3

Perangkat akuisisi data yang digunakan adalah kamera *dashcam* DDPAI Mola N3 yang dapat merekam video dengan resolusi 1600P dan kecepatan *frame* 30 fps. *Dashcam* dipasang pada bagian tengah depan kendaraan untuk memperoleh bidang pandang yang optimal terhadap rambu lalu lintas di sepanjang jalur perjalanan. Perangkat *dashcam* yang digunakan dalam penelitian ini ditunjukkan pada Gambar 12, sedangkan ilustrasi skema pemasangan dan arah pandang kamera terhadap jalan direpresentasikan pada Gambar 13. Penempatan kamera ini bertujuan untuk meminimalkan distorsi sudut pandang ekstrem serta memastikan rambu lalu lintas terekam dalam posisi yang konsisten relatif terhadap kendaraan.

Gambar 13. Ilustrasi Skenario Pengambilan Data

Proses pengambilan data dilakukan pada waktu malam hari dengan variasi kondisi pencahayaan jalan yang umum ditemui di lingkungan nyata. Data direkam pada beberapa ruas jalan di Kota Makassar, Kabupaten Gowa, Kabupaten Maros, dan Kabupaten Pangkep, Provinsi Sulawesi Selatan, yang memiliki karakteristik pencahayaan berbeda, mulai dari area dengan penerangan jalan yang memadai seperti pada area dengan lampu jalan, hingga area dengan intensitas cahaya rendah seperti pada area tanpa lampu jalan. Selain itu, perekaman dilakukan pada lingkungan jalan yang beragam, seperti jalan perkotaan dan jalan penghubung antarwilayah, sehingga dataset mencerminkan variasi latar belakang, jarak pandang, dan kondisi visual rambu lalu lintas yang berbeda-beda. Selama proses perekaman, kendaraan bergerak dengan kecepatan relatif antara 30 km/jam hingga 50 km/jam berdasarkan pengamatan pada speedometer, dimana kecepatan tersebut merepresentasikan kecepatan kendaraan pada umumnya saat melintas di ruas jalan raya. Variasi kondisi ini penting untuk memastikan bahwa dataset yang dikumpulkan tidak bersifat homogen dan mampu merepresentasikan tantangan visual yang muncul pada deteksi rambu lalu lintas di malam hari.

2.5.2 Anotasi dan Pembagian Dataset

Dataset citra rambu lalu lintas yang digunakan dalam penelitian ini diperoleh melalui proses ekstraksi *frame* dari data video hasil perekaman yang dijelaskan pada proses sebelumnya. Citra hasil ekstraksi tersebut kemudian digunakan sebagai dataset utama yang selanjutnya melalui proses anotasi untuk menghasilkan *ground truth* yang digunakan pada pelatihan dan evaluasi model. Contoh citra hasil ekstraksi *frame* dari video perekaman *dashcam* pada malam hari ditunjukkan pada Gambar 14 yang memperlihatkan kondisi visual rambu lalu lintas sebagaimana direkam oleh kamera kendaraan. Proses anotasi dilakukan dengan memberikan *bounding box* pada setiap objek rambu lalu lintas yang muncul pada citra, sesuai dengan kelas rambu yang telah

ditetapkan dalam ruang lingkup penelitian. Anotasi ini bertujuan untuk menandai lokasi dan kelas objek secara akurat sehingga model dapat mempelajari hubungan antara fitur visual citra dan label rambu lalu lintas yang benar.

Gambar 14. Contoh Hasil Ekstraksi Frame dari Video Dashcam pada Malam Hari

Proses anotasi dilakukan menggunakan platform Roboflow, yang menyediakan antarmuka anotasi dan pelabelan berbasis web serta dukungan ekspor label dalam berbagai format. Pada penelitian ini, format anotasi yang digunakan adalah format YOLO di mana setiap *bounding box* direpresentasikan oleh koordinat ternormalisasi terhadap ukuran citra beserta label kelas objek. Proses pelabelan objek rambu lalu lintas menggunakan platform Roboflow ditunjukkan pada Gambar 15 yang menggambarkan tahapan penentuan *bounding box* dan pemberian label kelas pada citra. Setelah proses anotasi selesai, dilakukan verifikasi dan pengecekan ulang terhadap hasil anotasi untuk memastikan kesesuaian antara kelas objek pada citra dan label yang diberikan.

Gambar 15. Proses Anotasi dan Pelabelan Rambu Lalu Lintas Menggunakan Platform Roboflow

Melalui proses anotasi dan pelabelan, dataset penelitian ini diklasifikasikan ke dalam 10 kelas rambu lalu lintas yang merepresentasikan jenis rambu yang umum ditemui pada kondisi jalan raya, sebagaimana dirangkum pada Tabel 1. Selanjutnya dataset yang telah dianotasi kemudian dibagi ke dalam tiga subset utama, yaitu *training set*, *validation set*, dan *testing set*. Dari total sebanyak 3544 citra yang digunakan dalam penelitian ini, dataset dibagi menjadi 2467 untuk data pelatihan, 538 untuk data validasi dan 539 untuk data Pengujian. Pembagian dataset ini bertujuan untuk memastikan bahwa proses pelatihan, penyesuaian parameter, dan evaluasi model dilakukan secara objektif menggunakan data yang saling terpisah. *Training set* digunakan untuk melatih model, *validation set* digunakan untuk memantau kinerja model selama pelatihan, sedangkan *testing set* digunakan untuk mengevaluasi performa akhir model pada data yang tidak pernah dilihat sebelumnya. Rasio pembagian dataset ditetapkan secara proporsional agar setiap subset tetap merepresentasikan distribusi kelas rambu lalu lintas yang ada.

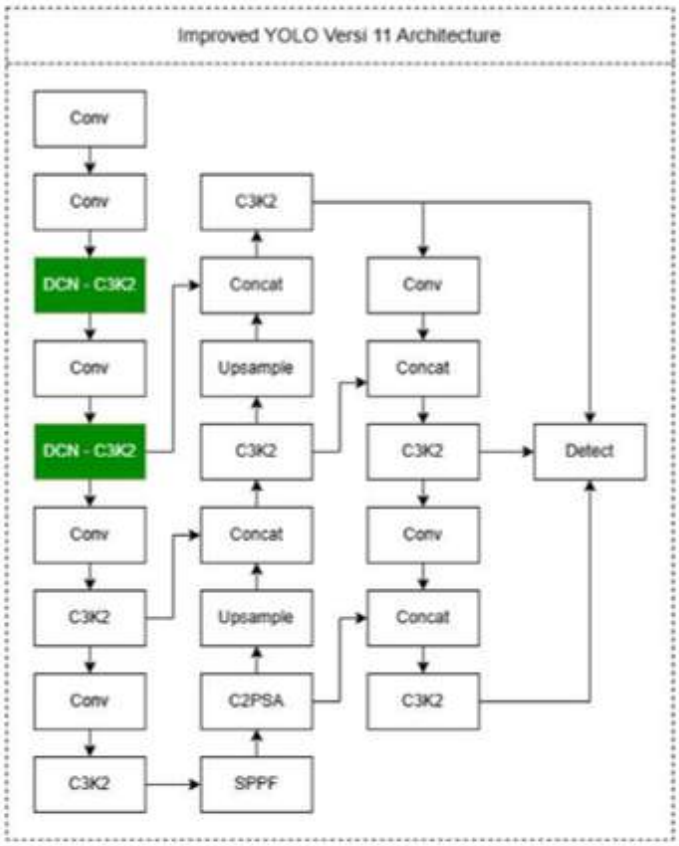
Tabel 1. Tabel Kelas Dataset

No. Indeks	Nama Kelas
0	Larangan Parkir
1	Larangan Berkendara dengan Kecepatan Lebih dari 40km/jam
2	Petunjuk Lokasi Putar Balik
3	Petunjuk Lokasi Masjid
4	Peringatan Hati-hati
5	Peringatan Tikungan Ke Kanan
6	Peringatan Tikungan Ke Kiri
7	Peringatan Jembatan atau Penyempitan Badan Jalan
8	Peringatan Banyak Lalu Lintas Pejalan Kaki
9	Peringatan Alat Pemberi Isyarat Lalu Lintas

Untuk mengurangi ketimpangan distribusi jumlah data antar kelas dilakukan penyesuaian dataset melalui teknik *downsampling* dan *upsampling*. *Downsampling* diterapkan dengan menghapus sebagian data pada kelas mayoritas, sehingga dominasi kelas tertentu dapat dikurangi. Sementara itu, *upsampling* dilakukan dengan menambah jumlah sampel pada kelas minoritas melalui augmentasi data, sehingga distribusi kelas pada data pelatihan menjadi lebih seimbang. Seluruh proses *downsampling* dan *upsampling* hanya diterapkan pada data pelatihan, sedangkan data validasi dan pengujian dibiarkan dalam kondisi asli tanpa penyesuaian. Pendekatan ini bertujuan untuk menjaga objektivitas evaluasi model.

2.6 Optimasi Algoritma You Only Look Once Versi 11

Pada penelitian ini, arsitektur YOLOv11 dioptimasi dengan menambahkan modul *Deformable Convolutional Network* (DCN) pada bagian C3k2 di *backbone*. Struktur lengkap dari rancangan arsitektur hasil optimasi tersebut dapat dilihat pada Gambar 16.



Gambar 16. Arsitektur Optimasi Algoritma You Only Look Once Versi 11

Setiap komponen pada arsitektur tersebut memiliki peran tersendiri dalam proses pendeteksian objek. *Backbone* berfungsi sebagai ekstraktor fitur utama, *neck* bertugas menggabungkan fitur multiskala, sedangkan *head* menghasilkan prediksi akhir berupa *bounding box* dan klasifikasi objek. Penjelasan lebih rinci mengenai fungsi dari setiap modul disajikan pada subbagian berikut.

2.6.1 Modul Convolution (Conv)

Modul Conv pada YOLOv11 disusun secara *sequential*, terdiri atas tiga tahapan utama, yaitu operasi konvolusi (Conv2D), *Batch Normalization* (BN), dan fungsi aktivasi *Sigmoid Linear Unit (SiLU)*.

a. Conv2d

Operasi konvolusi dua dimensi (*Conv2D*) bertujuan untuk mengekstraksi fitur spasial dari citra masukan dengan cara mengalikan kernel (filter) terhadap bagian tertentu dari citra, kemudian menjumlahkan hasilnya. Pada arsitektur YOLOv11. Secara matematis, operasi konvolusi 2D dapat dinyatakan sebagai berikut:

$$x_{i,j} = \sum_{m=0}^{2} \sum_{n=0}^{2} I_{(i+m),(j+n)} \cdot W_{m,n} \quad (8)$$

dimana:

$I_{(i+m),(j+n)}$: nilai piksel pada posisi $(i+m, j+n)$dari citra masukan,

$w_{m,n}$: bobot kernel pada posisi (m,n)

$x_{i,j}$: hasil konvolusi pada posisi (i,j).

b. BN

Setelah proses konvolusi, hasilnya akan mengalami distribusi nilai yang beragam. Untuk menstabilkan distribusi ini digunakan *Batch Normalization* (BN) yang menormalkan nilai keluaran berdasarkan rata-rata dan variansi. Persamaan BN adalah:

$$\hat{x_i} = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \quad (9)$$

$$y_i = \gamma \hat{x}_i + \beta \quad (10)$$

dimana:

x_i: nilai keluaran hasil konvolusi,

μ_B: nilai rata-rata mini-batch,

σ_B^2: variansi mini-batch,

ϵ: konstanta kecil untuk mencegah pembagian nol,

γ, β: parameter skala dan pergeseran yang dapat dilatih.

c. SiLU

Tahap terakhir pada blok *Conv* adalah penerapan fungsi aktivasi SiLU (*Sigmoid Linear Unit*). Fungsi ini mengombinasikan linearitas dan non-linearitas melalui perkalian antara input dan fungsi sigmoid-nya. Persamaan SiLU dituliskan sebagai:

$$\text{SiLU}(x) = x \cdot \sigma(x) \; dengan \; \sigma(x) = \frac{x}{1 + e^{-x}} \quad (11)$$

x : Nilai input hasil keluaran dari proses *Batch Normalization (BN)*.

$\sigma(x)$: Fungsi sigmoid yang memberikan pembobot non-linear terhadap nilai x

Setiap operasi konvolusi dalam modul *Conv* tidak hanya mengekstraksi fitur spasial, tetapi juga mengubah dimensi representasi data, baik dari sisi ukuran spasial (*height* dan *width*) maupun jumlah *channel* (*depth*). Ukuran keluaran dari proses konvolusi dua dimensi dapat dihitung menggunakan persamaan:

$$H_{out} = \lfloor \frac{H_{in} - K + 2P}{S} \rfloor + 1, \quad W_{out} = \lfloor \frac{W_{in} - K + 2P}{S} \rfloor + 1 \quad (12)$$

Dimana:

H_{in}, W_{in}: tinggi dan lebar citra masukan,

H_{out}, W_{out}: tinggi dan lebar keluaran,

K: ukuran kernel (misalnya 3),

P: jumlah *padding*,

S: ukuran *stride*.

Selain dimensi spasial, konvolusi juga memengaruhi jumlah *channel* keluaran (C_{out}). Pada arsitektur YOLOv11, jumlah channel keluaran setiap layer telah ditentukan secara eksplisit dalam konfigurasi model. Jumlah kernel konvolusi akan otomatis menyesuaikan agar sama dengan jumlah channel keluaran tersebut, karena setiap kernel menghasilkan satu *feature map*. Dengan demikian, hubungan antara jumlah kernel dan channel keluaran dapat dinyatakan sebagai:

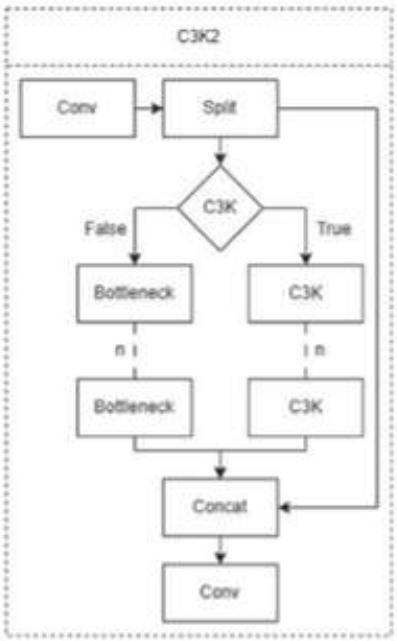
$$C_{out} = N_{kernel} \tag{13}$$

sehingga dimensi akhir dari hasil konvolusi adalah $(C_{out}, H_{out}, W_{out})$.

Sebagai contoh, pada layer awal YOLOv11 dengan masukan berukuran 640×640×3, kernel 3×3, *stride* 2, dan jumlah kernel 64, maka diperoleh keluaran berukuran 320×320×64. Ukuran spasial berkurang menjadi setengah akibat *stride* dua, sedangkan jumlah *channel* meningkat dari tiga menjadi enam puluh empat karena setiap kernel menghasilkan satu *fiture maps* baru. Peningkatan jumlah *channel* ini memperkaya representasi fitur yang dipelajari model, sehingga jaringan mampu mengenali pola visual yang lebih kompleks pada tahap-tahap berikutnya.

2.6.2 Modul C3k2

Modul C3k2 pada YOLOv11 merupakan salah satu komponen utama dalam proses ekstraksi fitur yang menggabungkan efisiensi struktur *Cross Stage Partial (CSP)* dengan fleksibilitas kernel adaptif dari blok C3k. Proses konvolusi (*Conv*) tidak dijelaskan kembali karena telah diuraikan pada bagian sebelumnya, sedangkan fokus pembahasan pada bagian ini mencakup empat tahapan utama, yaitu Split, Bottleneck (jika c3k=False), C3k (jika c3k=True), dan Concat. Alur pada modul C3k2 dapat dilihat pada Gambar 17.



Gambar 17. Alur C3k2

a. Tahap Split

Setelah melalui konvolusi 1×1 pada tahap sebelumnya, fitur hasil keluaran dibagi menjadi dua bagian menggunakan operasi *chunk* pada dimensi kanal. Operasi ini menghasilkan dua jalur: jalur utama (A) yang akan diproses lebih lanjut, dan jalur shortcut (B) yang dilewatkan langsung untuk menjaga stabilitas gradien. Persamaan proses pembagian fitur dapat dituliskan sebagai:

$$[A, B] = \text{split}(C) \tag{14}$$

dimana:

C = Jumlah kanal masukan

A, B = bagian kanal untuk proses selanjutnya

split(·) = fungsi pemisahan kanal

Tahapan ini merupakan implementasi dari prinsip *Cross Stage Partial (CSP)* yang bertujuan mengurangi beban komputasi dengan hanya memproses sebagian fitur melalui jalur utama.

b. Bottleneck

Pada kondisi c3k=*False*, jalur utama diproses menggunakan blok *Bottleneck* standar. Blok ini terdiri dari dua konvolusi berturut-turut, yaitu konvolusi 1×1 dan konvolusi 3×3 dan proses *residual connection* yang berfungsi untuk menjumlahkan fitur awal dan hasil *convolution*. Proses *bootleneck* dapat di lihat pada gambar 18. Persamaan proses *Bottleneck* dapat dituliskan sebagai:

$$Y = X + f_{3\times3}(f_{1\times1}(X)) \tag{15}$$

dimana:

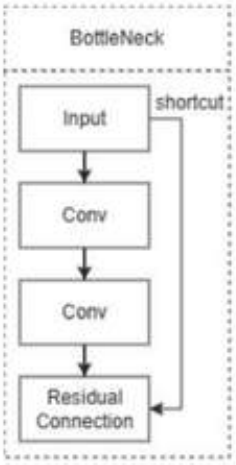
X= fitur masukan dari jalur transformasi

$f_{1\times1}$= konvolusi 1×1

$f_{3\times3}$= konvolusi 3×3

Y= hasil keluaran dari blok Bottleneck

$+$ = menunjukkan koneksi residu (*Residual connection*).



Gambar 18. Alur BottleNeck

c. C3k

Apabila argumen c3k=*True*, maka jalur transformasi akan menggunakan blok C3k, yaitu pengembangan dari struktur C3. Blok ini memanfaatkan dua konvolusi 1×1 untuk memisahkan dan menyeimbangkan fitur, serta blok *Bottleneck* di dalamnya yang prosesnya dapat di lihat pada gambar 19. Persamaan proses pada blok C3k dapat dituliskan sebagai:

$$Y = f_{1\times1}([f_{\text{Bottleneck}}(f_{1\times1}(X)), f_{1\times1}(X)]) \quad (16)$$

dimana:

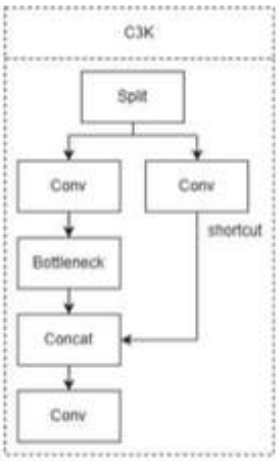
X= fitur masukan

$f_{1\times1}$= konvolusi 1×1

$f_{\text{Bottleneck}}$= blok Bottleneck

[,]= operasi *concatenation* antar fitur

Y= hasil keluaran setelah penggabungan dua jalur fitur



Gambar 19. Alur C3k

d. Concat

Tahap *concatenation* (*concat*) merupakan proses penggabungan dua fitur pada dimensi kanal (*channel dimension*). Dalam modul C3k2, satu jalur fitur berasal dari hasil pemrosesan melalui blok C3k atau *Bottleneck*, sedangkan jalur lainnya merupakan hasil konvolusi langsung tanpa pemrosesan tambahan. Kedua hasil tersebut kemudian digabung secara berdampingan sehingga jumlah kanal meningkat dua kali lipat.

$$C_{out} = [C_{in\,1}, C_{in\,2}] \quad (17)$$

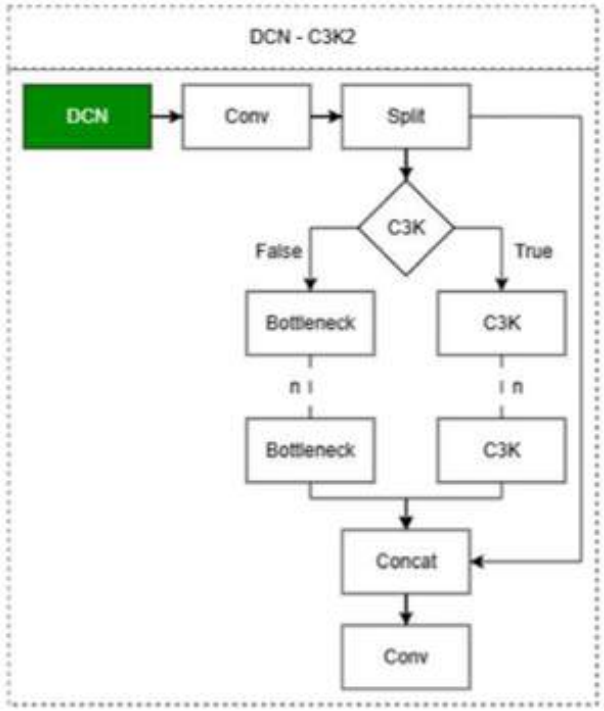
Misalkan $C_{in\,1}$ memiliki 32 kanal dan $C_{in\,2}$ juga memiliki 32 kanal maka C_{out} akan sama dengan 64.

Proses pada modul C3k2 diawali dengan konvolusi awal untuk menyesuaikan jumlah kanal masukan, kemudian fitur dibagi menjadi dua jalur melalui tahap *split*. Jalur pertama diproses menggunakan blok *Bottleneck* atau C3k sebanyak *n* kali sesuai nilai parameter c3k, sedangkan jalur kedua dilewatkan langsung sebagai *shortcut*. Kedua

hasil kemudian digabung melalui operasi *concat* pada dimensi kanal dan diakhiri dengan konvolusi 1 × 1untuk menghasilkan keluaran akhir.

2.6.3 Modul DCN-C3k2 (Integrasi Deformable Convolution)

Modul DCN-C3k2 merupakan hasil integrasi antara *Deformable Convolution Network* (DCN) dan blok C3k2 pada YOLOv11. Pada struktur ini, DCN ditempatkan sebelum proses C3k2 sebagai tahap awal untuk menyesuaikan fitur masukan terhadap bentuk dan posisi objek yang bervariasi. Dengan demikian, fitur yang dihasilkan dari DCN menjadi lebih adaptif sebelum diproses lebih lanjut oleh struktur C3k2. Setelah melalui DCN, hasil fitur diteruskan ke modul C3k2, yang selanjutnya menjalankan tahapan *split*, *Bottleneck/C3k*, *concat*, dan *conv 1×1* sebagaimana dijelaskan pada bagian sebelumnya. Integrasi ini memungkinkan modul untuk memanfaatkan keunggulan DCN dalam menangani deformasi spasial sebelum dilakukan ekstraksi fitur mendalam oleh C3k2. Proses integrasi antara DCN dan C3k2 secara umum dapat dilihat pada Gambar 20, yang menunjukkan urutan aliran data dimulai dari input yang melewati lapisan DCN hingga ke tahap akhir modul C3k2.



Gambar 20. Alur integrasi DCN dengan C3k2

a. DCN

DCN merupakan pengembangan dari konvolusi standar yang bertujuan untuk meningkatkan fleksibilitas area reseptif kernel konvolusi. Tidak seperti konvolusi konvensional yang memiliki pola sampling tetap, DCN memperkenalkan *offset* yang

dapat dipelajari secara adaptif sehingga posisi *sampling* dapat menyesuaikan bentuk objek atau fitur yang tidak beraturan. Secara matematis, DCN dinyatakan sebagai:

$$y(p_0) = \sum_{p_n \in R} x(p_0 + p_n + \Delta p_n) \cdot w(p_n) \quad (18)$$

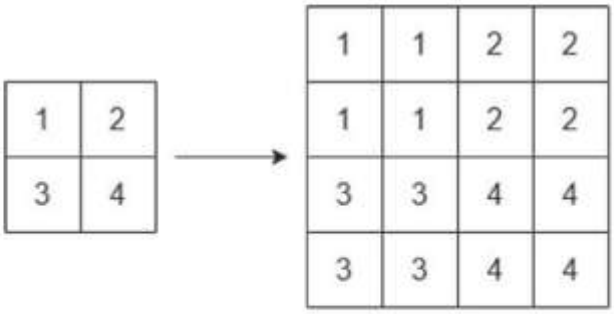
dimana:
$y(p_0)$ = nilai keluaran pada posisi p_0
$x(\cdot)$ = nilai fitur masukan.
$w(p_n)$ = bobot kernel pada posisi p_n.
Δp_n = offset spasial yang dipelajari untuk setiap posisi kernel.
p_0 = lokasi pusat dari area konvolusi.

Dengan formulasi ini, posisi sampling kernel menjadi tidak tetap, melainkan dapat bergeser mengikuti kontur dan orientasi objek di dalam citra. Hal ini memungkinkan DCN menangkap variasi bentuk dan posisi objek dengan lebih akurat dibandingkan konvolusi standar.

2.6.4 Modul Upsample

Proses *Upsampling* pada arsitektur YOLOv11 berfungsi untuk memperbesar ukuran spasial (*height* dan *width*) dari *feature map* yang sebelumnya telah diperkecil melalui konvolusi atau *downsampling*. Tahapan ini digunakan agar fitur dari beberapa level resolusi dapat digabungkan secara efektif. YOLOv11 menggunakan fungsi torch.nn.Upsample dari pustaka PyTorch dengan parameter mode='*nearest*'. Metode nearest-neighbor interpolation memperbesar ukuran fitur dengan cara menyalin nilai piksel dari posisi terdekat, tanpa melakukan perhitungan interpolasi. Setiap nilai piksel pada fitur asal direplikasi secara horizontal dan vertikal sesuai dengan faktor pembesaran (*scale factor*), misalnya dua kali lipat (scale_factor=2).

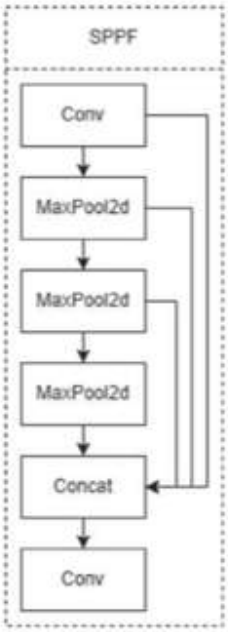
Sebagai contoh, jika *feature map* awal memiliki ukuran 2×2, maka setelah dilakukan *upsampling* dengan metode *nearest*, ukuran hasilnya menjadi 4×4, di mana setiap nilai dari fitur asal diperluas menjadi blok berukuran 2×2dengan nilai yang sama. Contoh visual proses ini dapat dilihat pada Gambar 21, yang memperlihatkan hasil replikasi piksel dari ukuran 2×2menjadi 4×4menggunakan metode *nearest-neighbor*.



Gambar 21. Contoh Visualisasi Upsample

2.6.5 Modul SPPF (Spatial Pyramid Pooling - Fast)

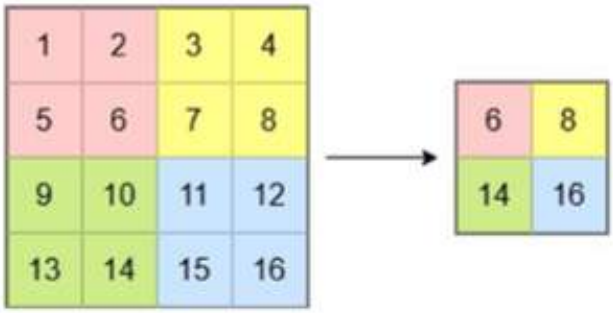
Modul SPPF (*Spatial Pyramid Pooling - Fast*) pada arsitektur YOLOv11 berfungsi untuk memperluas konteks spasial dari fitur yang dihasilkan oleh lapisan sebelumnya tanpa mengubah ukuran resolusi. Proses pada modul SPPF dimulai dari hasil keluaran konvolusi yang kemudian diproses melalui tiga tahap *maxpooling* secara berurutan. Setiap tahap *pooling* mempertahankan ukuran spasial yang sama, namun mengekstraksi informasi dari area sekitar yang semakin luas. Setelah melalui tiga tahap *maxpooling*, seluruh hasil (termasuk fitur awal sebelum *maxpooling*) digabungkan (*concatenate*) pada dimensi kanal, lalu dilanjutkan dengan konvolusi akhir untuk menyesuaikan jumlah kanal keluaran.



Gambar 22. Alur Modul SPFF

a. Max Pooling

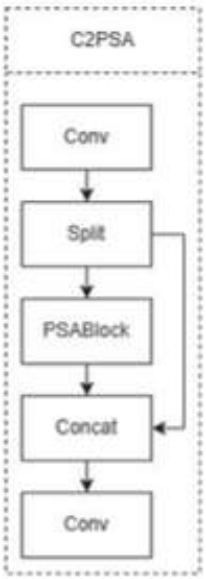
Proses *max pooling* bekerja dengan cara membagi *feature map* menjadi beberapa area kecil (disebut *jendela* atau *kernel*), kemudian mengambil nilai maksimum dari setiap area tersebut sebagai representasi fitur baru. Sebagai contoh, ilustrasi pada Gambar 23 memperlihatkan fitur masukan berukuran 4×4 yang diproses menggunakan kernel berukuran 2×2 dengan *stride* 2. Setiap blok 2×2 pada fitur masukan menghasilkan satu nilai tunggal, yaitu nilai tertinggi dari empat elemen di dalamnya. Misalnya, dari blok kiri atas berisi [1,2,5,6], nilai maksimum adalah 6; blok kanan atas [3,4,7,8], menghasilkan 8; dan seterusnya. Hasil akhirnya berupa fitur keluaran berukuran 2×2 dengan nilai [6,8; 14,16], yang masing-masing mewakili area dominan dari fitur masukan.



Gambar 23. Contoh Visualisasi Max Pooling

2.6.6 Modul C2PSA (Cross-Stage Partial Spatial Attention)

Modul C2PSA merupakan blok peningkatan representasi fitur yang mengombinasikan mekanisme *Cross-Stage Partial* (CSP) dengan *Position-Sensitive Spatial Attention* (PSA). Modul ini dirancang untuk memperkaya konteks spasial dan memperkuat hubungan antar piksel sehingga jaringan dapat secara adaptif menekankan region penting pada *feature map*. Pada tahap awal pemrosesan, fitur masukan dibagi menjadi dua jalur, di mana salah satu jalur dipertahankan sebagai *shortcut* untuk menjaga stabilitas gradien, sedangkan jalur lainnya diproses melalui mekanisme PSA yang terdiri atas komponen Attention dan Feed Forward Network (FFN); proses aliran data tersebut divisualisasikan secara terstruktur pada Gambar 24, yang menampilkan urutan pemisahan fitur, pemrosesan melalui blok PSA, hingga penggabungan akhir melalui operasi concatenation dan konvolusi untuk menghasilkan representasi fitur yang lebih informatif.



Gambar 24. Alur Modul C2PSA

a. PSABlock

PSABlock memproses *feature map* masukan melalui dua tahap utama, yaitu pemrosesan *Attention* dan *Feed Forward Network* (FFN), yang masing-masing dihubungkan menggunakan *residual connection*. Pada tahap pertama, fitur masukan Xdioperasikan melalui modul *Attention* untuk menghasilkan transformasi fitur, kemudian hasilnya digabungkan dengan masukan awal melalui *residual connection* sebagai berikut:

$$F_1 = X + \text{Attention}(X) \tag{19}$$

dimana:

X = merupakan fitur masukan

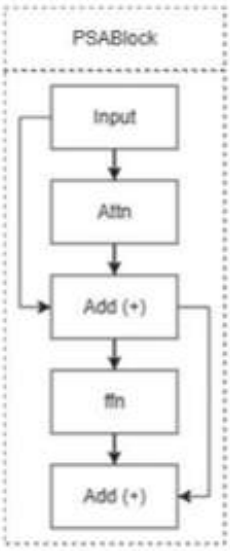
F_1 = fitur hasil integrasi antara masukan awal dan output attention.

Tahap berikutnya memproses F_1melalui FFN, kemudian hasilnya kembali dijumlahkan *dengan* fitur sebelumnya melalui residual connection kedua, yang dapat dituliskan sebagai:

$$F_{out} = F_1 + \text{FFN}(F_1) \tag{20}$$

dengan F_{out}merupakan keluaran akhir blok PSA.

Urutan langkah pemrosesan pada blok PSA, mulai dari input hingga output, divisualisasikan pada Gambar 25.



Gambar 25. Alur PSABlock

b. Attention

Proses Attention dimulai dengan mengoperasikan fitur masukan melalui lapisan konvolusi 1×1, yang menghasilkan representasi awal untuk pembentukan komponen query, key, dan value. Keluaran konvolusi tersebut kemudian diubah bentuknya (reshape) untuk menyesuaikan dimensi tensor, sebelum selanjutnya dipisahkan menjadi tiga bagian, yaitu Q, K, dan V. Pemisahan ini dilakukan berdasarkan proporsi dimensi

kanal yang telah ditentukan. Setelah diperoleh Q dan K, kedua komponen tersebut diproses melalui operasi perkalian matriks antara Q dan *transpose* dari K, yang menghasilkan skor keterkaitan antar elemen fitur. Perhitungan *skor attention* ini direpresentasikan secara matematis sebagai berikut:

$$S = (QK^T) \cdot \text{scale} \tag{21}$$

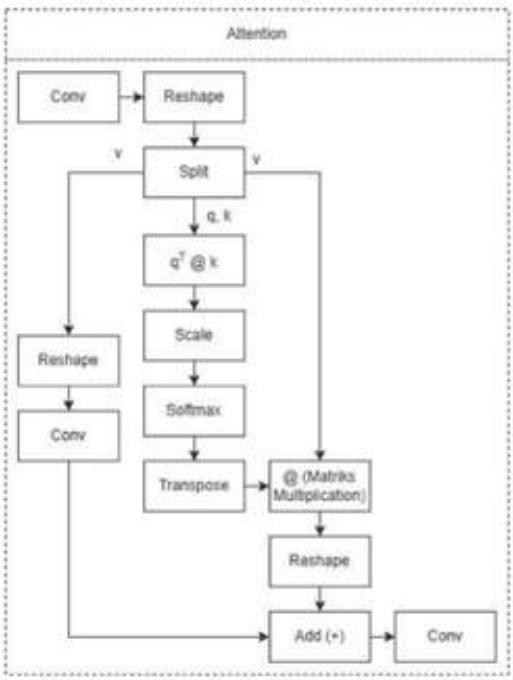
Nilai S kemudian dinormalisasi menggunakan fungsi *softmax* untuk menghasilkan matriks bobot *attention* A, sebagaimana dituliskan pada persamaan:

$$A = \text{Softmax}(S) \tag{22}$$

Matriks bobot A digunakan untuk memproses komponen value Vmelalui operasi perkalian matriks, sehingga menghasilkan keluaran fitur berbobot F_{att}, sesuai dengan persamaan berikut:

$$F_{att} = AV \tag{23}$$

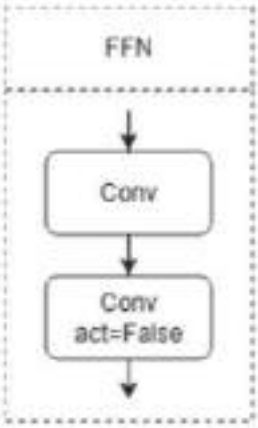
Keluaran F_{att} kemudian diubah kembali dimensinya (reshape) untuk menyesuaikan struktur *feature map* dua dimensi. Secara paralel, komponen Vjuga diubah bentuknya dan diproses melalui konvolusi depthwise 3×3 untuk menghasilkan positional encoding. Hasil positional encoding tersebut ditambahkan kembali ke fitur hasil attention melalui *residual addition*. Tahap akhir dari proses Attention menerapkan operasi konvolusi 1×1, yang berfungsi sebagai proyeksi linear untuk menghasilkan keluaran akhir dari blok *Attention*. Alur lengkap proses attention, mulai dari pembentukan Q, K, V, perhitungan *skor attention*, normalisasi *softmax*, hingga pembentukan keluaran akhir, ditunjukkan pada Gambar 26.



Gambar 26. Alur Proses Attention

c. FFN

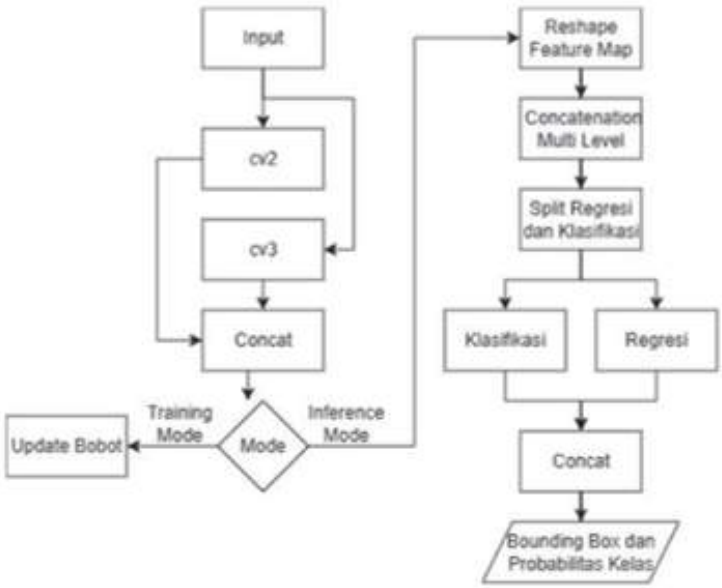
Feed Forward Network (FFN) memproses fitur masukan yang berasal dari keluaran *block Attention* melalui dua tahapan konvolusi berurutan. Proses dimulai dengan penerapan konvolusi 1×1 pertama yang bertujuan untuk mengubah dimensi kanal fitur dan menghasilkan representasi antara sebelum memasuki tahap berikutnya. Hasil keluaran tersebut kemudian diproses melalui konvolusi kedua berukuran 1×1, yang mengembalikan jumlah kanal ke dimensi semula. Pada konvolusi kedua ini tidak diterapkan fungsi aktivasi (act=*False*), sehingga menghasilkan transformasi linear murni sebagai keluaran FFN. Alur proses internal FFN ditunjukkan pada Gambar 27.



Gambar 27. Alur FFN

2.6.7 Modul Detect (Detection Head)

Modul *Detect* merupakan bagian akhir dari arsitektur YOLOv11 yang berfungsi menghasilkan prediksi objek berdasarkan *feature map* hasil pemrosesan *backbone* dan *neck*. Modul ini menerima tiga *feature map* multi skala (P3, P4, dan P5) dan memproses masing-masing level secara paralel untuk menghasilkan keluaran berupa regresi *bounding box* dan prediksi kelas objek. Secara umum, modul Detect bekerja dalam dua mode, yaitu *training mode* dan *inference mode*. Pada *training mode*, keluaran modul *Detect* digunakan secara langsung untuk perhitungan *loss function*, sedangkan pada *inference mode*, keluaran tersebut diproses lebih lanjut untuk menghasilkan koordinat *bounding box* dan probabilitas kelas yang merepresentasikan hasil prediksi objek. Alur kerja modul Detect secara keseluruhan ditunjukkan pada Gambar 28.



Gambar 28. Alur Modul Detect

a. Training Mode

Pada *training mode*, setiap *feature map* hasil keluaran *neck* diproses secara independen pada masing-masing level skala melalui dua jalur prediksi, yaitu jalur regresi *bounding box* (cv2) dan jalur klasifikasi (cv3). Kedua jalur menerima masukan *feature map* yang sama, kemudian menghasilkan keluaran regresi dan klasifikasi yang selanjutnya digabungkan melalui operasi *concatenation* pada dimensi kanal. Keluaran hasil *concatenation* pada setiap level *feature map* dipertahankan dalam bentuk terpisah tanpa penggabungan antar level. Seluruh keluaran tersebut kemudian diteruskan ke modul *loss* untuk dilakukan pencocokan antara prediksi dan *ground truth*, serta perhitungan nilai loss yang digunakan dalam proses *backpropagation* pada tahap pelatihan model.

b. Inference Mode

Pada *inference mode*, keluaran hasil *concatenation* dari setiap level *feature map* diproses lebih lanjut untuk menghasilkan prediksi objek akhir. Proses diawali dengan melakukan *reshape feature map* sehingga setiap lokasi spasial diperlakukan sebagai satu kandidat deteksi. Selanjutnya, seluruh kandidat deteksi dari berbagai level skala digabungkan melalui *concatenation* multi-level untuk membentuk satu representasi prediksi yang seragam. Representasi prediksi tersebut kemudian dipisahkan menjadi dua komponen, yaitu regresi dan klasifikasi. Pada komponen regresi dilakukan proses decoding untuk memperoleh koordinat *bounding box* yang merepresentasikan posisi objek pada citra masukan. Sementara itu, pada komponen klasifikasi dilakukan pemrosesan untuk menghasilkan probabilitas kelas objek. Hasil dari kedua komponen tersebut selanjutnya digabungkan kembali melalui operasi *concatenation*. Keluaran akhir dari *inference mode* pada modul Detect berupa koordinat *bounding box* dan probabilitas kelas yang merepresentasikan hasil prediksi objek.

2.7 Proses Pelatihan Model

Pada penelitian ini, proses pelatihan model dilakukan menggunakan arsitektur YOLOv11 serta versi yang telah ditingkatkan melalui integrasi modul *Deformable Convolutional Network* (DCN). Kedua model dilatih pada lingkungan komputasi yang sama, yaitu platform Kaggle dengan dukungan GPU P100, sehingga seluruh hasil eksperimen dapat dibandingkan secara objektif. Proses pelatihan ini menghasilkan model deteksi objek yang mampu mempelajari pola visual rambu lalu lintas pada kondisi malam hari, termasuk tantangan berupa noise, pencahayaan rendah, dan adanya motion blur.

Selama pelatihan, model dikonfigurasi menggunakan ukuran gambar tertentu yang telah disesuaikan dengan karakteristik dataset. Dalam penelitian ini digunakan resolusi input 640×640 piksel sesuai standar YOLO versi terbaru. Seluruh gambar diproses dalam kelompok kecil atau *batch size*, yang pada penelitian ini ditetapkan sebesar 16 untuk menyeimbangkan kebutuhan memori GPU dengan stabilitas pembelajaran. Setiap *batch* berisi sejumlah citra yang kemudian dipropagasikan melalui *backbone*, *neck*, dan *detection head* model, menghasilkan prediksi awal dalam bentuk *bounding box* serta probabilitas kelas.

Proses pelatihan berlangsung selama sejumlah *epoch*, yaitu satu putaran lengkap di mana seluruh dataset diperlihatkan kepada model. Penelitian ini menggunakan jumlah epoch yang sama untuk kedua model, sehingga tidak terdapat perbedaan durasi pelatihan yang dapat mempengaruhi hasil akhir. Selama setiap epoch, parameter model diperbarui melalui mekanisme *backpropagation* berdasarkan nilai *loss* yang dihitung dari selisih antara prediksi dan anotasi *ground truth* pada data validasi. YOLOv11 menggunakan beberapa komponen *loss*, termasuk loss regresi *bounding box*, loss klasifikasi, serta *Distribution Focal Loss* (DFL) yang digunakan untuk meningkatkan presisi regresi sisi *bounding box*. Ketiga loss tersebut bekerja secara bersamaan untuk mengoptimalkan kinerja model dalam tugas deteksi objek.

Selama pelatihan berlangsung, performa model dipantau melalui metrik evaluasi seperti *mean Average Precision* (mAP), precision, dan *recall*. Grafik metrik pelatihan dan validasi digunakan untuk memastikan bahwa model mengalami konvergensi dengan baik dan tidak menunjukkan gejala *overfitting*. Pemantauan yang konsisten ini penting mengingat kedua model YOLOv11 *baseline* dan YOLOv11 yang terintegrasi dengan DCN perlu melalui proses pelatihan yang identik agar hasil perbandingan kinerjanya relevan dan dapat dipertanggungjawabkan secara ilmiah.

Setelah seluruh proses pelatihan selesai, model disimpan dalam format yang digunakan oleh Ultralytics, yaitu file berformat .pt. File ini memuat bobot model, konfigurasi arsitektur, serta seluruh parameter hasil pembelajaran sehingga dapat digunakan kembali untuk proses pengujian, inferensi, ataupun pelatihan lanjutan. Dengan struktur penyimpanan tersebut, model dapat dijalankan kembali pada lingkungan apa pun yang mendukung PyTorch tanpa memerlukan konfigurasi ulang dari awal.

Secara keseluruhan, proses pelatihan model dilakukan secara sistematis dan konsisten, mulai dari pre-processing data, penentuan hyperparameter, perulangan epoch, hingga penyimpanan model. Dengan kesetaraan lingkungan pelatihan untuk YOLOv11 dan YOLOv11-DCN, hasil yang diperoleh pada tahap evaluasi mencerminkan

pengaruh arsitektur model secara murni, bukan pengaruh lingkungan atau perbedaan konfigurasi teknis.

2.8 Evaluasi dan Metrik Penilaian

Evaluasi dilakukan untuk menilai kinerja model dalam mendeteksi dan mengklasifikasikan rambu lalu lintas, di mana proses pengujian ini memastikan kemampuan model terhadap kondisi nyata yang bervariasi. Untuk merepresentasikan situasi di jalanraya, skenario pengujian dilakukan pada rentang kecepatan berkendara 30 km/jam hingga 50 km/jam, sehingga performa model dapat dinilai secara lebih nyata terhadap perubahan kondisi visual yang muncul akibat pergerakan kendaraan. Evaluasi ini menggunakan metrik deteksi objek, yaitu *precision* dan *recall*, yang menilai seberapa tepat model dalam mengklasifikasikan objek. Kinerja model selanjutnya diukur menggunakan *mean Average Precision* (mAP), baik pada ambang IoU 0.50 (mAP@50) maupun rentang nilai IoU 0.50 hingga 0.95 (mAP@50-95), sehingga memberikan gambaran terhadap kualitas deteksi objek.

Evaluasi selanjutnya adalah mengevaluasi kinerja model berdasarkan kualitas citra. Citra uji terlebih dahulu dievaluasi menggunakan *Perception based Image Quality Evaluator* (PIQE) untuk memperoleh skor kualitas citra. Skor PIQE tersebut kemudian digunakan untuk mengelompokkan citra ke dalam lima kategori kualitas, yaitu sangat baik dengan rentang skor 0-20, baik dengan rentang skor 21-40, sedang dengan rentang skor 41-60, buruk dengan rentang skor 61-80, dan sangat buruk dengan rentang skor 81-100. Setelah proses pengelompokan dilakukan, performa model pada setiap kategori dievaluasi menggunakan *precision*, *recall*, *mAP@50*, dan *mAP@50-95*. Evaluasi ini bertujuan untuk melihat kestabilan performa model terhadap variasi kualitas citra, khususnya pada citra malam hari yang rentan mengalami *noise*, *blur*, dan pencahayaan yang bervariasi.

Selain evaluasi berbasis performa deteksi dan klasifikasi, penelitian ini juga mengukur efisiensi komputasi model melalui *Inference time*, *Inference Throughput*, *Parameter Count*, dan *Floating Point Operations* (FLOPs). Keempat indikator ini digunakan untuk menilai kebutuhan komputasi serta kelayakan model untuk diterapkan pada skenario dunia nyata yang memerlukan proses deteksi yang cepat dan ringan secara komputasional. Melalui kombinasi metrik evaluasi tersebut, kinerja YOLOv11 dan YOLOv11 dengan modul *Deformable Convolutional Network* (DCN) dapat dibandingkan secara objektif.