BAB II

LANDASAN TEORI

2.1 Penelitian Terdahulu

Berbagai penelitian telah dilakukan untuk mengembangkan serta meningkatkan algoritma deteksi helm, terutama yang menggunakan metode YOLO (*You Only Look Once*) sebagai salah satu algoritma deteksi objek yang banyak diterapkan dalam bidang *computer vision* [34]. Seiring dengan perkembangan teknologi *computer vision* dan *deep learning*, alogitma YOLO terus mengalami peningkatan dari berbagai versi yang menawarkan performa lebih baik dalam hal akurasi [35]. Penelitian-penelitian ini fokus pada pengoptimalan model yang berguna untuk menghasilkan akurasi yang lebih tinggi sehingga dapat diterapkan secara efektif di berbagai kondisi nyata, baik dalam skenario statis maupun dinamis.

Berbagai Pendekatan telah diusulkan dalam penelitian terdahulu, diantaranya adalah modifikasi arsitektur modul dengan menambahkan modul tertentu seperti *attention mechanism* [24][28], *feature pyramid network* (FPN) [36], atau convolutional *block* [37] [21] yang lebih kompleks. Selain itu, diterapkan teknik augmentasi data untuk meningkatkan kemampuan generalisasi model terhadap berbagai variasi kondisi data, seperti perubahan cahaya, rotasi, skala objek, dan kondisi lingkungan lainnya [25]. Kemudian terdahap pengoptimalan *loss function* dan penyesuaian parameter *training* juga dilakukan yang berguna untuk meningkatkan stabilitas dan performa model selama *training* [21][38].

Fokus utama dari penelitian terdahulu adalah untuk mengatasi berbagai tantangan dalam deteksi helm, seperti sudut pandang yang terbatas, objek yang terhalang (*occlusion*), kondisi pencahayaan yang kurang optimal, serta variasi ukuran dan posisi objek didalam gambar. Tabel 2.1 merangkum hasil-hasil penelitian terdahulu yang membandingkan berbagai pendekatan dalam deteksi helm, dengan menampilkan metrik evaluasi akurasi (*mean Average precision* / mAP), dan kemampuan model dalam menghadapi berbagai kondisi lingkungan.

Tabel 2.1 Penelitian Terdahulu

Penulis	**Nama Jurnal & Nama Artikel**	**Dataset**	**Metode**	**Masalah**	**Hasil**	**Kelebihan**	**Kekurangan**
An, Y., Li, Z., Chen, L., & Chen, H. [36]	*Journal of Engineering Science and Technology Review 15 (1) (2022)* *"Detection Method of Helmet Wearing in Complex Scenes"*	2000 gambar (*hat* & *person*	yolov5 optimasi dengan CIoU loss, *augmentation*, modul *focus*, CSP, dan FPN-PAN	keterbatasan metode deteksi helm sebelumnya dan jangkauan pembaca RF yang terbatas	YOLOv5 unggul dengan mAP sebesar 94,9% dan kecepatan 30,5 FPS dibandingkan YOLOv3	Deteksi Real time dan mendeteksi dua kondisi kerja	Deteksi gambar banyak halangan (occlusion), pencahayaan buruk
H. Song [34]	*Sensors*, *22*(16), 6061 (2022) "*Multi-Scale Safety Helmet Detection Based on RSSE-YOLOv3*"	7500 gambar terdiri *multiple scenes* dan *multiple targets*	YOLOv3 + *Residual Spatial Squeeze and Exication (RSSE)* + *Complete Intersection over Union (CIOU)*	YOLOv3 + Darknet53 meningkatkan latency dan memperlambat kecepatan deteksi	precision (P) = 92,3%, recall (R) = 90,4%, F1 score = 91,3%, dan mean Average Precision (mAP) = 92,8%.	Menignkatkan kemampuan ekstraksi fitur dan proses deteksi	Dataset dari berbagai sumber yang menyebabkan perbedaan distribusi data
Mu, T., & Ding, L. [24]	*Academic Journal of Computing & Information Science (2025)* *"Motorcycle Helmet Detection Model Based on Improved YOLOv11n "*	5.489 gambar dengan label (motorperson, helmet, nonhelmet)	YOLOv11n dengan penambahan C3k2-SCConv, iAFF berbasis MS-CAM, MultiSEAM, serta ADown	YOLOv11n memiliki masalah deteksi motor yang berada jauh di background dan helm ukuran kecil	YOLOv11n, mencapai precision 88,9%, recall 82,5%, dan mAP sebesar 88,2%.	Model lebih ringan, dengan 2.69 parameters dan 5.3 Gflops	FPS turun dibandingkan baseline
Viriyasatr et al. [31]	*Defence Technology Academic Journal (2025)* *"An Investigation of YOLO-Based Helmet*	Menggunakan *Hard Hat Workers* Dataset total 1812 gambar dengan	model YOLOv11 GPU dan CPU untuk kebutuhan real-time.	Banyak sistem CCTV hanya menggunakan CPU *server* dan GPU mahal	YOLOv11n mencapai mAP sebesar 0,939 dan recall sebesar 0,902, dengan	Terdapat performa GPU dan CPU	Masih terdapat *missed detection* dan *false positive* pada *background*

Penulis	Nama Jurnal & Nama Artikel	Dataset	Metode	Masalah	Hasil	Kelebihan	Kekurangan
	Detection for Workers on GPU and CPU Platforms"	helm dan nonhelm			jumlah parameter 2,58M		
Alif, M. A. R. [35]	*Department of Computer Science (2024)* *"Yolov11 for Vehicle Detection: Advancements, Performance, and Applications in Intelligent Transportation Systems"*	Jumlah gambar 1321 dengan berbagai jenis kendaraan	YOLOv11 Optimalisasi: C2PSA, C3k2 block, SPPF, data augmentation, cosine annealing learning rate	Beberapa jenis kendaraan memiliki kemiripan visual, objek tertutup	mAP 0.743 F1 Score: 0.71 (confidence ~0.61) Recall: 0.93 (maksimum) Inference Speed: 290 FPS	YOLOv11 dengan C2PSA membantu model fokus pada area penting gambar	Recall turun pada confidence threshold tinggi
Licheng Sun, Heping Li, Liang Wang [28]	*CMC (Computers, Materials & Continua) (2024)* *"HWD-YOLO: A New Vision-Based Helmet Wearing Detection Method"*	Jumlah dataset 7581 dengan gambar berisi pekerja memakai helm dan tidak menggunakan helm	Menggunakan YOLOv5 dengan modul MSCAM, *DetecBlock*, dan *spatial transformer*	Helm sering terlihat kecil Digambar karena area luas dan kamera jauh	YOLOV5 yang telah dimodifikasi menghasilkan akurasi mAP: 96.3%, kecepatan deteksi: 33 FPS.	Model mampu membedakan helm keselamatan dengan topi biasa	Struktur model cukup kompleks dan loss function disarankan menggunakan GIoU
Wang et al. [25]	*Mach. Learn. Knowl. Extr. (2025)* *"Helmet Detection in Underground Coal Mines via Dynamic Background Perception with Limited Valid Samples"*	Jumlah 3076 gambar dengan jumlah scene 7	menggunakan model YOLOv10 dengan penerapan DBConv,GLFM, data augmentation	Background sangat kompleks dan objek helm sangat kecil	mencapai akurasi 94,4%, recall 89%, dan mAP sebesar 95,4%, serta berhasil menurunkan kesalahan deteksi latar belakang hingga 14%	Mampu menangkap detail local dan konteks global	Jumlah yang tidak menggunakan helm sangat sedikit

Penulis	Nama Jurnal & Nama Artikel	Dataset	Metode	Masalah	Hasil	Kelebihan	Kekurangan
Chun Shan, HongMing Liu, Yu Yu [37]	*Scientific Reports (2023)* *"Research on improved algorithm for helmet detection based on YOLOv5"*	Jumlah gambar 23.088 dengan kategori helmet dan head	Penyempurnaan algoritma YOLOv5 dengan BIFPN, ECA, dan *Decouping Head*	*False detection*, objek helm tidak terdeteksi, akurasi rendah pada kondisi kompleks	memiliki akurasi algoritma dengan mAP 95,9%, *Recall* = 92.4%, dan *Precision* = 94.7%	Deteksi objek kecil lebih akurat dan meningkatkan ekstraksi fitur	Weight model meningkat menjadi 27.9 MB, belum diuji secara luas di kondisi nyata
Han, J., Li, Z., Cui, G., & Zhao, J. [21]	*Appl. Sci., 14(17), 7923* (2024) *"EGS-YOLO: A Fast and Reliable Safety Helmet Detection Method Modified Based on YOLOv7"*	Jumlah gambar 5000 dengan label objek 75.750 dengan kategori helmet dan head	YOLOv7 ditingkatkan dengan *GhostModule*, *Squeeze and Excitation (SE),* dan *Enhanced IOU (EIOU)*	YOLOv7 memiliki parameter besar, komputasi tinggi, dan kurang efisien untuk deployment real-time	EGS-YOLO menghasilkan mAP = 91.1%, *Precision* = 89.9% dan *Recall* = 84.7% dan waktu eksekusi 3.9 ms	Optimalisasi mengurangi parameter model 37.3% dan komplektisitas turun 33.8%	Model masih memiliki ukuran cukup besar dengan 22.9 juta parameter
Zhang, Y., Qiu, Y., & Bai, H. [38]	*Electronics, 12(13), 2902 (2023)* *"FEFD-YOLOV5: A Helmet Detection Algorithm Combined with Feature Enhancement and Feature Denoising"*	Total 6140 gambar dengan kondisi pencahayaan yang berbeda beda	FEFD-YOLOv5 yaitu mengintegrasi *Small Object Detection (SDH), Attention Modules (SENet, CBAM),* dan *Denoising*	Informasi lokasi hilang saat downsampling dan fitur objek kecil menjadi blur	Model FEFD-YOLOV5 dengan mAP = 94.78%, dengan peningkatan 6.77% dibandingkan YOLOv5s asli.	Dengan SDH meningkatkan akurasi deteksi helm.	Parameter bertambah dan komputasi meningkat
Rihan et al. [39]	*Jurnal MAKE: Jurnal Teknik Mesin (2023)* *"Performance Analysis of YOLO, Faster R-CNN, and DETR for Automated Personal*	Total 5199 gambar dengan 9 kelas hardhat, vest, no helmet, no vest, dan person	model YOLOv11 ditingkatkan attention mechanism berupa SE dan loss function CIoU	Belum ada perbandingan YOLO, Faster R-CNN dan RT-DERR	Model YOLOv11 memiliki mAP = 0,770, *recall* 0,902	Menambahkan post processing logic untuk menentukan status pekerja yang compliant	Data augmentation tidak sama dan perbedaan pipeline training

Penulis	Nama Jurnal & Nama Artikel	Dataset	Metode	Masalah	Hasil	Kelebihan	Kekurangan
	Protective Equipment Detection"					dan non compliant	
Wang et al. [40]	*IEEE Access, vol. 10, pp. 60622-60632, (2022)* *"Investigation Into Recognition Algorithm of Helmet Violation Based on YOLOv5-CBAM-DCN"*	Total 7450 gambar yang diperoleh dari pengambilan langsung dilapangan	Implementasi metode YOLOv5 optimasi CBAM, DCN, dan fungsi *loss* DIoU	Akurasi detail rendah pada lingkungan kompleks dan sulit mengikuti bentuk objek yang tidak beraturan	Model YOLOv5-CBAM-DCN akurasi = 91.6% (peningkatan 2.3%	Model yang diusulkan mengingkatkan mAP hingga 0,916	Dataset terbatas dan perform menurun pada kondisi tertentu
Zhang, Y.-J., Xiao, F.-S., & Lu, Z.-M. [41]	*Sensors, 22(24), 9843 (2022)* *"Helmet Wearing State Detection Based on Improved Yolov5s"*	Total 8476 gambar dengan label person, helmet, hat_only, helmet_nowear, helmet_good, dan not_fastened	Penerapan algoritma YOLOv5s dimodifikasi dengan *redesign anchor box*, fungsi *loss* EIoU, dan *Attention Mechanism CoordAtt*	Dataset publik hanya menyediakan 2 kelas helmet dan person	YOLOv5s dimodifikasi dengan peningkatan *mean* mAP sebesar 3.9%, dengan hasil akhir 93.4%	Dataset memiliki klasifikasi lebih rinci dan peningkatan mAP	Beberapa kategori memiliki fitur visual yang hampir sama sehingga sulit dibedakan
Gopinath D. et al. [42]	International Journal of Intelligent Systems and Applications in Engineering (2024) "Detection of Helmet and License Plate Using Machine Learning"	Jumlah 11.000 gambar yang terdiri dari 5 objek	YOLOv8- TTA (Test Time Augmentation) untuk deteksi pelanggaran lalu lintas	Banyak kecelakaan terjadi karena pengendara tidak menggunakan helm, CCTV hanya merekam video tanpa deteksi	YOLOv8 + TTA menghasilkan Akurasi dengan mAP: 85% setelah 40,000 iterasi dengan 10,000+ gambar dan 4 kelas objek.	Fleksibel dalam penggunaan karema webcam, CCTV, kamera smartphone	Menggunakan model lama, kinerja sistem dipengaruhi jarak dan pencahayaan
P. Jin et al. [43]	*IEEE Xplore Vol. 12 (2024)*	Total 4400 gambar yang	Menerapkan ESCA-	Banyak pekerja berdekatan, dan	Hasil dari YOLO-ESCA	Dapat mendeteksi standar helm,	Penambahan small target layer

Penulis	Nama Jurnal & Nama Artikel	Dataset	Metode	Masalah	Hasil	Kelebihan	Kekurangan
	"YOLO-ESCA: A High-Performance Safety Helmet Standard Wearing Behavior Detection Model Based on Improved YOLOv5"	terdiri dari 3 objek yaitu *helmet*, *head*, *uncertainly*	YOLOv5n dengan optimasi Soft-NMS, CBAM, dan EIOU Loss function	objek masih saling menutupi	mencapai nilai mAP, yaitu 95.8% lebih baik daripada YOLOv5n.	mengurangi miss detection	menurunkan beberapa perform dan model ringan tapi masih bisa ditingkatkan

Tabel 2.1 menjelaskan berbagai pengembangan metode deteksi helm berbasis *deep learning* menunjukkan peningkatan performa yang signifikan, khususnya melalui pemanfaatan arsitektur YOLOv5 dan berbagai optimasinya. Optimalisasi YOLOv5 dengan penggunaan CIoU *loss*, teknik *data augmentation*, serta modul *Focus*, CSP, dan FPN-PAN mampu meningkatkan kemampuan model dalam mengekstraksi fitur objek serta memperbaiki akurasi deteksi pada berbagai skala objek. Penerapan metode tersebut menghasilkan performa mAP sebesar 94,9% dengan kecepatan 30,5 FPS, sehingga mampu mengatasi keterbatasan metode deteksi helm sebelumnya yang kurang optimal pada lingkungan kompleks serta memiliki jangkauan pemantauan terbatas [36]. Selain itu, pengembangan arsitektur YOLOv3 dengan integrasi *Residual Spatial Squeeze and Excitation* (RSSE) dan CIoU *loss* bertujuan meningkatkan kemampuan ekstraksi fitur spasial serta memperbaiki akurasi *bounding box*. Integrasi modul tersebut mampu meningkatkan *precision* menjadi 92,3%, recall 90,4%, F1-score 91,3%, dan mAP 92,8%, sehingga mampu mengurangi kesalahan deteksi pada objek helm yang memiliki variasi ukuran dan posisi dalam gambar [34].

Pengembangan lain pada arsitektur YOLO dilakukan dengan menambahkan berbagai mekanisme untuk meningkatkan performa deteksi pada kondisi kompleks seperti objek kecil, oklusi, dan latar belakang yang padat. Modifikasi YOLOv5 melalui penambahan modul MSCAM, DetectBlock, dan *Spatial transformer* digunakan untuk membantu model lebih fokus pada bagian objek yang dianggap penting serta meningkatkan kemampuan deteksi terhadap objek berukuran kecil. Penerapan modul tersebut mampu meningkatkan performa model dengan mAP sebesar 96,3% dan kecepatan deteksi 33 FPS, sehingga lebih efektif dalam mendeteksi helm yang berukuran kecil atau berada pada jarak jauh dari kamera [28]. Selain itu, integrasi BIFPN, ECA, dan *Decoupling Head* pada YOLOv5 bertujuan meningkatkan proses fusi fitur antar *layer* sehingga model mampu menangkap informasi objek pada berbagai skala dengan lebih baik. Pendekatan ini menghasilkan performa mAP 95,9%, recall 92,4%, dan precision 94,7%, yang menunjukkan peningkatan akurasi deteksi terutama pada kondisi lingkungan yang kompleks [37]. Penggunaan modul *attention* seperti SENet dan CBAM serta teknik

feature denoising pada model FEFD-YOLOv5 juga bertujuan mengurangi kehilangan informasi lokasi objek saat proses *downsampling* serta memperjelas fitur objek kecil. Integrasi metode tersebut mampu meningkatkan nilai mAP hingga mencapai 94,78%, sehingga membantu model mendeteksi objek helm secara lebih akurat pada berbagai kondisi pencahayaan [38].

Selain peningkatan akurasi, pengembangan metode juga berfokus pada efisiensi komputasi serta kemampuan implementasi *real-time*. Optimalisasi arsitektur YOLOv7 melalui integrasi *Ghost Module*, *Squeeze-and-Excitation* (SE), dan EIOU loss bertujuan mengurangi kompleksitas model serta meningkatkan efisiensi proses komputasi tanpa menurunkan performa deteksi. Pendekatan ini menghasilkan mAP sebesar 91,1% serta mampu mengurangi jumlah *parameter* model hingga 37,3%, sehingga lebih efisien untuk implementasi pada sistem pemantauan keselamatan kerja secara *real-time* [21]. Integrasi modul CBAM, DCN, serta fungsi loss DIoU pada YOLOv5 juga bertujuan meningkatkan kemampuan model dalam mengikuti bentuk objek yang tidak beraturan serta memperbaiki akurasi deteksi pada lingkungan kompleks. Pendekatan tersebut mampu meningkatkan akurasi model hingga 91,6% dibandingkan dengan model YOLOv5 dasar [40]. Selain itu, penerapan CoordAtt dan EIoU loss pada YOLOv5s mampu meningkatkan kemampuan model dalam mempelajari hubungan spasial antar fitur sehingga menghasilkan peningkatan mAP sebesar 3,9% dibandingkan model sebelumnya [41]. Peningkatan performa serupa juga diperoleh melalui model YOLO-ESCA yang mengombinasikan Soft-NMS, CBAM, dan EIOU loss, sehingga mampu mengurangi kesalahan deteksi pada objek yang saling menutupi dan menghasilkan mAP sebesar 95,8% [43].

Seiring dengan perkembangan tersebut, penelitian juga mulai berfokus pada keseimbangan antara akurasi, kecepatan, serta efisiensi komputasi melalui pengembangan arsitektur YOLO generasi terbaru seperti YOLOv11. Penambahan modul C3k2-SCConv, iAFF berbasis MSCAM, MultiSEAM, serta ADown pada YOLOv11n dikembangkan untuk meningkatkan kemampuan model dalam mengenali objek berukuran kecil yang berada pada area latar belakang dengan tetap mempertahankan efisiensi jumlah parameter model. Pendekatan ini menghasilkan *precision* 88,9%, *recall* 82,5%, dan mAP 88,2% dengan jumlah *parameter* yang

relatif kecil yaitu 2,69 juta *parameter*, sehingga cocok untuk implementasi sistem deteksi helm yang ringan [24]. Selain itu, penerapan YOLOv11 pada sistem deteksi berbasis CPU dan GPU menunjukkan keseimbangan yang baik antara akurasi dan efisiensi komputasi dengan mAP sebesar 0,939 dan recall 0,902, sehingga membantu implementasi pada sistem CCTV yang tidak memiliki GPU berperforma tinggi [31]. Integrasi *attention* SE serta optimasi CIoU loss pada model YOLOv11 juga mampu meningkatkan kemampuan model dalam membedakan objek pekerja yang menggunakan helm dan tidak menggunakan helm dengan mAP sebesar 0,770 dan recall 0,902, sehingga mendukung sistem pemantauan kepatuhan penggunaan alat keselamatan kerja [39].

Selain itu, beberapa pendekatan lain dikembangkan untuk mengatasi permasalahan spesifik seperti gangguan latar belakang, objek yang sangat kecil, serta keterbatasan data. Penggunaan YOLOv10 dengan modul DBConv dan *Global-Local Fusion Module* (GLFM) bertujuan meningkatkan kemampuan model dalam memahami konteks global dan detail lokal secara bersamaan sehingga dapat mengurangi kesalahan deteksi pada latar belakang yang kompleks. Pendekatan ini menghasilkan akurasi 94,4% dan mAP 95,4% serta mampu menurunkan kesalahan deteksi latar belakang hingga 14% [25]. Penerapan YOLOv8 dengan teknik *Test Time Augmentation* (TTA) juga bertujuan meningkatkan *robustnes* model terhadap variasi sudut pandang dan kondisi lingkungan, sehingga mampu menghasilkan mAP sebesar 85% pada sistem deteksi pelanggaran penggunaan helm [42]. Selain itu, integrasi modul C2PSA, C3k2 block, SPPF, serta teknik *data augmentation* pada YOLOv11 mampu meningkatkan kemampuan model dalam memfokuskan perhatian pada area objek penting serta mempercepat proses inferensi hingga mencapai 290 FPS, sehingga mendukung implementasi sistem deteksi berbasis *real-time* [35].

Berdasarkan analisis tersebut, Pada penelitian ini mengimplementasikan model YOLOv11s yang telah dimodifikasi untuk mengatasi keterbatasan penelitian sebelumnya. Penelitian ini berfokus pada peningkatan performa deteksi melalui pendekatan *attention mechanism*, optimalisasi arsitektur, serta penyesuaian *loss function*. Mekanisme yang digunakan meliputi MSCAM untuk meningkatkan fokus fitur penting, DetectBlock untuk meningkatkan deteksi objek kecil, serta

optimalisasi *backbone* dan *neck* untuk memperkuat ekstraksi dan fusi fitur. Selain itu, digunakan *loss function* GIoU untuk meningkatkan akurasi *bounding box*. Dengan pendekatan ini, diharapkan model mampu meningkatkan *precision*, *recall*, serta efisiensi komputasi sehingga lebih optimal dalam mendeteksi helm pada lingkungan lalu lintas yang kompleks.

2.2 Teori yang berkaitan

2.2.1 *Computer Vision*

Computer vision merupakan salah satu cabang ilmu dalam bidang teknologi informasi yang berfokus pada bagaimana sebuah sistem dapat "melihat" dan memahami informasi dari citra digital [44]. Bidang ini berkembang untuk meniru cara kerja penglihatan manusia dalam mengenali objek, memahami konteks *visual*, hingga mengambil keputusan berdasarkan data yang diperoleh dari gambar atau video [44]. Dalam praktiknya, *computer vision* tidak berdiri sendiri, melainkan berkaitan erat dengan pengolahan citra (*image processing*) yang berperan dalam meningkatkan kualitas gambar melalui proses seperti pengurangan *noise*, *filtering*, serta penyesuaian warna sebelum dilakukan analisis lebih lanjut [45].

1. *Image Detection*

Image detection merupakan salah satu bidang dalam *computer vision* yang digunakan untuk mengenali sekaligus menentukan lokasi objek pada gambar maupun *video*. Proses ini melibatkan dua tugas utama: *classification* (mengklasifikasikan objek) dan *localization* (menentukan posisi objek dengan bounding box) [46]. Dengan kemajuan *deep learning*, model berbasis CNN seperti R-CNN, YOLO, dan SSD kini dominan digunakan karena mampu melakukan deteksi secara otomatis dan akurat [47]. Meskipun demikian, tantangan seperti *occlusion*, variasi skala objek, pencahayaan, dan *noise* masih ada. Untuk mengatasinya, teknik seperti *attention mechanism* dan *multi-task learning* digunakan untuk meningkatkan efisiensi dan akurasi deteksi. Metrik seperti mAP, *precision*, *recall*, dan IoU digunakan untuk mengevaluasi kinerja model [48]. Seiring berjalannya waktu, *image detection* terus berkembang untuk aplikasi yang lebih cepat dan lebih akurat dalam berbagai bidang.

2. *Image Classification*

Image classification merupakan salah satu metode pada bidang *computer vision* yang digunakan untuk mengenali dan mengklasifikasikan objek pada gambar secara otomatis ke dalam kategori tertentu [49]. Proses ini meniru cara manusia dalam mengidentifikasi objek melalui penglihatan, namun dilakukan menggunakan algoritma pembelajaran mesin yang mampu mempelajari pola dan karakteristik dari data gambar. Perkembangannya sangat dipengaruhi oleh kemajuan *machine learning*, terutama *deep learning* dengan arsitektur jaringan saraf tiruan seperti *Convolutional Neural Network* (CNN) yang dirancang khusus untuk menangkap pola spasial pada gambar. Selain itu, metode lain seperti *Support Vector Machine* (SVM) juga sering digunakan karena kemampuannya dalam memisahkan data berdimensi tinggi secara optimal.

2.2.2 *Image Processing*

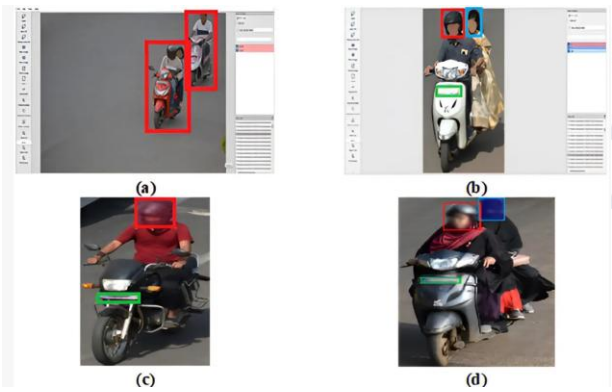
Image processing merupakan proses pengolahan gambar digital menggunakan algoritma dan pendekatan matematis untuk melakukan analisis maupun manipulasi gambar [50]. Tujuan *image processing* adalah untuk meningkatkan kualitas gambar, mengekstrak informasi yang berguna dari gambar, dan mengotomatiskan operasi yang berhubungan dengan gambar [50]. Langkah dalam *Image processing* mencakup mengolah dan menganalisis gambar. Proses diawali dengan pengambilan gambar menggunakan kamera, kemudian dilanjutkan dengan peningkatan kualitas visual, seperti memperbaiki kontras, mengurangi *noise*, maupun menghilangkan artefak pada gambar [24][51]. Pemulihan gambar bertujuan mengatasi degradasi seperti keburaman atau distorsi, sedangkan segmentasi membagi gambar menjadi beberapa bagian untuk mengidentifikasi objek atau fitur tertentu. Setelah itu, gambar direpresentasikan agar dapat dianalisis oleh komputer menggunakan algoritma untuk mengenali objek, mendeteksi pola, atau mengukur fitur [50]. Langkah lainnya meliputi sintesis dan kompresi gambar untuk efisiensi penyimpanan dan transmisi. *Image processing* digunakan secara luas dalam pencitraan medis, penginderaan jauh, visi komputer, dan multimedia [50].

2.2.3 *Deep Learning*

Deep learning merupakan salah satu metode pembelajaran mesin yang memanfaatkan jaringan saraf tiruan untuk meniru cara kerja otak manusia dalam memproses informasi [52]. Metode ini membantu pemodelan yang lebih akurat untuk data yang rumit [53]. *Deep learning* membantu model komputasi yang terdiri dari beberapa lapisan pemrosesan untuk mempelajari representasi data pada berbagai tingkat abstraksi [54]. Metode ini telah memberikan peningkatan yang signifikan dalam berbagai bidang, seperti pengenalan suara, pengenalan objek visual, deteksi objek, hingga penerapan pada penemuan obat dan analisis *genomic* [55]. *Deep learning* mampu mempelajari pola yang kompleks pada kumpulan data berukuran besar dengan memanfaatkan algoritma *backpropagation* untuk menyesuaikan parameter internal model pada setiap lapisan berdasarkan hasil dari lapisan sebelumnya. [56].

2.2.4. *Data Labelling*

Pelabelan data adalah merupakan langkah prapemrosesan data, data dimulai dari data mentah diberi label atau anotasi agar dapat digunakan untuk melatih model pembelajaran mesin. Proses ini sangat bergantung pada jenis data yang dikumpulkan, seperti gambar, audio, teks, atau data sensor, serta metode yang digunakan [57]. Pelabelan dapat dilakukan secara manual oleh manusia, melalui *crowdsourcing*, atau secara otomatis menggunakan algoritma [58]. Untuk data yang membutuhkan konteks waktu nyata, seperti data sensor, pelabelan sering kali dilakukan secara waktu nyata selama pengumpulan data untuk menjaga integritas dan relevansi data, untuk lebih jelas gambar dapat dilihat pada Gambar 2.1.

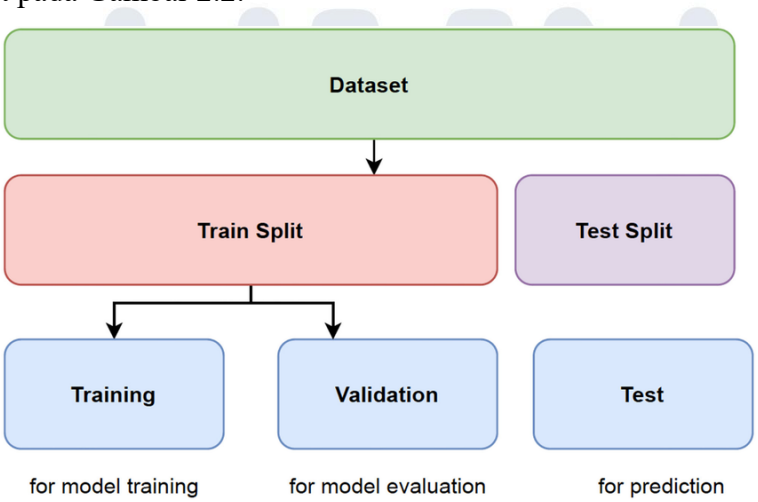


Gambar 2.1 *Data Labeling Image*
Sumber: [59]

Tantangan dalam pelabelan data termasuk memastikan akurasi pelabelan, menjaga konsistensi, dan mencegah kelelahan pengguna selama proses pelabelan. Pelabelan secara langsung menawarkan beberapa keuntungan, termasuk efisiensi waktu dan biaya, serta kemampuan untuk mengumpulkan data secara langsung di lapangan menggunakan antarmuka khusus seperti *Tangible User Interface* (TUI), yang mengintegrasikan sensor fisik dengan mekanisme pelabelan [60]. Pendekatan ini biasanya digunakan untuk pengumpulan data, seperti pemantauan aktivitas manusia, pelacakan emosi dan kesehatan, atau pengumpulan data lingkungan. Pelabelan yang tepat membantu dalam pengembangan model pembelajaran mesin yang lebih akurat, yang mendukung aplikasi di berbagai bidang. *Data labelling* memberikan anotasi yang dibutuhkan untuk melatih model dalam tugas-tugas seperti pengenalan gambar, pengenalan objek, dan analisis gambar medis [61].

2.2.5. *Data Splitting*

Data splitting adalah pendekatan mendasar dalam analisis statistik dan *machine learning*, yang penting untuk mengevaluasi kinerja model dan memastikan kemampuan generalisasinya. Proses ini melibatkan pembagian dataset ke dalam subset yang berbeda, biasanya set pelatihan dan pengujian, untuk menghindari *overfitting* dan untuk mengukur akurasi prediksi secara efektif [62], Proses *splitting* dapat dilihat pada Gambar 2.2.



Gambar 2.2 *Data Splitting Diagram*
Sumber: [63]

Rasio yang umum diadopsi untuk pemisahan termasuk 80:20 dan 70:30, meskipun ini sebagian besar bervariasi tergantung pada konteksnya. Rasio

optimal bergantung pada faktor-faktor seperti ukuran dataset dan jumlah parameter dalam model. Hasil ini menunjukkan bahwa rasio optimal menyeimbangkan kebutuhan akan data yang cukup untuk melatih model dan kebutuhan akan evaluasi yang kuat dengan menggunakan set *testing* [64].

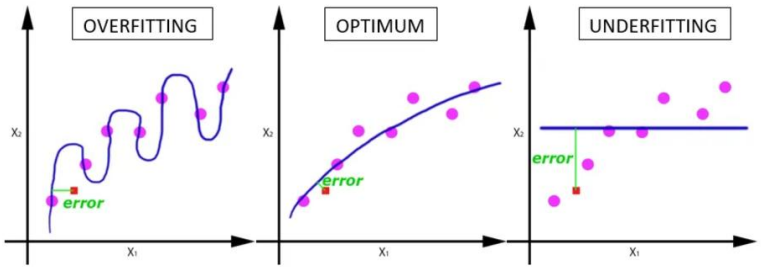
2.2.6. *Epoch*

Epoch merujuk pada satu putaran di mana seluruh dataset digunakan untuk melatih model. *Epoch training* melibatkan pengulangan dataset ini beberapa kali hingga model mencapai performa yang optimal [65]. Dalam konteks tradisional, pendekatan *multi-epoch* umum digunakan, terutama dalam pengaturan data terbatas seperti klasifikasi yang diawasi (*supervised classification*). Namun, pendekatan ini dapat menjadi tidak efisien ketika diterapkan pada pembelajaran *unsupervised learning* dengan dataset yang besar. efisiensi pelatihan dapat ditingkatkan melalui pelatihan satu *epoch* (*one epoch training*), di mana dataset diperbesar untuk memastikan setiap sampel hanya diproses sekali selama *training*. Pendekatan ini mengurangi risiko *overfitting*, karena tidak ada pengulangan sampel, sehingga performa model lebih representatif terhadap data nyata [66]. Untuk lebih jelasnya Gambar 2.3 merupakan grafik *epoch* Ketika terjadi *overfitting, optimum,* dan *underfitting* dan Tabel 2.2 [66] merupakan jumlah epoch per masing-masing jumlah dataset.

Tabel 2.2 Jumlah Epoch

Jumlah Dataset	***Rekomendasi Epoch***
< 1000 gambar	20 – 40 Epoch
1000 – 5000 gambar	40 – 100 Epoch
>5000 Gambar	100 – 150 Epoch

Berdasarkan Tabel 2.2, jumlah epoch yang digunakan dalam proses training umumnya disesuaikan dengan jumlah dataset yang dimiliki. Dataset dengan jumlah gambar yang lebih besar memerlukan epoch lebih banyak agar model dapat mempelajari pola data secara optimal dan menghasilkan performa deteksi yang lebih baik [66].



Gambar 2.3 Grafik *Epoch Training*
Sumber: [67]

Underfitting terjadi ketika model tidak cukup belajar dari data, sedangkan *overfitting* terjadi ketika model terlalu banyak belajar detail yang tidak relevan. Titik *optimum* tercapai ketika model seimbang, dengan performa baik pada data *training* dan *testing*. Ketiga kondisi ini dipengaruhi oleh jumlah *epoch* dalam *training* [66].

2.2.7. *Evaluation*

Matriks evaluasi membantu untuk memberikan bobot pada berbagai ide, dengan menilai masing-masing berdasarkan serangkaian kriteria yang telah ditetapkan, untuk mengidentifikasi ide yang paling relevan. Kriteria yang umum digunakan mencakup tingkat kompleksitas yang terkait dengan implementasi ide, serta potensi nilai tambah yang dapat diberikan, baik untuk pengguna maupun untuk organisasi secara keseluruhan [68]. Hal ini membantu dalam memilih ide yang tidak hanya informatif, tetapi juga berdampak positif dan berkelanjutan.

2.2.8. *Confusion Matrix*

Confusion matrix merupakan salah satu metode evaluasi yang digunakan untuk mengukur kinerja algoritma *machine learning*, khususnya pada tugas klasifikasi. Matriks ini memvisualisasikan distribusi hasil prediksi terhadap label aktual dalam format dua dimensi. Setiap baris dalam matriks merepresentasikan *actual label*, sementara setiap kolom sesuai dengan label yang diprediksi dari model [69]. *Confusion matrix* membantu penghitungan metrik kinerja seperti *precision*, *recall*, *F1-score*, dan akurasi dengan memanfaatkan nilai *True Positive* (TP), *True Negative* (TN), *False Positive* (FP), dan *False Negative* (FN) yang dijelaskan lebih rinci pada Tabel 2.3.

Tabel 2.3 *Confusion Matrix* [70]

Model AI	***Actual Positive* (1)**	**Akurasi Visi Komputer**
Predicted Positive (1)	*True Positive (TP)*	*False Positive (FP)*
Predicted Negative (0)	*False Negative (FN)*	*True Negative (TN)*

Tabel 2.3, *True Positive (TP)* mewakili contoh di mana model memprediksi dengan benar kelas positif, sementara *True Negative (TN)* menunjukkan prediksi yang benar dari kelas negatif. *False Negative (FP)* terjadi ketika model salah memprediksi kelas positif untuk contoh yang sebenarnya negatif, yang mengarah ke "*false alarm*". Sebaliknya, *False Negative (FN)* terjadi ketika model gagal memprediksi kelas positif untuk contoh yang sebenarnya positif, yang mengakibatkan *"missed detection"* [71].

2.2.9. *Recall*

Recall merupakan salah satu metrik evaluasi yang digunakan untuk mengukur kemampuan model dalam mendeteksi atau mengenali seluruh data yang relevan. Secara klasik, *recall* mengukur sejauh mana suatu model mampu mencakup seberapa besar proporsi data sebenarnya yang berhasil dihasilkan kembali oleh model tersebut. Dalam konteks ini, *manifold data* adalah representasi matematis dari ruang kemungkinan data yang sebenarnya, yang menjadi target untuk direplikasi oleh model *generative* [72]. *Recall* juga menangani ambiguitas dalam memisahkan kesalahan karena model gagal mencakup *manifold* sebenarnya (masalah recall) dari kesalahan akibat menghasilkan data yang tidak realistis [73]. Rumus dari *Recall* dapat dilihat pada Rumus 2.1 [74].

$$Recall = \frac{TP}{TP+FN} \tag{2.1}$$

Rumus 2.1 Perhitungan nilai *Recall*

Keterangan:

1. *TP (True Positive)*: Jumlah kasus positif yang berhasil diprediksi dengan benar oleh model.
2. *FN (False Negative)*: Jumlah kasus positif yang tidak terdeteksi oleh model (dianggap negatif secara salah).

Recall adalah metrik evaluasi yang digunakan untuk mengukur kemampuan model dalam mengidentifikasi seluruh data yang benar-benar positif. Nilai *recall*

yang tinggi menunjukkan bahwa model mampu menangkap sebagian besar kasus positif yang ada dalam data, meskipun dalam beberapa kondisi masih dapat muncul prediksi positif yang tidak tepat (*false positive*).

2.2.10. *Precision*

Precision merupakan metrik evaluasi yang digunakan untuk menilai ketepatan prediksi positif yang dihasilkan oleh model [75]. Secara matematis, *precision* dapat dinyatakan sebagai rasio antara *True Positives (TP)* dan *total Predicted Positives (TP + FP)*, dengan FP adalah *False Positives* Rumus dapat dilihat pada Rumus $Precision = \frac{TP}{TP+FP}$ (2.2) [74]. Metrik ini menjadi penting dalam konteks di mana konsekuensi dari prediksi positif yang salah (*False Positives*) dapat signifikan, seperti dalam diagnosis medis atau sistem deteksi keamanan [76]. *Precision* memastikan bahwa prediksi yang dihasilkan oleh model benar-benar relevan. Namun, *precision* memiliki keterbatasan karena tidak mempertimbangkan kemampuan model dalam menangani kasus negatif. Oleh karena itu, metrik ini biasanya digunakan bersama dengan *recall* agar dapat memberikan gambaran yang lebih menyeluruh mengenai performa model dalam melakukan klasifikasi [77]. Gabungan antara *Precision* dan *Recall* sering dirangkum dalam metrik *F1-Score.*

$$Precision = \frac{TP}{TP+FP} \quad (2.2)$$

Rumus 2.2 Perhitungan Nilai *Precision*

Keterangan:

1. *True Positive (TP)*: Jumlah kasus di mana model memprediksi positif, dan prediksi tersebut benar (contohnya, model mendeteksi objek tertentu yang memang benar-benar ada).
2. *False Positive (FP)*: Jumlah kasus di mana model memprediksi positif, tetapi prediksi tersebut salah (contohnya, model mendeteksi objek yang sebenarnya tidak ada).

Precision menilai akurasi hasil prediksi positif yang dilakukan model. Dengan membandingkan TP dengan total prediksi positif (TP + FP), *precision* menunjukkan seberapa andal model dalam memprediksi sesuatu sebagai positif.

2.2.11. mAP

Mean Average Precision (mAP) merupakan metrik evaluasi yang umum digunakan dalam bidang pengambilan informasi dan pengenalan pola, termasuk dalam pengambilan gambar berbasis *deep learning*. mAP mengevaluasi kemampuan model untuk memberikan peringkat yang relevan berdasarkan *query* tertentu [78]. Metode ini menghitung rata-rata presisi untuk semua kelas, dengan mempertimbangkan urutan relevansi dari data yang diambil. Tidak seperti metrik lain, seperti *recall* atau *F1-score*, mAP lebih independen terhadap ambang batas tertentu dan memiliki kelebihan dalam menangani variasi jumlah data relevan per query [79]. mAP sendiri mengukur kemampuan model dalam mendeteksi suatu kelas tertentu berdasarkan *Precision* dan *Recall* pada berbagai *threshold*. Rumus dari AP dapat dapat dilihat pada Rumus 2.3 [80].

$$AP = \frac{1}{11}\sum_{Recall_i} Precision(Recall_i) \tag{2.3}$$

Rumus 2.3 Perhitungan nilai AP

Rumus (2.3) digunakan untuk menghitung rata-rata presisi AP(*Average Precision*) pada 11 titik tetap *recall* yang berkisar dari 0,0 hingga 1,0. Pada setiap titik *recall*, nilai presisi tertinggi yang dicapai oleh model diambil, dan rata-rata dari semua nilai *precision* ini dihitung. Proses ini digunakan untuk mengevaluasi kinerja model dalam mendeteksi suatu kelas objek dengan memperhitungkan keseimbangan (*trade-off*) antara *precision* dan *recall*. AP yang tinggi menunjukkan bahwa model memiliki kinerja yang baik, yaitu mampu menghasilkan prediksi yang tepat (*precision* tinggi) sekaligus dapat menjangkau sebagian besar objek yang relevan dalam data (*recall* tinggi). [81]. Sehingga rumus mAP dapat dirumuskan pada Rumus 2.4 [80].

$$mAP = \frac{1}{N}\sum_{i=1}^{N} AP_i \tag{2.4}$$

Rumus 2.4 Perhitungan nilai mAP

Nilai *mean Average Precision* (mAP) berkisar antara 0 hingga 1, dengan nilai lebih tinggi menunjukkan model yang lebih akurat dalam mendeteksi objek di berbagai kelas. mAP dihitung pada berbagai *threshold Intersection over Union* (IoU), seperti mAP@0.5, yang menggunakan IoU minimal 50%, atau

mAP@[0.5:0.95], yang rata-rata mAP pada IoU dari 50% hingga 95%, memberikan gambaran kinerja model pada berbagai tingkat tumpang tindih antara prediksi dan objek yang benar.

2.2.12. *F1-Score*

F1-Score merupakan salah satu metrik evaluasi yang sering digunakan untuk mengukur kinerja model klasifikasi, terutama pada kasus dengan data yang tidak seimbang (*imbalanced data*) [82]. *F1-Score* menggabungkan dua metrik utama, yaitu *precision* dan *recall*, menjadi satu nilai evaluasi yang mencerminkan keseimbangan antara ketepatan prediksi positif yang dihasilkan model serta kemampuannya dalam menemukan seluruh objek positif yang sebenarnya ada dalam data. [82][83]. Dalam konsep evaluasi klasifikasi, *precision* menggambarkan seberapa akurat prediksi positif yang dihasilkan model [75], sedangkan *recall* menunjukkan kemampuan model dalam menemukan seluruh objek positif pada data [72]. Oleh karena itu, *F1-Score* digunakan untuk memberikan gambaran terhadap performa model dibandingkan hanya menggunakan salah satu metrik saja. Secara matematis, *F1-Score* didefinisikan sebagai harmonic mean dari *precision* dan *recall* sehingga memberikan penalti yang lebih besar apabila salah satu nilai memiliki nilai yang sangat rendah. Rumus *F1-Score* dapat dilihat pada Rumus 2.5 [82].

$$F1 - Score = 2\ x\ \frac{(Precision\ x\ Recall)}{(Precision + Recall)} \tag{2.5}$$

Rumus 2.5 Perhitungan nilai *F1-Score*

Keterangan:

1. *Precision* (P) : Rasio antara jumlah prediksi positif yang benar terhadap seluruh prediksi positif yang dilakukan model.
2. *Recall* (R) : Rasio antara jumlah prediksi positif yang benar terhadap seluruh data yang sebenarnya positif.

F1-Score akan menghasilkan nilai yang tinggi apabila *precision* dan *recall* sama-sama bernilai tinggi. Oleh karena itu, metrik ini bermanfaat dalam mengevaluasi kinerja model klasifikasi di berbagai bidang seperti *machine learning*, *computer vision*, dan *data analytics* [84]. Jika salah satu dari *precision*

atau *recall* memiliki nilai yang rendah, maka nilai *F1-Score* juga akan menurun karena sifat *harmonic mean* yang lebih sensitif terhadap nilai yang kecil. Dengan demikian, *F1-Score* membantu peneliti dalam menilai keseimbangan performa model dalam menghindari kesalahan prediksi positif maupun kegagalan mendeteksi objek yang seharusnya terdeteksi [82][84].

2.2.13. *User Acceptance Testing* (UAT)

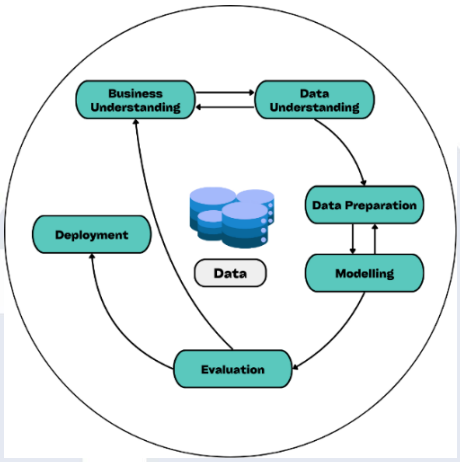
User Acceptance Testing (UAT), juga dikenal sebagai *application testing* atau *end-user testing*, adalah fase pengujian perangkat lunak di mana sistem diuji dalam kondisi nyata oleh audiens yang dituju. UAT sering kali merupakan fase terakhir dalam proses pengujian perangkat lunak dan dilakukan sebelum perangkat lunak yang telah diuji dirilis ke pasar yang dituju [85]. Pengujian UAT pada umumnya dilakukan sebelum peluncuran sebuah fitur baru di dalam aplikasi. Dengan melakukan ini, pengembang dapat memahami apakah rancangan yang dibuat sudah memenuhi harapan pengguna [86]. Pengujian UAT difokuskan pada lima variabel utama dalam mengevaluasi sistem, di antaranya adalah fungsionalitas sistem, kinerja sistem, pengalaman dan tampilan antarmuka sistem, Efisiensi dan produktivitas, serta keamanan dan keandalan sistem [87]. Umumnya UAT dilakukan dengan skala Likert melalui kuesioner yang diujicobakan menggunakan skenario UAT untuk memastikan keterbacaan, konsistensi respons, dan kemudahan pengisian [85].

2.3 Framework/Algoritma yang digunakan

2.3.1 *Cross-Industry Standard Process for Data Mining* (CRISP-DM)

CRISP-DM (*Cross-Industry Standard Process for Data Mining)* adalah pendekatan metodologis yang umum digunakan untuk mengekstraksi pengetahuan dari kumpulan data besar [88]. Tujuan utama dari CRISP-DM adalah mengolah data mentah menjadi informasi yang bernilai dan mendalam, sehingga dapat dimanfaatkan dalam proses pengambilan keputusan berbasis data [89]. Meskipun sering disamakan dengan istilah data mining, CRISP-DM lebih luas karena mencakup seluruh proses, mulai dari *business understanding* hingga *deployment* yang dapat dilihat pada Gambar 2.4, sementara data mining hanya berfokus pada algoritma pembelajaran mesin untuk menemukan pola dalam data. CRISP-DM

terdiri dari enam tahap utama: *Business Understanding, Data Understanding, Data Preparation, Modeling, Evaluation,* dan *Deployment*. Setiap tahap bertujuan untuk mengonversi masalah atau pertanyaan menjadi solusi berbasis data yang dapat diterapkan secara praktis, sekaligus mendorong pengembangan aplikasi baru yang inovatif berdasarkan wawasan yang diperoleh dari data [90].



Gambar 2.4 *Framework* CRISP-DM
Sumber: [91]

Gambar 2.4 berisikan sebuah alur pada *framework* CRISP-DM yang terdiri dari 6 tahapan utama yaitu *Business Understanding, Data Understanding, Data Preparation, Modeling, Evaluation,* dan *Deployment* [92]. Berikut adalah penjelasan detail dari setiap langkah dalam proses CRISP-DM [91] :

1. *Business Understanding*

   Tahap awal ini bertujuan untuk memahami tujuan bisnis dan bagaimana data mining dapat membantu mencapainya. Proses ini melibatkan identifikasi kebutuhan bisnis, tujuan analitik, dan kriteria keberhasilan. Hasil dari tahap ini mencakup pemahaman yang jelas mengenai apa yang ingin dicapai dengan data mining serta strategi untuk mencapainya.
2. *Data Understanding*

   Langkah ini fokus pada eksplorasi awal data yang tersedia. Melibatkan pengumpulan data dari berbagai sumber, memahami karakteristiknya, dan mendeteksi masalah seperti data yang hilang atau *outlier*. Visualisasi data dan

statistik deskriptif sering digunakan untuk memahami pola awal dalam data. Hasil dari tahap ini adalah insight tentang kualitas dan karakteristik data.

3. *Data Preparation*

   Pada tahap ini, data yang telah dieksplorasi diproses lebih lanjut untuk digunakan dalam modeling. Aktivitas meliputi pembersihan data (*cleaning*), integrasi data, transformasi atribut, penghilangan data yang tidak relevan, dan pengisian nilai yang hilang. Hasil akhir dari tahap ini adalah dataset terstruktur yang siap untuk tahap modeling.

4. *Modeling*

   Pada tahap ini, algoritma *data mining* diterapkan untuk membangun model. Model dipilih berdasarkan tujuan analitik, seperti klasifikasi, *clustering*, atau regresi. Parameter model diuji dan disesuaikan untuk mendapatkan performa terbaik. Hasil dari tahap ini adalah model yang dapat memprediksi atau menemukan pola dalam data.

5. *Evaluation*

   Tahap evaluasi menilai apakah model yang dihasilkan memenuhi kebutuhan bisnis yang telah ditentukan. Melibatkan pengujian model terhadap data yang belum pernah dilihat sebelumnya untuk mengevaluasi performa dan validitasnya. Model yang tidak memenuhi tujuan bisnis akan dimodifikasi atau diganti. Hasil dari tahap ini adalah model akhir yang siap digunakan.

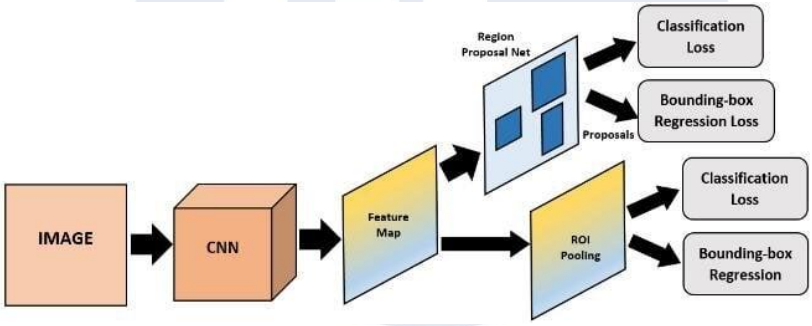
6. *Deployment*

   Langkah terakhir adalah mengimplementasikan model yang telah dievaluasi ke dalam lingkungan operasional. Proses ini melibatkan penyajian hasil kepada pemangku kepentingan, integrasi model dengan sistem bisnis, atau pembuatan laporan analisis yang dapat digunakan untuk pengambilan keputusan. Hasil dari tahap ini adalah pengetahuan yang dapat langsung diterapkan untuk mencapai tujuan bisnis.

Tahap tersebut mulai dari *business understanding* hingga *deployment*, memastikan bahwa data yang digunakan memiliki kualitas tinggi, relevan, dan dapat memberikan wawasan yang bermakna. Setiap langkah saling terhubung, di mana hasil dari satu tahap menjadi input untuk tahap berikutnya.

2.3.2 Faster R-CNN

Algoritma *Faster Region-based Convolutional Neural Network* (Faster R-CNN) merupakan algoritma *two-stage object detection* yang diperkenalkan oleh Ren dkk. pada tahun 2015 sebagai pengembangan dari R-CNN dan Fast R-CNN. Faster R-CNN mengintegrasikan proses Region Proposal ke dalam jaringan menggunakan *Region Proposal Network* (RPN) sehingga menghasilkan proses deteksi objek yang lebih cepat dibandingkan metode sebelumnya dengan tetap mempertahankan akurasi yang tinggi. Arsitektur Faster R-CNN terdiri atas beberapa komponen utama yaitu *Image*, *CNN Backbone*, *Feature Map*, *Region Proposal Network* (RPN), *RoI Pooling*, serta *Prediction Head* seperti ditunjukkan pada Gambar 2.5.

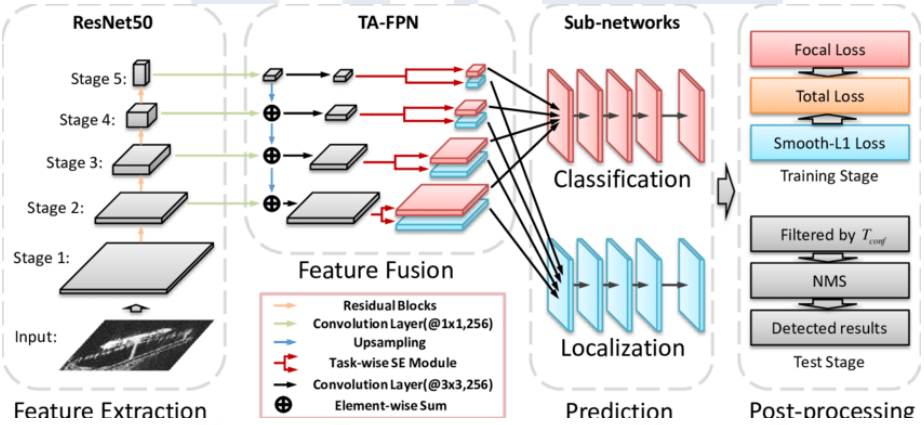


Gambar 2. 5 Arsitektur Faster R-CNN
Sumber: [93]

Berdasarkan Gambar 2.5, arsitektur *Faster* R-CNN terdiri atas beberapa komponen utama, yaitu Image, CNN *Backbone*, *Feature Map*, *Region Proposal Network* (RPN), *RoI Pooling*, *Classification Loss*, dan *Bounding Box Regression Loss*. *Image* merupakan gambar masukan yang diproses oleh CNN *Backbone* untuk mengekstraksi fitur penting sehingga menghasilkan *Feature Map*. *Feature map* kemudian digunakan oleh RPN untuk menghasilkan region proposal atau kandidat lokasi objek beserta koordinat awal *bounding box*. Selanjutnya, proposal tersebut diproses menggunakan *RoI Pooling* agar memiliki ukuran fitur yang seragam sebelum dilakukan proses klasifikasi objek dan penyempurnaan koordinat *bounding box*. Selama proses pelatihan, model menggunakan *Classification Loss* untuk mengukur kesalahan klasifikasi objek dan *Bounding Box Regression Loss* untuk mengoptimalkan ketepatan posisi *bounding box* yang diprediksi.

2.3.3 RetinaNet

Algoritma RetinaNet merupakan salah satu algoritma *one-stage object detection* yang diperkenalkan oleh Tsung-Yi Lin dkk. pada tahun 2017. RetinaNet dikembangkan untuk mengatasi permasalahan ketidakseimbangan antara jumlah sampel positif dan negatif pada proses deteksi objek melalui penerapan *Focal Loss*, sehingga model dapat lebih fokus mempelajari sampel yang sulit diklasifikasikan. Dengan menggabungkan *backbone* ResNet sebagai ekstraktor fitur dan *Feature Pyramid Network* (FPN) untuk menghasilkan representasi fitur multi-skala, RetinaNet mampu mendeteksi objek dengan berbagai ukuran secara efektif. Seperti ditunjukkan pada Gambar 2.6, arsitektur RetinaNet terdiri atas empat bagian utama, yaitu *Feature Extraction*, *Feature Fusion*, *Prediction*, dan *Post-processing*.



Gambar 2 6 Arsitektur RetinaNet
Sumber: [94]

Berdasarkan Gambar 2.6, arsitektur RetinaNet terdiri atas beberapa komponen utama, yaitu *Feature Extraction*, *Feature Fusion*, *Prediction*, dan *Post-processing*. Pada tahap *Feature Extraction*, citra masukan diproses menggunakan *backbone* ResNet50 untuk mengekstraksi fitur pada berbagai tingkatan sehingga menghasilkan *feature map* dengan informasi spasial. Selanjutnya, *feature map* tersebut digabungkan pada tahap *Feature Fusion* menggunakan *Task-Aware Feature Pyramid Network* (TA-FPN) melalui proses *upsampling*, *convolution* layer, residual block, dan task-wise SE module sehingga menghasilkan representasi fitur multi-skala yang lebih baik untuk mendeteksi objek dengan ukuran yang bervariasi. Tahap *Prediction* terdiri atas dua *sub-network*, yaitu *Classification* yang bertugas memprediksi kelas objek dan *Localization* yang berfungsi menentukan

koordinat *bounding box*. Selama proses training, RetinaNet menggunakan *Focal Loss* untuk mengatasi ketidakseimbangan jumlah sampel positif dan negatif serta *Smooth L1 Loss* untuk mengoptimalkan regresi *bounding box*. Pada tahap *Post-processing*, hasil prediksi disaring berdasarkan nilai *confidence threshold* dan diproses menggunakan *Non-Maximum Suppression* (NMS) untuk menghilangkan deteksi yang saling tumpang tindih sehingga menghasilkan deteksi objek akhir yang lebih akurat.

2.3.4 YOLO

YOLO pertama kali diperkenalkan oleh Redmon et al. sebagai metode deteksi objek satu tahap end-to-end secara real-time. Pendekatan ini menjadi dasar perkembangan selanjutnya dari seri YOLO [95]. Hingga saat ini, seri YOLO beserta variannya telah menjadi detektor objek real-time utama yang digunakan secara luas. YOLO merupakan algoritma untuk deteksi objek dalam gambar yang dirancang untuk mendeteksi berbagai objek secara *real-time*. Dibandingkan dengan metode tradisional yang memproses gambar dalam beberapa tahap , YOLO mengubah pendekatan tersebut dengan memproses seluruh gambar secara langsung dalam satu kali pemrosesan (*single-pass*), sehingga lebih cepat dan efisien [96].

YOLO memiliki kemampuan untuk mendeteksi objek secara cepat, bahkan dalam video atau *real-time*. YOLO berguna dalam deteksi objek satu kali *forward pass* melalui jaringan. Pendekatan ini membagi gambar menjadi *grid*, kemudian menentukan *bounding box, confidence score*, dan klasifikasi kelas objek untuk setiap sel *grid* [97]. Didalam YOLO sendiri terdapat dua tugas utama yaitu klasifikasi dan lokalisasi [98].

1. Klasifikasi

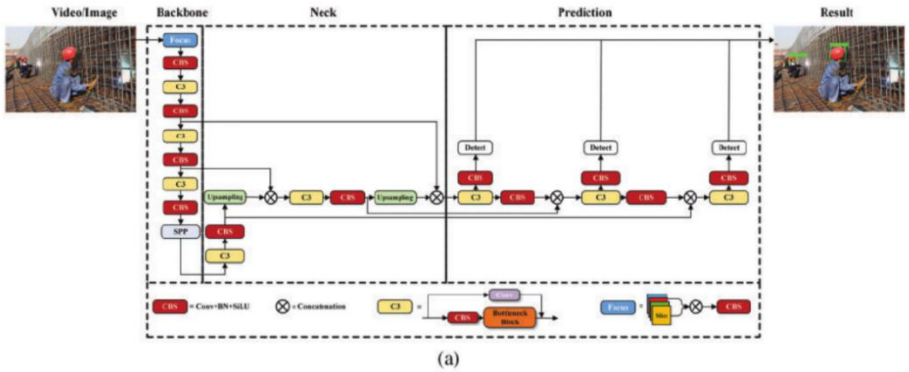
   YOLO mengidentifikasi kelas objek dalam gambar, seperti mobil, orang, atau hewan. Tugas ini dilakukan dengan memberikan label pada objek yang terdeteksi berdasarkan pola yang dipelajari selama pelatihan. Klasifikasi dilakukan secara *end-to-end* dalam satu jaringan, yang menjadikannya lebih cepat dan efisien dibandingkan metode yang memisahkan tugas ini.
2. Lokalisasi

YOLO juga menentukan posisi objek dengan memprediksi *bounding box* (kotak pembatas) untuk setiap objek. Setiap kotak menunjukkan lokasi dan ukuran objek dalam gambar, membantu YOLO untuk mendeteksi banyak objek sekaligus. Proses ini dilakukan dengan membagi gambar menjadi *grid-grid* kecil, di mana setiap grid bertanggung jawab untuk mendeteksi objek di wilayah tersebut.

Salah satu versi yang cukup banyak digunakan adalah YOLOv5, yang menawarkan keseimbangan yang baik antara akurasi dan efisiensi komputasi jika dibandingkan dengan beberapa versi YOLO lainnya. Misalnya, YOLOv8 membawa peningkatan dalam kemampuan seperti segmentasi *instance*, estimasi pose, dan klasifikasi dengan basis pengembangan dari YOLOv5 [99]. Demikian pula, YOLOv7, yang juga dikembangkan dari kode YOLOv5, menambahkan implementasi informasi gradien yang dapat diprogram. Karena keunggulan ini, YOLOv5 sering dijadikan dasar untuk berbagai aplikasi, termasuk pengembangan metode deteksi berbasis visi untuk mendeteksi penggunaan helm [100].

2.3.5 YOLOv5

Algoritma deteksi YOLOv5 adalah model ringan yang dirilis oleh Ultralytics pada tahun 2020. YOLOv5 memiliki empat struktur, yaitu YOLOv5s, YOLOw5m, YOLOw5l, dan YOLOv5x, yang mengontrol keempat struktur tersebut melalui dua parameter *depth_multiple* dan *width_multiple* [101]. YOLOv5, seperti pada gambar 2.7, mencakup input, *Backbone, Neck*, dan Prediksi.



Gambar 2.7 Arsitektur YOLOv5
Sumber : [28]

Pada bagian *backbone* Gambar 2.7, sendiri adalah komponen penting yang bertanggung jawab untuk mengekstraksi fitur dari gambar input. Terdapat beberapa komponen yang ada pada arsitektur YOLOv5 [28][102]:

1. *Focus*

   Lapisan ini memotong gambar input menjadi potongan-potongan kecil dan menggabungkannya kembali untuk meningkatkan resolusi. Ini membantu dalam menangkap detail yang lebih halus dari gambar.

2. *CBS*

   Merupakan kombinasi dari lapisan konvolusi, normalisasi batch, dan fungsi aktivasi SiLU. *Convolutional layer* berguna sebagai lapisan untuk menerapkan filter pada gambar untuk mengekstraksi fitur seperti *backbone, neck, dan prediction*. Kemudian batch *normalization* merupakan sebuah proses untuk menormalkan output dari *Convulutional layer* yang berguna untuk mempercepat pelatihan dan meningkatkan stabilitas model. Kemudian Silu *Activation* yang berfungsi dalam membantu jaringan saraf (*neural networks*) untuk membantu model mempelajari pola yang lebih kompleks dalam data.

3. *C3 Module*

   Merupakan modul yang terdiri dari beberapa blok *bottleneck* yang digunakan untuk mengekstraksi fitur dengan efisien. Didalam *C3 module* terdapat 2 bagian yaitu *Bottleneck* yang berguna untuk mengurangi dimensi fitur untuk mengurangi beban komputasi, kemudian mengembalikannya ke dimensi asli setelah pemrosesan dan *Residual Connections* yang membantu dalam mengatasi masalah gradien yang menghilang dan memastikan informasi penting tidak hilang selama pemrosesan.

4. *Spatial Pyramid Pooling (SPP)*

   Merupakan sebuah lapisan yang membantu dalam mengekstraksi fitur multi-skala dari gambar. Di dalam SPP terdapat dua komponen utama, yaitu *pooling layer* yang berfungsi untuk mengurangi resolusi *feature map* sambil tetap mempertahankan informasi penting, serta *pyramid structure* yang membantu model menangkap informasi pada berbagai skala, sehingga efektif untuk mendeteksi objek dengan ukuran yang bervariasi.

Pada bagian *Neck* bertanggung jawab untuk menggabungkan fitur dari berbagai lapisan *backbone* dan mempersiapkannya untuk prediksi akhir. Didalam *neck* terdapat komponen - komponen yang digunakan yaitu [28] [103]:

1. *Upsampling*

   Lapisan ini meningkatkan resolusi peta fitur yang dihasilkan oleh backbone. *Upsampling* dilakukan untuk memastikan bahwa fitur dari berbagai skala dapat digabungkan dengan baik. Teknik yang umum digunakan adalah interpolasi bilinear atau dekonvolusi.

2. CBS dan C3

   Sama hal nya pada bagian *backbone* CBS dan C3 digunakan lagi untuk menggabungkan fitur dari berbagai skala dan memastikan model dapat mengoptimalkan dan meningkatkan kualitas fitur yang telah diekstraksi, memastikan bahwa informasi yang lebih mendalam dan detail tersedia untuk tahap prediksi.

3. *Concatenation*

   Concatenation membantu model menggabungkan peta fitur dari berbagai lapisan *backbone* dan hasil *upsampling*. Dengan mengintegrasikan fitur dari berbagai skala, model dapat menangkap informasi yang lebih lengkap dan detail tentang objek dalam gambar.

Pada bagian *prediction* berisi komponen yang bertanggung jawab untuk membuat prediksi akhir berdasarkan fitur yang telah diekstraksi dan digabungkan oleh *backbone* dan *neck*. Sehingga komponen-komponen yang terdapat pada *prediction* adalah sebagai berikut [28] [104]:

1. *Detect*

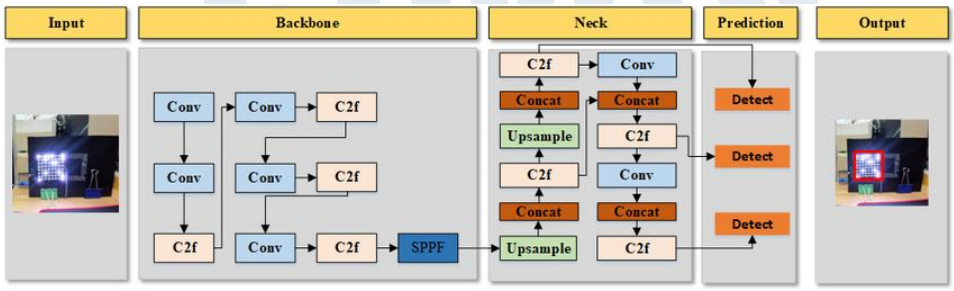
   Lapisan ini adalah inti dari bagian prediksi. Lapisan ini bertanggung jawab untuk mendeteksi objek dalam gambar berdasarkan fitur yang telah diproses oleh *backbone* dan *neck*. Lapisan ini menggunakan beberapa filter untuk menghasilkan prediksi *bounding box*, kelas objek, dan skor kepercayaan untuk setiap objek yang terdeteksi.

2. CBS dan *Concatenation*

Sama halnya yang digunakan pada *Backbone* dan *Neck* Dimana modul-modul ini bertujuan untuk meningkatkan kualitas dan efisiensi dalam ekstraksi fitur. CBS membantu memperbaiki fitur dengan menangkap pola non-linear dalam data, sementara *concatenation* menggabungkan fitur dari berbagai skala untuk memastikan informasi yang lebih lengkap dan detail tersedia untuk prediksi akhir. Penggunaan modul-modul ini di kedua bagian jaringan memastikan bahwa model dapat memproses gambar dengan cepat dan efisien, sekaligus meningkatkan akurasi deteksi objek dengan menangkap informasi penting dari berbagai skala.

2.3.6 YOLOv8

YOLOv8 merupakan salah satu model deteksi objek modern yang dikembangkan oleh Ultralytics pada tahun 2023 sebagai pengembangan lanjutan dari YOLOv5. Model ini dirancang untuk meningkatkan performa deteksi objek secara real-time dengan tetap mempertahankan keseimbangan antara akurasi dan kecepatan [105]. Berbeda dengan pendekatan deteksi objek tradisional yang dilakukan melalui beberapa tahap, YOLOv8 menggunakan konsep *single-stage detector*, di mana proses klasifikasi dan lokalisasi objek dilakukan dalam satu kali pemrosesan oleh jaringan saraf [105]. Pendekatan ini membuat YOLOv8 lebih efisien dan cocok digunakan dalam aplikasi *real-time* seperti pengawasan, sistem transportasi, dan analisis gambar [106]. Arsitektur YOLOv8 dapat dilihat pada Gambar 2.8.



Gambar 2.8 Arsitektur YOLOv8
Sumber : [106]

Pada bagian *Backbone* Gambar 2.8, model berfungsi untuk mengekstraksi fitur utama dari citra input. Pada arsitektur YOLOv8, backbone dirancang menggunakan pendekatan *Convolutional Neural Network* (CNN) yang mampu

menangkap informasi dari tingkat rendah hingga tinggi, seperti tekstur, bentuk, hingga pola objek. Berdasarkan arsitektur yang digunakan, terdapat beberapa komponen utama, yaitu [105][106]:

1. *Convolution Layer* (Conv)

   Lapisan ini melakukan operasi konvolusi dengan menggunakan kernel untuk mengekstraksi fitur dari gambar input. Pada tahap awal, conv biasanya menangkap fitur dasar seperti garis, warna, dan tekstur. Seiring bertambahnya kedalaman layer, fitur yang dihasilkan menjadi lebih kompleks seperti pola bentuk objek. Selain itu, *convolution* juga sering dikombinasikan dengan aktivasi dan normalisasi untuk meningkatkan stabilitas pelatihan.

2. *Cross Stage Partial - Faster* (C2f)

   C2f merupakan pengembangan dari konsep CSPNet yang bertujuan untuk meningkatkan efisiensi komputasi serta memperkuat aliran informasi antar *layer*. Modul ini membantu model mempertahankan informasi penting dari *layer* sebelumnya sekaligus mengurangi redundansi perhitungan.

3. *Spatial Pyramid Pooling Fast* (SPPF)

   SPPF digunakan untuk memperluas cakupan pemahaman model terhadap konteks global dalam gambar. Dengan menggabungkan beberapa operasi *pooling* dalam satu *layer*, modul ini membantu model mengenali objek dalam berbagai ukuran tanpa perlu mengubah resolusi input. Versi "*Fast*" dari SPPF juga dirancang agar lebih ringan dan cepat dibandingkan metode sebelumnya.

Bagian *Neck* berfungsi untuk menggabungkan dan memperkaya *feature map* dari *backbone* sehingga siap digunakan untuk proses deteksi. YOLOv8 menggunakan pendekatan *multi-scale feature fusion* yang terinspirasi dari FPN dan PANet, sehingga mampu mengintegrasikan informasi dari berbagai *level* kedalaman jaringan. Komponen pada *neck* meliputi [105][106]:

1. *Upsample*

   Upsample digunakan untuk meningkatkan resolusi *feature map* dari *layer* yang lebih dalam. Hal ini penting karena *layer* dalam biasanya memiliki resolusi kecil namun kaya informasi semantik. Dengan *upsampling*, fitur tersebut dapat

disesuaikan ukurannya agar dapat digabungkan dengan *feature map* dari *layer* yang lebih awal.

2. *Concatenation* (Concat)

   Concatenation berfungsi untuk menggabungkan *feature map* dari berbagai layer menjadi satu representasi yang lebih lengkap. Proses ini membantu model mengombinasikan detail *spasial* dari *layer* awal dengan informasi semantik dari *layer* dalam, sehingga meningkatkan kemampuan dalam mendeteksi objek dengan berbagai ukuran.

3. C2f

   Setelah proses penggabungan, C2f kembali digunakan untuk memproses *feature map* hasil *concat*. Modul ini membantu menyaring informasi yang relevan dan memperkuat representasi fitur agar lebih efektif digunakan pada tahap prediksi.

4. Conv

   Lapisan *convolution* pada *neck* berfungsi untuk menyempurnakan hasil penggabungan fitur. Conv membantu mengurangi *noise*, menyesuaikan dimensi *feature map*, serta memastikan bahwa informasi yang diteruskan ke tahap berikutnya sudah dalam kondisi optimal.

Bagian *Head* atau *Prediction* merupakan tahap akhir yang bertugas menghasilkan *output* deteksi objek. YOLOv8 menggunakan pendekatan *anchor-free*, yang menyederhanakan proses prediksi dibandingkan metode sebelumnya. Komponen utama pada bagian ini adalah [105][106]:

1. *Detect Layer*

   Detect layer menghasilkan prediksi berupa koordinat *bounding box*, nilai *confidence*, serta kelas objek. Setiap *detect layer* bekerja pada *feature map* dengan resolusi berbeda, sehingga mampu menangani variasi ukuran objek. Proses ini dilakukan secara langsung dalam satu tahap tanpa memerlukan *region* proposal seperti metode dua tahap.

2. *Anchor-Free Mechanism*

Pada pendekatan ini, model tidak lagi menggunakan *anchor box* sebagai acuan awal. Sebagai gantinya, model langsung memprediksi posisi objek berdasarkan titik pusatnya. Hal ini mengurangi kompleksitas perhitungan, mempercepat proses pelatihan, serta membuat model lebih fleksibel terhadap berbagai bentuk dan rasio objek.

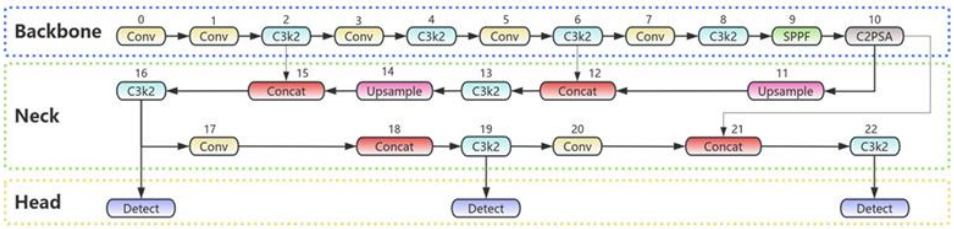
3. *Multi-Scale Prediction*

   Prediksi dilakukan pada beberapa skala *feature map* secara bersamaan. Skala resolusi tinggi digunakan untuk mendeteksi objek kecil, sedangkan resolusi rendah digunakan untuk objek besar. Dengan cara ini, model mampu memberikan hasil deteksi yang lebih konsisten di berbagai kondisi.

Hasil akhir dari model berupa objek yang berhasil terdeteksi pada gambar input, yang ditunjukkan melalui *bounding box*, label kelas, serta nilai *confidence*. Output ini menunjukkan bahwa model mampu melakukan klasifikasi dan lokalisasi objek secara bersamaan dalam satu kali proses (single-stage detection) [105][106].

2.3.7 YOLOv11

Seiring perkembangannya, YOLO terus mengalami peningkatan dari YOLOv11 hingga YOLOv10, baik dari sisi akurasi, efisiensi komputasi, maupun fleksibilitas penggunaan. YOLOv11 sebagai versi terbaru menghadirkan berbagai penyempurnaan arsitektur yang bertujuan untuk meningkatkan performa deteksi secara menyeluruh. Model ini tidak hanya digunakan untuk *object detection*, tetapi juga mendukung berbagai tugas lain seperti *instance segmentation*, *pose estimation*, *image classification*, hingga *oriented object detection (OBB)* [26]. Selain itu, YOLOv11 dirancang agar dapat digunakan pada berbagai skala perangkat, mulai dari *edge devices* hingga sistem komputasi berkinerja tinggi [107]. Secara umum, arsitektur YOLOv11 terdiri dari tiga komponen utama, yaitu *backbone*, *neck*, dan *head*. Ketiga komponen ini bekerja secara berurutan dalam mengekstraksi fitur, menggabungkan informasi, dan menghasilkan prediksi akhir [35][108]. Untuk arsitektur YOLOv11 dapat dilihat pada Gambar 2.9.



Gambar 2.9 Arsitektur YOLOv11
Sumber : [35]

Bagian *backbone* yang tertera pada Gambar 2.9 berperan sebagai tahap awal dalam arsitektur YOLOv11 yang bertugas mengekstraksi fitur dari gambar input. Proses ini dilakukan melalui serangkaian lapisan konvolusi dan blok khusus yang dirancang untuk menangkap informasi visual mulai dari pola sederhana hingga kompleks. Pada YOLOv11, backbone mengalami peningkatan melalui penggunaan blok yang lebih efisien sehingga mampu menghasilkan *feature map* yang lebih representatif tanpa menambah beban komputasi secara signifikan. Komponen pada *backbone* meliputi [35][108]:

1. *Convolutional Layer* (Conv)

   Lapisan konvolusi digunakan sebagai fondasi utama dalam proses ekstraksi fitur. Pada tahap awal, conv bekerja untuk mengurangi dimensi spasial citra sekaligus meningkatkan jumlah *channel* sehingga informasi penting dapat dipertahankan. Selain itu, *conv* juga membantu mendeteksi pola dasar seperti tepi, tekstur, dan bentuk sederhana yang nantinya akan dikembangkan pada layer berikutnya.

2. C3k2 *Block*

   C3k2 merupakan pengembangan dari konsep *Cross Stage Partial* (CSP) yang dirancang lebih ringan dan efisien. Berbeda dengan blok sebelumnya, C3k2 menggunakan dua konvolusi berukuran kecil untuk menggantikan satu konvolusi besar. Pendekatan ini membuat proses komputasi menjadi lebih cepat tanpa mengurangi kualitas fitur yang dihasilkan. Selain itu, blok ini juga membantu menjaga aliran gradien agar tetap stabil selama proses pelatihan.

3. *Spatial Pyramid Pooling - Fast* (SPPF)

SPPF berfungsi untuk menangkap informasi konteks dari berbagai skala dalam satu waktu. Modul ini melakukan *pooling* dengan beberapa ukuran kernel sehingga model dapat memahami objek baik yang berukuran kecil maupun besar. Dengan adanya SPPF, *backbone* mampu menghasilkan representasi fitur yang lebih kaya tanpa meningkatkan kompleksitas secara berlebihan.

4. *Spatial Attention Module* (C2PSA)

   C2PSA merupakan modul *attention* yang membantu model untuk lebih fokus pada bagian penting dari citra. Dengan mekanisme ini, model dapat memberikan perhatian lebih pada area yang mengandung objek dan mengurangi pengaruh informasi yang kurang relevan. Hal ini sangat berguna terutama ketika objek berada di lingkungan yang kompleks atau memiliki gangguan latar belakang.

Bagian neck berfungsi sebagai penghubung antara backbone dan head yang bertugas menggabungkan serta memperkaya *feature map* dari berbagai skala. YOLOv11 menggunakan pendekatan *multi-scale feature fusion* sehingga informasi dari layer yang berbeda dapat diintegrasikan dengan lebih optimal. Proses ini penting untuk meningkatkan kemampuan model dalam mendeteksi objek yang memiliki ukuran beragam. Komponen pada neck meliputi [35][108]:

1. *Upsample*

   Upsample digunakan untuk meningkatkan resolusi *feature map* dari layer yang lebih dalam. *Layer* dalam biasanya memiliki informasi semantik yang kuat namun resolusinya kecil. Dengan *upsampling*, ukuran *feature map* disesuaikan sehingga dapat digabungkan dengan *layer* lain yang memiliki resolusi lebih tinggi.

2. *Concatenation* (Concat)

   Concatenation berfungsi untuk menggabungkan *feature map* dari berbagai level menjadi satu kesatuan. Proses ini membantu model untuk memanfaatkan kombinasi informasi detail (dari *layer* awal) dan informasi semantik (dari *layer* dalam), sehingga hasil fitur menjadi lebih lengkap dan informatif.

3. C3k2 *Block*

Setelah proses penggabungan, C3k2 digunakan kembali untuk memproses *feature map* hasil concat. Modul ini membantu menyaring informasi penting dan meningkatkan kualitas representasi fitur. Selain itu, penggunaan C3k2 juga menjaga efisiensi model dari sisi komputasi.

4. *Attention Mechanism* (C2PSA)

   Pada bagian *neck*, *attention mechanism* kembali digunakan untuk memperkuat fokus model terhadap area penting. Dengan adanya mekanisme ini, model menjadi lebih sensitif terhadap objek kecil atau objek yang sebagian tertutup, sehingga akurasi deteksi dapat meningkat.

Bagian *head* merupakan tahap akhir dalam arsitektur YOLOv11 yang bertugas menghasilkan prediksi berupa *bounding box* dan klasifikasi objek. *Feature map* yang telah diproses oleh *backbone* dan *neck* akan digunakan untuk menentukan lokasi dan kategori objek secara langsung. Komponen pada *head* meliputi [35][108]:

1. C3k2 *Block*

   Pada bagian *head*, C3k2 digunakan untuk mengolah *feature map* dari berbagai skala sebelum dilakukan prediksi. Modul ini membantu memperdalam representasi fitur sehingga model dapat membedakan objek dengan lebih akurat, terutama pada kondisi yang kompleks.

2. *Convolution, BatchNorm, SiLU* (CBS)

   CBS merupakan kombinasi dari *convolution*, *batch normalization*, dan fungsi aktivasi SiLU. Lapisan ini berfungsi untuk menstabilkan distribusi data, mempercepat proses pelatihan, serta meningkatkan kemampuan model dalam menangkap dan mempelajari pola *non-linear*. Dengan CBS, *feature map* menjadi lebih halus dan siap digunakan untuk tahap prediksi.

3. *Final Convolution Layer*

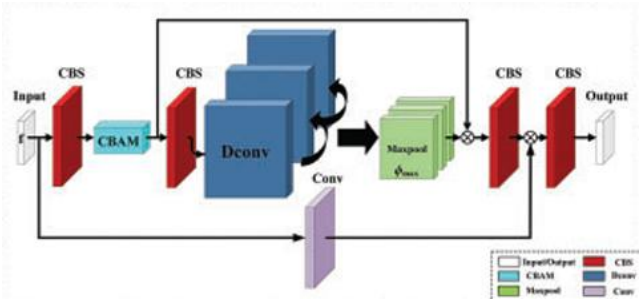
   Lapisan konvolusi terakhir bertugas menyesuaikan jumlah *output* sesuai kebutuhan deteksi, seperti koordinat *bounding box* dan jumlah kelas. Proses ini memastikan bahwa setiap *feature map* dapat diterjemahkan menjadi prediksi yang relevan.

2.4 Attention Mechanism

Attention Mechanism bertujuan untuk meniru cara manusia yang memberi fokus lebih pada informasi penting dan mengabaikan informasi yang kurang relevan. Dalam praktiknya, mekanisme ini membuat jaringan hanya fokus pada bagian atau objek yang esensial dan menekan informasi yang tidak terlalu penting, serupa dengan cara manusia bekerja [109]. Sejumlah model *Attention Mechanism* telah diperkenalkan. Salah satu yang pertama adalah *spatial transformer network* yang digunakan untuk mengekstrak informasi spasial penting dari gambar input. Model ini memetakan gambar ke ruang lain dan menggunakan *spatial attention module* (SAM) untuk mengekstraksi informasi fitur berdasarkan koordinat posisi spasial [110]. Kemudian *convolutional block attention module* (CBAM) diperkenalkan sebagai modul ringan yang menggabungkan perhatian spasial dan saluran, memadukan keunggulan dari keduanya. Meski demikian, masih banyak ruang untuk perbaikan dalam *attention mechanism* [111]. Salah satu topik menarik adalah bagaimana teknik-teknik *attention mechanism* yang dikembangkan di satu area dapat diterapkan di area lain. Salah satunya adalah bagaimana memilih dan menerapkan *attention mechanism* dalam deteksi pemakaian helm, yang menjadi masalah menarik untuk diselidiki lebih lanjut.

2.4.1 MSCAM Module

Multi-scale Convolutional Attention Module (MSCAM) adalah sebuah pendekatan dalam bidang deep learning, yang menggabungkan konsep perhatian (*attention*) dengan skala fitur multi-resolusi untuk meningkatkan kinerja model dalam mendeteksi informasi penting pada berbagai tingkatan resolusi dalam data input [112]. MSCAM dirancang untuk mengatasi masalah yang muncul pada model jaringan saraf konvolusional tradisional, yang sering kali kurang sensitif terhadap informasi yang berada pada berbagai skala (resolusi) dalam gambar [113]. Gambar arsitektur MSCAM dapat dilihat pada gambar 2.10 yang berisikan berbagai komponen dalam MSCAM yang terpecah menjadi 3 yaitu [114] [28]:



Gambar 2.10 Arsitektur MSCAM
Sumber : [28]

1. *Multi Scale Feature Extraction*

   MSCAM pertama-tama mengekstrak fitur dari gambar input pada berbagai skala. Ini biasanya dilakukan dengan menggunakan beberapa lapisan konvolusi dengan kernel yang berbeda-beda untuk mendapatkan informasi pada skala kecil hingga besar.

2. *Attention Mechanism*

   Setelah ekstraksi fitur multi-skala, MSCAM menggunakan *attention mechanism* untuk memberikan bobot yang lebih tinggi pada fitur-fitur penting pada setiap skala. Dengan demikian, perhatian pada setiap skala dapat disesuaikan untuk fokus pada fitur-fitur yang lebih relevan.

3. *Fusion of Multi-scale Attention*

   Fitur-fitur yang telah diimplementasikan kemudian digabungkan untuk menghasilkan representasi akhir yang mempertimbangkan informasi dari berbagai skala. Penggabungan ini membantu model untuk memanfaatkan kelebihan dari setiap skala dengan lebih efektif.

Proses MSCAM *module* dapat dilihat pada Rumus 2.6 yang menggambarkan proses pembentukan output. Kemudian proses *dilated convolution* dijelaskan secara berurut-turut pada Rumus 2.7 hingga Rumus 2.9 dengan variasi kernel yang berbeda [28].

$$Output = CBS(Concat(Conf(f); CBS(Concat(CBAM\big(CBS(f)\big) \tag{2.6}$$

$$\emptyset_{max} = \left(DCONV_1\left(CBS\left(CBAM\big(CBS(f)\big)\right)\right)\right) \tag{2.7}$$

$$\emptyset_{max} = \left(DCONV_2 \left(CBS \left(CBAM(CBS(f)) \right) \right) \right) \quad (2.8)$$

$$\emptyset_{max} = \left(DCONV_3 \left(CBS \left(CBAM(CBS(f)) \right) \right) \right) \quad (2.9)$$

Rumus 2.6 Perhitungan nilai MSCAM *Module*

Rumus 2.6 dimulai dengan ekstraksi fitur menggunakan konvolusi, batch normalization, dan fungsi aktivasi SiLU (CBS), diikuti dengan penerapan mekanisme perhatian CBAM untuk menekankan fitur yang lebih relevan. *Dilated convolutions* (Dconv1, Dconv2, Dconv3) digunakan untuk menangkap informasi dari skala yang lebih luas tanpa mengurangi resolusi, sementara *max pooling* (ϕmax) mengurangi dimensi fitur yang tidak penting. Hasil dari beberapa skala ini kemudian digabungkan (concat) dan diproses lebih lanjut untuk menghasilkan output yang informatif, yang digunakan untuk deteksi objek yang lebih efektif dalam berbagai kondisi. Berdasarkan Rumus 2.5 maka dapat proses MSCAM dapat dijabarkan dalam bentuk pseudocode pada tabel 2.4.

Tabel 2.4 *Pseudocode* MSCAM

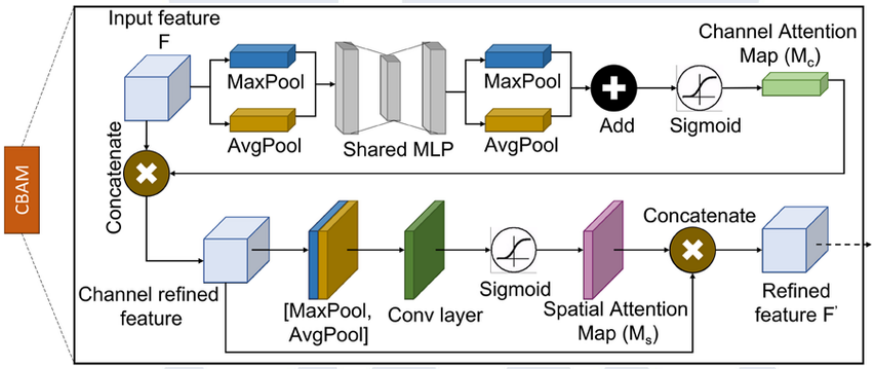
(1)	*Downsampling* peta fitur input *F* untuk mendapatkan peta fitur dengan berbagai skala: F_1 (skala 1), F_2 (skala 2), F_3 (skala 4), F_4 (skala 8)
(2)	Terapkan *ChannelAttention* pada masing-masing skala untuk menghitung setiap peta fitur skala
(3)	Kalikan *ChannelAttention* dengan peta fitur masing-masing untuk mendapatkan *Enhanced feature map*
(4)	Gabungkan semua peta fitur yang telah ditingkatkan.
(5)	terapkan *convolution* pada *concatenated feature map* $Fconcat$ untuk mendapatkan final output $Fout$.

MSCAM (*Multi-Scale Channel Attention Module*) dimulai dengan melakukan *downsampling* peta fitur input F pada berbagai skala (1, 2, 4, 8). Kemudian, Channel Attention diterapkan pada setiap skala untuk menyoroti saluran yang penting, diikuti dengan perkalian peta perhatian saluran dengan peta fitur untuk memperkuat fitur yang relevan. Selanjutnya, semua peta fitur yang telah diperhatikan digabungkan untuk mengintegrasikan informasi dari berbagai skala, dan akhirnya, konvolusi diterapkan pada peta fitur gabungan untuk menghasilkan output yang lebih informatif. Hasil akhirnya adalah *feature map* yang lebih

informatif, sehingga membantu model dalam menyoroti fitur-fitur penting dari objek pada berbagai skala di dalam gambar.

2.4.2 CBAM *Module*

CBAM (*Convolutional Block Attention Module*) merupakan salah satu metode *attention mechanism* dalam *deep learning* yang bertujuan untuk meningkatkan kualitas representasi fitur dengan memberikan bobot lebih besar pada informasi yang relevan serta mengurangi pengaruh informasi yang kurang penting [38]. CBAM bekerja dengan memanfaatkan dua dimensi utama pada *feature map*, yaitu dimensi *channel* dan dimensi spasial, sehingga model dapat memahami tidak hanya fitur apa yang penting tetapi juga lokasi fitur tersebut dalam citra [115]. Arsitektur pada CBAM dapat dilihat pada Gambar 2.11 yang berisikan beberapa komponen CBAM yang terdiri dari dua bagian utama, yaitu [38][115]:



Gambar 2.11 Arsitektur CBAM
Sumber : [116]

1. *Channel Attention*

   Channel attention berfungsi untuk menentukan fitur mana yang paling penting pada setiap *channel*. Pada tahap ini, *feature map* akan diproses menggunakan dua jenis *pooling*, yaitu *average pooling* dan *max pooling*, untuk menangkap informasi global dan informasi paling dominan. Hasil dari kedua proses tersebut kemudian dimasukkan ke dalam jaringan *Multi-Layer Perceptron* (MLP) yang sama untuk menghasilkan bobot perhatian pada setiap *channel*. Bobot ini selanjutnya dikalikan dengan *feature map* awal sehingga *channel* yang penting akan diperkuat, sedangkan *channel* yang kurang relevan akan dikurangi pengaruhnya.

2. *Spatial Attention*

   Spatial Attention bertugas untuk menentukan lokasi penting pada *feature map*. Berbeda dengan *channel attention* yang fokus pada "apa", *spatial attention* lebih menekankan pada "di mana" informasi penting tersebut berada. Pada tahap ini, dilakukan proses *pooling* pada dimensi *channel* menggunakan *average pooling* dan *max pooling* untuk menghasilkan *feature map* dua dimensi. Kedua peta tersebut kemudian digabungkan dan diproses menggunakan konvolusi (biasanya kernel 7×7) untuk menghasilkan peta perhatian spasial. Hasil ini digunakan untuk memperkuat area penting dan mereduksi area yang tidak relevan.

Mekanisme *channel attention* pada CBAM dirumuskan Rumus 2.10 yang berfungsi untuk menekankan informasi penting pada setiap *channel* fitur [117]. Mekanisme *spatial attention* dijelaskan pada Rumus 2.11 yang bertujuan untuk mengfokuskan *attention* pada area *spatial* yang relevan [117].

$$F' = M_c(F) \otimes F \tag{2.10}$$

$$F'' = M_s(F') \otimes F' \tag{2.11}$$

Rumus 2.7 Perhitungan nilai CBAM *Module*

Keterangan:

1. F = *feature map input*
2. Mc(F) = *channel attention*
3. Ms(F') = *spatial attention*
4. $\otimes$ = perkalian elemen (*element-wise multiplication*)

Rumus 2.7 menunjukkan proses pemberian *attention* yang dilakukan secara bertahap pada feature map. Pada Rumus pertama, $F' = M_c(F) \otimes F$, feature map awal Fdikalikan dengan bobot *channel attention* $M_c(F)$, sehingga menghasilkan feature map baru F'yang telah diperkuat pada channel-channel yang dianggap penting. Selanjutnya, pada Rumus kedua $F'' = M_s(F') \otimes F'$, feature map hasil sebelumnya F'kembali diproses menggunakan spatial attention $M_s(F')$, yang berfungsi untuk menyoroti lokasi atau area penting pada fitur tersebut. Hasil akhirnya adalah F'', yaitu feature map yang telah melalui dua tahap penyaringan,

baik dari sisi channel maupun spasial, sehingga informasi yang dipertahankan menjadi lebih relevan dan mendukung peningkatan performa model dalam memahami objek. Proses kerja attention mechanism CBAM dapat dilihat pada Tabel 2.5 [117].

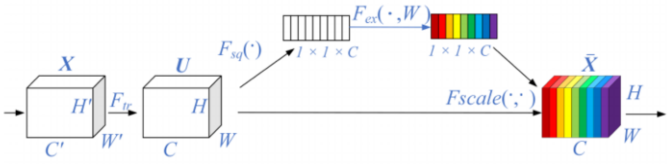
Tabel 2.5 *Pseudocode* CBAM

(1)	Input peta fitur F dari hasil ekstraksi *backbone*
(2)	Terapkan *Channel Attention* menggunakan *average pooling* dan *max pooling*
(3)	Kalikan hasil *channel attention* dengan peta fitur untuk mendapatkan F' *(refined feature)*
(4)	Terapkan *Spatial Attention* pada F' menggunakan *pooling* pada dimensi *channel*.
(5)	Kalikan hasil *spatial attention* dengan F' untuk menghasilkan *output* akhir F''.

CBAM dimulai dengan *feature map F* dari hasil ekstraksi fitur, kemudian diterapkan *channel attention* untuk menekankan *channel* yang penting menggunakan *average* dan *max pooling*. Setelah itu, *feature map* diperkuat kembali dengan *spatial attention* untuk menentukan area penting pada citra. Kombinasi keduanya menghasilkan *feature map* yang lebih fokus dan informatif, sehingga membantu meningkatkan akurasi model dalam mendeteksi objek.

2.4.3 SEBlock

SEBlock merupakan salah satu metode *attention mechanism* pada deep learning yang berfokus pada dimensi channel [21]. Modul ini bertujuan untuk meningkatkan kualitas representasi fitur dengan cara memberikan bobot yang berbeda pada setiap *channel*, sehingga model dapat lebih menekankan fitur yang penting dan mengurangi pengaruh fitur yang kurang relevan [21] [118]. Dengan pendekatan ini, hubungan antar *channel* dapat dimodelkan secara adaptif, sehingga meningkatkan kemampuan jaringan dalam mengekstraksi informasi yang lebih bermakna. Untuk lebih jelas gambar arsitektur SEBlock dapat dilihat pada Gambar 2.10.



Gambar 2.12 Arsitektur SEBlock
Sumber: [119]

Komponen dalam SEBlock pada Gambar 2.12 terdiri dari beberapa bagian utama, yaitu [21][119]:

1. *Squeeze* (*Global Average Pooling*)

   Squeeze berfungsi untuk merangkum informasi spasial dari *feature map* ke dalam bentuk yang lebih ringkas. Pada tahap ini, setiap *channel* pada *feature map* diproses menggunakan *Global Average Pooling* (GAP) sehingga ukuran *feature map* berubah dari $[C, H, W]$menjadi $[C, 1, 1]$. Proses ini menghasilkan representasi global dari masing-masing *channel*, yang mencerminkan seberapa penting *channel* tersebut secara keseluruhan dalam gambar.

2. *Excitation* (*Fully Connected Layer*)

   Excitation bertujuan untuk mempelajari hubungan antar *channel* dan menghasilkan bobot perhatian. Nilai hasil *squeeze* kemudian diproses menggunakan dua *fully connected layer*. *Layer* pertama digunakan untuk mereduksi dimensi (biasanya dengan rasio tertentu seperti $r = 16$) dan diikuti dengan aktivasi ReLU untuk menangkap hubungan non-linear. Selanjutnya, *layer* kedua mengembalikan dimensi ke ukuran semula dan menggunakan fungsi sigmoid untuk menghasilkan bobot pada setiap *channel* dengan rentang nilai antara 0 hingga 1.

3. *Scaling* (*Feature Recalibration*)

   Scaling merupakan tahap akhir, yaitu mengalikan bobot hasil *excitation* dengan *feature map* awal secara elemen per *channel*. Proses ini bertujuan untuk memperkuat *channel* yang penting dan melemahkan *channel* yang kurang relevan. Hasilnya adalah *feature map* yang telah disesuaikan sehingga lebih informatif dan optimal untuk proses selanjutnya.

Rumus pada SEBlock *module* dapat dilihat pada Rumus 2.12 hingga Rumus 2.14. Proses *squeeze* direpresentasikan pada Rumus 2.12, di mana feature map U_c dikompresi menjadi representasi channel Z_c melalui operasi *global pooling*. Tahap *excitation* ditunjukkan pada Rumus 2.13 yang berfungsi untuk mempelajari hubungan antara *channel* dan menghasilkan bobot *attention* S_c. Selanjutnya proses

scale dijelaskan pada Rumus 2.14 , di mana feature map awal $U_c$dikalikan dengan bobot $S_c$untuk menghasilkan fitur akhir yang telah diperkuat [21].

$$Z_c = F_{sq}(U_c) \tag{2.12}$$

$$S_c = F_{ex}(Z_c) \tag{2.13}$$

$$\tilde{X}_c = F_{scale}(U_c, S_c) \tag{2.14}$$

Rumus 2.8 Perhitungan Nilai SEBlock Module

Keterangan:

1. U_c= *feature map input*
2. Z_c= *hasil squeeze (global pooling)*
3. S_c= *bobot channel (excitation)*
4. $\tilde{X}_c$= *feature map hasil akhir*
5. F_{sq}= *operasi squeeze*
6. F_{ex}= *operasi excitation*
7. F_{scale}= *operasi scaling*

Rumus 2.8 menunjukkan proses perhatian pada SEBlock yang dilakukan dalam tiga tahap. Pertama, feature map $U_c$diproses melalui *squeeze* untuk mendapatkan representasi *global* Z_c. Selanjutnya, $Z_c$diproses pada tahap *excitation* untuk menghasilkan bobot *channel* S_c. Terakhir, bobot tersebut dikalikan dengan *feature map* awal untuk menghasilkan *feature map* baru yang lebih terfokus pada *channel* penting. Proses kerja pada attention mechanism SEBlock dapat dilihat pada tabel 2.6 [21].

Tabel 2.6 *Pseudocode* SEBlock

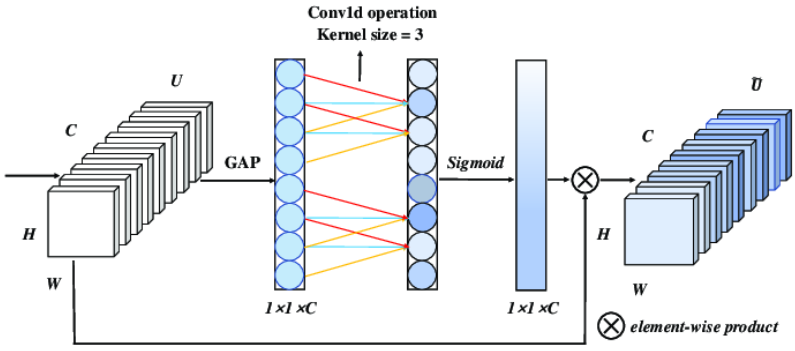
(1)	*Input* peta fitur *U*dari hasil ekstraksi *backbone*
(2)	Lakukan *Global Average Pooling* untuk mendapatkan vektor *channel Z*
(3)	Proses *Z* menggunakan MLP untuk menghasilkan bobot *channel*
(4)	Terapkan fungsi aktivasi (ReLU dan sigmoid) untuk mendapatkan bobot akhir.
(5)	Kalikan bobot *channel* dengan peta fitur awal untuk menghasilkan output akhir**.**

SEBlock pada Tabel 2.6, dimulai dengan menerima *feature map* dari hasil ekstraksi fitur, kemudian dilakukan proses squeeze untuk merangkum informasi global tiap *channel*. Setelah itu, *excitation* digunakan untuk menentukan bobot

kepentingan masing-masing channel. Terakhir, bobot tersebut diterapkan kembali ke *feature map* awal melalui proses *scaling*. Hasil akhirnya adalah feature map yang lebih terarah dan informatif, sehingga membantu meningkatkan performa model dalam mendeteksi objek.

2.4.4 ECA Module

Efficient Channel Attention (ECA) merupakan salah satu pengembangan dari *attention mechanism* yang berfokus pada peningkatan kualitas representasi fitur pada dimensi *channel* dengan cara yang lebih efisien [37]. Modul ini dirancang sebagai penyempurnaan dari SEBlock dengan tujuan mengurangi kompleksitas tanpa mengorbankan performa model. Berbeda dengan SEBlock yang menggunakan reduksi dimensi, ECA mempertahankan jumlah channel asli sehingga informasi antar channel tetap terjaga [37]. Dengan pendekatan ini, ECA mampu memodelkan hubungan antar *channel* secara lebih efektif dan ringan, sehingga cocok digunakan pada model dengan kebutuhan komputasi yang efisien [120]. Arsitektur *attention mechanism* ECA dapat dilihat pada Gambar 2.13 yang berisikan beberapa Komponen dalam ECA yang terdiri dari beberapa bagian utama, yaitu [120]:



Gambar 2.13 Arsitektur ECA
Sumber: [121]

1. *Global Average Pooling*

   Tahap awal pada ECA adalah melakukan *Global Average Pooling* (GAP) pada *feature map input*. Proses ini bertujuan untuk merangkum informasi global dari setiap *channel* tanpa mengubah jumlah *channel* yang ada. Hasil dari GAP berupa vektor satu dimensi yang merepresentasikan distribusi informasi pada

masing-masing channel, sehingga tetap mempertahankan karakteristik asli *feature map*

2. *Local Cross-Channel Interaction*

   Setelah proses *pooling*, ECA tidak menggunakan *fully connected layer* seperti pada SEBlock, melainkan memanfaatkan konvolusi satu dimensi (1D convolution). Konvolusi ini digunakan untuk menangkap hubungan antar *channel* secara lokal dengan mempertimbangkan sejumlah tetangga *channel* terdekat. *Parameter* penting pada tahap ini adalah nilai k, yang menentukan seberapa luas interaksi antar *channel*. Dengan pendekatan ini, ECA dapat menangkap dependensi *channel* tanpa menambah kompleksitas *parameter* secara signifikan.

3. *Adaptive Kernel Size* (k)

   Nilai kernel k pada ECA tidak ditentukan secara sembarangan, melainkan dihitung secara adaptif berdasarkan jumlah *channel* pada *feature map*. Hal ini bertujuan agar cakupan interaksi antar *channel* tetap proporsional terhadap ukuran fitur. Umumnya, nilai k dipilih sebagai bilangan ganjil terdekat dari hasil perhitungan fungsi tertentu, sehingga mampu menyeimbangkan antara efisiensi dan kemampuan representasi.

4. *Feature Recalibration* (*Scaling*)

   Tahap akhir adalah melakukan *scaling*, yaitu mengalikan bobot hasil perhatian dengan *feature map* awal. Bobot tersebut diperoleh setelah melalui fungsi aktivasi sigmoid, sehingga memiliki rentang nilai antara 0 hingga 1. Proses ini akan memperkuat *channel* yang penting dan menekan *channel* yang kurang relevan, sehingga menghasilkan *feature map* yang lebih informatif.

Rumus pada *attention mechanism* ECA dapat dilihat pada Rumus 2.15 hingga Rumus 2.17. Proses ekstraksi informasi *channel* direpresentasikan pada Rumus 2.15, dimana feature map F diringkas menjadi vektor z menggunakan operasi *Global Average Pooling* (GAP). Selanjutnya, proses pembobotan *channel* ditunjukkan pada Rumus 2.16, di mana vektor z diproses menggunakan operasi 1D *convolution* dengan kernel berukuran k dan fungsi aktivasi sigmoid untuk

menghasilkan *attention weight* s. Proses akhir dijelaskan pada Rumus 2.17, di mana *feature map* awal F dikalikan dengan *attention weight* s untuk menghasilkan *feature map* F' yang telah diperkuat [122].

$$z = GAP(F) \tag{2.15}$$

$$s = \sigma(Conv1D_k(z)) \tag{2.16}$$

$$F' = s \otimes F \tag{2.17}$$

Rumus 2.9 Perhitungan Nilai ECA *Module*

Keterangan:

1. F= feature map input
2. z= hasil global average pooling
3. $Conv1D_k$= konvolusi 1D dengan kernel k
4. s= bobot channel (attention)
5. σ= fungsi sigmoid
6. $\otimes$= perkalian elemen (element-wise multiplication)
7. F'= feature map hasil akhir

Rumus 2.9 menunjukkan bahwa *feature map* F terlebih dahulu diringkas menggunakan *global average pooling* untuk menghasilkan representasi *channel* z. Selanjutnya, dilakukan konvolusi satu dimensi untuk menangkap hubungan antar *channel* secara lokal dan menghasilkan bobot perhatian s. Bobot ini kemudian digunakan untuk mengalikan *feature map* awal sehingga menghasilkan *feature map* baru yang telah diperkuat pada *channel* yang relevan. Cara kerja *attention mechanism* modul ECA dapat dilihat pada Tabel 2.7 [120].

Tabel 2.7 *Pseudocode* ECA

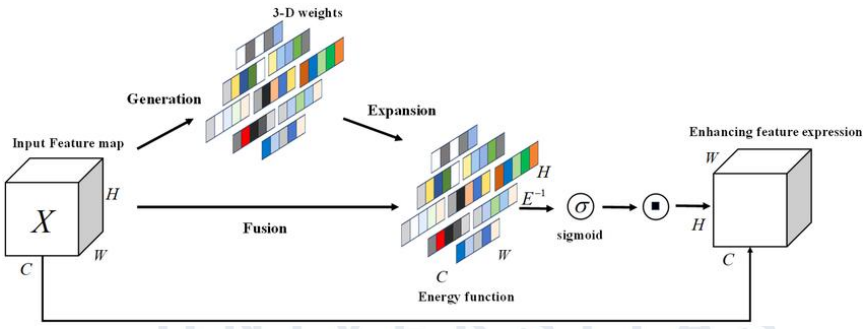
(1)	*Input* peta fitur F dari hasil ekstraksi *backbone.*
(2)	Lakukan *Global Average Pooling* untuk mendapatkan vektor *channel* z.
(3)	Tentukan nilai kernel k berdasarkan jumlah *channel.*
(4)	Terapkan konvolusi 1D pada z untuk mendapatkan bobot *channel.*
(5)	Gunakan fungsi sigmoid untuk menghasilkan bobot akhir.
(6)	Kalikan bobot dengan peta fitur awal untuk menghasilkan *output* F'

Pseudocode ECA pada Tabel 2.7, dimulai dengan menerima *feature map* dari hasil ekstraksi fitur, kemudian dilakukan *global average pooling* untuk merangkum

informasi tiap *channel*. Setelah itu, hubungan antar *channel* dipelajari menggunakan konvolusi satu dimensi tanpa reduksi dimensi. Bobot yang dihasilkan digunakan untuk memperkuat *feature map* awal melalui proses *scaling*. Hasil akhirnya adalah *feature map* yang lebih efisien dan tetap kaya informasi, sehingga mampu meningkatkan performa model terutama dalam menangkap objek dengan ukuran kecil maupun detail yang kompleks.

2.4.5 *SimAM Module*

Simple Attention Module (SimAM) merupakan salah satu metode *attention mechanism* dalam *deep learning* yang bertujuan untuk meningkatkan kualitas representasi fitur dengan cara memberikan bobot berdasarkan tingkat kepentingan setiap *neuron* pada *feature map* [123]. Berbeda dengan metode *attention* lain yang menambahkan *parameter* tambahan, SimAM dirancang tanpa penambahan *parameter* (*parameter-free*), sehingga lebih efisien secara komputasi. SimAM bekerja dengan mempertimbangkan dimensi *channel* dan spasial secara bersamaan (*3D attention*), sehingga model dapat memahami informasi secara lebih menyeluruh [123]. Arsitektur attention mechanism simAM dapat dilihat pada Gambar 2.14 yang berisikan beberapa komponen dalam SimAM terdiri dari beberapa tahapan utama, yaitu [123][124]:



Gambar 2.14 Arsitektur SimAM
Sumber: [125]

1. *Energy Function Calculation*

   Pada tahap ini, setiap neuron dalam *feature map* dihitung tingkat kepentingannya menggunakan fungsi energi. Fungsi ini mengukur seberapa baik suatu neuron dapat dibedakan dari neuron lain dalam satu *channel*. Semakin kecil nilai energi yang dihasilkan, maka semakin penting neuron

tersebut. Pendekatan ini didasarkan pada konsep bahwa neuron yang membawa informasi penting memiliki perbedaan yang signifikan dibandingkan neuron di sekitarnya.

2. *Attention Weight Estimation*

   Setelah nilai energi diperoleh, langkah selanjutnya adalah mengubah nilai tersebut menjadi bobot perhatian (*attention weight*). Neuron dengan energi rendah akan mendapatkan bobot lebih tinggi, sehingga kontribusinya dalam feature map diperkuat. Sebaliknya, neuron dengan energi tinggi akan memiliki bobot lebih kecil karena dianggap kurang relevan

3. *Feature Refinement*

   Pada tahap ini, bobot *attention* yang telah diperoleh dikalikan dengan *feature map* awal. Proses ini bertujuan untuk memperkuat fitur penting dan mengurangi pengaruh fitur yang tidak relevan. Hasilnya adalah *feature map* yang lebih bersih, fokus, dan informatif untuk tahap selanjutnya.

Rumus pada SimAM module dapat dilihat pada Rumus 2.18, yang merepresentasikan perhitungan *energy function* untuk menentukan tingkat *attention* setiap neuron berdasarkan hubungan antar *feature* serta penambahan regularisasi untuk menjaga kestabilan bobot [126].

$$e_t = \frac{1}{M-1} \sum_{i=1}^{M-1} (-1 - (\omega_t x_i + b_t))^2 + (1 - (\omega_t t + b_t))^2 + \lambda \omega_t^2 \quad (2.18)$$

Rumus 2.10 Perhitungan Nilai SimAM *Module*

Keterangan:

1. t= neuron target
2. x_i= neuron lain dalam satu channel
3. M= jumlah neuron dalam satu channel
4. ω_t, b_t= parameter linear
5. λ= parameter regularisasi
6. e_t= nilai energi neuron

Rumus 2.10 menunjukkan bahwa setiap neuron dihitung nilai energinya untuk mengetahui tingkat kepentingannya. Neuron yang memiliki nilai energi

rendah dianggap lebih penting karena memiliki perbedaan yang jelas dibandingkan neuron lainnya. Nilai ini kemudian digunakan untuk menghasilkan bobot perhatian tanpa memerlukan parameter tambahan. Proses kerja attention mechanism simAM dapat dilihat pada Tabel 2.8 [126].

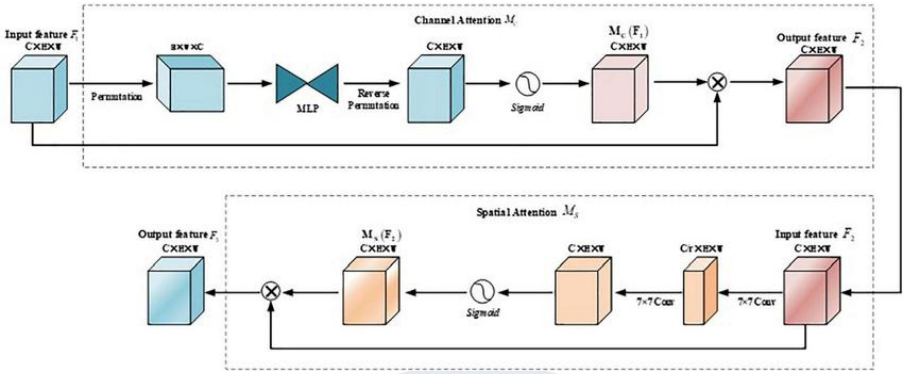
Tabel 2.8 *Pseudocode* SimAM

(1)	*Input* peta fitur *F* dari hasil ekstraksi *backbone.*
(2)	Hitung nilai energi setiap neuron dalam *feature map.*
(3)	Tentukan bobot *attention* berdasarkan nilai energi.
(4)	Normalisasi bobot *attention* untuk membentuk *attention map.*
(5)	Kalikan *attention map* dengan *feature map* untuk menghasilkan output akhir.

Pseudocode SimAM pada Tabel 2.8, dimulai dengan menerima *feature map* dari hasil ekstraksi fitur, kemudian setiap neuron dianalisis menggunakan fungsi energi untuk menentukan tingkat kepentingannya. Selanjutnya, bobot *attention* diberikan berdasarkan nilai tersebut dan digunakan untuk memperkuat fitur yang relevan. Hasil akhir berupa *feature map* yang lebih fokus dan informatif, sehingga model dapat meningkatkan performa dalam mendeteksi objek tanpa menambah kompleksitas jaringan.

2.4.6 GAM Module

Global Attention Mechanism (GAM) merupakan salah satu metode *attention mechanism* dalam *deep learning* yang bertujuan untuk meningkatkan kualitas representasi fitur dengan memodelkan hubungan antar *channel* dan spasial secara lebih menyeluruh [127]. Berbeda dengan metode seperti SE yang hanya berfokus pada *channel* atau CBAM yang memproses *channel* dan spasial secara terpisah, GAM mengintegrasikan keduanya secara lebih mendalam sehingga mampu mempertahankan informasi konteks global [127]. Pendekatan ini membuat GAM lebih efektif dalam menangkap fitur-fitur penting, khususnya pada objek berukuran kecil atau objek yang mengalami *occlusion* (tertutup sebagian). Gambar arsitektur GAM dapat dilihat pada Gambar 2.15 dimana komponen dalam GAM terdiri dari dua bagian utama, yaitu [128]:



Gambar 2.15 Arsitektur GAM
Sumber : [128]

1. *Channel Attention*

   Channel attention pada GAM bertujuan untuk menentukan tingkat kepentingan setiap *channel* dengan mempertimbangkan hubungan global antar fitur. Berbeda dengan pendekatan sederhana, GAM terlebih dahulu melakukan transformasi dimensi *feature map* agar informasi dapat dipelajari secara lebih fleksibel. Selanjutnya, *feature map* diproses menggunakan *Multi-Layer Perceptron* (MLP) dengan struktur encoder-decoder untuk menghasilkan bobot perhatian yang adaptif. Setelah itu, dimensi dikembalikan ke bentuk semula dan dikalikan dengan *feature map* awal, sehingga *channel* yang relevan akan diperkuat dan hubungan antar *channel* dapat ditangkap dengan lebih baik.

2. *Spatial Attention*

   Spatial attention pada GAM berfungsi untuk menentukan lokasi penting pada *feature map* dengan mempertahankan informasi spasial secara maksimal. Modul ini menggunakan dua lapisan konvolusi berukuran besar (biasanya 7×7) untuk menangkap hubungan spasial pada berbagai skala. Berbeda dengan CBAM yang menggunakan *pooling*, GAM tidak menggunakan pooling agar tidak kehilangan informasi penting. Untuk mengurangi kompleksitas akibat peningkatan parameter, digunakan teknik *group convolution* dengan pengacakan *channel*. Hasilnya adalah peta perhatian spasial yang mampu menyoroti area penting tanpa mengorbankan detail informasi.

Rumus pada GAM module dapat dilihat pada Rumus 2.19 dan Rumus 2.20. Proses *channel attention* direpresentasikan pada Rumus 2.19, dimana feature map

F_1dikalikan dengan $M_c(F_1)$untuk menghasilkan F_2. Kemudian proses *spatial attention* ditunjukkan pada Rumus 2.20 [129], di mana F_2dikalikan dengan $M_s(F_2)$untuk menghasilkan feature map akhir F_3 [129].

$$F_2 = M_c(F_1) \otimes F_1 \tag{2.19}$$

$$F_3 = M_s(F_2) \otimes F_2 \tag{2.20}$$

Rumus 2.11 Perhitungan Nilai GAM *Module*

Keterangan:

1. F_1= feature map input
2. F_2= feature map hasil channel attention
3. F_3= feature map hasil akhir
4. M_c= channel attention
5. M_s= spatial attention
6. $\otimes$= perkalian elemen (*element-wise multiplication*)

Rumus 2.11 menunjukkan bahwa proses *attention* dilakukan secara bertahap. Pada tahap pertama, *feature map* awal F_1diperkuat menggunakan *channel attention* untuk menghasilkan F_2. Selanjutnya, F_2diproses kembali menggunakan *spatial attention* untuk menghasilkan F_3yang lebih terfokus pada area penting. Cara kerja attention mechanism modul GAM dapat dilihat pada tabel 2.9 [129].

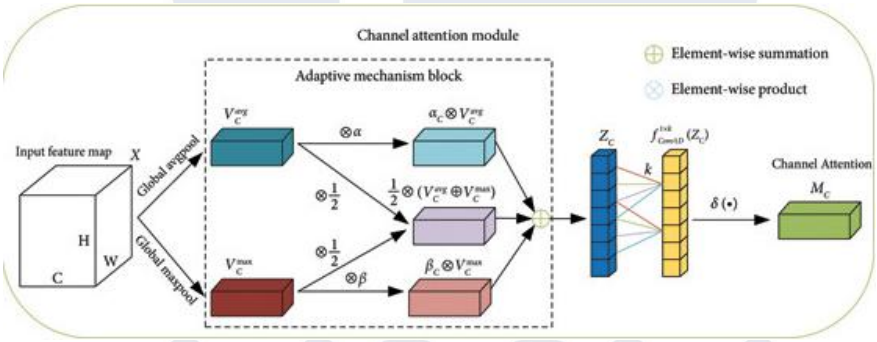
Tabel 2.9 Pseudocode GAM

(1)	*Input* peta fitur F_1dari hasil ekstraksi *backbone.*
(2)	Terapkan *channel attention* menggunakan transformasi dimensi dan MLP.
(3)	Kalikan hasil *channel attention* dengan *feature map* untuk mendapatkan F_2.
(4)	Terapkan *spatial attention* menggunakan *convolution* tanpa *pooling.*
(5)	Kalikan hasil *spatial attention* dengan F_2untuk menghasilkan *output* akhir F_3.

Pseudocode GAM Tabel 2.9, dimulai dengan feature map dari hasil ekstraksi fitur, kemudian diterapkan channel attention untuk memahami hubungan antar channel secara global. Setelah itu, feature map diproses menggunakan spatial attention untuk menyoroti lokasi penting tanpa menghilangkan informasi detail. Kombinasi kedua proses ini menghasilkan feature map yang lebih kaya konteks dan informatif, sehingga dapat meningkatkan performa model dalam mendeteksi objek, terutama pada kondisi kompleks.

2.4.7 CA Module

Channel Attention Module (CA *Module*) merupakan salah satu mekanisme *attention* dalam *deep learning* yang bertujuan untuk meningkatkan kualitas representasi fitur dengan cara memilih *channel* yang paling relevan [43]. Modul ini bekerja dengan memberikan bobot berbeda pada setiap *channel* sehingga fitur yang lebih penting akan diperkuat, sementara fitur yang kurang relevan akan ditekan [43]. Berbeda dengan pendekatan sederhana seperti SE Block yang hanya memanfaatkan *global average pooling*, CA *Module* menggabungkan informasi dari *average pooling* dan *max pooling* untuk menangkap karakteristik fitur secara lebih teratur [43]. Pendekatan ini membantu model memahami baik informasi umum maupun fitur yang paling menonjol pada setiap *channel*. Gambar arsitektur modul CA dapat dilihat pada Gambar 2.16 yang berisikan berberapa komponen dalam CA Module terdiri dari beberapa tahapan utama, yaitu:



Gambar 2.16 Arsitektur CA
Sumber: [130]

1. *Global Feature Extraction*

   Feature map hasil ekstraksi *backbone* diproses menggunakan dua jenis *pooling*, yaitu *global average pooling* dan *global max pooling*. Kedua operasi ini bertujuan untuk merangkum informasi penting dari setiap *channel* ke dalam representasi global sehingga hubungan antar *channel* dapat dipahami secara lebih menyeluruh.

2. *Channel Weight Learning* (MLP)

   Hasil dari kedua pooling tersebut kemudian dimasukkan ke dalam jaringan MLP yang terdiri dari dua lapisan *fully connected*. Lapisan pertama berfungsi untuk mereduksi dimensi *channel*, sedangkan lapisan kedua mengembalikan

dimensi ke ukuran semula. Kedua jalur (*average* dan *max*) menggunakan MLP yang sama, kemudian hasilnya digabungkan dan diaktifkan menggunakan fungsi sigmoid untuk menghasilkan bobot *attention* yang adaptif.

3. *Feature Recalibration*

   Bobot *attention* yang dihasilkan selanjutnya dikalikan dengan *feature map* awal secara elemen-per-elemen untuk memperkuat *channel* yang relevan. Selain itu, digunakan *residual connection* dengan menambahkan kembali *feature map* awal agar informasi asli tetap terjaga dan proses pelatihan menjadi lebih stabil.

Rumus pada CA *module* dapat dilihat pada Rumus 2.21 yang menunjukkan pembentukan *attention weight* β dan Rumus 2.22 menunjukkan penerapan *attention weight* tersebut pada *feature map*.

$$\beta = \sigma(MLP(P_{avg}(x)) + MLP(P_{max}(x))) \quad (2.21)$$

$$F_2 = x \otimes \beta + x \quad (2.22)$$

Rumus 2.12 Perhitungan Nilai CA *Module*

Keterangan:

1. x= feature map input
2. F_2= feature map hasil channel attention
3. P_{avg}= global average pooling
4. P_{max}= global max pooling
5. MLP= Multi-Layer Perceptron
6. σ= fungsi sigmoid
7. $\otimes$= perkalian elemen (element-wise multiplication)

Rumus 2.12 menunjukkan bahwa bobot attention diperoleh dari kombinasi informasi global rata-rata dan maksimum, kemudian digunakan untuk menyesuaikan kontribusi masing-masing channel terhadap feature map.

Tabel 2.10 *Pseudocode* CA

(1)	*Input feature map* $x$dari *backbone*.
(2)	Terapkan *global average pooling* dan *global max pooling*.
(3)	Masukkan kedua hasil *pooling* ke dalam MLP yang sama.

(4)	Jumlahkan hasil MLP dan aktifkan dengan sigmoid untuk mendapatkan bobot *channel* β.
(5)	Kalikan bobot β dengan *feature map* x.
(6)	Tambahkan *residual connection* untuk menghasilkan *output* akhir F_2.

Pseudocode CA *Module* dapat dilihat pada Tabel 2.10, dimulai dengan mengekstraksi informasi global dari setiap *channel* menggunakan dua pendekatan *pooling* yang berbeda, kemudian mempelajari bobot kepentingan *channel* melalui MLP. Setelah itu, *feature map* dikalibrasi ulang dengan memberikan penekanan pada *channel* yang relevan. Dengan mekanisme ini, model mampu menghasilkan representasi fitur yang lebih informatif dan adaptif, sehingga meningkatkan performa dalam berbagai tugas seperti klasifikasi maupun deteksi objek.

2.5 DetectBlock Module

DetectBlock berguna untuk membantu beberapa fitur yang hilang atau berkurang pada *backbone* dan *neck* terutama untuk objek kecil atau jauh, seperti helm yang terdeteksi pada jarak jauh. Sehingga Detectblock berguna dalam mempertahankan informasi fitur yang dalam dan meningkatkan akurasi deteksi [131]. DetecBlock terdiri dari *dilate convolution*, *attention mechanism*, dan lapisan C3. *Dilate convolution* memperbesar medan persepsi tanpa mengurangi resolusi, memungkinkan deteksi objek kecil atau jauh. *Attention mechanism* (seperti CBAM) memberikan bobot lebih pada informasi penting di dimensi spasial dan saluran. Lapisan C3, yang menggabungkan Bottleneck dan CSP, digunakan untuk mempelajari fitur residual dan memperoleh informasi fitur yang lebih dalam [132]. Pseudocode pada DetectBlok dapat dirumuskan pada Tabel 2.11.

Tabel 2.11 *Pseudocode* DetectBlock

(1)	Input: Mengambil *feature map* F dengan dimensi [H,W,C] di mana H adalah tinggi, W adalah lebar, dan C adalah jumlah saluran (channels).
(2)	Konvolusi Fitur: Terapkan dua lapisan convolution berturut-turut dengan kernel 3×3 untuk ekstraksi fitur dari peta fitur F.
(3)	*Spatial Attention*: Terapkan mekanisme *Spatial Attention* pada peta fitur F untuk fokus pada area penting dan meningkatkan kualitas representasi fitur.
(4)	*Global Average Pooling*: Gunakan *Global Average Pooling* untuk mengurangi dimensi peta fitur F dan menangkap informasi global yang relevan.
(5)	*Fully Connected Layer*: Terapkan lapisan *Fully Connected* (FC) pada peta fitur yang sudah diproses untuk memperbaiki representasi fitur dan menyiapkan untuk prediksi.

(6)	*Prediksi Bounding Box dan Confidence*: Terapkan convolution pada peta fitur hasil FC untuk menghasilkan prediksi *bounding box* dan skor *confidence* untuk setiap objek yang terdeteksi.
(7)	Output: Hasilkan prediksi akhir berupa koordinat *bounding box* dan *confidence score* berdasarkan peta fitur yang telah diproses.

Pseudocode DetectBlock Tabel 2.10, menggambarkan langkah-langkah utama dalam proses deteksi menggunakan *feature map*. Pertama, fitur diekstraksi melalui dua convolution layers, diikuti dengan mekanisme Spatial Attention yang meningkatkan spatial details. Kemudian, *Global Average Pooling* digunakan untuk mengurangi dimensi, menangkap *global contex*t. *Fully Connected layer* memperbaiki *feature representation*, yang kemudian digunakan untuk memprediksi *bounding box* dan *confidence score* melalui *convolution layer* lainnya. Output akhir berupa koordinat *bounding box* dan *confidence score*, yang memberikan lokasi dan kemungkinan objek yang terdeteksi. Proses ini memungkinkan model untuk fokus pada fitur yang relevan dan menghasilkan prediksi objek yang akurat.

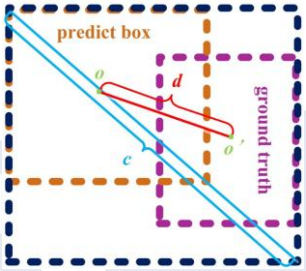
2.6 Loss Function

Loss function adalah komponen fundamental dalam *machine learning* dan optimasi, yang digunakan untuk mengukur adanya perbedaan antara prediksi model (y_{pred}) dengan target sebenarnya (y_{true}). Tujuan utama dari loss function ini adalah untuk meminimalkan nilai fungsi loss sehingga model dapat menghasilkan prediksi yang lebih akurat [133].

2.6.1 CIoU

CIoU Loss (*Complete Intersection over Union Loss*) adalah sebuah fungsi loss yang digunakan untuk meningkatkan akurasi dalam masalah deteksi objek, khususnya dalam prediksi lokasi *bounding box*. Fungsi ini dikembangkan sebagai perbaikan dari fungsi loss sebelumnya, seperti IoU (*Intersection over Union*) dan GIoU (*Generalized Intersection over Union*) [134]. Meskipun IoU dan GIoU sudah banyak digunakan untuk menghitung kesamaan antara dua kotak pembatas, keduanya memiliki keterbatasan dalam hal pengukuran kualitas prediksi, terutama ketika kotak pembatas yang diprediksi dan kotak target tidak memiliki area tumpang tindih yang signifikan atau bentuk yang sangat berbeda.

CIoU mengatasi masalah dengan memasukkan lebih banyak informasi untuk menilai kesesuaian antara prediksi dan *ground truth*. Dengan memperhitungkan jarak tengah antara dua kotak, rasio aspek (*aspect ratio*), dan area tumpang tindih [135], CIoU memberikan ukuran yang lebih lengkap mengenai kualitas prediksi kotak pembatas. Penjelasan lebih dapat dilihat pada gambar 2.17.



Gambar 2.17 CIoU *Loss Function Bounding Box*
Sumber: [136]

Gambar 2.17, menjelaskan CIoU (*Complete Intersection over Union*), *loss function* dalam deteksi objek untuk mengukur kesesuaian antara kotak prediksi (*predict box*) dan *ground truth box*. Kotak oranye mewakili *predict box,* sedangkan kotak ungu menunjukkan *ground truth*. CIoU mempertimbangkan tiga aspek utama: IoU (*Intersection over Union*) untuk mengukur tumpang tindih, jarak pusat (*distance*) (d garis merah) untuk mengukur posisi relatif kedua kotak, dan aspek rasio (*aspect ratio*) menggunakan diagonal kotak minimum (c, garis biru). Rumus CIoU mencakup pengurangan nilai akibat center *distance* $\frac{d^2}{C^2}$ dan perbedaan aspek rasio $\alpha \, . \, v$ Rumus dapat dilihat pada Rumus 2.23 sampai Rumus 2.25. Selain mengoptimalkan IoU. Tujuannya adalah memaksimalkan tumpang tindih, menyelaraskan posisi pusat, dan menyesuaikan dimensi kotak prediksi dengan *ground truth* [136].

$$LCIoU = 1 - IoU + \frac{d^2}{C^2} + \alpha \, . \, v \tag{2.23}$$

Rumus 2.13 Fungsi CIoU

$$v = \frac{4}{\pi^2}\left(arctan\frac{w^{gt}}{h^{gt}} - arctan\frac{w}{h}\right)^2 \tag{2.24}$$

Rumus 2.14 Fungsi v

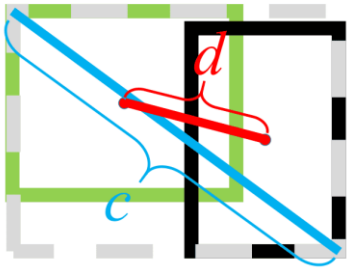
$$\alpha = \frac{v}{(1-IoU)+v} \tag{2.25}$$

Rumus 2.15 Fungsi α

Rumus *Complete IoU Loss* (CIoU) adalah pengembangan dari metrik *Intersection over Union* (IoU) yang mempertimbangkan tiga faktor utama: kesesuaian area, jarak antar pusat kotak, dan rasio lebar-tinggi. Komponen $1 - IoU$ mengukur ketidaksesuaian area, $\frac{d^2}{C^2}$ memberikan koreksi berdasarkan jarak antar pusat kotak yang dinormalisasi oleh diagonal minimum *enclosing box* (C), dan $\alpha . v$ memperbaiki perbedaan rasio lebar-tinggi kotak menggunakan fungsi *arctangent*. Dengan menggabungkan ketiga komponen ini, CIoU Loss meningkatkan kemampuan model untuk mendeteksi posisi, ukuran, dan bentuk objek secara lebih akurat, terutama untuk objek kecil atau bentuk yang kompleks [137].

2.6.2 DIoU

Distance Intersection over Union (DIoU) *Loss* merupakan pengembangan dari IoU dan GIoU yang bertujuan untuk meningkatkan akurasi regresi *bounding box* dengan mempertimbangkan tidak hanya area tumpang tindih, tetapi juga jarak antara pusat *bounding box* prediksi dan *ground truth* [40]. Metode digunakan untuk mengatasi keterbatasan GIoU yang belum mampu merepresentasikan hubungan spasial secara optimal, khususnya ketika posisi *bounding box* sudah saling mencakup namun belum sejajar secara tepat [138]. Dengan menambahkan komponen jarak, DIoU mampu mempercepat proses konvergensi serta memberikan arah optimasi yang lebih jelas dalam pelatihan model deteksi objek [40][138]. Penjelasan lebih lanjut dapat dilihat pada Gambar 2.18.



Gambar 2.18 DIOU *Loss Function Bounding Box*
Sumber : [40]

Gambar 2.18 menunjukkan konsep *Distance Intersection over Union* (DIoU) dalam proses evaluasi kesesuaian antara *bounding box* prediksi dan *ground truth box* pada deteksi objek. Pada gambar terlihat dua kotak pembatas, yaitu kotak hitam sebagai *predict box* dan kotak hijau sebagai *ground truth box*. DIoU tidak hanya mempertimbangkan tingkat tumpang tindih menggunakan *Intersection over Union* (IoU), tetapi juga memperhitungkan jarak antara titik pusat kedua kotak yang dilambangkan dengan d (garis merah). Selain itu, terdapat c yang menunjukkan panjang diagonal dari *bounding box* minimum yang mencakup kedua kotak tersebut (ditunjukkan oleh garis biru). Dengan memasukkan komponen jarak pusat d yang dinormalisasi oleh diagonal c, DIoU memberikan penalti ketika posisi kotak prediksi jauh dari *ground truth* sehingga membantu model mempercepat proses konvergensi dan meningkatkan akurasi dalam menentukan lokasi objek. Rumus DIoU dapat dilihat pada Rumus 2.26 [40].

$$L_{DIoU} = 1 - IoU + \frac{d^2(B_p,B_t)}{r^2} \tag{2.26}$$

Rumus 2.16 Persamaan Nilai DIoU

Keterangan:

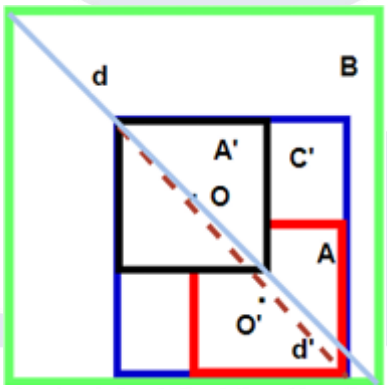
1. IoU: nilai *Intersection over Union* antara *predicted box* dan *ground truth box*.
2. $d(B_p, B_t)$: jarak *Euclidean* antara titik pusat *predicted box* (Bp) dan *ground truth box* (Bt).
3. r: panjang diagonal dari minimum *enclosing box* yang mencakup kedua *bounding box*.

Pada rumus tersebut, IoU digunakan untuk mengukur tingkat *overlap* antara *bounding box* hasil prediksi (Bp) dan *ground truth* (Bt). Semakin besar nilai IoU, maka nilai *loss* yang dihasilkan akan semakin kecil. Selanjutnya, d(Bp,Bt) merupakan jarak Euclidean antara titik pusat *bounding box* prediksi dan *ground truth*, sedangkan r adalah panjang diagonal dari *bounding box* minimum yang mencakup kedua kotak tersebut. Komponen $\frac{d^2(B_p,B_t)}{r^2}$ berfungsi sebagai penalti jarak yang menilai seberapa jauh posisi kedua kotak. Dengan adanya komponen ini, DIoU tidak hanya mempertimbangkan luas area tumpang tindih tetapi juga memperhitungkan jarak antara pusat kedua kotak, sehingga membantu model

mendekatkan posisi *bounding box* prediksi ke *ground truth* secara lebih cepat selama proses pelatihan [40].

2.6.3 *GIoU*

Generalized Intersection over Union Loss (GIoU) *loss* merupakan fungsi *loss* yang digunakan untuk meningkatkan akurasi prediksi *bounding box* pada deteksi objek [139]. Fungsi ini dikembangkan untuk mengatasi keterbatasan IoU yang hanya berfokus pada tingkat *overlap* antara *bounding box* prediksi dan *ground truth* [139]. Pada metode IoU, jika kedua kotak tidak saling tumpang tindih maka nilai IoU akan bernilai 0, sehingga tidak memberikan informasi mengenai seberapa jauh posisi kedua kotak tersebut [28]. Permasalahan ini membuat proses optimasi model menjadi kurang efektif karena model tidak mendapatkan informasi gradien yang cukup ketika kotak prediksi dan *ground truth* tidak saling overlap [28]. Oleh karena itu, GIoU diperkenalkan untuk memberikan informasi tambahan mengenai hubungan spasial antara kedua kotak dengan menambahkan komponen penalti berdasarkan area yang tidak saling tumpang tindih [139]. Penjelasan lebih dapat dilihat pada gambar 2.19.



Gambar 2.19 GIoU *Loss Function Bounding Box*
Sumber: [140]

Gambar 2.19 berisikan GIoU Loss Function Bounding box, dimana kotak hitam menunjukkan *ground truth*, kotak biru menunjukkan *predict box*, sedangkan kotak hijau (C) merupakan *bounding box* minimum yang mencakup kedua kotak tersebut. Area A' menunjukkan bagian yang saling tumpang tindih (*intersection*), sementara area lain di dalam C yang tidak termasuk dalam kedua kotak merupakan area penalti. GIoU menghitung kesamaan kedua kotak dengan mempertimbangkan

nilai IoU serta menambahkan penalti berdasarkan area kosong dalam C, sehingga model dapat menyesuaikan posisi *bounding box* prediksi agar lebih mendekati *ground truth*. Rumus GIoU *Loss* dapat dilihat pada Rumus 2.27 [140].

$$L_{GIoU} = 1 - IoU + \frac{|C-(B_{gt} \cup B_{pred})|}{|C|} \quad (2.27)$$

Rumus 2.17 Fungsi GIoU

Keterangan:

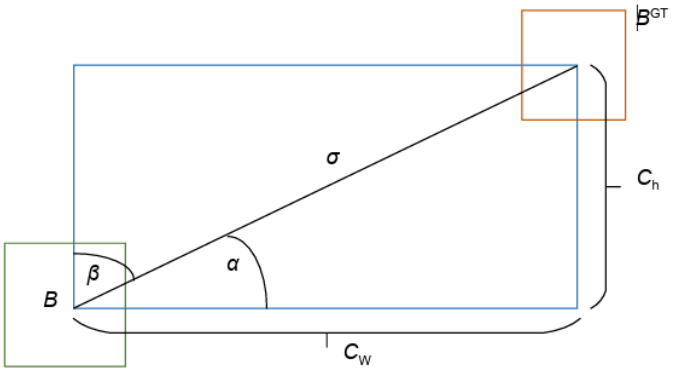
1. IoU : *Intersection over Union* antara *bounding box* prediksi dan *ground truth*.
2. C : *bounding box* minimum yang mencakup *bounding box* prediksi dan *ground truth*.
3. B_{gt}: *bounding box ground truth*.
4. B_{pred}: *bounding box* hasil prediksi model.

Rumus 2.17 merupakan pengembangan dari metrik IoU dengan menambahkan komponen penalti berdasarkan area yang tidak saling tumpang tindih antara *bounding box* prediksi dan *ground truth*. Komponen 1 − IoU digunakan untuk mengukur ketidaksesuaian area tumpang tindih antara kedua kotak, sedangkan komponen $\frac{|C-(B_{gt} \cup B_{pred})|}{|C|}$ mengukur rasio area kosong dalam kotak pembungkus minimum C yang tidak ditempati oleh kedua *bounding box* tersebut. Dengan memasukkan komponen penalti ini, GIoU dapat memberikan informasi gradien bahkan ketika kedua kotak tidak saling tumpang tindih, sehingga membantu model menyesuaikan posisi *bounding box* prediksi agar mendekati *ground truth* secara lebih efektif selama proses pelatihan [140].

2.6.4 SIoU

SIoU merupakan fungsi loss yang digunakan untuk meningkatkan akurasi regresi *bounding box* pada deteksi objek [141]. Fungsi ini dikembangkan untuk mengatasi keterbatasan loss function berbasis IoU sebelumnya yang hanya mempertimbangkan beberapa metrik seperti jarak pusat *bounding box*, area overlap, dan rasio aspek antara *bounding box* prediksi dan *ground truth* [141]. Pada metode sebelumnya seperti GIoU atau CIoU, proses optimasi model belum mempertimbangkan arah perbedaan posisi (*direction mismatch*) antara *bounding*

box prediksi dan *ground truth*. Akibatnya, selama proses pelatihan *bounding box* prediksi dapat bergerak secara tidak terarah (*wandering*) sebelum mencapai posisi yang benar, sehingga proses konvergensi menjadi lebih lambat dan kurang efisien. Penjelasan lebih dapat dilihat pada Gambar 2.18



Gambar 2.20 SIoU *Loss Function Bounding Box*
Sumber: [141]

Gambar 2.20 menunjukkan konsep perhitungan *angle cost* dan *distance cost* pada *loss function* SIoU. Kotak B merupakan *bounding box* prediksi, sedangkan B^{gt} adalah *bounding box ground truth*. Garis diagonal σ menunjukkan jarak antara pusat kedua *bounding box*, sementara sudut α dan β menunjukkan arah perbedaan posisi yang digunakan untuk menentukan arah konvergensi *bounding box* selama pelatihan. Kotak biru C merupakan *bounding box* minimum yang mencakup kedua kotak dengan ukuran C_w dan C_h, yang digunakan dalam perhitungan komponen *distance* pada SIoU. Rumus SIoU *Loss* dapat dilihat pada Rumus 2.28 [141].

$$L_{box} = 1 - IoU + \Delta + \frac{\Omega}{2} \tag{2.28}$$

Rumus 2.18 Fungsi SIoU

Keterangan:

1. IoU : *Intersection over Union* antara *bounding box* prediksi dan *ground truth*
2. Δ : *distance cost* yang mengukur jarak antara pusat *bounding box* prediksi dan *ground truth*
3. Ω : *shape cost* yang mengukur perbedaan ukuran dan rasio aspek *bounding box* prediksi dan *ground truth*

Nilai IoU dihitung sebagai perbandingan antara area irisan kedua *bounding box* dengan area gabungan keduanya.

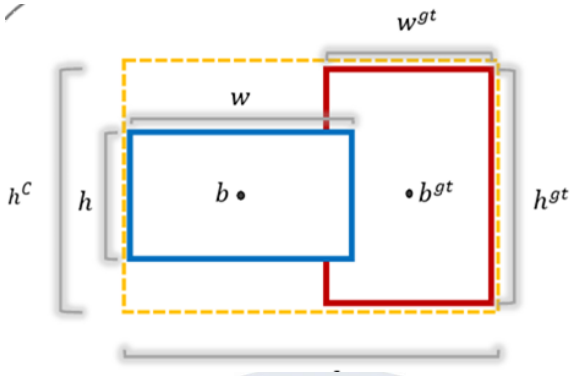
$$IoU = \frac{|B \cap B_{GT}|}{|B \cup B_{GT}|} \tag{2.29}$$

Rumus 2.19 Fungsi IoU

Pada Rumus 2.29, komponen 1 − IoU pada Rumus 2.29 digunakan untuk mengukur tingkat ketidaksesuaian pada area tumpang tindih antara *bounding box* prediksi dan *ground truth*. Komponen Δ (*distance cost*) digunakan untuk memperhitungkan jarak antara pusat kedua *bounding box* dengan mempertimbangkan arah perbedaan posisi. Sedangkan komponen Ω (*shape cost*) digunakan untuk mengukur kesesuaian ukuran dan rasio aspek *bounding box* prediksi terhadap *ground truth*. Dengan menggabungkan komponen sudut, jarak, bentuk, dan tingkat *overlap*, SIoU mampu mengarahkan proses konvergensi *bounding box* secara lebih efektif sehingga menghasilkan proses pelatihan model yang leih cepat dan akurat dalam mendeteksi objek.

2.6.5 EIoU

EIoU merupakan *loss function* yang digunakan untuk meningkatkan akurasi regresi *bounding box* pada deteksi objek [43]. Metode ini merupakan pengembangan dari CIoU yang bertujuan mempercepat proses konvergensi serta meningkatkan akurasi lokalisasi objek. Pada metode sebelumnya seperti IoU, GIoU, dan CIoU, perhitungan *loss* belum secara langsung meminimalkan perbedaan ukuran *bounding box* antara prediksi dan *ground truth* sehingga proses optimasi dapat berjalan lebih lambat. EIoU mengatasi hal tersebut dengan memisahkan komponen penalti ukuran *bounding box* secara langsung dalam tiga komponen utama yaitu IoU *loss*, *distance loss*, dan *aspect ratio loss* [43]. IoU *loss* mengukur tingkat tumpang tindih antara *bounding box* prediksi dan *ground truth*, *distance loss* menghitung jarak antara pusat kedua *bounding box*, sedangkan *aspect ratio loss* mengukur perbedaan lebar dan tinggi antara *bounding box* prediksi dan *ground truth* sehingga proses konvergensi model menjadi lebih cepat dan akurat [43]. Penjelasan lebih lanjut dapat dilihat pada Gambar 2.21.



Gambar 2.21 EIoU *Loss Function Bounding Box*
Sumber: [142]

Gambar 2.21 menunjukkan konsep perhitungan EIoU pada regresi *bounding box*. Kotak biru B merupakan *bounding box* prediksi dengan lebar w dan tinggi h, sedangkan kotak merah B^{gt} merupakan *bounding box ground truth* dengan lebar w^{gt} dan tinggi h^{gt}. Titik b dan b^{gt} menunjukkan pusat dari masing-masing *bounding box* yang digunakan untuk menghitung jarak pusat pada komponen *distance loss*. Kotak garis putus-putus kuning C merupakan *bounding box* pembungkus terkecil yang mencakup kedua kotak dengan ukuran w^c dan h^c, yang digunakan untuk proses normalisasi dalam perhitungan *loss*. Pada metode EIoU, perhitungan *loss* tidak hanya mempertimbangkan tingkat *overlap* (IoU) dan jarak pusat *bounding box*, tetapi juga secara langsung menghitung perbedaan lebar dan tinggi antara *bounding box* prediksi dan *ground truth* sehingga proses konvergensi model menjadi lebih cepat dan akurat. Rumus EIoU Loss dapat dilihat pada Rumus 2.30 [142].

$$L_{EIoU} = 1 - IoU + \frac{\rho^2(b,b_{gt})}{(w_c^2+h_c^2)} + \frac{\rho^2(w,w_{gt})}{w_c^2} + \frac{\rho^2(h,h_{gt})}{h_c^2} \quad (2.30)$$

Rumus 2.20 Fungsi EIoU

Keterangan:

1. IoU : *Intersection over Union* antara *bounding box* prediksi dan *ground truth*.
2. $\rho^2(b, b_{gt})$: jarak Euclidean antara pusat *bounding box* prediksi dan *ground truth*.
3. w dan h : lebar dan tinggi *bounding box* prediksi.
4. w_{gt} dan h_{gt}: lebar dan tinggi *bounding box ground truth*.
5. w_c^2 dan h_c^2 : lebar dan tinggi *bounding box*.

Nilai IoU dihitung sebagai perbandingan antara area irisan kedua bounding box dengan area gabungan keduanya.

$$IoU = \frac{|B \cap B_{GT}|}{|B \cup B_{GT}|} \tag{2.31}$$

Rumus 2.21 Fungsi IoU

Pada Rumus 2.30, komponen 1 - IoU pada Rumus 2.31 digunakan untuk mengukur ketidaksesuaian area tumpang tindih antara *bounding box* prediksi dan *ground truth*. Komponen *distance loss* digunakan untuk menghitung jarak antara titik pusat kedua *bounding box* yang dinormalisasi dengan ukuran *bounding box* pembungkus. Sedangkan komponen *aspect ratio loss* secara langsung menghitung perbedaan lebar dan tinggi antara *bounding box* prediksi dan *ground truth*. Dengan memisahkan komponen ukuran *bounding box* secara eksplisit, EIoU mampu meningkatkan kecepatan konvergensi serta menghasilkan lokalisasi objek yang lebih akurat dibandingkan metode IoU sebelumnya.

2.7 Tools/software yang digunakan

2.7.1 Python

Python adalah bahasa pemrograman yang dikembangkan oleh Guido van Rossum pada tahun 1991. Bahasa ini dikenal memiliki sintaks yang sederhana sehingga mudah dipahami. Python juga mendukung berbagai paradigma pemrograman, seperti prosedural, berorientasi objek, dan fungsional [143], Python sangat fleksibel dan digunakan dalam berbagai bidang. Dengan pustaka yang kaya seperti NumPy, Pandas, dan TensorFlow, Python menjadi pilihan utama untuk pengembangan web, analisis data, *machine learning*, automasi, hingga pengembangan game [143]. Sebagai bahasa yang bersifat *interpreted*, Python memudahkan *debugging* dan membantu pengembang menjalankan kode di berbagai platform seperti *Windows*, Mac, dan Linux tanpa banyak perubahan [144].

2.7.2 Roboflow

Roboflow adalah platform yang dirancang untuk memudahkan pengelolaan dataset dan proses pelatihan model dalam proyek *computer vision*, mulai dari tahap anotasi, augmentasi data, hingga ekspor dataset ke berbagai format yang dibutuhkan [145]. Roboflow sangat relevan dalam konteks YOLO (*You Only Look*

Once), salah satu algoritma deteksi objek paling populer. Dengan Roboflow, user dapat mengimpor dataset, melakukan anotasi secara efisien, dan menerapkan augmentasi otomatis yang mendukung performa YOLO. Roboflow membantu dalam menkonversi dataset ke format YOLO secara langsung, seperti YOLOv5 atau YOLOv8, sehingga mempermudah proses pelatihan model [146].

2.7.3 Google Colab

Google Colab adalah platform berbasis *cloud* yang dikembangkan oleh Google untuk membuat dan menjalankan notebook interaktif yang memuat kode, teks penjelasan, visualisasi, serta output secara langsung melalui browser tanpa memerlukan instalasi tambahan. Google Colab mendukung berbagai bahasa pemrograman, terutama Python, dan banyak digunakan dalam bidang *data science*, *machine learning*, analisis data, serta pendidikan [144]. Dengan sifatnya yang interaktif, pengguna dapat menulis kode, menjalankannya, dan langsung melihat hasilnya dalam satu tempat sehingga memudahkan proses eksplorasi, pengujian, dan dokumentasi program. Selain itu, Google Colab juga mendukung visualisasi data menggunakan pustaka seperti Matplotlib atau Seaborn [145], serta menyediakan akses GPU dan TPU untuk mempercepat proses komputasi dan pelatihan model *machine learning*.