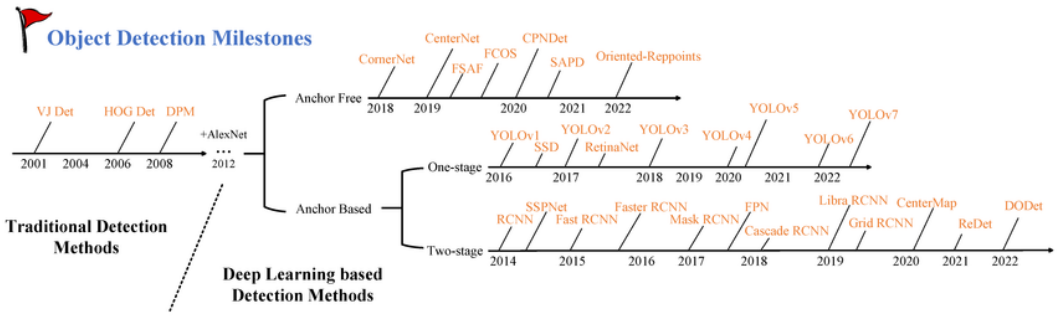
BAB II

LANDASAN TEORI

2.1 Deteksi Objek (*Object Detection*)

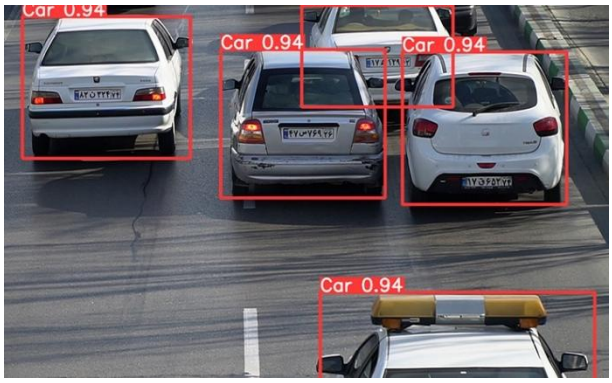
Object detection, merupakan sebuah landasan dari *computer vision* yang berfungsi untuk mengidentifikasi contoh-contoh objek dari kelas tertentu (misalnya, manusia, bangunan, atau mobil) dalam gambar dan video digital (Zhao dkk., 2019). *Object detection* memiliki peran dalam berbagai aplikasi, termasuk *autonomous driving*, *surveillance systems*, dan robotika. Kemampuan utama dari *object detection* adalah memberikan wawasan dan pemahaman bagi komputer mengenai gambar visual. Pada Gambar 2.1 direpresentasikan awal mula evolusi algoritma deteksi objek hingga model-model terbaru deteksi objek.



Gambar 2.1 Evolusi dari Object Detection (X. Wang dkk., 2023)

Gambar 2.1 menunjukkan munculnya *Neural Network* khususnya *convolutional neural network* (CNN) sangat berpengaruh terhadap landasan dasar untuk model-model deteksi objek berikutnya seperti YOLO, RCNN bahkan Fast RCNN. *Neural network* mengekstrak fitur unik dan khas dari data gambar yang diberikan sehingga mampu mendeteksi objek dalam berbagai skala, orientasi, dan

kontras dengan benar (Purwono dkk., 2022). Proses deteksi objek secara langsung ditunjukkan pada Gambar 2.2.



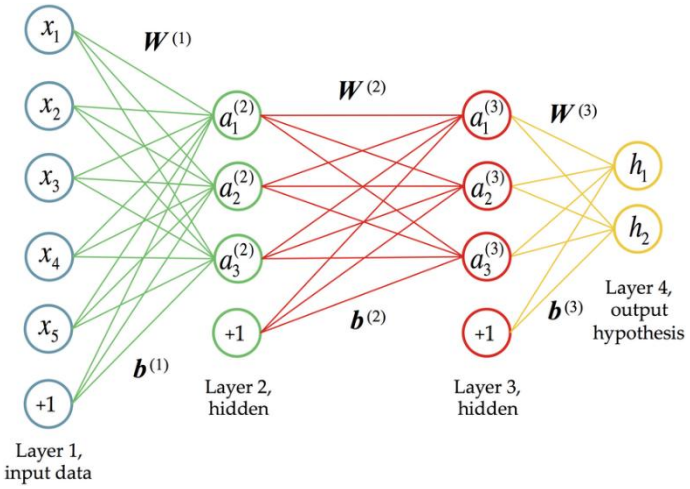
Gambar 2.2 Deteksi Kendaraan Secara Real-Time

*Moving object detectio*n berfokus pada identifikasi dan pelacakan objek yang bergerak dalam sebuah urutan video. Hal ini memberikan tantangan yang kompleks karena sifat dinamis dari video dan kebutuhan akan pemrosesan data secara *real-time* (Al-Dweik dkk., 2023; Cypto dkk., 2022; Wojke dkk., 2017). Mendeteksi dan melacak objek bergerak secara akurat sangat penting untuk aplikasi seperti pengawasan video, pemantauan lalu lintas, ban konveyor pabrik, dan pengenalan gerakan tubuh.

2.2 Neural Network

Neural network (NN) adalah model komputasi yang terinspirasi oleh jaringan saraf biologis yang ditemukan di otak manusia. Jaringan ini terdiri dari simpul-simpul yang saling berhubungan atau “neuron” yang disusun secara berlapis-lapis (Blalock dkk., 2020; Isaac Abiodun dkk., 2018). Neuron-neuron ini memproses dan mengirimkan informasi melalui koneksi berbobot. NN belajar dari data dengan menyesuaikan bobot koneksi antar neuron. Koneksi neuron yang berlapis ini *neural network* ditunjukkan pada Gambar 2.3 berguna untuk mengenali pola, membuat

prediksi, dan melakukan tugas-tugas yang kompleks (Doan Van, 2023; Isaac Abiodun dkk., 2018).



Gambar 2.3 Lapisan didalam *neural network* (Sheehan & Song, 2016)

Neural Network ini biasanya terdiri dari tiga layer atau lapisan yaitu:

1. *Input Layer* (Lapisan Input): *Input Layer* berfungsi sebagai titik masuk data ke dalam *neural network*. Setiap node di lapisan ini berhubungan dengan fitur atau atribut dari data *input*. Fungsi utama dari *Input Layer* adalah untuk menerima dan merepresentasikan data mentah dalam format yang dapat diproses oleh lapisan berikutnya (Blalock dkk., 2020; Isaac Abiodun dkk., 2018; Purwono dkk., 2022).
2. *Hidden Layer* (Lapisan Tersembunyi): *Hidden Layer* adalah lapisan perantara antara lapisan *input* dan *output*. Lapisan ini bertanggung jawab untuk melakukan komputasi dan transformasi yang kompleks pada data *input*. Setiap lapisan tersembunyi terdiri dari beberapa *node* (neuron) yang saling terhubung dengan koneksi berbobot (*weighted connections*) (Blalock dkk., 2020; Isaac Abiodun dkk., 2018; Purwono dkk., 2022).. Jumlah *Hidden Layer* dan *node* di

setiap lapisan dapat bervariasi tergantung pada kompleksitas masalah yang sedang dipecahkan. *Hidden Layer* belajar untuk mengekstrak fitur dan pola yang relevan dari data *input*, memungkinkan jaringan untuk membuat prediksi atau klasifikasi yang akurat.

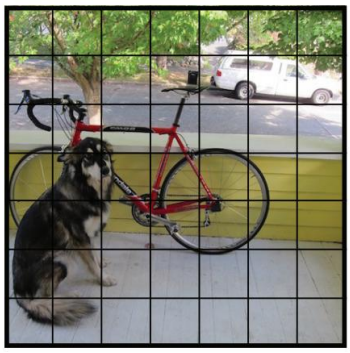
3. *Output Layer* (Lapisan Keluaran): *Output layer* adalah lapisan akhir dari jaringan saraf. Lapisan ini menghasilkan prediksi atau klasifikasi jaringan berdasarkan informasi yang diproses dari *hidden layer* (Blalock dkk., 2020; Isaac Abiodun dkk., 2018). Jumlah *node* di *output* layer tergantung pada sifat tugas. Sebagai contoh, dalam masalah klasifikasi biner *output* layer mungkin memiliki satu *node* yang mewakili dua kelas yang memungkinkan. Sementara dalam masalah klasifikasi multi-kelas, lapisan ini mungkin memiliki beberapa *node* yang mewakili setiap kelas yang berbeda atau unik.

Interaksi antar lapisan-lapisan ini memungkinkan *neural network* untuk mempelajari hubungan yang kompleks antara data *input* dan *output*. *Input* layer menerima data mentah, *hidden layer* mengekstrak representasi yang berarti, dan *output layer* menghasilkan prediksi akhir. Kekuatan koneksi antar *node*, yang diwakili oleh bobot, disesuaikan selama proses pelatihan untuk mengoptimalkan kinerja jaringan (Isaac Abiodun dkk., 2018).

2.3 YOLO

YOLO (*You Only Look Once*) adalah metode pendeteksian objek yang bekerja dengan cara membagi sebuah Gambar ke dalam kisi-kisi yang terdiri dari bagian-bagian kecil berdimensi S×S. Setiap bagian bertanggung jawab untuk

mengidentifikasi objek apa pun yang berada di bagian tengahnya (Bochkovskiy dkk., 2020). Ilustrasi pembagian kisi pada Gambar ditunjukkan Gambar 2.4.

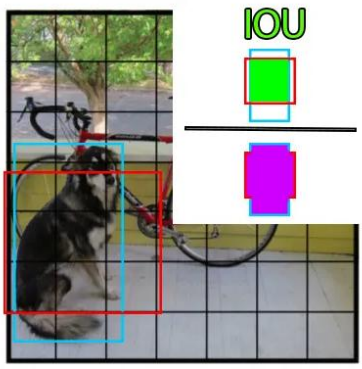


Gambar 2.4 Gambar input yang dibagi ke dalam kisi-kisi (Redmon dkk., 2016)

Pada setiap bagian, model memprediksi sekumpulan kotak pembatas atau *bounding box* (biasanya dua kotak secara *default*) yang berpotensi melingkupi sebuah objek. Setiap kotak dilengkapi dengan nilai kepercayaan (*classification score*) mulai dari 0 (tidak ada objek) hingga 1 (kepastian penuh), yang menunjukkan seberapa yakin model tersebut bahwa kotak tersebut berisi objek.

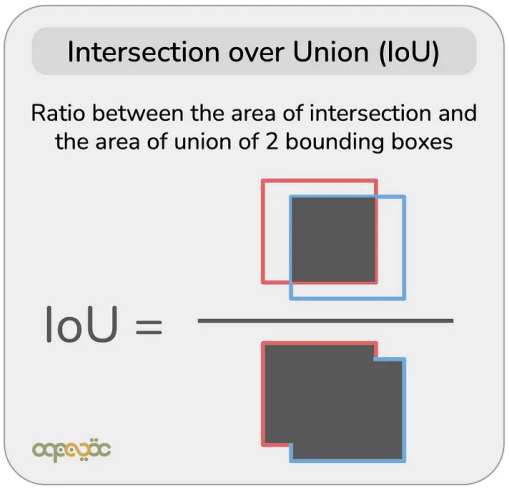
Setiap kotak pembatas ditentukan oleh lima variabel:

1. Posisi x adalah lokasi horizontal pusat kotak, relatif terhadap seluruh Gambar.
2. Posisi y adalah lokasi vertikal pusat kotak, relatif terhadap seluruh Gambar.
3. *Width* merupakan lebar kotak, relatif terhadap seluruh Gambar.
4. *Height* merupakan tinggi kotak, relatif terhadap seluruh Gambar.
5. *Confidence* merupakan ukuran seberapa baik kotak yang diprediksi tumpang tindih dengan objek yang sebenarnya atau dikenal sebagai kotak *ground truth box*. Hal ini dihitung dengan menggunakan metrik *Intersection Over Union* (IOU), yang membandingkan area tumpang tindih antara dua kotak dengan area gabungannya seperti yang ditunjukkan pada Gambar 2.5.



Gambar 2.5 Contoh IOU dalam deteksi anjing (Redmon et al., 2016)

Gambar 2.5 adalah contoh dari IOU: area perpotongan (*intesection*) antara *ground truth box* dan kotak prediksi dalam area warna hijau dibagi dengan area penyatuan kedua kotak, dalam warna ungu . Nilai IOU akan berada di antara 0 dan 1, 0 jika keduanya tidak tumpang tindih sempurna dan 1 jika keduanya merupakan kotak yang sama. IOU yang lebih tinggi lebih baik karena merupakan prediksi yang lebih akurat. Persamaan dalam mencari nilai IOU ditunjukkan pada Gambar 2.6.

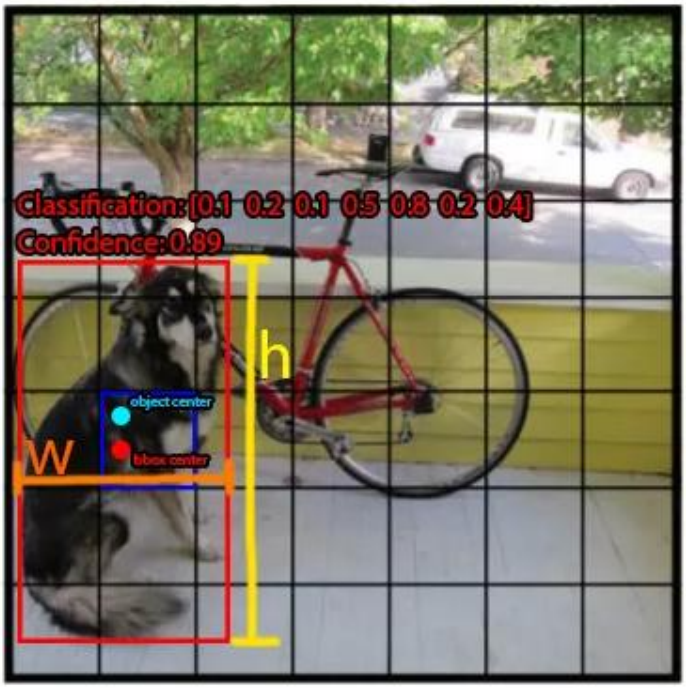


Gambar 2.6 Persamaan IoU (Anwar, 2022)

Gambar 2.6 menjelaskan persamaan nilai *Intersection over Union* (IoU), yang dihitung sebagai rasio antara area perpotongan (*intersection*) antara *ground truth box* dan kotak prediksi, dibagi dengan area gabungan (*union*) dari kedua kotak

tersebut (Derrenger dkk., 2024; Gillani dkk., 2022; Jocher & Bình, 2023). Selain memprediksi lokasi dan keyakinan objek dalam setiap *grid cell*, YOLO juga mencoba mengklasifikasikan jenis objek. Klasifikasi ini direpresentasikan sebagai sebuah *list* angka. Setiap angka berhubungan dengan kelas yang berbeda di dalam *dataset*. *List* ini memiliki nilai *value* 1 untuk kelas yang diprediksi dan 0 untuk yang lainnya. Setiap *grid cell* hanya dapat memprediksi satu kelas, meskipun terdapat beberapa objek dari kelas yang berbeda. Keterbatasan ini dapat menyebabkan algoritma salah mengklasifikasikan objek ketika sebuah sel berisi lebih dari satu jenis objek (Derrenger dkk., 2024; Jocher & Bình, 2023).

Karena hal tersebut, *output* untuk setiap *grid cell* menggabungkan prediksi kelas dan informasi kotak pembatas. Jika kita memiliki 'C' kelas dan memprediksi 'B' *bounding box* per sel. Selanjutnya, B dikalikan dengan 5 untuk memperhitungkan 5 nilai yang mengGambarkan setiap kotak (x, y, lebar, tinggi, dan keyakinan). Karena Gambar dibagi menjadi S x S sel, *output* keseluruhan model adalah *array* 3 dimensi (atau disebut tensor) dengan dimensi S x S x (C + B * 5) tensor. Tensor ini yang disimpan sebagai prediksi untuk semua sel pada Gambar (Derrenger dkk., 2024; Jocher & Bình, 2023). Ilustasi dari *output* model ditunjukkan pada Gambar 2.7.



Gambar 2.7 Ilustrasi *Output* Model dari Deteksi Anjing (Redmon dkk., 2016)

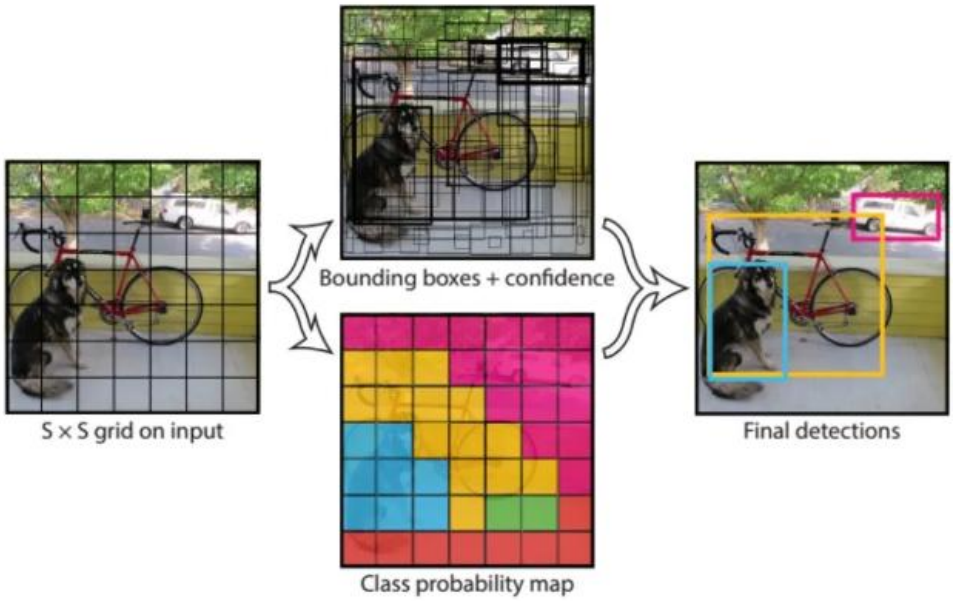
Gambar 2.7 mengilustrasikan keluaran model menggunakan skenario yang disederhanakan di mana hanya satu *bounding box* yang diprediksi per sel. Pada Gambar yang diberikan, pusat anjing yang sebenarnya ditandai dengan lingkaran biru muda berlabel "pusat objek". *Grid cell* yang bersesuaian, disorot dengan warna biru tua, bertanggung jawab untuk mendeteksi anjing dan menghasilkan *bounding box*.

Bounding box yang diprediksi ditentukan oleh empat elemen:

1. Pusat (x, y) direpresentasikan oleh titik merah, yang menunjukkan titik pusat dari *bounding box* yang diprediksi. Perhatikan bahwa model memprediksi pusat dan dimensi kotak, bukan koordinat sudutnya.
2. Lebar (w) direpresentasikan oleh penanda oranye yang menunjukkan lebar kotak relatif terhadap ukuran Gambar.

3. Tinggi (h) direpresentasikan oleh penanda kuning menunjukkan tinggi kotak relatif terhadap ukuran Gambar.

Selanjutnya, model menghasilkan prediksi klasifikasi untuk objek yang terdeteksi. Prediksi ini dikodekan sebagai *one-hot vector* yang memproses data kategorikal menjadi data *vector* numerik yang bisa dipahami komputer. Dalam Gambar 2.7 terdapat tujuh kelas yang potensial, dan model mengklasifikasikan objek ke kelas ke-5. Hal ini terlihat dari *one-hot vector* dengan 7 elemen di mana elemen ke-5 mendapatkan nilai 0,8 atau mendekati nilai 1. Hal ini menandakan bahwa model tersebut sangat yakin bahwa objek tersebut masuk ke dalam kelas ke-5 yang dalam contoh ini berupa anjing. Proses deteksi secara beberapa objek secara bersamaan ditunjukkan pada Gambar 2.8.



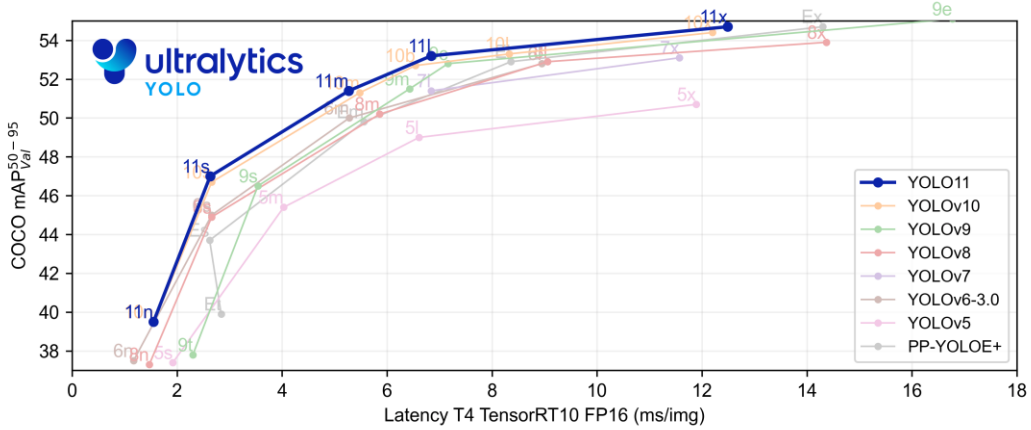
Gambar 2.8 Proses klasifikasi objek oleh YOLO (Redmon dkk., 2016)

Gambar 2.8 menunjukkan proses menyeluruh dari deteksi dan klasifikasi semua kotak pembatas dan prediksi kelas objek yang sebenarnya akan dibuat dan hasil akhirnya. Perlu diingat bahwa ini hanyalah sebuah contoh untuk menunjukkan

jenis keluaran yang mungkin dihasilkan, sehingga nilainya mungkin tidak akurat dengan nilai sebenarnya.

2.3.1 YOLOv11

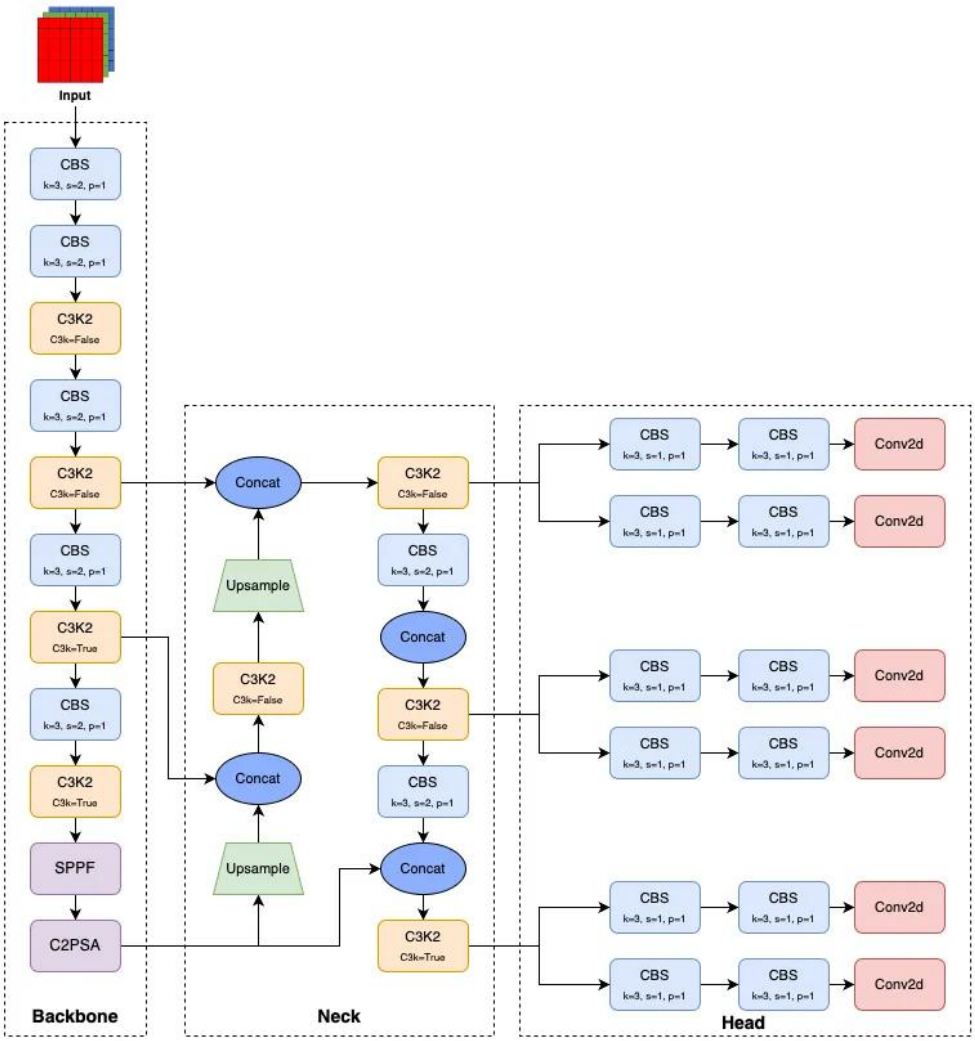
YOLOv11 adalah versi terbaru dari seri YOLO (*You Only Look Once*) yang dikembangkan oleh Ultralytics dan dirilis pada tanggal 30 September 2024 (Derrenger dkk., 2024; Sapkota dkk., 2024). Model ini dirancang untuk memberikan kinerja deteksi objek *real-time* yang lebih baik dengan menyempurnakan model dari YOLO generasi-generasi sebelumnya. YOLOv11 memperkenalkan beberapa peningkatan dalam arsitektur dan metode pelatihan, yang memungkinkan model ini mencapai akurasi yang lebih tinggi dengan parameter yang lebih sedikit. Parameter yang lebih sedikit tentunya membuat *interface* FPS lebih rendah (Derrenger dkk., 2024; Sapkota dkk., 2024). Perbandingan performa antara model YOLO ditunjukkan pada Gambar 2.9.



Gambar 2.9 Perbandingan Performa antar Model YOLO (Derrenger dkk., 2024)

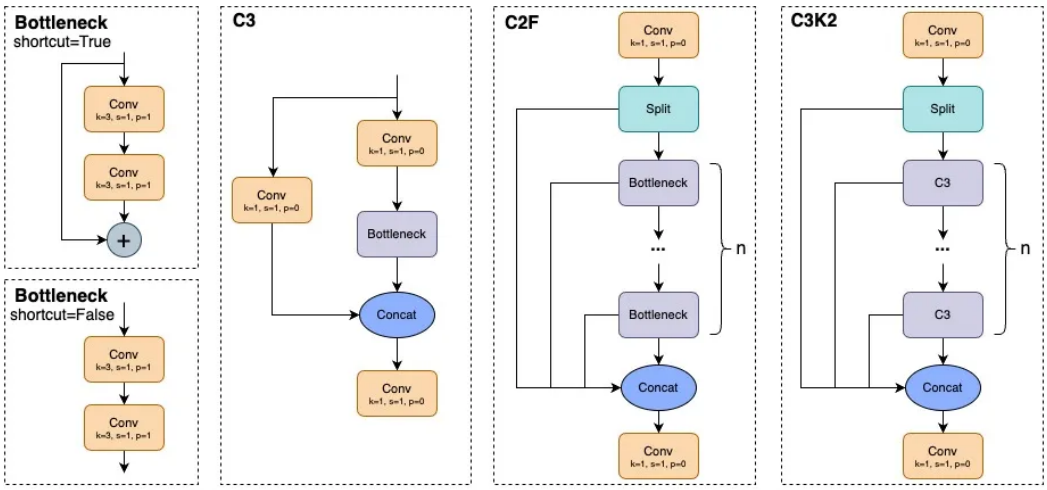
YOLO11 mencapai akurasi yang lebih besar dengan parameter yang lebih sedikit melalui perubahan dalam desain model dan teknik optimasi. Arsitektur yang ditingkatkan memungkinkan ekstraksi dan pemrosesan fitur yang lebih efisien,

menghasilkan rata-rata *Average Precision* (mAP) yang lebih tinggi pada *dataset* seperti COCO dengan menggunakan parameter 22% lebih sedikit daripada YOLOv8m. Hal ini membuat YOLO11 lebih efisien secara komputasi tanpa mengorbankan akurasi, sehingga lebih cocok untuk digunakan pada perangkat yang memiliki sumber daya terbatas (Derrenger dkk., 2024). Arsitektur YOLOv11 ditunjukkan pada Gambar 2.10.



Gambar 2.10 Arsitektur YOLOv11(Tsoi, 2024)

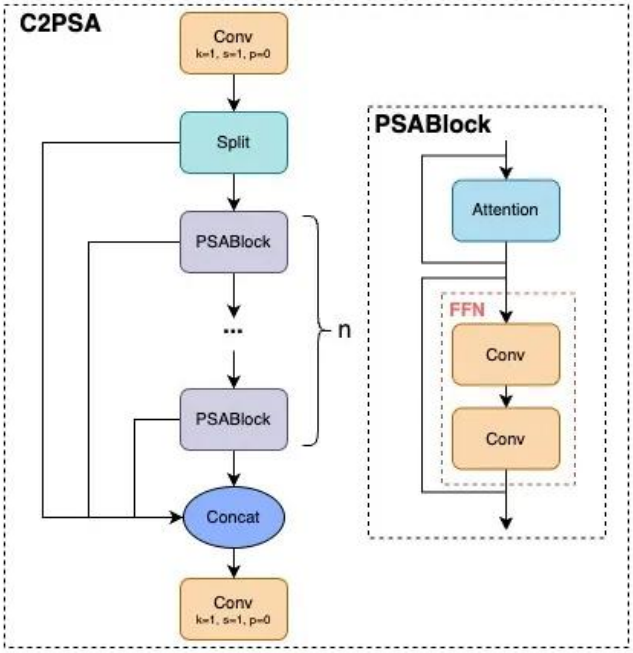
Gambar 2.10 menunjukkan arsitektur YOLOv11 yang terdiri dari tiga komponen utama yaitu *Backbone*, *Neck*, dan *Head*. YOLOv11 menghadirkan sejumlah inovasi dalam arsitekturnya untuk meningkatkan efisiensi dan akurasi deteksi objek. Modul CSP (*Cross Stage Partial*) yang menjadi modul utama pada YOLOv5 disempurnakan menjadi modul C2F (Modul CSP dengan 2 Layer Konvulsi) pada YOLOv8 (Gillani dkk., 2022; Jocher & Bình, 2023). Modul C2F disempurnakan lagi menjadi modul C3K2 (Modul CSP dengan 3 layer konvulsi dan *Kernel* 2x2) pada YOLOv11 (Gillani dkk., 2022; Sapkota dkk., 2024). Detail arsitektur dari modul C3, C2F, C3K2 ditunjukkan pada Gambar 2.11.



Gambar 2.11 Arsitektur Modul C3, C2F dan C3K2 (Tsoi, 2024)

Pada bagian *backbone*, model ini menggunakan kombinasi modul CBS (Modul CSP dengan 1 layer konvulsi, *batch normalization*, dan aktivasi SiLU (*Sigmoid Linear Unit*)) dan C3K2 secara berulang yang berperan dalam mengekstraksi fitur dari tingkat rendah hingga tinggi (Tsoi, 2024). Struktur ini dilengkapi dengan Spatial Pyramid Pooling - Fast (SPPF) dan C2PSA pada akhir struktur backbone agar didapat pemrosesan yang lebih optimal (Derrenger dkk., 2024; Tsoi, 2024).

Sementara itu, bagian *neck* menggabungkan fitur yang diekstrak oleh *backbone* menggunakan modul C2PSA dan modul C3K2 yang merupakan pengembangan dari modul C2F . Modul C2PSA merupakan modul dengan 2 blok konvolusi serta mekanisme perhatian spasial (*spatial attention*) untuk meningkatkan efisiensi ekstraksi dan pemrosesan fitur. Modul ini mencakup lapisan konvolusi dan serangkaian blok PSA (Positional Self-Attention) (Derrenger dkk., 2024; Tsoi, 2024). Detail arsitektur dari modul C2PSA ditunjukkan pada Gambar 2.12.



Gambar 2.12 Arsitektur Modul C2PSA (Tsoi, 2024)

Fitur-fitur utama dari model YOLOv11 diantara lain:

1. Ekstraksi Fitur yang Ditingkatkan: YOLO11 menggunakan arsitektur *backbone* dan *neck* yang ditingkatkan. Hal ini meningkatkan kemampuan ekstraksi fitur untuk deteksi objek yang lebih tepat dan kinerja tugas yang kompleks (Derrenger dkk., 2024).

2. Optimal untuk Efisiensi dan Kecepatan: YOLO11 memperkenalkan desain arsitektur yang disempurnakan dan pelatihan yang dioptimalkan, menghasilkan kecepatan pemrosesan yang lebih cepat dan menjaga keseimbangan optimal antara akurasi dan kinerja (Derrenger dkk., 2024).
3. Akurasi yang lebih baik dengan parameter yang lebih sedikit: YOLO11m mencapai rata-rata *Average Precision* (mAP) yang lebih tinggi pada *dataset* COCO dengan menggunakan 22% lebih sedikit parameter daripada YOLOv8m, menjadikannya efisien secara komputasi tanpa mengorbankan akurasi (Derrenger dkk., 2024).
4. Kemampuan beradaptasi di berbagai *environment*: YOLO11 dapat digunakan dengan baik di berbagai *environment*, termasuk perangkat *edge*, *platform cloud*, dan sistem yang mendukung GPU NVIDIA atau perangkat *mobile* (Derrenger dkk., 2024).

Studi komparasi model YOLO oleh (Sapkota dkk., 2024) menunjukkan peningkatan performa dibandingkan model sebelumnya, seperti YOLOv8 hingga YOLOv10. Nilai *mean Average Precision* (mAP) dan kecepatan pemrosesan YOLOv11s mencapai mAP50 sebesar 0,933, mendekati performa terbaik YOLOv9 Gelan-e dengan 0,935. Selain itu, YOLOv11n mencatatkan waktu *inference* tercepat yaitu 2,4 ms. Waktu ini lebih cepat dibandingkan YOLOv10 (5,5 ms) dan YOLOv8 (4,1 ms). Hal ini menjadikan YOLOv11 ideal untuk aplikasi *real-time* di perangkat *mobile* dan sistem otomatisasi yang membutuhkan kecepatan tinggi. Model ini juga menunjukkan fleksibilitas dalam berbagai kondisi lingkungan dan dapat diimplementasikan di perangkat *edge*, *cloud*, maupun GPU, sehingga cocok

untuk mendukung otomatisasi di berbagai sektor, termasuk pertanian dan industri (Sapkota dkk., 2024).

2.4 Metrik Kinerja (Performance metrics)

Metrik kinerja selama pelatihan model YOLO mencakup indikator utama seperti presisi, *recall*, mAP (*mean Average Precision*), dan berbagai nilai *loss* (misalnya, *classification loss*, *bounding box loss*). Variabel ini menjadi digunakan untuk mengukur seberapa baik model dalam mendeteksi objek dalam Gambar.

2.4.1 Precision

Precision adalah ukuran yang menunjukkan seberapa banyak dari hasil prediksi positif model yang benar-benar positif. Fungsi utama dari metrik *Precision* digunakan untuk meminimalkan jumlah *false positives* (kesalahan ketika model memprediksi positif tetapi sebenarnya negatif) (Padilla dkk., 2020).

Persamaan 1 merupakan persamaan matematis dari *precision* (Derrenger dkk., 2024; Jocher dkk., 2023; Jocher & Bình, 2023; Sapkota dkk., 2024):

$$Precision = \frac{TP}{TP + FP} \quad (1)$$

di mana 'TP' atau *True Positive* adalah jumlah prediksi positif yang benar, dan 'FP' atau *False Positive* adalah data yang seharusnya negatif/salah diklasifikasikan sebagai positif.

2.4.2 Recall

Recall adalah metrik yang menunjukkan seberapa banyak data positif yang benar-benar terdeteksi oleh model. Metrik ini berguna ketika kita ingin meminimalkan *false negatives* (kesalahan ketika model memprediksi negatif padahal sebenarnya positif) (Ramos dkk., 2024b).

Persamaan 2 merupakan persamaan matematis dari *Recall* (Derrenger dkk., 2024; Jocher dkk., 2023; Jocher & Bình, 2023; Sapkota dkk., 2024):

$$Recall = \frac{TP}{TP + FN} \quad (2)$$

di mana 'TP' atau *True Positive* adalah jumlah prediksi positif yang benar, dan 'FN' atau *False Negative* adalah data yang seharusnya positif/benar diklasifikasikan sebagai negatif.

2.4.3 F1-Score

F1-Score adalah metrik gabungan yang memperhitungkan keseimbangan antara *precision* dan *recall*. Metrik ini adalah rata-rata dari nilai *precision* dan *recall*, sehingga *F1 Score* memberikan Gambaran yang lebih menyeluruh tentang performa model, terutama saat *precision* dan *recall* memiliki nilai yang sangat berbeda. Nilai F1-score yang tinggi menunjukkan bahwa model memiliki baik *precision* maupun *recall* yang tinggi.(Ramos dkk., 2024b).

Persamaan 3 merupakan persamaan matematis dari F1-Score (Gillani dkk., 2022; Sapkota dkk., 2024) adalah:

$$F1 - Score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \quad (3)$$

dimana, *Precision* melambangkan proporsi *true positive* relatif terhadap jumlah gabungan dari *true positive* dan *false positive*, sedangkan *Recall* mewakili proporsi *true positive* relatif terhadap jumlah *true positive* dan *false negative*.

2.4.4 *Mean Average Precision* (mAP)

Mean Average Precision (mAP) merupakan metrik yang digunakan dalam evaluasi sistem pendeteksian objek. mAP mencakup *precision* maupun *recall*

dengan menghitung *Average Precision* (AP) di seluruh kelas (Ramos dkk., 2024b). mAP adalah metrik yang informatif karena menginterpretasikan keseimbangan antara kedua aspek tersebut dengan mengukur area di bawah kurva *precision-recall* (Padilla dkk., 2020).

Persamaan 4 merupakan persamaan matematis dari AP (Derrenger dkk., 2024; Jocher dkk., 2023; Jocher & Bình, 2023; Sapkota dkk., 2024):

$$AP = \int_0^1 Precision(r), dr \tag{4}$$

Representasi matematis dari AP, seperti dijelaskan pada Persamaan 4, menggambarkan integral dari *precision* terhadap tingkat *recall*, yang dilambangkan sebagai *precision*(r) di mana mewakili tingkat *recall* yang spesifik. Setelah itu nilai AP digunakan untuk menghitung mAP dalam persamaan 5.

Persamaan 5 merupakan persamaan matematis dari mAP (Derrenger dkk., 2024; Jocher dkk., 2023; Jocher & Bình, 2023; Sapkota dkk., 2024):

$$mAP = \frac{\sum_{i=1}^{k} AP_i}{k} \tag{5}$$

dimana, semua nilai AP bagi masing-masing kelas dijumlahkan kemudian dibagi dengan total jumlah kelas (k).

Nilai mAP yang lebih tinggi menunjukkan performa yang lebih baik dalam pendeteksian objek. Dimana nilai *precision* dan *recall* harmonis atau seimbang. Selain itu, mAP *memiliki* variasi yang disesuaikan dengan berbagai ambang batas *Intersection over* Union (IoU). mAP@0.5 berfokus pada ambang batas IoU sebesar 0,5, sedangkan mAP@0.50-0.95 mencakup nilai rata-rata *precision* di berbagai ambang batas IoU mulai dari 0,5 hingga 0,95 (Zoph dkk., 2020).

2.4.5 *Fitness Score*

Fitness score merupakan nilai yang menunjukkan kemampuan model secara keseluruhan dengan menggunakan nilai mAP50 dan mAP50-95 sebagai nilai utamanya (Derrenger dkk., 2024; Jocher dkk., 2023; Jocher & Bình, 2023; Sapkota dkk., 2024). Nilai ini digunakan pada proses *tuning* menggunakan Algoritma Genetika untuk menentukan *parent* terbaik (Gen, 2006; Safarik dkk., 2018; Santoso dkk., 2023). Pada penelitian ini, formula *fitness score* disempurnakan dengan penambahan metrik F1-score pada kalkulasinya. Hal ini bertujuan agar keseimbangan *precision* (klasifikasi) dan *recall* (keseimbangan) model masuk ke dalam penilaian performa model (Derrenger dkk., 2024; Jocher & Bình, 2023). Oleh karena itu, metrik ini juga akan digunakan sebagai metrik evaluasi dan perbandingan performa model akhir.

Persamaan 6 merupakan persamaan matematis dari *fitness score* (Derrenger dkk., 2024; Jocher dkk., 2023; Jocher & Bình, 2023) dengan penambahan variabel F1-Score:

$$Fitness = (0.2 \times F1) + (0.1 \times mAP50) + (0.7 \times mAP50{:}95) \quad (6)$$

di mana, persentase perbandingan pengaruh antar nilai F1-Score, mAP50 dan mAP50-95 yaitu 20:10:70. Semakin dekat nilai Fitness dengan 1, semakin baik pula performa deteksi, klasifikasi dan keseimbangan keduanya dari model tersebut.

2.4.6 *Frame Rate per Second* (FPS)

Kinerja kecepatan pemrosesan dihitung menggunakan frame per detik (FPS)

Persamaan 7 merupakan persamaan matematis dari FPS (Liberatori dkk., 2022):

$$FPS = \frac{Total\ Jumlah\ Frame}{Total\ Waktu\ Deteksi(s)} \quad (7)$$

Secara umum, semakin tinggi FPS semakin cepat pula model dalam mendeteksi objek. Selain itu, nilai FPS yang tinggi membuat model dapat mendeteksi objek bergerak lebih baik karena objek dapat segera diproses outputnya oleh model (Huang dkk., 2018; Suhaimin dkk., 2021; H. Yang & Han, 2020).

2.5 Hyperparameter Tuning

Hyperparameter tuning mengacu pada proses pemilihan parameter terbaik untuk model agar didapatkan kinerja yang paling efisien. Proses ini melibatkan pengujian sistematis dari *hyperparameter* yang merupakan pengaturan yang mengontrol proses pelatihan model tidak dipelajari secara langsung dari data. *Hyperparameter* memiliki dampak yang signifikan terhadap akurasi model, stabilitas pelatihan, waktu komputasi, dan waktu respon *interface* (Dalal dkk., 2024; Javed dkk., 2021; Ramos dkk., 2024b).

Berikut merupakan *Hyperparameter* yang dapat dikostumisasi pada model YOLO:

1. Ukuran Gambar (*Image Size*): *image size* pada model YOLO mengacu pada dimensi Gambar *input* yang diberikan kepada model selama pelatihan dan inferensi. Ukuran Gambar *default* YOLO adalah 640x640 (Derrenger dkk., 2024; Jocher & Bình, 2023). Ukuran gambar mempengaruhi kinerja akurasi mendeteksi objek kecil dan besar dari model. Ukuran Gambar yang lebih besar umumnya memungkinkan model untuk mendeteksi objek yang lebih kecil dengan detail yang lebih baik, tetapi juga meningkatkan beban komputasi. Sebaliknya, Gambar yang lebih kecil dapat mempercepat pelatihan dan

kesimpulan, tetapi dapat mengorbankan akurasi pada objek yang kecil (Dalal dkk., 2024; Goodflellow dkk., 2016; Javed dkk., 2021).

2. Ukuran *Batch* (*Batch Size*): Ukuran *batch* dalam YOLO mengacu pada jumlah Gambar yang diproses secara bersamaan dalam satu iterasi selama pelatihan. Nilai ukuran *batch default* YOLO adalah 16 (Derrenger dkk., 2024; Jocher & Bình, 2023). Ukuran *batch* mempengaruhi penggunaan memori GPU, stabilitas pembelajaran, dan kecepatan konvergensi model. Ukuran *batch* yang lebih besar memungkinkan model untuk menghitung gradien yang lebih akurat karena jumlah sampel gambar lebih banyak. Namun, meningkatkan ukuran *batch* juga meningkatkan penggunaan memori dan dapat menyebabkan model tidak dapat dilatih oleh GPU dengan kapasitas terbatas (Isa dkk., 2022; Ramos dkk., 2024b). Ukuran *batch* yang lebih kecil menghemat penggunaan memori, namun dapat mengurangi stabilitas pembelajaran. Hal ini dikarenakan gradien yang dihasilkan dari *batch* yang lebih kecil cenderung memiliki varians yang lebih tinggi, sehingga kurang akurat dalam merepresentasikan gradien keseluruhan dari data pelatihan (Isa dkk., 2022). Pada model YOLO, memilih ukuran *batch* yang optimal berguna untuk mencapai keseimbangan antara akurasi dan waktu pelatihan, terutama ketika menggunakan perangkat keras dengan sumber daya terbatas (Feurer & Hutter, 2019; Goodflellow dkk., 2016).
3. *Learning rate* (laju pembelajaran) menentukan kecepatan model memperbarui nilai *weight* selama pelatihan. Persamaan matematis dari perubahan *weigth* ditunjukkan pada persamaan 8:

$$w_{t+1} = w_t - \eta . \nabla L(w_t) \quad (8)$$

di mana, wt merupakan bobot (*weight*) pada iterasi ke-t. Lalu η merupakan nilai *hyperparameter* dan ∇L(wt) merupakan gradien dari fungsi *loss* L pada *weight* iterasi ke-t. Menggunakan *learning rate* yang lebih tinggi dapat menyebabkan model konvergen lebih cepat, tetapi juga dapat menyebabkan model kehilangan *weight* optimal, yang berdampak negatif pada performa (Motta dkk., 2020; Ramos dkk., 2024b; Van & Hoang, 2023). laju pembelajaran yang lebih rendah dapat memastikan konvergensi yang lebih stabil, tetapi juga dapat menyebabkan model membutuhkan waktu lebih lama untuk mencapai solusi terbaik.

4. *Weight Decay* adalah teknik regulasi yang digunakan untuk mengurangi *overfitting* dengan menambahkan penalti pada bobot model selama pelatihan. Dalam model YOLO, *weight decay* mempengaruhi kinerja deteksi dengan membantu model menjaga generalisasi, terutama pada *dataset* yang besar dan kompleks. Penggunaan *weight decay* yang tepat dapat meningkatkan stabilitas pelatihan dan akurasi deteksi tanpa terlalu meningkatkan kompleksitas model. Penelitian sebelumnya menunjukkan bahwa penyesuaian *weight decay* pada YOLO menghasilkan perbaikan signifikan pada kemampuan mendeteksi objek kecil dalam gambar dengan latar belakang kompleks (Feurer & Hutter, 2019; Motta dkk., 2020; R. Anita Jasmine dkk., 2022). Namun, penggunaan *weight decay* yang terlalu besar dapat memperlambat konvergensi dan mengurangi kemampuan model untuk menangkap fitur penting dalam data (Liao dkk., 2022; Ramos dkk., 2024b; L. Yang & Shami, 2020).

5. *Epoch*: Jumlah *epoch* mengacu pada jumlah total iterasi di seluruh *dataset* selama pelatihan. Jumlah *epoch* yang lebih tinggi dapat meningkatkan kinerja model tetapi juga dapat berisiko *overfitting*. Sebaliknya, pelatihan dengan jumlah epoch yang lebih sedikit dapat lebih efisien dari segi waktu, namun dapat menyebabkan model menjadi kurang pas (*underfitting*) (Dalal dkk., 2024; Ramos dkk., 2024b; Van & Hoang, 2023).
6. *Optimizer* mengontrol bagaimana parameter model diperbarui berdasarkan penurunan gradien. Memilih *optimizer* yang tepat sangat penting dalam menentukan kecepatan konvergensi, stabilitas pelatihan, dan kemampuan generalisasi model (Dalal dkk., 2024; Isa dkk., 2022; Javed dkk., 2021; Ramos dkk., 2024a; Sun dkk., 2021). Beberapa *optimizer* yang tersedia pada pelatihan model YOLOv11 termasuk *Stochastic Gradient Descent* (SGD), Adam, AdamW, dan RMSprop (Derrenger dkk., 2024; Jocher & Bình, 2023).
    a. *Stochastic Gradient Descent* (SGD) adalah metode optimasi yang secara bertahap memodifikasi parameter model ke arah *loss function negative gradient*, sebanding dengan *learning rate* (Mustapha dkk., 2020). Meskipun sederhana dan efisien, metode ini terkadang terjebak pada nilai *local minima* (titik di mana nilai loss funtion mencapai titik terendah dalam suatu *wilayah*, tetapi bukan titik terendah global).
    b. *Adaptive Moment Estimation* (Adam) bekerja memperbarui model *weight* dengan menggunakan gabungan antara *learning rate* dan momentum yang telah disesuaikan. Adam menyesuaikan *learning rate* untuk setiap *weight*, dengan mempertimbangkan ukuran gradien dan riwayat gradien

sebelumnya, sehingga menghasilkan proses pengoptimalan yang lebih efektif dan hemat sumber daya (Llugsi dkk., 2021; SEN & OZKURT, 2020).

c. AdamW merupakan variasi dari algoritma optimisasi Adam. Dengan menambahkan istilah peluruhan bobot (*weight decay*) ke dalam fungsi kehilangan, algoritma ini bertujuan untuk mengurangi overfitting dan meningkatkan kemampuan generalisasi model (Llugsi dkk., 2021)

d. *Root Mean Square Propagation* (RMSProp). Pengoptimal ini adalah berbasis momentum dan menggunakan gradien kuadrat rata-rata dari *loss function* untuk mengadaptasi laju pembelajaran secara individual untuk setiap bobot (Chattopadhyay & Maitra, 2022). Metode ini membantu dalam menemukan rute yang sesuai dan efisien seminimal mungkin.

Beberapa metode yang umum digunakan untuk optimasi *hyperparameter* antara lain:

Metode *One Factor At a Time* (OFAT) menguji setiap *hyperparameter* secara individual untuk melihat bagaimana pengaruhnya parameter tersebut terhadap performa model (Suryawanshi & Eswari, 2022).

Metode *Random Search* menguji metode secara acak dengan mengambil sampel *hyperparameter* mencoba kombinasi *hyperparameter* yang berbeda untuk menemukan konfigurasi yang optimal (P dkk., 2022).

Metode *Grid Search* adalah metode optimasi *hyperparameter* yang melibatkan eksplorasi menyeluruh (*exhaustive search*) dari semua kombinasi *hyperparameter* yang ditentukan pengguna (P dkk., 2022). Metode ini

mencoba menemukan konfigurasi terbaik berdasarkan metrik evaluasi yang dipilih.

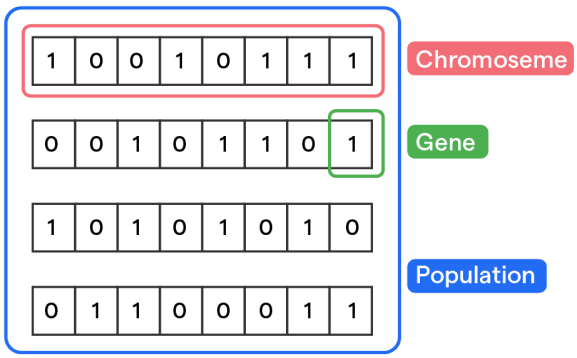
Metode lain yang lebih modern adalah menggunakan *Genetic Algorithm* juga dapat diterapkan untuk mengeksplorasi ruang *hyperparameter* yang lebih kompleks dan besar, sehingga menghasilkan kinerja model yang lebih baik.

2.5.1 Algoritma Genetika (*Genetic Algorithm*)

Algoritma Genetika atau *Genetic Algorithm* (GA) adalah salah satu metode optimasi berbasis populasi yang terinspirasi dari teori evolusi biologi, seperti seleksi alam dan mutasi genetik (Gen, 2006; Katoch dkk., 2021; Sheehan & Song, 2016). Metode ini dirancang untuk menemukan solusi optimal dari suatu masalah dengan memanfaatkan mekanisme evolusi seperti seleksi, *crossover*, dan mutasi. Algoritma GA dimulai dengan sekelompok solusi acak yang disebut populasi awal. Setiap solusi (kromosom) dievaluasi berdasarkan sebuah fungsi yang disebut fungsi *fitness*. Beberapa solusi dengan *fitness* yang tinggi akan dipilih untuk generasi/iterasi berikutnya (Gen, 2006; Katoch dkk., 2021).

Langkah-langkah utama dalam Algoritma Genetika meliputi:

1. Penciptaan Populasi: Proses ini meliputi pembuatan populasi awal yang terdiri dari solusi atau gen acak. Setiap solusi direpresentasikan dalam bentuk kromosom atau serangkaian angka yang berbeda tergantung pada jenis masalah yang diselesaikan (Gen, 2006; Katoch dkk., 2021).



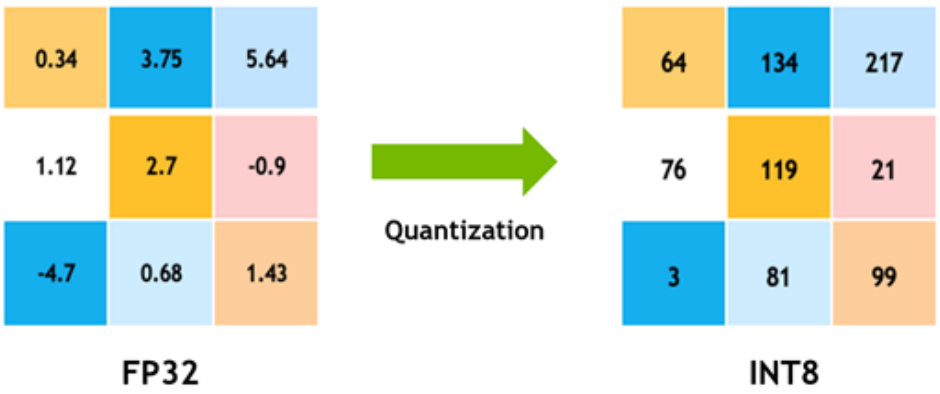
Gambar 2.13 Bagian-Bagian pada Algoritma Genetik (Gen, 2006)

2. Seleksi: proses ini memilih kromosom berdasarkan nilai *fitness* mereka. Fungsi *fitness* menilai setiap kromosom berdasarkan kemampuan atau kecocokannya terhadap masalah yang ingin dipecahkan, dan memberikan nilai kuantitatif yang menunjukkan kualitas solusi tersebut. Teknik seleksi nilai *fitness* yang umum digunakan adalah seleksi peringkat, seleksi turnamen, dan elitisme. (Gen, 2006; Katoch dkk., 2021).
3. Rekombinasi (*Crossover*): Proses ini menggabungkan dua atau lebih solusi (orang tua) untuk membentuk satu atau lebih solusi baru (keturunan). Proses ini meniru pertukaran gen antar individu dalam organisme biologis. Teknik persilangan yang umum digunakan seperti persilangan satu titik, persilangan dua titik, atau persilangan seragam (Gen, 2006; Katoch dkk., 2021).
4. Mutasi: Proses ini melakukan perubahan acak pada kromosom yang digunakan untuk mengeksplorasi posibilitas solusi. Proses mutasi dilakukan dengan mengganti satu atau beberapa gen pada kromosom dengan nilai acak yang baru (Gen, 2006; Katoch dkk., 2021).
5. Evaluasi dan Penggantian: Setelah populasi baru terbentuk, setiap solusi dievaluasi kembali dengan menggunakan fungsi *fitness*. Solusi dengan *fitness*

terbaik akan dipertahankan dan populasi lama digantikan dengan populasi baru untuk melanjutkan ke generasi berikutnya sampai kriteria penghentian terpenuhi (Gen, 2006; Katoch dkk., 2021).

2.6 Integer Quantization

Integer Quantization adalah teknik optimasi yang mengubah representasi parameter model dari *floating-point* (biasanya 32-bit) ke representasi *integer* (8-bit) (Javed dkk., 2021; Kluska & Zięba, 2020; Krishnamoorthi, 2018). Ada sekitar 4 miliar nilai dalam rangkaian kombinasi nilai FP32 mulai dari $-3{,}4 \times 10^{38}$ hingga $3{,}4 \times 10^{38}$. Sedangkan dengan INT8, kita hanya melihat 256 nilai dalam rangkaian nilai yang mungkin mulai dari -128 hingga 128. Karena kumpulan nilai bit yang jauh lebih kecil, perkalian matriks dapat terjadi lebih cepat. *Quantization* pada model deteksi objek YOLO dilakukan dengan merepresentasikan nilai bobot (*weights*) dan aktivasi (*activations*) sebagai bilangan bulat 8-bit, yang secara signifikan mengurangi kebutuhan memori dan mempercepat inferensi (Dalal dkk., 2024; Zhu dkk., 2023). Proses ini menghasilkan model yang lebih kecil dan lebih efisien dalam hal konsumsi memori dan kecepatan inferensi tanpa mengorbankan banyak akurasi. Integer *quantization* sering digunakan pada perangkat dengan daya dan komputasi yang terbatas, seperti perangkat *edge* atau perangkat seluler.



Gambar 2.14 Proses *Integer Quantization* (Zmora dkk., 2021)

Integer quantization mengubah *weight* dan *activations* pada *neural network* ke dalam bentuk bilangan bulat dengan menggunakan skala tertentu. *WeightIP* yang awalnya dalam format *floating-point* dipetakan ke rentang bilangan bulat yang lebih kecil. Pengubahan format *weight* ini mampu menghemat hingga 25 persen memori (Liberatori dkk., 2022). Proses ini memungkinkan operasi matematika seperti konvolusi dan perkalian matriks dilakukan dalam format bilangan bulat, yang lebih cepat dan lebih hemat daya pada perangkat keras (Kluska & Zięba, 2020).

Persamaan 9 merupakan persamaan matematis dari Quantization (Zmora dkk., 2021):

$$x_q = clip(round\left(\frac{x_f}{scale}\right)) \tag{9}$$

di mana, x_q melambangkan nilai *weight* yang sudah di kuantisasi (bilangan bulat 8-bit), x_f adalah nilai *weight* asli dalam format FP32 (*Floting Point* 32-bit). Fungsi $clip()$ berfungsi untuk membatasi nilai output selalu berada didalam rentang 8-bit yaitu [-128, 127]. Sedangkan, *round*() adalah fungsi pembulatan yang mengubah hasil pembagian $\frac{x_f}{scale}$ menjadi bilangan bulat terdekat. *scale* adalah

scaling factor atau faktor skala, yang digunakan untuk mengubah rentang nilai bobot asli menjadi rentang kuantisasi (Dalal dkk., 2024; Zmora dkk., 2021).

Fungsi untuk mencari nilai *scale* (Zmora dkk., 2021) ditunjukkan pada persamaan 10:

$$scale = \frac{(2 * amax)}{256} \tag{10}$$

di mana, amax merupakan nilai absolut terbesar dari rentang data *floating point* itu sendiri. Nilai 256 merupakan cakupan diskrit dari nilai integer 8-bit yaitu [-128, 127] yaitu 256.

Proses ini dapat dilakukan setelah pelatihan model (*post-training quantization*) atau selama pelatihan (*quantization-aware training*). Teknik ini sering diterapkan pada model *deep learning* yang digunakan pada perangkat keras untuk meningkatkan efisiensi komputasi dan mengurangi kebutuhan memori, seperti pada TensorFlow Lite untuk aplikasi Android atau perangkat yang disematkan.

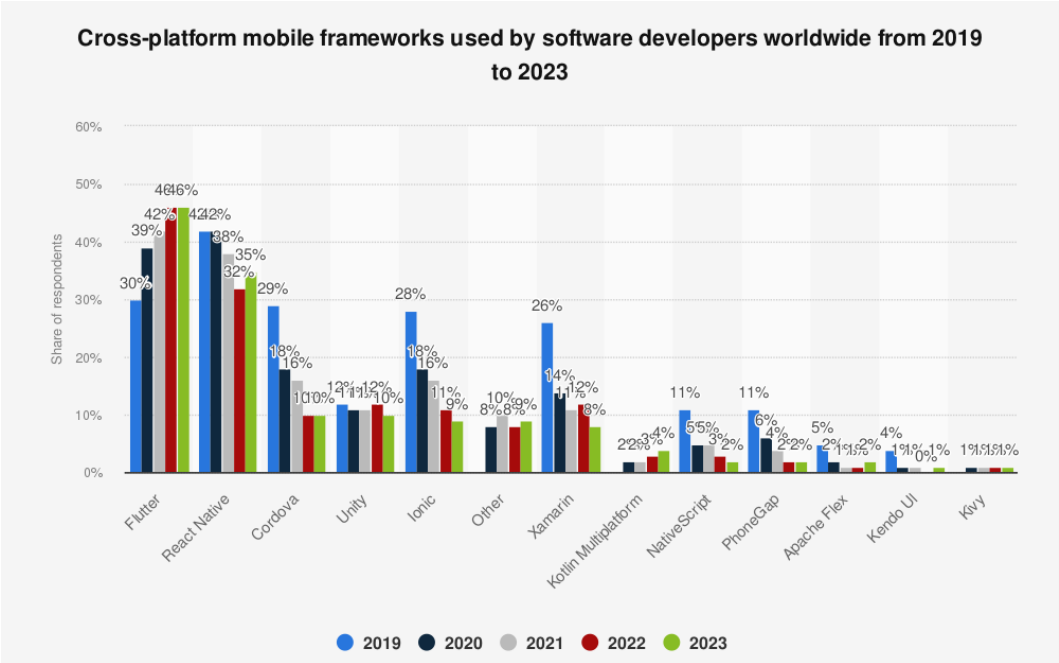
2.7 Flutter

Flutter adalah *framework open-source* yang dikembangkan oleh Google yang dirilis pada tahun 2017 untuk membangun aplikasi multiplatform dari satu basis kode yang mencakup perangkat seluler, *website*, *desktop*, dan *embedded* Flutter menggunakan *rendering* berbasis Impeller untuk memberikan performa yang lebih cepat dan stabil. Hal ini memungkinkan aplikasi yang dikembangkan dengan Flutter memiliki performa yang lebih baik dan latensi yang sama atau bahkan lebih rendah dibanding bahasa *native* lain pada berbagai perangkat (Fallas dkk., 2024).

Flutter menggunakan bahasa pemrograman Dart sebagai bahasa utama *framework*. Bahasa pemrograman Dart mendukung kompilasi *ahead-of-time*

(AOT) yang menghasilkan kode mesin asli. Ketika aplikasi Flutter dibangun, kode Dart akan dikompilasi menjadi kode mesin asli untuk platform target (seperti ARM atau x86). Keunggulan Dart ini membuat Flutter mampu membuat aplikasi dengan kecepatan dan responsifitas yang setara dengan aplikasi *native* di berbagai platform.

Flutter memiliki fitur "hot reload" yang memungkinkan pengembang untuk melihat perubahan kode secara instan tanpa harus me-*restart* aplikasi. Hal ini mempercepat siklus pengembangan dan *debugging* aplikasi. Flutter juga mendukung integrasi dengan berbagai layanan Google seperti Firebase, Google Maps, dan Google Play, sehingga memudahkan pengembangan aplikasi dengan ekosistem Google secara mulus.



Gambar 2.15 Platform Mobile Framework yang di Gunakan Programmer dari Tahun 2019 – 2023 (Jadaun dkk., 2023)

Penelitian terbaru telah membahas penggunaan Flutter yang terus meningkat. Studi oleh (Vailshery, 2024) menyatakan bahwa Flutter merupakan *framework mobile* lintas platform terpopuler kedua di antara para developer di seluruh dunia.

2.8 Penelitian Terkait

Saat ini terdapat banyak penelitian yang mencoba mengoptimalkan *hyperparameter* pada model YOLO untuk meningkatkan performa model. Aspek-aspek penting yang dapat meningkatkan akurasi dalam *hyperparameter tuning* model diantara lain seperti *learning rate*, *image size*, *batch size*, dan *optimizer* (Isa dkk., 2022; R. Anita Jasmine dkk., 2022; Ramos dkk., 2024b; Van & Hoang, 2023; Y. Y. Wang dkk., 2022; L. Xu dkk., 2023). Sedangkan, *integer quantization* dapat digunakan untuk menurunkan kompleksitas model dan membuat model membutuhkan lebih sedikit daya komputasi. *Interface time* dari model YOLO yang sudah di *quantization* menjadi jauh lebih tinggi (Chen & Lin, 2019; Liberatori dkk., 2022; Wai dkk., 2019; Yue dkk., 2022; Zmora dkk., 2021).

Tabel 2.1 *State of the art* penelitian terkait

Penulis	Model	*Hyperparameter*	Tipe *Quantization*	Keterangan
(Isa dkk., 2022)	YOLOv5	*Learning Rate*, *Optimizer* (SGD, Adam), *Image Size*	-	**Kelebihan**: Kenaikan tingkat konvergensi model. Kenaikan mAP dari 97,7% menjadi 98,6%. **Kekurangan**: Hanya menguji *learning rate* dan *optimizer* saja. Belum menerapkan *quantization*.
(R. Anita Jasmine dkk., 2022)	YOLO CNN	*Batch Size*, *Optimizer* (SGD, Adam),		**Kelebihan** : Kenaikan mAP dari 97% menjadi 98%. Nilai *recall* tinggi di 0,984.

Penulis	Model	*Hyperparameter*	Tipe *Quantization*	Keterangan
				Kekurangan: Model terlalu kompleks. Kebutuhan daya komputasi model tinggi. Belum menerapkan *quantization*.
(Ramos dkk., 2024b)	YOLOv8	*Learning Rate*, *Optimizer* (SGD, Adam, Adamw, RMSprop), *Epoch*, *Batch Size*	-	**Kelebihan**: kenaikan nilai in *precision* 1.39 %, *recall* 1.48 %, F1-score 1.44 %, dan mAP 0.70 % pada mAP50 dan 5.09 % pada mAP50-95 **Kekurangan** : Belum menerapkan *Quantization*.
(Van & Hoang, 2023)	YOLOv5 dan YOLOv7	*Learning Rate*, *Epoch*, *Image Size*	-	**Kelebihan**: mAP_0.5:0.95, naik 4.42% dibanding YOLOv5x *Default* dan 8.37% dibandingkan YOLOv7 **Kekurangan**: mAP_0.5 turun 0.2% dibanding YOLOv5x *Default*. Penurunan mAP_0.5:0.95 dibanding YOLOv5m (1.29%). Belum menerapkan *Quantization*.
(L. Xu dkk., 2023)	G-YOLOv5	*Learning Rate*, *Weight Decay*,	-	**Kelebihan** : Peningkatan mAP sebesar 1.7%. Peningkatan F1-score sebesar 1.0%. Peningkatan mAP untuk *pedestrian* sebesar 2.6%.. Peningkatan mAP untuk *cyclist* sebesar 2.3%. **Kekurangan**: Model terlalu kompleks. Kebutuhan daya

Penulis	Model	*Hyperparameter*	Tipe *Quantization*	Keterangan
				komputasi model tinggi. Belum menerapkan *quantization*.
(Chen & Lin, 2019)	YOLOv3-Live		INT8	**Kelebihan**: Kenaikan FPS dari 6 FPS ke 28 FPS **Kekurangan**: Penurunan nilai mAP 87,79% ke 69,79%. Belum menerapkan *Hyperparameter Tuning*.
(Liberatori dkk., 2022)	YOLOv4	-	INT8	**Kelebihan**: Kenaikan FPS dari 0,99 FPS menjadi 1,97 FPS **Kekurangan**: Turunnya mAP dari 0,618 menjadi 0,74. Belum menerapkan *Hyperparameter Tuning*.
(Wai dkk., 2019)	Tiny YOLOv2	-	INT8	**Kelebihan**: Hanya 0.3% mAP yang loss atau turun. **Kekurangan**: Model YOLO yang dipakai cukup obsolet. Belum menerapkan *optimizer* dan *Hyperparameter Tuning*.
(Yue dkk., 2022)	YOLOv4-GD	-	FP16	**Kelebihan**: kenaikan mAP dari 97,38% ke 97,42. Penurunan *respon time* dari 207,92 ms ke 32,75 ms. Kenaikan FPS dari 4,81 FPS ke 30,53 FPS. Kekurangan: Deteksi beberapa kelas objek tidak terkenali dengan baik. Kesalahan ekstraksi Belum menerapkan

Penulis	Model	*Hyperparameter*	Tipe *Quantization*	Keterangan
				Hyperparameter tuning.
(Zhu dkk., 2023)	YOLOv7 dan QBS-YOLO	-	QAT INT8	**Kelebihan:** mAP50 dari 79% ke 84,8%. FPS dari 40 FPS ke 57 FPS. Penurunan *respon time* dari 11,8ms ke 8,3ms. **Kekurangan:** Penggunaan *Quantization Aware Training* membuat model jauh menjadi lebih kompleks. Proses *training* model memakan waktu lebih lama dan membutuhkan Hardware yang lebih memadai.
(Dalal dkk., 2024)	YOLO-Tranfer Learning	*Optimzer* (SGD), *Learning rate*, *Batch Size*, *Epoch*	INT8	**Kelebihan:** Penerapan *Hyperparameter tuning* pada model. dibandingkan model YOLOv5, Kenaikan mAP50 dari 0,704 ke 0,819 dan mAP50-95 dari 0,349 ke 0,569 **Kekurangan:** Terjadinya *overfitting* karena *dataset* kurang besar. Meskipun model sudah dioptimalkan dengan *quantization*, penerapan pada perangkat dengan kemampuan komputasi rendah masih membutuhkan pengujian lebih lanjut.
(Javed dkk., 2021)	Tiny-YOLOv2 dan QuantYOLO	*Learning Rate*, *Batch Size*, *Optimizer* (Adam), *Image Size*	INT8	**Kelebihan:** Peningkatan FPS dari 6,53 FPS ke 25,29 FPS. Penurunan konsumsi daya dari 4,06 watt ke 1,92 watt.

Penulis	Model	*Hyperparameter*	Tipe *Quantization*	Keterangan
				Kekurangan: Penurunan akurasi dari 73,2% ke 70,1%. *Quantization* tidak dilakukan pada setiap layer, masih ada layer dengan *Full Precision*(FP32/32 bit *floating point*).

Pada Tabel 2.1, beberapa penelitian terkait optimalisasi model YOLO untuk deteksi objek telah dilakukan dengan menerapkan berbagai metode *hyperparameter tuning* dan teknik *quantization*. Penelitian oleh (Isa et al., 2022) dan (Van & Hoang, 2023) berfokus pada *tuning* parameter seperti *learning rate* dan *image size* menghasilkan peningkatan mAP meskipun belum menerapkan *quantization*. Penelitian oleh (Xu et al., 2023) mengusulkan G-YOLOv5 dengan *tuning* parameter *learning rate* dan *weight decay*, yang terbukti efektif meningkatkan mAP sebesar 1.7%. Penelitian seperti (Chen & Lin, 2019) dan (Liberatori et al., 2022) telah menerapkan *integer quantization* pada YOLOv3 dan YOLOv4 yang berfokus pada peningkatan FPS dan penurunan waktu inferensi tetapi terjadi penurunan mAP yang signifikan. Penelitian oleh (Dalal et al., 2024) dan (Javed et al., 2021) menggabungkan *hyperparameter tuning* dan integer *quantization* (INT8) untuk model Tiny-YOLOv2 berhasil meningkatkan FPS hingga lebih dari 20 FPS dengan konsumsi daya yang lebih rendah. Namun, tingkat kompleksitas model masih terbilang tinggi akibat beberapa layer yang masih menggunakan *Full Precision* 32-bit. Dari penelitian-penelitian tersebut, belum ada yang secara khusus mengintegrasikan optimalisasi *hyperparameter tuning* dan integer *quantization*

pada model YOLOv11 untuk deteksi objek secara *real-time* pada perangkat dengan daya komputasi rendah.

Perbandingan penelitian terkait dapat dilihat melalui matriks penelitian pada Tabel 2.2.

Tabel 2.2 Perbandingan target capaian penelitian dengan penelitian terkait

Penelitian	Model	*Hyperparmeter Tuning*				*Quantization*
		Learning rate	*Image Size*	*Batch Size*	*Optimizer*	
(Isa dkk., 2022)	YOLOv5	√	√	-	√	-
(R. Anita Jasmine dkk., 2022)	YOLO CNN	-	-	√	√	-
(Ramos dkk., 2024b)	YOLOv8	√	-	√	√	-
(Van & Hoang, 2023)	YOLOv5 dan YOLOv7	√	√	-	-	-
(L. Xu dkk., 2023)	G-YOLOv5	√	-	-	-	-
(Chen & Lin, 2019)	YOLOv3-Live	-	-	-	-	√
(Liberatori dkk., 2022)	YOLOv4	-	-	-	-	√
(Wai dkk., 2019)	Tiny YOLOv2	-	-	-	-	√
(Yue dkk., 2022)	YOLOv4-GD	-	-	-	-	√
(Zhu dkk., 2023)	YOLOv7 dan QBS-YOLO	-	-	-	-	√
(Dalal dkk., 2024)	YOLO-Tranfer Learning	√	-	√	√	√
(Javed dkk., 2021)	Tiny-YOLOv2 dan QuantYOLO	√	√	√	√	√

Pada Tabel 2.2, perbandingan penelitian terkait menunjukkan bahwa beberapa penelitian lebih berfokus pada *hyperparameter tuning* tanpa mempertimbangkan *quantization*, seperti yang dilakukan oleh (Isa et al., 2022), (Ramos et al., 2024),

dan (Van & Hoang, 2023). Penelitian lain yang difokuskan pada *quantization*, seperti (Chen & Lin, 2019), (Liberatori et al., 2022), dan (Yue et al., 2022) berhasil meningkatkan kecepatan inferensi, tetapi mengalami penurunan mAP yang signifikan karena tidak dilengkapi dengan *tuning hyperparameter*. Sebaliknya, penelitian oleh (Dalal et al., 2024) dan (Javed et al., 2021) menunjukkan kombinasi keduanya, dengan hasil yang lebih seimbang antara kecepatan dan akurasi. Penelitian ini mengintegrasikan *learning rate*, *batch size*, *optimizer*, dan *quantization* untuk mendapatkan performa optimal. Penelitian oleh (Javed et al., 2021) bahkan berhasil menurunkan konsumsi daya model Tiny-YOLOv2 yang cocok untuk diterapkan pada perangkat dengan daya komputasi rendah. Hal ini menunjukkan bahwa upaya untuk menggabungkan *hyperparameter tuning* dan *quantization* seperti yang dilakukan dalam penelitian ini dapat menjadi strategi yang efektif untuk mencapai performa optimal tanpa mengorbankan kecepatan inferensi dan akurasi deteksi. Maka dari itu, penelitian ini berfokus pada penggabungan *hyperparameter tuning* dan *quantization* untuk mencapai hasil yang lebih baik dari segi akurasi dan waktu *interface* pada model YOLO terbaru yaitu YOLOv11 ketika model digunakan pada perangkat dengan daya komputasi yang rendah.