



UNIVERSITAS INDONESIA

**PENGEMBANGAN SISTEM DETEKSI MULTI OBJEK JENIS
IKAN PADA VIDEO BERBASIS YOLOV7**

SKRIPSI

Muhammad Daffa Ajiputra

1906355781

**FAKULTAS TEKNIK
PROGRAM STUDI TEKNIK KOMPUTER**

DEPOK

MEI 2023



UNIVERSITAS INDONESIA

**PENGEMBANGAN SISTEM DETEKSI MULTI OBJEK JENIS
IKAN PADA VIDEO BERBASIS YOLOV7**

SKRIPSI

**Diajukan sebagai salah satu syarat untuk memperoleh gelar
Sarjana Teknik**

**Muhammad Daffa Ajiputra
1906355781**

**FAKULTAS TEKNIK
PROGRAM STUDI TEKNIK KOMPUTER**


**DEPOK
MEI 2023**

HALAMAN PERNYATAAN ORISINALITAS

**Skripsi ini adalah hasil karya saya sendiri,
dan semua sumber baik yang dikutip maupun dirujuk
telah saya nyatakan dengan benar.**

Nama : Muhammad Daffa Ajiputra

NPM : 1906355781

Tanda Tangan :  _____

Tanggal : 30 Mei 2023

LEMBAR PENGESAHAN

Skripsi ini diajukan oleh:

Nama : Muhammad Daffa Ajiputra

NPM : 1906355781

Program Studi : Teknik Komputer

Judul Skripsi : PENGEMBANGAN SISTEM DETEKSI MULTI OBJEK JENIS IKAN PADA VIDEO BERBASIS YOLOV7

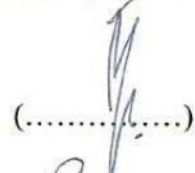
Telah berhasil dipertahankan di hadapan Dewan Penguji dan diterima sebagai bagian persyaratan yang diperlukan untuk memperoleh gelar Sarjana pada Program Studi Teknik Komputer Fakultas Teknik Universitas Indonesia.

DEWAN PENGUJI

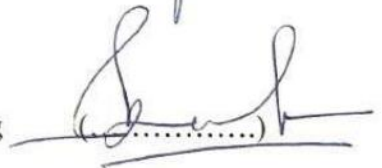
Pembimbing : Dr. Prima Dewi Purnamasari, S.T., M.T., M. Sc



Penguji 1 : Dr. Eng. Mia Rizkinia, S.T., M.T.



Penguji 2 : Prof. Dr. Ir. Anak Agung Putri Ratna, M.Eng



Ditetapkan di : Depok

Tanggal : 13 Juni 2023

KATA PENGANTAR

Puji dan syukur dipanjatkan ke hadirat Allah SWT atas berkat, rahmat, dan nikmat-Nya sehingga buku Skripsi ini dapat diselesaikan tepat waktu dengan judul “Pengembangan Sistem Deteksi Multi Objek Jenis Ikan pada Video Berbasis YOLOv7”. Buku skripsi ini dibentuk sebagai salah satu syarat kelulusan Fakultas Teknik Universitas Indonesia, khususnya bagi Program Studi S1 Teknik Komputer dalam Departemen Teknik Elektro.

Buku skripsi ini tidak akan terselesaikan tanpa adanya bantuan dan dukungan dari pihak lain. Pada kesempatan ini, penulis ingin memberikan apresiasi setinggi-tingginya kepada pihak-pihak terkait yang terdiri dari namun tidak terbatas pada:

1. Orang Tua dan keluarga penulis yang selalu memberikan dukungan dan fasilitas selama menjalani masa perkuliahan.
2. Ibu Dr. Prima Dewi Purnamasari S.T., M.Sc. selaku dosen pembimbing yang selalu mengarahkan dan memberikan bantuan, kritik, serta saran dalam pengerjaan buku skripsi.
3. Bapak Dr. Ruki Harwahyu S.T., M.T., M.Sc. selaku pembimbing akademis yang sudah memberikan arahan dan bantuan selama perkuliahan di program studi Teknik Komputer FTUI.
4. Tim peneliti proyek Fishmon baik Ibu Dr. Eng. Mia Rizkinia, S.T, M.T., pihak Aiseeyou, Ahmad Zufar Asshiddiqi, Karenina Kamila, dan Timothy Christian Panggabean yang memberikan bantuan dan wawasan dalam perancangan sistem.
5. Rekan-rekan seperbimbingan, grup “Tekkom Warrior”, asisten laboratorium digital serta rekan-rekan Teknik komputer 2019 yang selalu memberikan bantuan dan motivasi selama menjalani perkuliahan di program studi Teknik Komputer.

Penulis menyadari bahwa dalam menjalankan proses skripsi serta pembuatan buku terdapat beberapa kesalahan yang dilakukan. Oleh karena itu, penulis ingin memohon maaf kepada pihak yang merasakan kekurangan baik dari

penulis, pelaksanaan skripsi ataupun dari buku skripsi yang dibuat. Segala kritik dan saran konstruktif dapat disampaikan kepada penulis agar penulis menjadi pribadi yang lebih baik di waktu yang akan datang. Penulis juga berharap untuk mendapatkan kelancaran dalam menjalani kehidupan pasca-kampus serta menjadi pribadi yang berguna bagi agama, nusa dan bangsa.

Depok, Mei 2023

A handwritten signature in dark ink, appearing to read 'Daffa', with a horizontal line underneath.

Muhammad Daffa Ajiputra

HALAMAN PERNYATAAN PERSETUJUAN PUBLIKASI KARYA ILMIAH UNTUK KEPENTINGAN AKADEMIS

Sebagai sivitas akademik Universitas Indonesia, saya yang bertandatangan di bawah ini:

Nama : Muhammad Daffa Ajiputra
NPM : 1906355781
Program Studi : Teknik Komputer
Fakultas : Fakultas Teknik
Jenis Karya : Skripsi

demi pengembangan ilmu pengetahuan, menyetujui untuk memberikan kepada Universitas Indonesia **Hak Bebas Royalti Noneksklusif (Non-exclusive Royalty-Free Right)** atas karya ilmiah saya yang berjudul:

PENGEMBANGAN SISTEM DETEKSI MULTI OBJEK UNTUK PENDETEKSIAN JENIS IKAN BERBASIS YOLOV7

beserta perangkat yang ada (jika diperlukan). Dengan Hak Bebas Royalti Noneksklusif ini Universitas Indonesia berhak menyimpan, mengalihmedia/formatkan, mengelola dalam bentuk pangkalan data (database), merawat, dan memublikasikan tugas akhir saya tanpa meminta izin dari saya selama tetap mencantumkan nama saya sebagai penulis/pencipta dan sebagai pemilik Hak Cipta.

Demikian pernyataan ini saya buat dengan sebenarnya.

Dibuat di : Depok
Pada tanggal : 30 Mei 2023
Yang menyatakan



Muhammad Daffa Ajiputra

ABSTRAK

Nama : Muhammad Daffa Ajiputra
Program Studi : Teknik Komputer
Judul : Pengembangan Sistem Deteksi Multi Objek Jenis Ikan pada Video Berbasis YOLOv7

Indonesia merupakan salah satu negara pengekspor ikan terbesar di dunia yang membuat sektor perikanan Indonesia memiliki banyak ancaman. *Illegal, unreported, unregulated (IUU) fishing* adalah salah satu permasalahan yang memiliki dampak yang cukup signifikan karena membuat kerugian yang cukup besar di sektor perikanan Indonesia. Untuk mencegah permasalahan tersebut, sudah banyak solusi yang diajukan, salah satunya adalah penerapan teknologi seperti kamera pengawas, tetapi solusi tersebut belum memiliki dampak yang signifikan dalam mengurangi dan menghilangkan IUU *fishing*. Oleh karena itu, penelitian ini dilakukan untuk mengembangkan sistem deteksi multi objek untuk pendeteksian jenis ikan berbasis YOLOv7, sebuah model kecerdasan buatan yang dapat melakukan pendeteksian jenis ikan untuk melakukan pengawasan terhadap jumlah ikan yang ditangkap oleh nelayan sehingga IUU *fishing* dapat berkurang secara signifikan. Dari pengujian yang dilakukan, model YOLOv7 menjadi varian model YOLOv7 terbaik yang dapat digunakan untuk melakukan pendeteksian jenis ikan dengan nilai mAP yang dapat mencapai 86,1% dan *inference time* hingga 14,5 ms sehingga menghasilkan jumlah FPS yang dapat mencapai 69 FPS. Nilai tersebut berhasil didapatkan dengan menggunakan bentuk data *polygon annotation*, metode *object detection*, ukuran citra 800 piksel, dan jumlah *epochs* sebesar 1000 dengan *patience* 50. Namun, model YOLOv7 memiliki *inference time* yang sangat lambat hingga 797.6 ms ketika dipasang pada Jetson Nano meskipun akurasi pendeteksian memiliki hasil yang sama.

Kata Kunci: *deep learning, computer vision, object detection, YOLOv7*

ABSTRACT

Name : Muhammad Daffa Ajiputra
Study Program : Computer Engineering
Title : Development of Fish Variant Multi Object Detection System for Video based on YOLOv7

Indonesia is one of the world's largest exporters of fish, which exposes Indonesia's fishing sector to many threats. Illegal, unreported, unregulated (IUU) fishing is one of the problems that resulted in a significant impact in a form of a big loss that is created for the Indonesian fisheries sector. To prevent that problem, there are a lot of solutions that have been proposed, one of which is the application of technology such as surveillance cameras, but it still doesn't have a big impact to reduce and eliminate IUU fishing. Therefore, this research is conducted to develop a multi-object detection system for the detection of fish species based on YOLOv7, an artificial intelligence model that can detect a fish to supervise the number of fish that is caught by the fisherman so IUU fishing can reduce significantly. From the testing, the YOLOv7 model becomes the best YOLOv7 model variant that can be used to detect a fish with the value of mAP that can reach up to 86.1% and the value of inference time up to 14.5 ms that can produce an FPS total up to 69 FPS. The value can be achieved by doing some modifications in data annotation, the training model method, image size, and iteration on training. However, the YOLOv7 model has a very slow inference time up to 797.6 ms when it's installed in Jetson Nano even though the detection accuracy has the same value.

Key words: *deep learning, computer vision, object detection, YOLOv7*

DAFTAR ISI

HALAMAN PERNYATAAN ORISINALITAS	ii
LEMBAR PENGESAHAN	iii
KATA PENGANTAR.....	iv
HALAMAN PERNYATAAN PERSETUJUAN PUBLIKASI KARYA ILMIAH UNTUK KEPENTINGAN AKADEMIS	vi
ABSTRAK	vii
ABSTRACT	viii
DAFTAR ISI.....	ix
DAFTAR GAMBAR.....	xi
DAFTAR TABEL	xvi
DAFTAR ISTILAH	xvii
DAFTAR SINGKATAN.....	xix
BAB 1 PENDAHULUAN	1
1.1. Latar Belakang	1
1.2. Rumusan Masalah	3
1.3. Tujuan Penelitian	3
1.4. Batasan Masalah.....	4
1.5. Metodologi Penelitian	4
1.6. Sistematika Penulisan	5
BAB 2 SISTEM DETEKSI MULTI OBJEK BERBASIS YOLOV7	7
2.1. Machine Learning	7
2.1.1. Deep Learning	8
2.2. Computer Vision	12
2.2.1. Object Detection.....	14
2.3. Convolutional Neural Networks	20
2.4. YOLO.....	23
2.4.1. YOLOv7.....	25
2.5. Fishnet.....	30
2.6. <i>Polygon annotation</i>	31
2.7. Penelitian Terkait	32

BAB 3 PERANCANGAN SISTEM DETEKSI MULTI OBJEK BERBASIS YOLOV7	34
3.1. <i>System Requirements</i>	34
3.2. Pengumpulan Data	35
3.3. <i>Preprocessing</i> Data	36
3.4. Rancangan Proses Pelatihan Model	42
3.5. Rancangan Proses Pengujian Model	45
3.6. Rancangan Proses Implementasi Model pada Jetson Nano	46
BAB 4 IMPLEMENTASI DAN ANALISIS SISTEM DETEKSI MULTI OBJEK BERBASIS YOLOV7	50
4.1. Implementasi Sistem	50
4.2. <i>Dataset</i> Pengujian	51
4.3. Skenario Pengujian.....	52
4.4. Pengujian Model	53
4.4.1. Pengujian Pengaruh Data Berbentuk <i>Polygon annotation</i> Terhadap Akurasi dan Jumlah FPS	54
4.4.2. Pengujian Perbandingan Metode <i>Object Detection</i> dengan <i>Semantic Segmentation</i> Terhadap Akurasi dan Jumlah FPS.....	57
4.4.3. Pengujian Pengaruh Ukuran Citra Pelatihan Model Terhadap Akurasi dan Jumlah FPS	61
4.4.4. Pengujian Pengaruh Penambahan Epochs dalam Memicu <i>Early Stopping</i> Terhadap Akurasi dan Jumlah FPS.....	66
4.4.5. Pengujian Pendeteksian Model dengan Jetson Nano	74
4.5. Dampak Terhadap Lingkungan, Masyarakat dan/atau Bidang Lain.....	78
BAB 5 KESIMPULAN	80
5.1. Kesimpulan	80
5.2. Saran.....	81
DAFTAR REFERENSI	82

DAFTAR GAMBAR

Gambar 2.1 Perbandingan alur <i>machine learning</i> tradisional dan <i>deep learning</i> [13]	8
Gambar 2.2 Contoh <i>backpropagation</i> pada 4-layer neural networks [13]	12
Gambar 2.3 Contoh <i>Region of Interest</i> pada citra sebuah anjing [13]	15
Gambar 2.4 Hasil prediksi sebelum dan sesudah diterapkan <i>non-maximum suppression</i> [13]	17
Gambar 2.5 Contoh <i>Precision-Recall Curve</i> dengan <i>IoU Threshold</i> = 0.5	18
Gambar 2.6 Proses ekstraksi fitur pada <i>convolutional layers</i> [13]	21
Gambar 2.7 Contoh <i>max pooling feature map</i> berukuran 4 X 4 dengan <i>window</i> berukuran 2 x 2 [13]	22
Gambar 2.8 Contoh <i>global average pooling feature map</i> berukuran 4 X 4 [13]..	22
Gambar 2.9 Arsitektur <i>fully connected layers</i> [13]	23
Gambar 2.10 Ilustrasi pendeteksian menggunakan algoritma YOLO [12]	24
Gambar 2.11 Arsitektur awal model YOLO [12]	25
Gambar 2.12 Arsitektur model YOLOv7-Tiny (Digambar ulang berdasarkan [8])	27
Gambar 2.13 <i>Layer</i> modul <i>Convolutional</i> YOLOv7-Tiny (Digambar ulang berdasarkan [8])	27
Gambar 2.14 Arsitektur ELAN model YOLOv7-Tiny (Digambar ulang berdasarkan [8])	27
Gambar 2.15 <i>Layer</i> modul SPPCSPC YOLOv7-Tiny (Digambar ulang berdasarkan [8]).....	27
Gambar 2.16 Arsitektur model YOLOv7 (Digambar ulang berdasarkan [8])	28

Gambar 2.17 <i>Layer</i> modul <i>Convolutional</i> YOLOv7 (Digambar ulang berdasarkan [8]).....	28
Gambar 2.18 Arsitektur ELAN pada bagian <i>backbone</i> (Atas) dan <i>head</i> (Bawah) model YOLOv7 (Digambar ulang berdasarkan [8])	29
Gambar 2.19 <i>Layer</i> modul SPPCSPC YOLOv7 (Digambar ulang berdasarkan [8])	29
Gambar 2.20 <i>Layer</i> modul MP pada YOLOv7 (Digambar ulang berdasarkan [8])	29
Gambar 2.21 Arsitektur model YOLOv7-X (Digambar ulang berdasarkan [8])..	30
Gambar 2.22 Arsitektur ELAN model YOLOv7-X (Digambar ulang berdasarkan [8]).....	30
Gambar 2.23 Contoh data <i>Polygon Annotation</i> pada kerusakan sebuah mobil [18]	31
Gambar 3.1 Rancangan sistem pendeteksian jenis ikan.....	34
Gambar 3.2 Contoh citra dari <i>dataset</i> Fishnet [17].....	36
Gambar 3.3 Tahapan <i>pre-processing</i> data	37
Gambar 3.4 Kode untuk mengekstrak <i>frame</i> pada berkas video	38
Gambar 3.5 Perintah untuk membuat dan mengaktifkan <i>virtual environment</i> pada <i>anaconda</i> [19]	38
Gambar 3.6 Perintah untuk memasang dan mengaktifkan Label Studio [19]	38
Gambar 3.7 Halaman utama Label Studio	39
Gambar 3.8 Tampilan UI untuk metode anotasi yang dapat digunakan	39
Gambar 3.9 Contoh citra yang belum diberikan label	40
Gambar 3.10 Pemberian <i>Polygon Annotation</i> pada citra	40

Gambar 3.11 Hasil akhir label <i>Polygon Annotation</i> pada citra.....	41
Gambar 3.12 Skema pelatihan model YOLOv7	42
Gambar 3.13 Kode untuk mengonversi <i>dataset</i> dari <i>Polygon Annotation</i> menjadi <i>Bounding Box</i>	44
Gambar 3.14 Perintah untuk mencari <i>Swap Memory</i> [21]	47
Gambar 3.15 Perintah untuk mencari ruang pada memori penyimpanan [21]	47
Gambar 3.16 Perintah untuk membuat <i>Swap File</i> [21].....	48
Gambar 3.17 Perintah untuk mengaktifkan <i>Swap File</i> menjadi <i>Swap Memory</i> [21]	48
Gambar 4.1 Rincian skenario pengujian	53
Gambar 4.2 Grafik mAP0.5 pengujian bentuk data pada varian model	55
Gambar 4.3 Grafik <i>inference time</i> pengujian bentuk data pada varian model.....	55
Gambar 4.4 Grafik <i>F1-score</i> pengujian bentuk data pada varian model	55
Gambar 4.5 Contoh citra pengujian dengan label yang benar	56
Gambar 4.6 Hasil pendeteksian model YOLOv7 yang dilatih dengan data a) <i>Bounding box</i> dan b) <i>Polygon annotation</i>	56
Gambar 4.7 Grafik mAP pengujian metode pelatihan pada varian model	58
Gambar 4.8 Grafik <i>inference time</i> pengujian metode pelatihan pada varian model	59
Gambar 4.9 Grafik <i>F1-score</i> pengujian metode pelatihan pada varian model	59
Gambar 4.10 Hasil pengujian model YOLOv7-X menggunakan metode <i>semantic segmentation</i> dengan a) citra pengujian metode <i>semantic segmentation</i> dengan Label yang Benar b) hasil pendeteksian.....	60

Gambar 4.11 Hasil pengujian model YOLOv7-X menggunakan metode <i>object detection</i> dengan a) Citra pengujian metode <i>object detection</i> dengan label yang benar b) Hasil pendeteksian	60
Gambar 4.12 Grafik <i>inference time</i> pengujian ukuran citra pelatihan model pada varian model.....	62
Gambar 4.13 Grafik <i>F1-score</i> pengujian ukuran citra pelatihan model pada varian model.....	62
Gambar 4.14 Perbandingan tren peningkatan nilai a) mAP0.5 dan b) mAP0.5:0.95 YOLOv7-Tiny dengan ukuran citra 640 piksel (Merah) dan 800 piksel (Oranye).....	63
Gambar 4.15 Perbandingan tren peningkatan nilai a) mAP0.5 dan b) mAP0.5:0.95 YOLOv7 dengan ukuran citra 640 piksel (Biru Muda) dan 800 piksel (Biru Tua)	63
Gambar 4.16 Perbandingan tren peningkatan nilai a) mAP0.5 dan b) mAP0.5:0.95 YOLOv7-X dengan ukuran citra 640 piksel (Hijau) dan 800 piksel (Merah Muda)	63
Gambar 4.17 Hasil pengujian model YOLOv7-Tiny dengan ukuran citra a) 640 piksel dan b) 800 piksel.....	64
Gambar 4.18 Hasil pendeteksian model YOLOv7 dengan ukuran citra a) 640 piksel dan b) 800 piksel	64
Gambar 4.19 Hasil pendeteksian model YOLOv7-X dengan ukuran citra a) 640 piksel dan b) 800 piksel.....	65
Gambar 4.20 Grafik mAP0.5 pengujian peningkatan epochs dan perbedaan <i>patience</i> saat epochs sebesar 1000 pada varian model YOLOv7	67
Gambar 4.21 Grafik <i>inference time</i> pengujian peningkatan epochs dan perbedaan <i>patience</i> saat epochs sebesar 1000 pada varian model YOLOv7.....	68
Gambar 4.22 Grafik <i>F1-score</i> pengujian peningkatan epochs dan perbedaan <i>patience</i> saat epochs sebesar 1000 pada varian model YOLOv7.....	68

Gambar 4.23 Contoh citra pada pengujian dengan label yang benar.....	69
Gambar 4.24 Hasil pengujian pada model YOLOv7 dengan a) <i>epochs</i> 300, b) <i>epochs</i> 1000 dengan <i>patience</i> 50, c) <i>epochs</i> 1000 dengan <i>patience</i> 100	69
Gambar 4.25 <i>Confusion matrix</i> hasil pengujian model YOLOv7 dengan <i>epochs</i> sebesar 300.....	71
Gambar 4.26 <i>Confusion matrix</i> hasil pengujian model YOLOv7 dengan <i>epochs</i> sebesar 1000 dengan <i>patience</i> sebesar 50	72
Gambar 4.27 <i>Confusion matrix</i> hasil pengujian model YOLOv7 dengan <i>epochs</i> sebesar 1000 dengan <i>patience</i> 100.....	73
Gambar 4.28 Grafik <i>inference time</i> pengujian model pada Jetson Nano.....	75
Gambar 4.29 Contoh <i>frame</i> hasil <i>inference</i> pada video hiv00167.mp4.....	76
Gambar 4.30 Citra hasil ekstraksi <i>frame</i> video pengujian dengan label yang benar	77
Gambar 4.31 Hasil pengujian akurasi pendeteksian pada a) Jetson Nano dan b) Komputer pelatihan.....	77

DAFTAR TABEL

Tabel 2.1 Persebaran bagian <i>dataset</i> Fishnet	31
Tabel 2.2 Hasil <i>average precision</i> untuk setiap kelas yang dilatih[9].	32
Tabel 2.3 <i>Confusion matrix</i> dari hasil pengujian [9].....	32
Tabel 2.4 Akurasi pendeteksian dari sampel secara langsung [9]	33
Tabel 3.1 Spesifikasi komputer.....	35
Tabel 4.1 Spesifikasi Jetson Nano untuk pengujian model	50
Tabel 4.2 Persebaran citra pada <i>dataset</i> yang dikembangkan	51
Tabel 4.3 Jumlah data tiap kelas pada <i>dataset</i> yang dikembangkan.....	52
Tabel 4.4 Hasil pengujian model dengan <i>dataset</i> berbentuk <i>Polygon annotation</i> dan <i>Bounding Box</i>	54
Tabel 4.5 Hasil pengujian model dengan metode <i>Object Detection</i> dan <i>Semantic Segmentation</i>	58
Tabel 4.6 Hasil pengujian model dengan ukuran citra 640 piksel dan 800 piksel	62
Tabel 4.7 Hasil pengujian model dengan peningkatan <i>epochs</i>	67
Tabel 4.8 Perbandingan <i>F1-score</i> model YOLOv7 dalam mendeteksi kelas yang termasuk ikan	70
Tabel 4.9 Hasil pengujian model pada Jetson Nano dan komputer pelatihan	75
Tabel 4.10 Hasil pengujian akurasi model pada Jetson Nano dan komputer pelatihan	77

DAFTAR ISTILAH

Istilah	Penjelasan
<i>Bounding Box</i>	Salah satu jenis label yang digunakan pada deteksi objek berupa kotak yang mengelilingi objek yang terdeteksi.
<i>Computer Vision</i>	Cabang dari <i>machine learning</i> yang memungkinkan sebuah komputer melakukan identifikasi pada sebuah citra atau video.
CUDA	<i>Framework</i> yang digunakan untuk mengaktifkan utilisasi GPU dalam membentuk model <i>machine learning</i>
CUDNN	<i>Library</i> yang digunakan untuk mempercepat proses pembentukan model.
<i>Dataset</i>	Kumpulan data yang digunakan sebagai masukan untuk proses pelatihan dan pengujian model.
<i>Hyperparameter</i>	Parameter yang digunakan untuk mengatur proses pelatihan model.
Label	Tanda yang digunakan untuk memberikan kelas pada sebuah data yang akan dipelajari model sehingga dapat diidentifikasi.
<i>Machine Learning</i>	Sub-bidang dari <i>Artificial Intelligence</i> (AI) untuk membuat komputer bisa mereplika kemampuan yang dimiliki seorang manusia dengan mempelajari data dalam jumlah yang banyak.

Model	Hasil pelatihan berupa sebuah <i>weights</i> untuk menemukan pola dalam mengidentifikasi objek.
<i>Object Detection</i>	Pengembangan dari <i>image classification</i> untuk membuat model dapat mengetahui lokasi dan jumlah objek pada sebuah citra.
<i>Polygon Annotation</i>	Salah satu jenis label yang digunakan pada deteksi objek berupa bentuk dari objek yang akan dideteksi.
<i>Semantic Segmentation</i>	Pengembangan <i>object detection</i> dengan membentuk hasil pendeteksian yang serupa dengan objek yang terdeteksi.
<i>Virtual Environment</i>	Sebuah <i>tools</i> untuk membentuk area khusus pengembangan sebuah proyek.
YOLOv7	Versi ketujuh dari algoritma YOLO (<i>You Only Look Once</i>) yang merupakan salah satu algoritma deteksi objek buatan Joseph Redmon.

DAFTAR SINGKATAN

Singkatan	Kepanjangan
CNN	<i>Convolutional Neural Networks</i>
CPU	<i>Central Processing Unit</i>
ELAN	<i>Efficient Layer Aggregation Networks</i>
FPS	<i>Frames Per Second</i>
GPU	<i>Graphical Processing Unit</i>
IoU	<i>Intersection over Union</i>
mAP	<i>Mean Average Precision</i>
RAM	<i>Random Access Memory</i>
SGD	<i>Stochastic Gradient Descent</i>

BAB 1

PENDAHULUAN

1.1. Latar Belakang

Indonesia merupakan salah satu negara pengekspor ikan terbesar di dunia. Tercatat pada tahun 2021, Indonesia berada di peringkat ke tujuh negara pengekspor ikan terbesar di dunia untuk ikan yang sudah difillet maupun ikan yang dibekukan dengan valuasi sebesar 558 juta US Dollar [1]. Salah satu komoditi ekspor ikan Indonesia yang memiliki nilai valuasi yang tinggi adalah ikan tuna. Nilai valuasi ekspor ikan tuna dari Indonesia pada tahun 2021 mencapai 325.4 juta US Dollar dengan volume ekspor mencapai 48.3 Mega Ton untuk ikan tuna yang difillet dan sudah dibekukan [2].

Di sisi lain, dengan banyaknya komoditi ekspor laut Indonesia, Indonesia pun tidak terlepas dari kasus *illegal, unreported, unregulated (IUU) fishing*. Tercatat pada tahun 2021, Kementerian kelautan dan perikanan menangkap 167 Kapal Indonesia maupun Kapal Asing seperti kapal dari Vietnam dan Malaysia yang melakukan IUU *fishing* [3]. Diperkirakan IUU *fishing* yang terjadi selama 2021 membuat Indonesia mengalami kerugian sebesar 74 Juta US Dollar [4]. Kementerian Kelautan dan Perikanan sendiri sudah menerapkan beberapa solusi untuk mencegah terjadinya IUU *fishing* di wilayah Indonesia, sebagai contoh pada Maret 2021, terdapat 8 kapal Malaysia yang ditenggelamkan akibat *illegal fishing* [5]. Selain itu, negara-negara di ASEAN juga saling bekerja sama untuk mendorong keterbukaan informasi di sektor perikanan melalui sistem pemantauan kapal serta sistem pelaporan dan pengawasan elektronik.

Penerapan teknologi sendiri sudah menjadi salah satu metode yang digunakan untuk mencegah terjadinya tindak kejahatan di berbagai bidang, salah satu contohnya adalah *computer vision*. *Computer vision* merupakan bidang pada *artificial intelligence (AI)* yang memungkinkan komputer untuk mengobservasi dan memahami informasi penting yang ada pada suatu citra, video dan masukan berbentuk visual lainnya [6]. Hingga saat ini, sudah terbentuk berbagai metode

penerapan *computer vision* di berbagai aspek kehidupan, salah satunya adalah *object detection*.

Object detection merupakan metode *computer vision* yang berfungsi untuk melakukan pendeteksian sebuah objek pada suatu citra atau video [7]. Algoritma *object detection* pertama kali ditemukan pada tahun 2001 oleh pasangan Viola-Jones yang disebut sebagai *VJ-Detectors* yang saat itu masih terbuat secara manual. *Object detection* mengalami transisi pada 2012 menggunakan Teknik otomasi ketika *Alexnet* ditemukan. Sejak saat itu, algoritma *object detection* berbasis *deep learning* banyak ditemukan salah satunya adalah YOLO (*You Only Look Once*). YOLO pertama kali ditemukan oleh Joseph Redmon pada tahun 2016. Algoritma YOLO terus mengalami perkembangan dan saat ini, sudah ada algoritma YOLO versi ketujuh (YOLOv7) yang merupakan pengembangan dari YOLO versi kelima.

YOLOv7 merupakan perkembangan terkini dari algoritma YOLO. Tujuan utama dari pengembangan YOLOv7 adalah mengurangi jumlah parameter dan proses komputasi yang dibutuhkan untuk membentuk model *machine learning* tanpa mengurangi kecepatan dan akurasi pendeteksian. Berdasarkan jurnal yang dipublikasikan, YOLOv7 memiliki kecepatan dan akurasi pendeteksian yang lebih baik dibandingkan beberapa algoritma yang sudah dikembangkan sebelumnya seperti YOLOR, YOLOX, Scaled-YOLOv4, YOLOv5, DETR, dan lain-lain tanpa menggunakan *pre-trained weights* saat pelatihan model [8].

Terdapat beberapa penelitian yang bertujuan untuk melakukan pendeteksian jenis ikan menggunakan *object detection* dengan algoritma YOLO sebelum YOLOv7, seperti YOLOv4 dan YOLOv5. Sebagai contoh, penelitian yang dilakukan menggunakan algoritma YOLOv4-Tiny pada tahun 2022 yang digunakan untuk mendeteksi jenis ikan di sebuah akuarium dengan Raspberry Pi 4 [9]. Pada tahun yang sama pula, terdapat penelitian yang dilakukan dengan YOLOv5s, YOLOv5m dan YOLOv5l untuk mendeteksi jenis ikan yang berada di bawah air [10]. Rata-rata dari penelitian tersebut berfokus untuk melakukan pendeteksian pada ikan yang bukan merupakan komoditi ekspor seperti ikan tuna. Selain itu, sudut pandang dari citra yang digunakan sebagai *dataset* sangat dekat

dengan ikan sehingga terdapat kemungkinan ketika bentuk objek tidak sesuai atau jarak objek cukup jauh, objek tersebut tidak dapat diklasifikasikan.

Dari latar belakang tersebut, penelitian ini dilakukan untuk membentuk sistem deteksi multi objek ikan pada sebuah video menggunakan algoritma YOLOv7. Pemilihan YOLOv7 sebagai algoritma yang digunakan didasari oleh algoritma tersebut yang memiliki beberapa keunggulan dibandingkan dengan algoritma deteksi objek sebelumnya. Sistem deteksi ini akan mendeteksi beberapa objek seperti manusia dan jenis ikan yang terdapat pada berkas video yang berasal dari hasil tangkapan kamera yang terpasang pada kapal. Sudut pandang dari kamera sendiri akan diarahkan ke satu arah yang merupakan tempat ikan tersebut disimpan sehingga seluruh aktivitas baik penangkapan ikan maupun penyimpanan ikan dapat dipantau. Nantinya, model tersebut dapat dikembangkan untuk melakukan perhitungan jumlah ikan yang ditangkap oleh nelayan pada kapal tersebut sehingga jika terjadi perbedaan antara jumlah ikan yang terdeteksi dengan jumlah ikan yang dilaporkan, dapat diindikasikan terjadi proses IUU *fishing* pada kapal tersebut.

1.2. Rumusan Masalah

Rumusan masalah dari penelitian ini adalah sebagai berikut:

1. Bagaimana cara membuat sebuah sistem deteksi multi objek berbasis YOLOv7 sebagai sistem pendeteksian jenis ikan?
2. Apakah dengan menggunakan metode *semantic segmentation* dapat meningkatkan hasil pendeteksian dibandingkan dengan metode *object detection*?
3. Varian model YOLOv7 manakah yang memiliki hasil terbaik dan tingkat kesalahan yang kecil dalam melakukan pendeteksian jenis ikan?

1.3. Tujuan Penelitian

Tujuan dari penelitian ini adalah sebagai berikut:

1. Merancang sistem pendeteksian multi objek dengan algoritma YOLOv7 yang dapat membedakan beberapa jenis ikan hasil tangkapan nelayan.

2. Membandingkan hasil pendeteksian model YOLOv7 menggunakan *semantic segmentation* dan *object detection* sebagai metode pelatihan model untuk mendeteksi objek.
3. Membandingkan dan mengevaluasi model YOLOv7 sehingga diperoleh varian YOLOv7 terbaik yang akan digunakan dalam pembentukan sistem pendeteksian jenis ikan secara multi objek.

1.4. Batasan Masalah

Batasan masalah dalam penelitian ini adalah sebagai berikut:

1. Sistem pendeteksian harus bisa mendeteksi enam jenis objek (*human, tuna, skipjack tuna, tongkol, squid, unknown*) berdasarkan himpunan data yang akan dikumpulkan dari video kamera cctv yang terpasang pada kapal.
2. Sistem pendeteksian akan mendeteksi melalui kamera cctv yang akan memberikan keluaran berupa sebuah kotak pembatas dengan label untuk objek yang terdeteksi.
3. Proses pelatihan dan pengujian model YOLOv7 dilakukan terhadap suatu citra dari video yang ditangkap melalui kamera cctv pada kapal dengan enam jenis objek (*human, tuna, skipjack tuna, tongkol, squid, unknown*) yang mungkin terdapat pada citra.

1.5. Metodologi Penelitian

Metode yang digunakan selama penelitian ini adalah sebagai berikut:

1. Studi literatur
Tahapan studi literatur mencakup pencarian dan pengumpulan referensi yang berkaitan dengan semua penerapan teknologi yang diterapkan pada penelitian ini.
2. Konsultasi dengan dosen pembimbing dan tim peneliti
Konsultasi dilakukan secara rutin setiap pekan dengan dosen pembimbing untuk mendapatkan umpan balik kemajuan penelitian, menerima saran dan membahas hambatan yang dialami mengenai topik-topik yang berkaitan

dengan penelitian ini. Konsultasi juga dilakukan dengan tim peneliti untuk mendapatkan wawasan tambahan terkait model yang akan digunakan.

3. Pengumpulan dan pelabelan data

Pengumpulan data dilakukan dengan mengobservasi tiap *frame* video hasil tangkapan kamera pada kapal yang terdiri dari jenis ikan yang akan diklasifikasikan dan akan dilabeli sesuai dengan jenis ikan tersebut.

4. Desain Sistem

Desain sistem pendeteksian jenis ikan dilakukan dengan membuat rancangan sistem yang akan diimplementasikan sesuai batasan masalah yang telah ditentukan.

5. Rancangan proses pelatihan model

Rancangan proses pelatihan model YOLOv7 dibentuk sesuai dengan studi literatur dan referensi-referensi lain sehingga dapat menghasilkan model yang maksimal.

6. Rancangan proses pengujian dan perbandingan model

Rancangan proses pengujian dan perbandingan model YOLOv7, YOLOv7-Tiny dan YOLOv7-X dibentuk untuk membandingkan hasil pelatihan model dengan seimbang tanpa mengunggulkan varian mana pun.

7. Analisis

Analisis diberikan terhadap rancangan sistem pendeteksian jenis ikan yang dibentuk dan ketiga varian model YOLOv7 yang telah melewati proses pelatihan model.

8. Kesimpulan

Pemberian kesimpulan terhadap keseluruhan penelitian yang telah dilakukan berdasarkan hasil yang diperoleh dan analisis yang didapatkan dari rancangan sistem pendeteksian jenis ikan dan perbandingan varian model YOLOv7 agar dapat diterapkan pada sistem tersebut.

1.6. Sistematika Penulisan

Sistematika Penulisan pada penelitian ini dibagi ke dalam 5 bab, yaitu:

BAB I PENDAHULUAN

Bab ini menjelaskan latar belakang, tujuan, batasan masalah, dan metodologi yang digunakan sebagai acuan pelaksanaan penelitian.

BAB II DASAR TEORI

Bab ini menjelaskan teori yang digunakan dalam pembuatan rancangan sistem pendeteksian jenis ikan dan perbandingan model YOLOv7 yang digunakan.

BAB III PERANCANGAN SISTEM

Bab ini menjelaskan rincian dari rancangan sistem pendeteksian jenis ikan yang telah dibentuk, data dan pemrosesan data yang diterapkan dalam pelatihan dan pengujian data, serta rancangan proses pelatihan dan pengujian varian model YOLOv7.

BAB IV IMPLEMENTASI DAN ANALISIS

Bab ini menjelaskan implementasi dari sistem yang telah dirancang serta menampilkan hasil serta analisis dari pengujian sistem pada beberapa skenario yang diuji.

BAB V KESIMPULAN

Bab ini menjelaskan kesimpulan dari hasil penelitian yang telah dilakukan beserta beberapa saran yang dapat diterapkan untuk pengembangan sistem yang telah terbentuk.

BAB 2

SISTEM DETEKSI MULTI OBJEK BERBASIS YOLOV7

2.1. Machine Learning

Machine learning merupakan salah satu Sub-bidang dari *Artificial Intelligence* (AI) yang bertujuan untuk membuat komputer bisa mereplika kemampuan yang dimiliki seorang manusia dengan menginstruksikannya untuk mempelajari sebuah data dalam jumlah yang banyak [11]. Menurut Mitchell (1997), kata “learning” untuk komputer didefinisikan sebagai “Sebuah program belajar dari *experience* yang terdiri dari beberapa kelas *tasks* dan *performance measure*, jika *tasks* yang diukur dengan *performance measure* akan meningkat sesuai dengan *experience*” [12].

Machine learning memungkinkan kita untuk menyelesaikan *tasks* yang terlalu sulit untuk diselesaikan hanya dengan suatu program yang dibentuk oleh seorang manusia dan bersifat tetap. *Tasks* didefinisikan sebagai bagaimana suatu sistem *machine learning* harus mengolah sebuah *example* yang didefinisikan sebagai kumpulan dari *features* yang telah diukur secara kuantitatif dari beberapa objek atau kejadian yang akan diproses oleh sistem *machine learning*. Sebagai contoh, *example* yang digunakan merupakan suatu citra, maka *features* dari sebuah citra adalah piksel-piksel yang ada pada citra tersebut. Salah satu *tasks* yang paling umum diselesaikan menggunakan *machine learning* adalah klasifikasi. *Tasks* jenis ini meminta suatu komputer untuk coba mengklasifikasikan suatu masukan ke beberapa kategori yang ada.

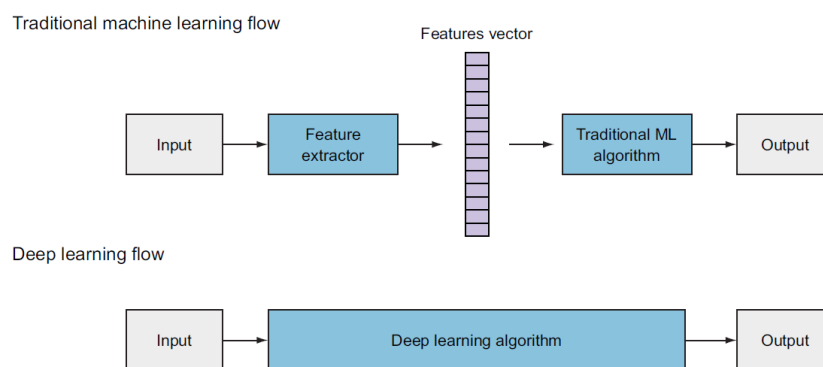
Untuk mengevaluasi kemampuan suatu algoritma *machine learning*, kita perlu membentuk *performance measure* yang sesuai dengan *tasks* yang terbentuk pada sistem. Sebagai contoh, pada *tasks* klasifikasi salah satu parameter pengukurannya adalah *accuracy*, yaitu proporsi dari *examples* yang keluarannya tepat dari model yang terbentuk. Selain itu, terdapat juga *error rate*, yaitu proporsi dari *examples* yang keluarannya salah dari model yang terbentuk. Lalu, untuk mengukur seberapa baik model yang terbentuk dalam penggunaannya di dunia

nyata, biasanya digunakan sebuah *test set* berupa data yang berbeda dari data yang digunakan untuk melatih sistem *machine learning*.

Algoritma *machine learning* sendiri dapat dibedakan sesuai dengan *experience* apa yang diberikan pada algoritma tersebut di mana sebuah *experience* akan berbentuk sebuah *dataset*, yaitu kumpulan dari banyak *examples*. Secara umum, *experience* yang dapat diterima oleh sebuah *machine learning* terbagi menjadi *unsupervised* dan *supervised*. Algoritma *unsupervised learning* adalah algoritma yang mendapatkan *experience* dari *dataset* yang memiliki banyak *features* dan mempelajari properti dari *dataset* tersebut, sedangkan algoritma *supervised learning* adalah algoritma yang mendapatkan *experience* dari *dataset* yang memiliki sebuah *features*, tetapi juga terikat dengan sebuah label.

2.1.1. Deep Learning

Deep learning merupakan cabang dari *machine learning* yang menggunakan banyak layer dari *neural networks* [11]. *Deep learning* merupakan pengembangan dari alur *machine learning* tradisional yang masih menggunakan *feature extractor* untuk bisa mengambil *features* yang ada pada suatu masukan. Algoritma *deep learning* sendiri sudah memiliki *feature extractor* pada layer *neural networks* nya sehingga proses ekstraksi *features* dari masukan pada algoritma *deep learning* dilakukan secara otomatis.



Gambar 2.1 Perbandingan alur *machine learning* tradisional dan *deep learning* [13]

Deep learning terdiri dari beberapa layer *neural networks* di mana suatu *neural networks* terdiri dari beberapa *perceptron*. *Perceptron* merupakan *neural*

network yang paling sederhana. *Perceptron* memiliki cara kerja yang mirip dengan neuron yang ada pada otak manusia di mana neuron manusia akan menerima sinyal elektrik dari *dendrite*, mengatur sinyal elektrik tersebut menjadi beberapa bagian dan mengirimkannya menuju *synapsis* untuk dikirimkan ke neuron lain ketika sinyal masukan sudah melebihi batasannya. Dari cara kerja tersebut, *perceptron* bekerja dengan dua fungsi, yaitu mengalkulasi *weighted sum* masukan yang merepresentasikan kekuatan sinyal dan menerapkan *step function* untuk menentukan apakah sinyal tersebut sudah melewati batas yang ditentukan atau belum. Oleh karena itu, suatu *perceptron* memiliki empat bagian utama, yaitu:

- A. *Input vector*, yaitu vektor dari *feature* yang berhasil diekstrak dari masukan. Umumnya, disimbolkan dengan X sebagai sebuah vektor dari masukan (x_1, x_2, \dots, x_n) .
- B. *Weights vector*, yaitu representasi seberapa pentingnya *example* tersebut dibandingkan dengan *example* lain yang ada pada *dataset*. Direpresentasikan dengan sebuah vektor dari *weights* (w_1, w_2, \dots, w_n) .
- C. *Neuron functions*, yaitu kalkulasi dari sinyal masukan berupa penjumlahan dari *weighted sum* dan *step activation function*.
- D. *Output*, yaitu hasil *neuron functions* yang diatur oleh *activation function*.

Sebuah *perceptron* merupakan sebuah fungsi linear yang sangat efektif digunakan untuk suatu *dataset* linear di mana *dataset* tersebut dapat dibagi oleh sebuah garis lurus. Akan tetapi, jika *dataset* yang digunakan memiliki bentuk yang lebih kompleks, contohnya adalah sebuah citra atau video, sebuah *perceptron* kurang efektif untuk menyelesaikan permasalahan tersebut. Oleh karena itu, untuk menyelesaikan permasalahan tersebut, dibutuhkan lebih dari satu *perceptron* untuk bisa menyesuaikan fungsi yang ada pada *dataset* atau bisa disebut sebagai *multilayer perceptron* [13].

Deep learning menggunakan *multilayer perceptron* dalam membentuk arsitekturnya. *Multilayer perceptron* bekerja dengan menumpuk neuron menjadi beberapa layer yang biasa disebut sebagai *hidden layer*. Layer yang ada pada *hidden*

layer saling terhubung satu sama lain dengan sebuah *weights connections*. Komponen utama yang ada pada suatu *multilayer perceptron* adalah:

- A. *Input layer*, yaitu layer yang terdiri dari *features* yang dimiliki oleh masukan data.
- B. *Hidden layers*, yaitu layer yang terdiri dari neuron-neuron yang akan mengekstrak vektor dari *feature* yang akan diproses pada *input layer*.
- C. *Weight connections (edges)*, yaitu nilai yang terdapat pada tiap hubungan antar *nodes* pada *neural networks* untuk menggambarkan pentingnya *nodes* tersebut pada hasil prediksi.
- D. *Output layer*, yaitu hasil dari pemrosesan pada *hidden layers* yang dapat berbentuk suatu nilai riil atau kumpulan dari sebuah probabilitas.

Deep learning membutuhkan tiga tahap yang akan dilakukan secara terus menerus untuk menghasilkan model yang maksimal. Tahap pertama, melakukan kalkulasi dari *weights sum* tiap *nodes* dan *activation* untuk menghasilkan suatu prediksi atau disebut sebagai proses *feedforward* [14]. *Weights sum* adalah penjumlahan dari vektor *weights* yang ada pada suatu *perceptrons* untuk tiap masukan. Persamaan yang digunakan untuk menentukan *weights sum* adalah:

$$\mathbf{s}_i = \mathbf{w}_i^T \mathbf{x}_i + \mathbf{b}_i \quad (2.1)$$

Setelah mendapatkan *weights sum* tiap *nodes*, *weights sum* tersebut akan dikalikan dengan non-linear *activation function* yang direpresentasikan dengan h sehingga didapatkan persamaan untuk mendapatkan hasil prediksi tiap masukan adalah:

$$\mathbf{y}'_i = \mathbf{h}(\mathbf{s}_i) \quad (2.2)$$

Tahap kedua, mengukur tingkat kesalahan dari hasil prediksi *neural networks* dibandingkan dengan keluaran yang seharusnya. Caranya adalah dengan menerapkan suatu *error function* atau bisa juga disebut sebagai *loss functions*. Jika hasil prediksi memiliki nilai *loss* yang tinggi, model yang terbentuk kurang baik. Salah satu *loss function* yang umum digunakan adalah *cross-entropy*. *Cross-entropy* adalah *loss function* yang umum digunakan dalam permasalahan klasifikasi. *Cross-entropy* akan menghitung perbedaan antara dua distribusi probabilitas, yaitu

predicted distribution untuk distribusi hasil prediksi dan *true distribution* untuk distribusi hasil sebetulnya [13]. Persamaan untuk *cross-entropy* adalah:

$$E(W, b) = - \sum_{i=1}^m y'_i \log(p_i) \quad (2.3)$$

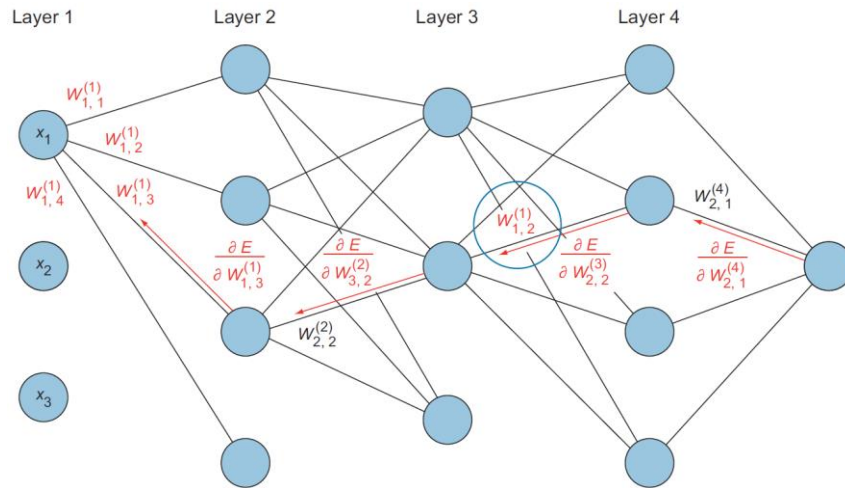
di mana y' adalah probabilitas sesungguhnya, p adalah probabilitas hasil prediksi dan m adalah jumlah kelas yang dilatih. Untuk mengetahui kesalahan *cross-entropy* untuk semua data (n) pada *dataset*, persamaan yang digunakan akan menjadi:

$$E(W, b) = - \sum_{i=1}^n \sum_{j=1}^m y'_{ij} \log(p_{ij}) \quad (2.4)$$

Tahap ketiga, menerapkan *optimization functions* seperti *gradient descent* atau jenis lainnya untuk memodifikasi *neural networks* secara berulang hingga di titik paling optimalnya yang dilakukan dengan mengalkulasi turunan *loss function* E_n untuk example n terhadap *weights* w menggunakan *chain rule* dari *output layer* menuju *input layer* yang bisa disebut sebagai *backpropagation*. Secara umum, *backpropagation* melakukan turunan dari *error* yang didapatkan terhadap suatu *weights*. Setelah mengalami *backpropagation*, *weights* yang dimiliki suatu node akan diperbaharui mengikuti persamaan:

$$w_{next-step} = w_{current} + \Delta w \quad (2.5)$$

Ketika *backpropagation* diterapkan pada suatu *multilayer perceptron*, terdapat banyak *weights* yang perlu dikalkulasi. Oleh karena itu, digunakan operasi *chain rule* untuk mencari nilai turunan dari *error* terhadap *weights* sesuai dengan *node* yang akan dilewati ketika kembali menuju *input layer*.



Gambar 2.2 Contoh *backpropagation* pada 4-layer neural networks [13]

Sebagai contoh, pada Gambar 2.2, untuk mencari tahu *weights* dari masukan setelah diterapkan *backpropagation*, turunan *error* terhadap *weights* dari *output layer* menuju masukan yang dituju adalah:

$$\frac{dE}{dw_{1,3}^{(1)}} = \frac{dE}{dw_{2,1}^{(4)}} \times \frac{dw_{2,1}^{(4)}}{dw_{2,2}^{(3)}} \times \frac{dw_{2,2}^{(3)}}{dw_{3,2}^{(2)}} \times \frac{dw_{3,2}^{(2)}}{dw_{1,3}^{(1)}} \quad (2.6)$$

2.2. Computer Vision

Computer vision adalah salah satu Sub-bidang dari *artificial intelligence* yang berfokus pada bagian visual untuk bisa menciptakan suatu teknologi yang bisa memahami suatu benda di dunia nyata melalui video atau citra. Teknologi *computer vision* dibentuk berdasarkan *visual perception*, yaitu bentuk aksi untuk melakukan observasi sebuah pola atau objek menggunakan suatu masukan visual. *Computer vision* dilatih tidak hanya untuk melihat lingkungan di sekitar kita, tetapi juga membentuk sistem yang bisa memahami lingkungan di sekitarnya melalui masukan visual [13].

Computer vision menjadi salah satu topik yang memiliki lingkup penelitian yang aktif hingga saat ini dikarenakan menjadi salah satu topik yang cukup menantang untuk sebuah komputer. Hal tersebut dikarenakan *computer vision* merupakan *inverse problem*, di mana kita mencari beberapa informasi yang tidak

cukup untuk diketahui dan mengembalikan informasi tersebut untuk menemukan solusi sepenuhnya dari permasalahan tersebut [14]. Dibutuhkan sebuah model yang didasari oleh konsep fisika dan *probabilistic* atau sebuah *machine learning* dengan *dataset* yang besar untuk bisa menghilangkan keambiguan dari solusi yang mungkin digunakan, tetapi memodelkan suatu data visualisasi dunia nyata jauh lebih kompleks dari yang dibayangkan.

Secara umum, *computer vision* memiliki prinsip yang sama dengan sistem penglihatan pada manusia. Sistem penglihatan seorang manusia terdiri dari mata yang berfungsi untuk menangkap visual yang ada di lingkungan sekitar makhluk tersebut serta otak yang berfungsi untuk memahami objek dari visual yang dilihat oleh mata. Dari konsep tersebut, *computer vision* memiliki dua komponen untuk diterapkan, yaitu *sensing devices* dan *interpreting device*.

Sensing devices adalah perangkat yang berfungsi seperti mata, yaitu untuk menangkap visual yang ada di sekitar perangkat tersebut. *Sensing devices* yang digunakan untuk menerapkan *computer vision* perlu disesuaikan dengan pengaplikasian yang diinginkan. Beberapa perangkat yang umum digunakan sebagai *sensing devices* seperti kamera, radar, CT scan, dan lain-lain.

Interpreting device adalah perangkat yang berfungsi seperti otak, yaitu untuk memahami dan menafsirkan objek yang berhasil ditangkap oleh *sensing devices*. Pada *computer vision*, algoritma yang digunakan merupakan *interpreting device* pada sistem *computer vision*. Karena prinsip kerja yang mirip layaknya otak seorang manusia, ilmuwan menerapkan konsep yang sama untuk membuat algoritma yang bertindak sebagai *interpreting device*. Dari konsep tersebut, terciptalah teknologi yang biasa dikenal sebagai *artificial neural networks* (ANNs). Ketika sebuah jaringan neuron memiliki jutaan hingga miliaran neuron didalamnya, jaringan tersebut akan menghasilkan suatu algoritma yang mampu melakukan pembelajaran yang biasa disebut *deep learning* [13].

2.2.1. Object Detection

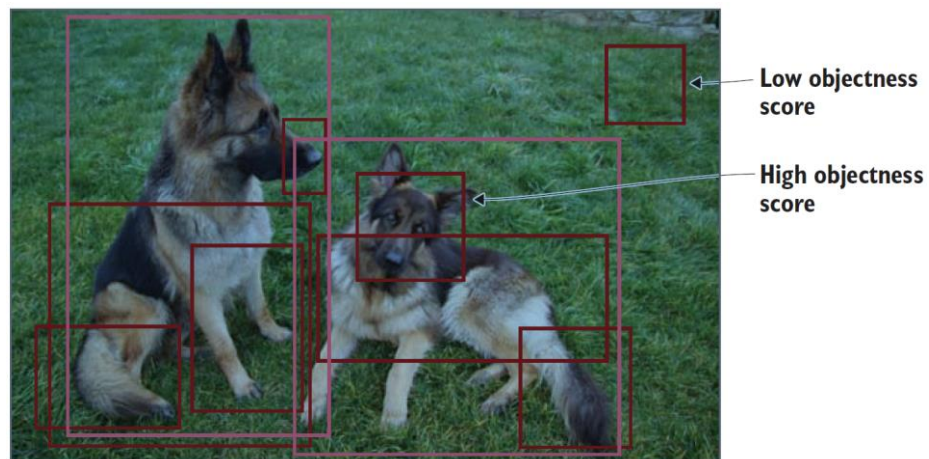
Object detection merupakan salah satu contoh pengaplikasian teknologi *computer vision* di kehidupan nyata. *Object detection* merupakan pengembangan dari *image classification* di mana *object detection* berfungsi untuk membuat *interpreting device* yang diterapkan pada sistem *computer vision* bisa mengetahui lokasi dan jumlah objek yang terdapat di suatu citra. Secara umum, *object detection* akan mengidentifikasi lokasi dan kelas objek dengan memecah citra tersebut menjadi bagian yang lebih kecil dan memberikan kelas untuk setiap bagian yang dipecah sehingga objek yang terdapat pada citra tersebut dapat terlabeli [13].

Framework object detection memiliki empat komponen utama, diantaranya:

A. Region Proposals

Region proposals adalah suatu bagian yang diberikan oleh model *deep learning* berbentuk suatu *region of interest* (RoIs) yang akan diproses lebih lanjut oleh sistem. *Region of interest* adalah bagian pada suatu citra yang sistem perkirakan memiliki sebuah objek yang dikalkulasi menggunakan *objectness score*.

Pada tahap ini, sistem akan membentuk ribuan *bounding boxes* untuk dianalisis dan diklasifikasikan oleh *neural networks*. *Objectness score* dari tiap *bounding boxes* adalah hasil analisis dari *neural networks* yang akan mengklasifikasikan *bounding boxes* tersebut sebagai *foreground* (Objek) atau *background* (Bukan Objek). Jika *bounding boxes* tersebut melewati *threshold* yang ditetapkan pada *neural networks*, *bounding boxes* tersebut akan diklasifikasikan sebagai *foreground* dan akan diteruskan ke tahap berikutnya oleh *neural networks*.



Gambar 2.3 Contoh *Region of Interest* pada citra sebuah anjing [13]

Terdapat beberapa parameter yang perlu diperhatikan, seperti *threshold* yang terkonfigurasi. Apabila terlalu kecil, akan terlalu banyak *bounding boxes* yang terbentuk oleh *neural networks* yang memungkinkan semua objek terdeteksi, tetapi akan membuat proses pendeteksian lebih lambat karena membutuhkan proses komputasional yang kompleks. Oleh karena itu, penetapan parameter perlu disesuaikan dengan permasalahan yang ingin diselesaikan untuk mengurangi jumlah RoIs yang terbentuk.

B. *Feature extraction and network predictions*

Komponen ini adalah lanjutan dari *region proposal* di mana *bounding boxes* yang termasuk sebagai *foreground region* akan diekstrak *feature*-nya dan akan digunakan untuk menentukan kelas dari objek yang dikenali pada citra tersebut. *Feature* pada *foreground region* akan diekstrak oleh sebuah *pretrained model image classification* agar hasil yang didapatkan akan tergeneralisasi secara merata. Setelah mengekstraksi *feature*, *foreground region* akan dianalisis oleh *neural networks* dan dibuat dua prediksi untuk tiap *foreground region*, yaitu:

- *Bounding-box prediction*, yaitu prediksi untuk menentukan lokasi dari *foreground region* pada citra tersebut. Prediksi akan berbentuk sebuah *tuple* (x, y, w, h) di mana x dan y adalah koordinat titik tengah dari *foreground region*, sedangkan w dan h adalah panjang dan lebar dari region tersebut.

- *Class prediction*, yaitu prediksi yang umumnya terbentuk dengan *activation function softmax* yang akan memprediksi probabilitas tiap kelas yang dilatih untuk objek pada citra tersebut.

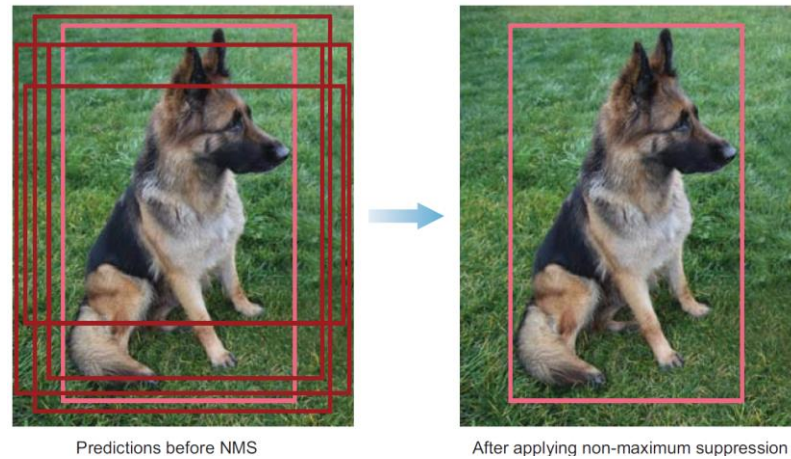
Sebuah objek yang dikenali pada suatu citra dapat memiliki beberapa *foreground region* yang secara tepat melakukan klasifikasi, tetapi tidak semua *region* akan digunakan untuk melakukan deteksi. Di beberapa permasalahan, hanya dibutuhkan satu *foreground region* saja dengan tingkat prediksi tertinggi untuk melakukan suatu pendeteksian karena *foreground region* yang banyak akan membuat objek tersebut terhitung dengan jumlah lebih dari satu. Oleh karena itu, *foreground region* yang bukan merupakan *region* dengan tingkat prediksi tertinggi perlu dihapuskan untuk mencegah hal tersebut.

C. *Non-maximum suppression (NMS)*

Non-maximum suppression (NMS) adalah komponen yang digunakan untuk memastikan bahwa hanya ada satu *bounding boxes* untuk tiap objek. Cara kerja dari komponen adalah NMS akan mencari semua *bounding boxes* yang mengarah ke sebuah objek dan mencari *bounding boxes* dengan probabilitas prediksi tertinggi dan akan menghilangkan semua *bounding boxes* yang memiliki probabilitas prediksi lebih rendah.

Cara kerja dari *non-maximum suppression* dalam mencari probabilitas prediksi tertinggi adalah:

- Menghilangkan semua *bounding boxes* yang probabilitas prediksinya tidak melewati *confidence threshold* yang sudah disesuaikan dengan kebutuhan.
- Mencari semua *bounding boxes* yang tersisa dan memilih *bounding boxes* dengan probabilitas tertinggi
- Mengalkulasi area yang saling meniban antara *bounding boxes* hasil prediksi (*predicted*) dengan *bounding boxes* acuan (*ground truth*) dan mencari komparasi nilai area tersebut. Nilai komparasi itu disebut sebagai *intersection over union (IoU)*.
- Menghilangkan *bounding boxes* yang memiliki nilai IoU lebih rendah dibandingkan *threshold* yang diterapkan.



Gambar 2.4 Hasil prediksi sebelum dan sesudah diterapkan *non-maximum suppression* [13]

D. Evaluation Metrics

Komponen ini adalah komponen yang akan mengevaluasi hasil prediksi dari tiga komponen sebelumnya. Beberapa satuan yang umum digunakan dalam mengevaluasi hasil prediksi sebuah model *object detection* adalah:

- *Frames Per Second* (FPS)

Frames Per Second adalah satuan evaluasi yang mengukur kecepatan sebuah model *object detection* melakukan pendeteksian. Jumlah FPS dapat dikalkulasi menggunakan nilai *inference time* dengan persamaan:

$$FPS = \frac{1000}{inference\ time\ (ms)} \quad (2.7)$$

- *Intersection over Union* (IoU)

Intersection over Union adalah satuan evaluasi yang akan menilai *overlap* yang terjadi antara *bounding boxes* hasil prediksi (*predicted*) dengan *bounding boxes* acuan (*ground truth*). IoU akan menentukan apakah hasil pendeteksian valid (*True Positive*) atau tidak (*False Positive*) di mana range nilainya antara 0-1 dan semakin besar nilainya, semakin baik hasil pendeteksian. Nilai IoU suatu *bounding boxes* dapat ditentukan menggunakan persamaan:

$$IoU = \frac{B_{ground\ truth} \cap B_{predicted}}{B_{ground\ truth} \cup B_{predicted}} \quad (2.8)$$

Jika suatu *bounding boxes* memiliki nilai yang lebih rendah dibandingkan dengan *threshold* yang diterapkan, hasil pendeteksian dianggap tidak valid (*False Positive*), sedangkan jika melebihi *threshold*, hasil pendeteksian dianggap valid (*True Positive*).

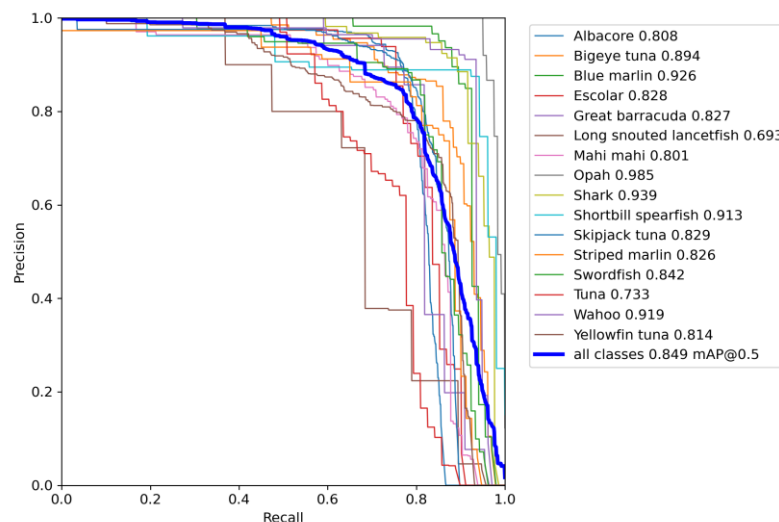
- *Precision-recall Curve (PR-Curve)*

Precision-recall Curve adalah kurva perbandingan antara *precision* dan *recall* dari suatu model *object detection* yang akan diproses untuk setiap *confidence threshold*. *Precision* adalah satuan evaluasi yang mengukur kemampuan model untuk mengidentifikasi sebuah objek yang relevan pada sebuah citra. Nilai *precision* sebuah model *object detection* dapat ditentukan menggunakan persamaan:

$$\text{Precision} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}} \quad (2.9)$$

Lalu, *recall* adalah satuan evaluasi yang mengukur kemampuan model untuk mencari semua objek yang relevan pada sebuah citra. Nilai *recall* dapat ditentukan menggunakan persamaan:

$$\text{Recall} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}} \quad (2.10)$$



Gambar 2.5 Contoh *Precision-Recall Curve* dengan *IoU Threshold* = 0.5

- *Mean Average Precision* (mAP)

Dari kurva *precision-recall*, kita dapat menentukan *average precision* (AP), yaitu nilai rata-rata presisi untuk setiap kelas pada pelatihan model sesuai dengan *threshold* yang ditentukan menggunakan nilai dari *interpolated precision* untuk setiap level *recall*. Nilai *interpolated precision* dapat dicari menggunakan persamaan:

$$p_{interpolated} = \max_{r' \geq r} [p(r')] \quad (2.11)$$

Dari nilai tersebut, kita dapat menemukan nilai *average precision* dengan mencari rata-rata dari *interpolated precision* terhadap *recall* untuk setiap kelas yang dapat dicari menggunakan persamaan:

$$AP = \frac{1}{n} \sum_{i=1}^{n-1} (r_{i+1} - r_i) p_{interpolated}(r_{i+1}) \quad (2.12)$$

Nilai *average precision* hanya merepresentasikan rata-rata presisi sebuah kelas pada model. Umumnya, sebuah model deteksi objek memiliki beberapa kelas objek sehingga perlu dicari rata-rata *average precision* seluruh kelas untuk mengetahui akurasi model secara keseluruhan yang umum disebut sebagai *mean Average Precision* (mAP). Nilai *mean Average Precision* (mAP) ditentukan dengan persamaan:

$$mAP = \frac{1}{k} \sum_{l=1}^k AP_l \quad (2.13)$$

Pada persamaan tersebut, k merepresentasikan jumlah kelas yang dilatih pada model dan AP_l adalah *average precision* dari masing-masing kelas. Nilai mAP umumnya ditentukan sesuai dengan *IoU threshold* yang digunakan, seperti mAP0.5 dan mAP0.5:0.95. Nilai mAP0.5 adalah rata-rata dari AP setiap kelas ketika *IoU threshold* bernilai 0.5. Kemudian, mAP0.5:0.95 merupakan salah satu parameter evaluasi pada COCO 2017 *challenge* di mana nilai mAP tersebut merupakan hasil

rata-rata dari AP setiap kelas pada 10 IoU *threshold* pada rentang 0.5 sampai 0.95 dengan *step size* sebesar 0.05. Kalkulasi nilai mAP 0.5:0.95 dilakukan dengan mengambil AP setiap kelas pada IoU *threshold* bernilai 0.5 dan akan dicari nilai *precision* tiap nilai *recall* pada rentang 0 – 1 dengan *step size* sebesar 0.01. Proses tersebut akan dilakukan seterusnya dengan peningkatan IoU *threshold* sebesar 0.05 seperti 0.55, 0.6, 0.65 hingga mencapai 0.95 yang kemudian hasil mAP0.5:0.95 akan ditentukan dari rata-rata semua kelas yang dilatih dan AP dari setiap IoU *threshold* [15].

- *F1 Score*

F1 Score adalah satuan evaluasi untuk mengetahui nilai *harmonic mean* dari *precision* dan *recall*. *F1 score* memiliki rentang nilai antara 0-1. Semakin besar nilai *F1 score*, model tersebut memiliki *precision* dan *recall* yang sempurna dan juga memiliki *accuracy* pendeteksian yang baik. *F1 score* dapat ditentukan dengan persamaan:

$$F1 - Score = 2 * \frac{precision * recall}{precision + recall} \quad (2.14)$$

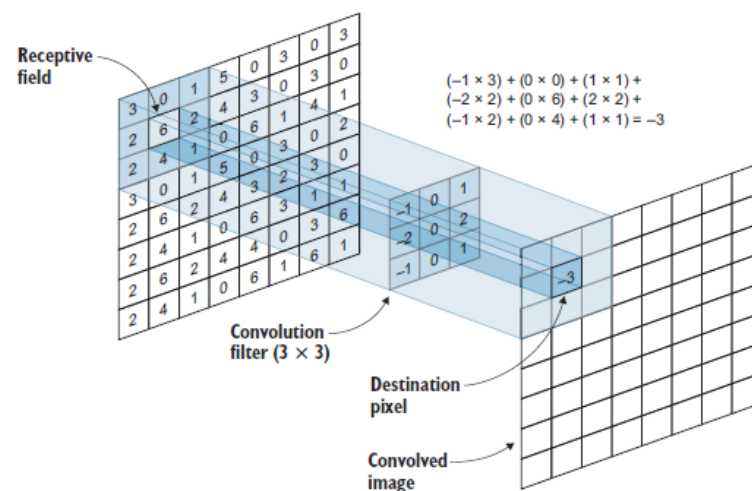
2.3. Convolutional Neural Networks

Convolutional neural networks merupakan sebuah *neural networks* yang menerapkan operasi *convolution* dalam membentuk sebuah arsitektur *neural networks*. *Convolution* merupakan sebuah operasi linear matematis dua buah fungsi untuk mencari fungsi baru. *Convolutional neural networks* umum digunakan untuk sebuah data yang bisa diterapkan sebuah *grid cell* didalamnya, seperti data *time series* yang didefinisikan dengan data dalam *grid* 1D atau data citra yang dapat didefinisikan sebagai data 2D dari sebuah piksel [12]. Secara umum, arsitektur *convolutional neural networks* terbagi menjadi 3 *layers* utama, yaitu:

A. Convolutional layers

Convolutional layers adalah *layers* yang berfungsi untuk mengekstraksi fitur-fitur pada sebuah citra untuk melakukan identifikasi objek yang ada pada citra tersebut. *Convolutional layers* menggunakan sebuah *window* yang disebut sebagai

convolutional filters untuk mengekstrak fitur pada sebuah citra di mana filter ini akan memeriksa semua piksel yang ada semua citra. Filter tersebut memiliki *weights* tersendiri untuk setiap fitur pada citra dan akan melakukan operasi *dot product* antara filter dengan satu fitur tertentu yang akan dijumlahkan jika semua operasi *dot product* sudah dilakukan. *Layers* ini akan menghasilkan sebuah citra baru yang disebut sebagai *convolved image* atau *feature map* yang fiturnya berasal dari operasi *dot product* antara filter dengan fitur citra masukan.



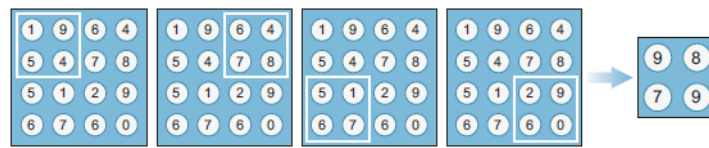
Gambar 2.6 Proses ekstraksi fitur pada *convolutional layers* [13]

B. Pooling layers

Pooling layers adalah *layers* yang berfungsi untuk mengurangi jumlah parameter agar proses komputasi yang dibutuhkan tidak terlalu kompleks. *Pooling layers* akan mengubah ukuran dari *feature map* menggunakan operasi statistika seperti *max* atau *average* untuk mengurangi jumlah parameter yang akan diteruskan menuju *layer* berikutnya di mana hanya parameter-parameter yang penting saja yang akan diteruskan menuju *layer* berikutnya. *Pooling layers* akan memodifikasi nilai *feature map* saja tanpa mengurangi jumlah *feature map* yang dihasilkan dari *convolutional layers*. *Pooling layers* terbagi menjadi dua jenis utama, yaitu:

- *Max Pooling*

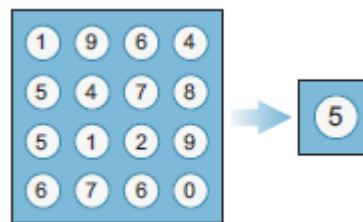
Max pooling adalah jenis *pooling layers* yang akan memeriksa *feature map* hasil *convolutional layers* dan mengambil nilai fitur yang tertinggi pada *window* sehingga terbentuk *feature map* baru yang hanya akan berisi nilai fitur hasil *pooling*.



Gambar 2.7 Contoh *max pooling feature map* berukuran 4 X 4 dengan *window* berukuran 2 x 2 [13]

- *Average Pooling*

Average Pooling adalah jenis *pooling layers* yang akan memeriksa *feature map* hasil *convolutional layers* dan mengambil rata-rata nilai fitur pada *window*. Terdapat beberapa jenis *average pooling*, contohnya adalah *global average pooling* yang akan merata-rata semua nilai fitur pada *feature map* sehingga hanya ada satu nilai yang diteruskan ke *layer* berikutnya.

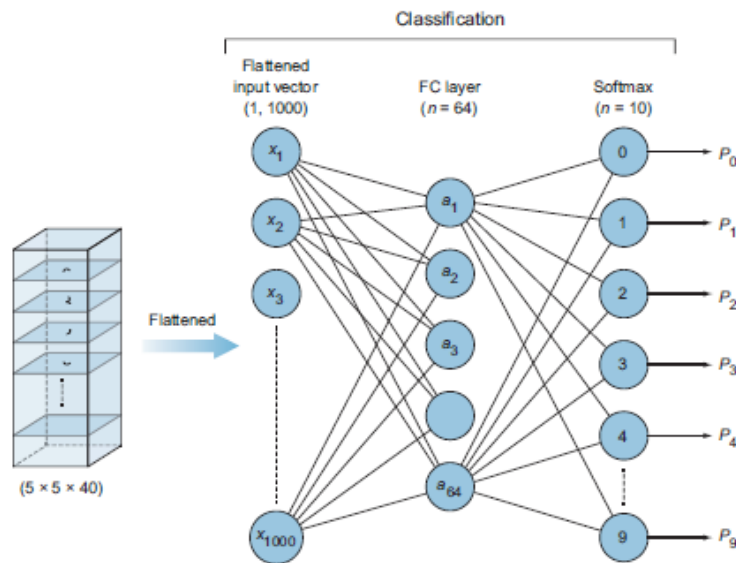


Gambar 2.8 Contoh *global average pooling feature map* berukuran 4 X 4 [13]

C. *Fully connected layers*

Fully connected layers adalah *layer* yang berfungsi untuk melakukan klasifikasi. *Layer* ini akan menerima hasil *feature extraction* yang dilakukan pada *convolutional* dan *pooling layers* dan akan memprosesnya sehingga hasil ekstraksi dapat digunakan untuk melakukan klasifikasi. Arsitektur *fully connected layers* terdiri dari tiga komponen utama, yaitu:

- *Input flattened vector*, hasil ekstraksi fitur akan dimasukkan ke *flatten layer* agar nilainya dapat diubah menjadi sebuah vektor berukuran (1, n) di mana n adalah nilai sesuai dengan dimensi hasil ekstraksi.
- *Hidden layer*, vektor dari hasil ekstraksi akan dihubungkan satu sama lain yang akan menghasilkan kelas klasifikasi membentuk neuron-neuron.
- *Output layer*, neuron yang terbentuk akan dihubungkan dengan *activation function* yang sesuai dan menghasilkan *node-node* kelas objek.

Gambar 2.9 Arsitektur *fully connected layers* [13]

2.4. YOLO

You Only Look Once (YOLO) adalah salah satu keluarga model *deep learning* yang umum digunakan untuk melakukan *object detection*. YOLO bukan model *object detection* dengan tingkat akurasi tertinggi dibandingkan model lain seperti R-CNN, tetapi YOLO merupakan salah satu model dengan waktu pendeteksian tercepat sehingga banyak diterapkan pada sistem *real-time* seperti kamera pengawas [11].

YOLO menggunakan pendekatan yang berbeda dalam melakukan pendeteksian di mana YOLO akan membagi citra menjadi beberapa bagian menggunakan sebuah *grid cell* berukuran $S \times S$. Tiap *grid cell* tersebut akan melakukan pendeteksian terhadap objek dengan membentuk beberapa *bounding boxes* dan menentukan nilai *confidence* dari *bounding boxes* tersebut. Nilai *confidence* akan menggambarkan kepastian model untuk mendeteksi objek dan keakuratan model dalam memprediksi objek pada *box* tersebut. Nilai *confidence* umumnya didefinisikan dengan persamaan:

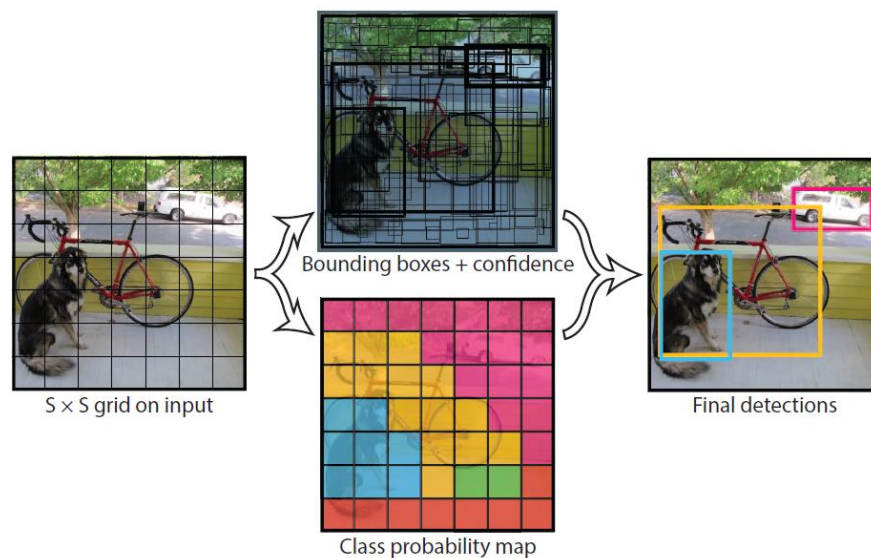
$$Conf = Pr(Object) * IoU_{pred}^{truth} \quad (2.15)$$

Bounding boxes yang terbentuk memiliki lima unsur utama, yaitu x , y , w , h , dan *confidence* di mana x dan y merepresentasikan titik tengah *bounding boxes* terhadap *grid cell*, w dan h merepresentasikan tinggi dan lebar hasil prediksi terhadap keseluruhan citra serta *confidence* merepresentasikan IoU *bounding boxes* hasil prediksi dengan *bounding boxes* acuan.

Grid cell yang memiliki objek didalamnya akan melakukan prediksi untuk menentukan kelas objek tersebut dengan hanya satu kemungkinan kelas yang akan coba diprediksi untuk tiap *grid cell* berapapun jumlah *bounding boxes* yang terbentuk. Prediksi tersebut dapat direpresentasikan dengan $\Pr(\text{Class}_i|\text{Object})$ yang akan dikalikan nilai *confidence* tiap *box* pada saat pengujian sehingga akan dihasilkan nilai *confidence* untuk tiap kelas objek dengan persamaan:

$$\text{Conf}(\text{Class}_i) = \Pr(\text{Class}_i) * \text{IoU}_{\text{pred}}^{\text{truth}} \quad (2.16)$$

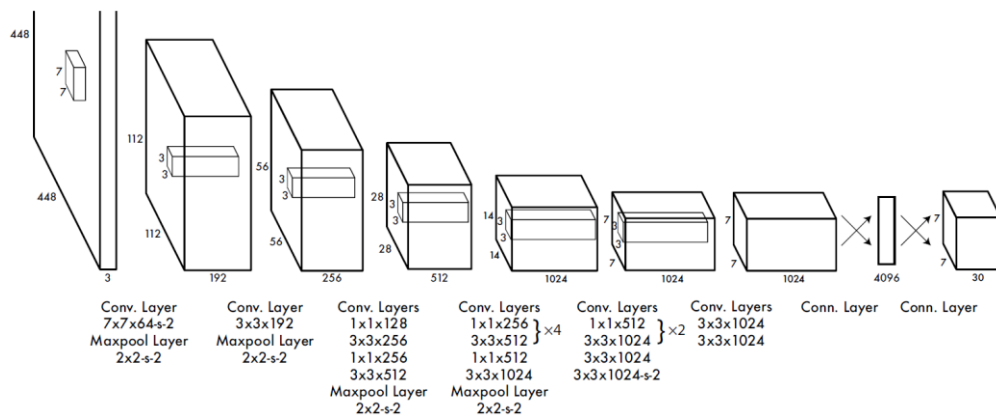
Nilai prediksi tersebut akan merepresentasikan probabilitas dari sebuah kelas objek tampil pada *box* dan bagaimana kualitas dari *box* hasil prediksi sesuai dengan objek yang ada.



Gambar 2.10 Ilustrasi pendeteksian menggunakan algoritma YOLO [12]

YOLO menggunakan *convolutional neural networks* sebagai arsitektur utama yang akan mengekstraksi fitur citra dan menampilkan hasil prediksi.

Convolutional neural networks yang digunakan mengambil referensi dari arsitektur model *GoogLeNet* untuk melakukan *image classification* dengan 24 *convolution layers* dan dua *fully connected layers*. YOLO juga mengganti salah satu modul yang digunakan *GoogLeNet* berupa modul *Inception* menjadi 1 X 1 *reduction layers* ditambah 3 X 3 *convolutional layers*. Arsitektur tersebut akan menghasilkan sebuah kumpulan prediksi berukuran 7 X 7 X 30 dalam bentuk sebuah *tensor*.



Gambar 2.11 Arsitektur awal model YOLO [12]

Walaupun YOLO merupakan salah satu algoritma dengan kecepatan pendeteksian yang tercepat, versi awal YOLO memiliki beberapa kelemahan. Pertama, satu *grid cell* hanya bisa melakukan pendeteksian untuk satu kelas objek saja sehingga jika terdapat lebih dari satu kelas objek, YOLO tidak bekerja dengan optimal. Kedua, jika *grid cell* mendeteksi kumpulan dari objek-objek kecil seperti sebuah burung, YOLO juga tidak bekerja dengan optimal dikarenakan *bounding boxes* yang terbentuk membutuhkan *spatial constraints* yang kuat sehingga terdapat limitasi dari objek yang dapat dideteksi pada *grid cell* yang sama [11].

2.4.1. YOLOv7

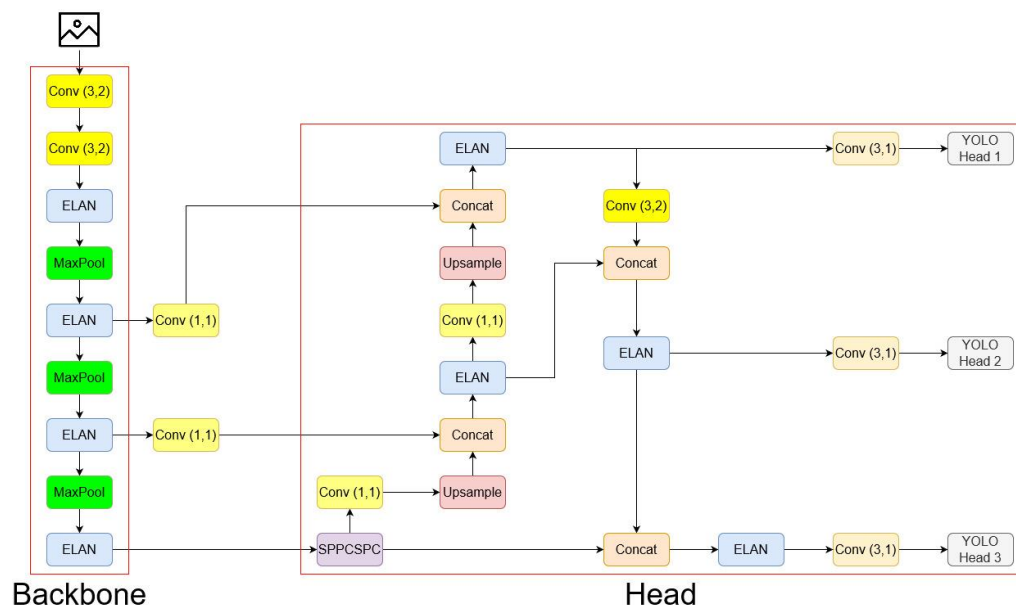
YOLOv7 merupakan salah satu pengembangan dari keluarga model YOLO yang dipublikasikan pada tahun 2022. Pengembangan YOLOv7 berfokus pada bagaimana mengoptimalkan proses *training* yang dilakukan sebuah model dengan memperkuat *training cost* untuk meningkatkan akurasi tanpa meningkatkan *inference cost* menggunakan *optimized modules* dan *optimized methods*. Pengujian yang dilakukan menggunakan YOLOv7 menunjukkan bahwa model YOLOv7

memiliki performa yang lebih baik dibandingkan dengan beberapa model lain seperti YOLOv5, YOLOR, YOLOX, PPYOLOE, dan beberapa model lain. Sebagai contoh, YOLOv7-Tiny-SiLU memiliki nilai *average precision* yang lebih baik dibandingkan dengan YOLOv5-N dengan peningkatan sebesar 10,7% yang diikuti dengan hasil *inference* yang lebih cepat 127 FPS [8].

Secara umum, arsitektur YOLOv7 terbagi menjadi dua bagian, yaitu bagian *backbone* dan *head*. Bagian *backbone* adalah bagian pada arsitektur yang akan melakukan ekstraksi fitur untuk mendapatkan objek yang akan dideteksi, sedangkan *head* adalah bagian arsitektur yang akan melakukan pendeteksian pada model deteksi objek yang dihubungkan oleh bagian *neck* [16]. YOLOv7 terbagi menjadi dua tipe utama, yaitu YOLOv7-P5 dan YOLOv7-P6. Perbedaan utama dari kedua tipe itu adalah perangkat yang umum digunakan dalam mengeksekusi model di mana YOLOv7-P5 lebih digunakan pada perangkat GPU *consumer* atau *edge*, sedangkan YOLOv7-P6 lebih digunakan pada GPU untuk *Cloud Server*. Oleh karena itu, penelitian ini akan lebih berfokus pada tipe YOLOv7-P5. YOLOv7-P5 memiliki beberapa varian, diantaranya:

A. YOLOv7-Tiny

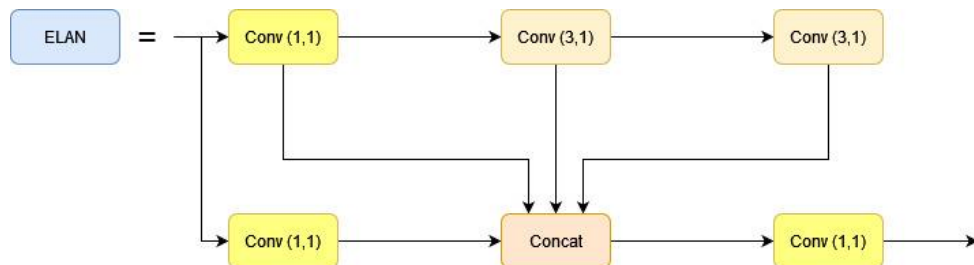
YOLOv7-Tiny merupakan varian model YOLOv7 dengan arsitektur paling sederhana. Karena memiliki arsitektur yang sederhana, YOLOv7-Tiny umum digunakan sebagai model kecerdasan buatan pada perangkat-perangkat *edge devices* seperti mikrokontroler atau mini-PC. Arsitektur YOLOv7-Tiny terdiri dari bagian *backbone* dan *head* di mana *backbone* YOLOv7-Tiny terdiri dari 28 *layer*, sedangkan *head* YOLOv7-Tiny terdiri dari 49 *layer*. Berikut merupakan visualisasi arsitektur YOLOv7-Tiny.



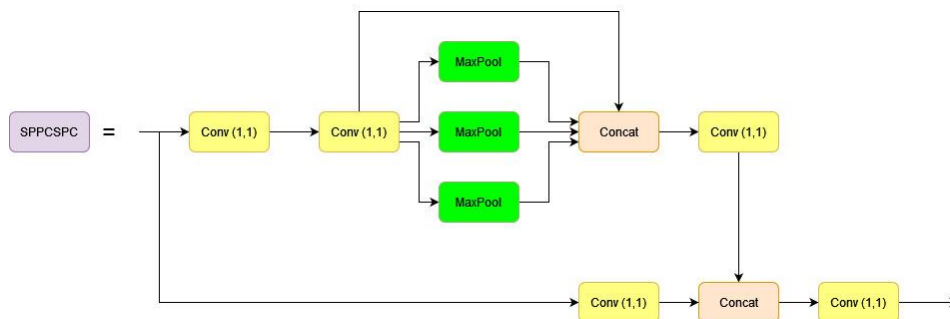
Gambar 2.12 Arsitektur model YOLOv7-Tiny (Digambar ulang berdasarkan [8])



Gambar 2.13 Layer modul Convolutional YOLOV7-Tiny (Digambar ulang berdasarkan [8])



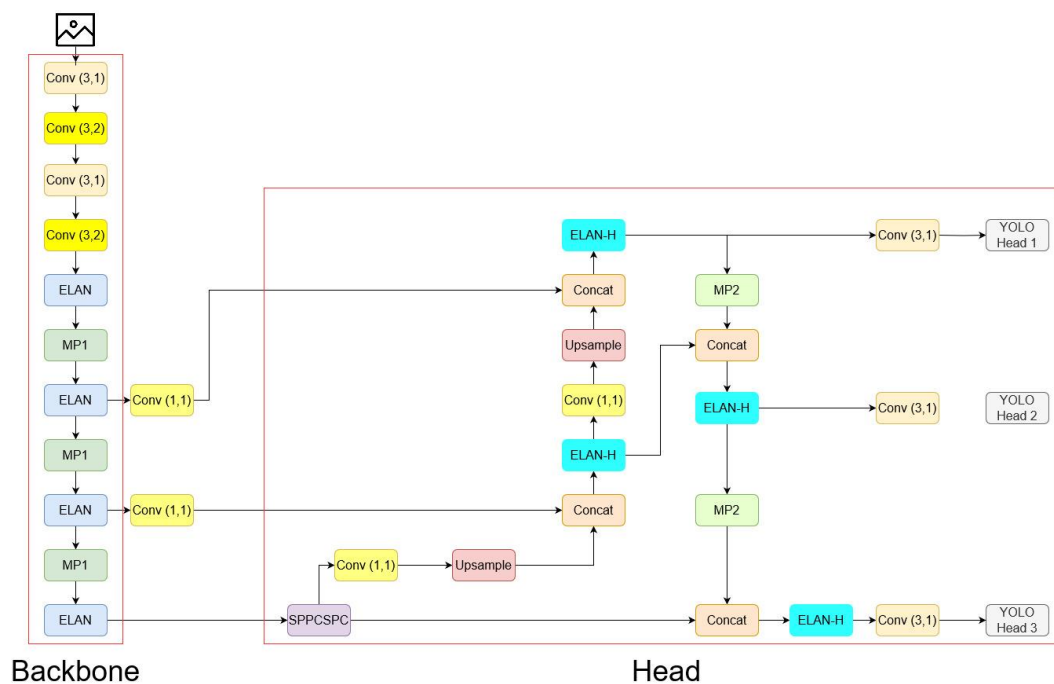
Gambar 2.14 Arsitektur ELAN model YOLOv7-Tiny (Digambar ulang berdasarkan [8])



Gambar 2.15 Layer modul SPPCSPC YOLOv7-Tiny (Digambar ulang berdasarkan [8])

B. YOLOv7

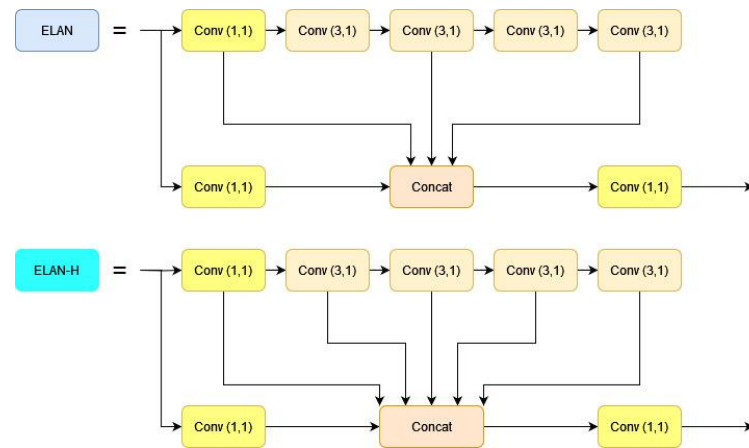
YOLOv7 merupakan standar dari seluruh varian model YOLOv7. YOLOv7 memiliki arsitektur yang lebih kompleks dibandingkan dengan YOLOv7-Tiny, tetapi bukan varian dengan arsitektur yang paling kompleks. Perbedaan antara arsitektur YOLOv7-Tiny dengan YOLOv7 adalah tidak adanya modul MP pada YOLOv7-Tiny, yaitu modul untuk menggabungkan hasil *pooling* dengan modul *convolutional*. Selain itu, YOLOv7 memiliki dua arsitektur ELAN pada bagian *backbone* dan *head*. Bagian *backbone* YOLOv7 terdiri dari 50 layer, sedangkan bagian *head* YOLOv7 terdiri dari 55 layer. Berikut merupakan visualisasi arsitektur YOLOv7.



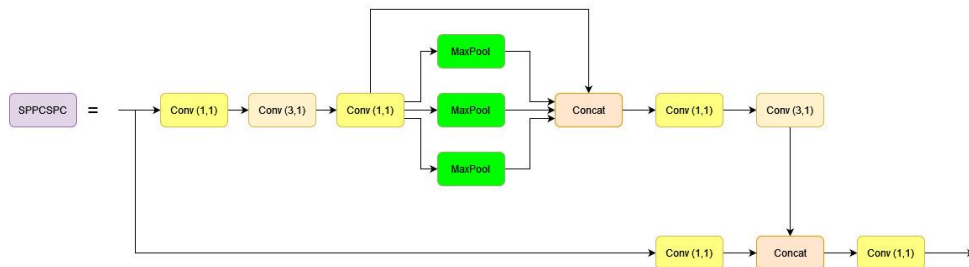
Gambar 2.16 Arsitektur model YOLOv7 (Digambar ulang berdasarkan [8])



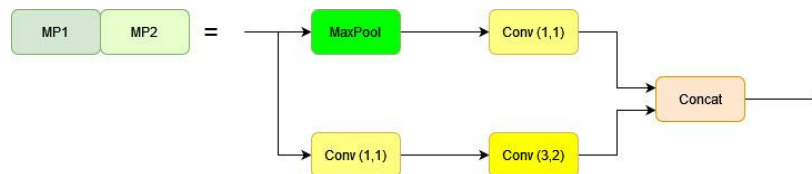
Gambar 2.17 Layer modul Convolutional YOLOv7 (Digambar ulang berdasarkan [8])



Gambar 2.18 Arsitektur ELAN pada bagian *backbone* (Atas) dan *head* (Bawah) model YOLOv7 (Digambar ulang berdasarkan [8])



Gambar 2.19 Layer modul SPPCSPC YOLOv7 (Digambar ulang berdasarkan [8])

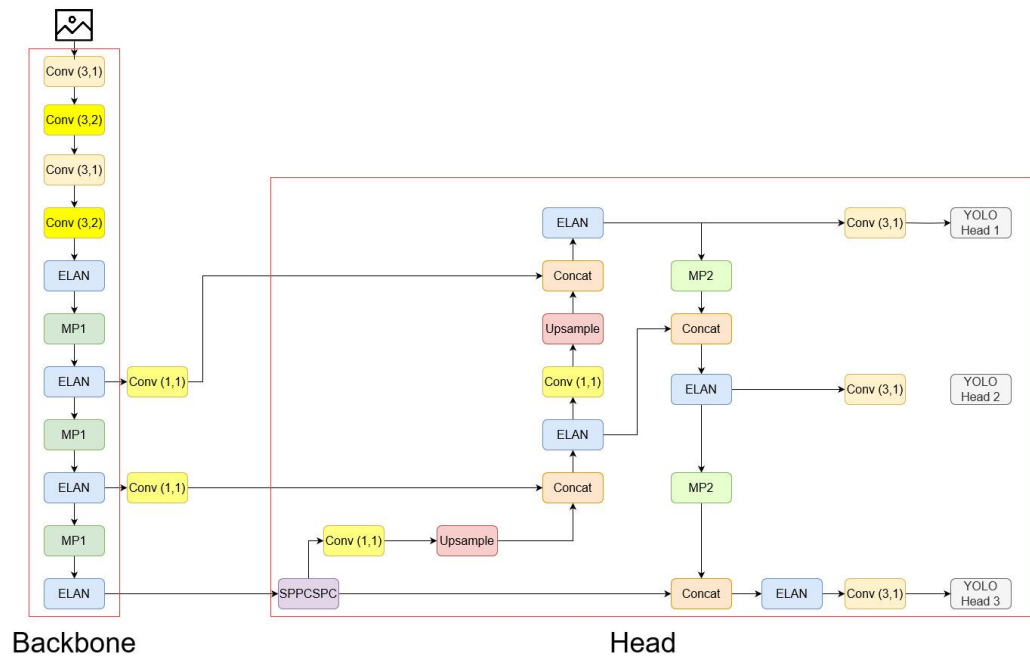


Gambar 2.20 Layer modul MP pada YOLOv7 (Digambar ulang berdasarkan [8])

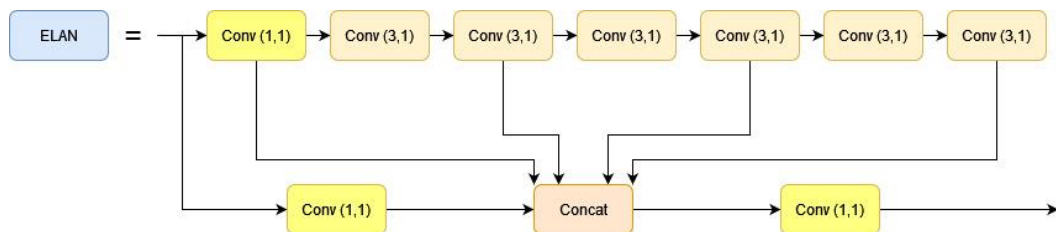
C. YOLOv7-X

YOLOv7-X merupakan varian model YOLOv7 yang paling kompleks pada tipe YOLOv7-P5. YOLOv7-X terbentuk dengan melakukan modifikasi varian YOLOv7 pada bagian *neck* dan menambahkan metode *compound scaling* pada model. Secara umum, arsitektur YOLOv7 dan YOLOv7-X tidak memiliki perbedaan yang signifikan. Perbedaan kedua model tersebut hanya pada arsitektur ELAN yang digunakan di mana YOLOv7-X memiliki lebih banyak layer pada arsitektur ELAN-nya dan tidak terbagi menjadi ELAN untuk bagian *backbone* dan

head. Modul lain seperti SPPCSPC dan MP tidak memiliki perbedaan dengan model YOLOv7. Berikut merupakan visualisasi arsitektur YOLOv7-X.



Gambar 2.21 Arsitektur model YOLOv7-X (Digambar ulang berdasarkan [8])



Gambar 2.22 Arsitektur ELAN model YOLOv7-X (Digambar ulang berdasarkan [8])

2.5. Fishnet

Fishnet merupakan *dataset* yang terdiri dari berkas citra kumpulan ikan dari kamera pengawasan pada kapal yang sudah dianotasi dengan *bounding boxes* untuk tiap objeknya. *Dataset* ini terdiri dari 34 kelas objek yang mencakup spesies-spesies ikan serta manusia yang didapatkan dari beberapa pihak perikanan dan kelautan. Versi terbaru *dataset* fishnet memiliki 143.818 citra dengan 549.209 *bounding boxes* di mana terdapat 4 *bounding boxes* di setiap citra pada *dataset*. Berkas *dataset* Fishnet sendiri memiliki ukuran total sebesar 30 GB [17].

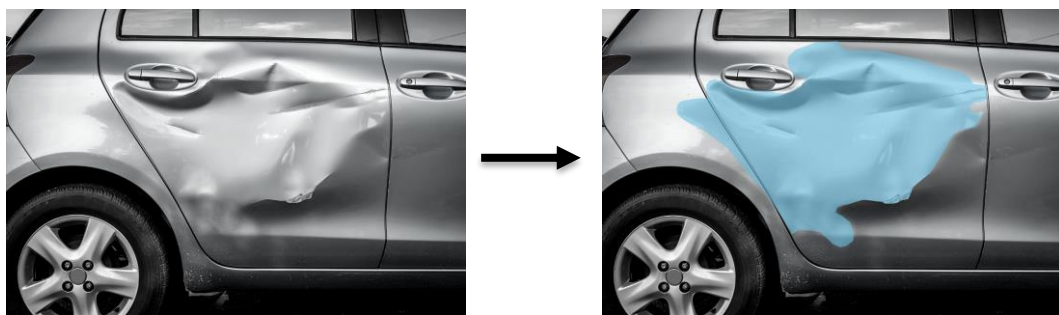
Dataset Fishnet terbagi menjadi tiga bagian, yaitu *training*, *validation*, *testing*. Ketiga bagian tersebut disesuaikan untuk proses pelatihan dan pengujian model *deep learning* yang ingin dibentuk. Perbandingan jumlah citra untuk *training* dan *validation-testing* adalah 7:3 di mana citra pada bagian *validation* dan *testing* dibagi kembali dengan perbandingan 7:3.

Tabel 2.1 Persebaran bagian *dataset* Fishnet

Nama Bagian	Jumlah Citra
<i>Training</i>	100.673
<i>validation</i>	30.202
<i>Testing</i>	12.943
Total	143.818

2.6. *Polygon annotation*

Polygon annotation adalah cara dalam pemberian label pada sebuah data untuk pembentukan model *machine learning* dengan menentukan koordinat x dan y dari tiap titik tepi objek yang ingin dideteksi [18]. Dengan menentukan koordinat x dan y dari tiap titik tepi objek, bentuk label yang dihasilkan akan memiliki tingkat presisi yang sama objek yang sesungguhnya. Karena hal tersebut, metode *polygon annotation* umum digunakan sebagai metode pemberian anotasi pada *dataset* model *object detection* atau *object recognition* di kehidupan nyata seperti pada CT scan, memperhatikan pertumbuhan sebuah tanaman, perhitungan biaya reparasi kerusakan kendaraan, dan lain-lain.



Gambar 2.23 Contoh data *Polygon Annotation* pada kerusakan sebuah mobil [18]

Polygon annotation memiliki beberapa keunggulan seperti fleksibel ketika diterapkan pada sebuah bentuk yang tidak biasa dan membuat piksel yang bukan bagian dari objek tidak diikutkan pada label sehingga hasil pendeteksian bisa lebih

baik. Namun, waktu yang dibutuhkan untuk melakukan anotasi lebih lama sesuai dengan kompleksitas dari objek yang dianotasi dan tidak semua *annotator* memiliki fitur untuk melakukan *polygon annotation* pada objek yang memiliki lubang seperti ban, donat dan lain-lain.

2.7. Penelitian Terkait

Belum ada penelitian yang membahas terkait model YOLOv7 dalam melakukan pendeteksian jenis ikan, tetapi terdapat beberapa jurnal yang menggunakan model YOLO untuk melakukan pendeteksian jenis ikan. Salah satu penelitiannya adalah dengan model YOLOv4-Tiny melakukan uji coba untuk mengembangkan sistem pendeteksian jenis ikan pada sebuah akuarium menggunakan Raspberry Pi 4 pada tahun 2022. Penelitian tersebut dilakukan untuk melakukan pendeteksian terhadap 3 jenis ikan, yaitu *dwarf gourami*, *guppy* dan *zebrafish* menggunakan Raspberry Pi 4 yang terhubung pada sebuah kamera dengan model YOLOv4-Tiny [9]. *Dataset* yang digunakan untuk pelatihan model didapatkan dari hasil penggabungan data dari beberapa sumber seperti *Kaggle* dan *Fishbase* serta pengambilan langsung dari akuarium dengan jumlah data sekitar 3300 data. Berikut adalah hasil penelitian yang didapatkan.

Tabel 2.2 Hasil *average precision* untuk setiap kelas yang dilatih[9].

No	Spesies	Average Precision (%)
1	<i>Dwarf Gourami</i>	97.97
2	<i>Guppy</i>	97.91
3	<i>Zebrafish</i>	97.54
Mean Average Precision (%)		97.81

Tabel 2.3 *Confusion matrix* dari hasil pengujian [9]

Total = 48		Predicted Species			
		<i>Dwarf Gourami</i>	<i>Guppy</i>	<i>Zebrafish</i>	<i>Unknown (No Detection)</i>
Actual Species	<i>Dwarf Gourami</i>	12	0	0	0
	<i>Guppy</i>	0	11	0	1
	<i>Zebrafish</i>	0	0	11	1
	<i>Unknown</i>	0	0	2	10

Tabel 2.4 Akurasi pendeteksian dari sampel secara langsung [9]

No	Spesies	Accuracy (%)
1	<i>Dwarf Gourami</i>	100
2	<i>Guppy</i>	97.92
3	<i>Zebrafish</i>	93.75
4	<i>Unknown</i>	91.67
Global Accuracy (%)		91.67

Terdapat beberapa saran yang diberikan untuk pengembangan lebih lanjut penelitian ini, diantaranya adalah menggunakan *dataset* yang lebih besar untuk meningkatkan akurasi model dan mengurangi terlihatnya *false positive*. Kelas objek yang dideteksi juga dapat ditambah agar model bisa lebih tergeneralisir. Penggunaan algoritma yang lebih baru juga dapat meningkatkan performa dan akurasi dari model yang terbentuk.

BAB 3

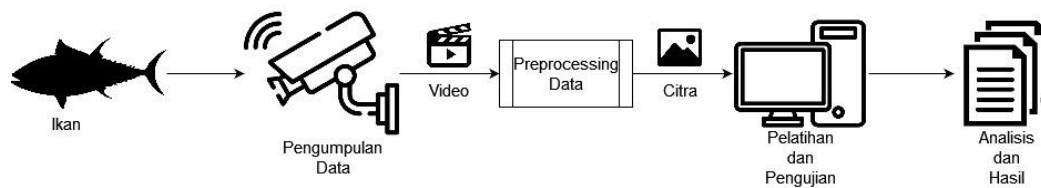
PERANCANGAN SISTEM DETEKSI MULTI OBJEK BERBASIS YOLOV7

3.1. *System Requirements*

Secara umum, sistem pendeteksian jenis ikan memiliki beberapa persyaratan baik dari segi perangkat keras maupun perangkat lunak. Dari segi perangkat keras, terdapat dua komponen yang dibutuhkan untuk membuat sistem, yaitu:

1. Komputer yang bertujuan untuk memuat model pendeteksian jenis ikan.
2. Kamera yang berfungsi sebagai alat pendeteksi sekaligus pengambilan data untuk pelatihan dan pengujian model.

Rancangan sistem diilustrasikan secara visual pada Gambar 3.1.



Gambar 3.1 Rancangan sistem pendeteksian jenis ikan

Dari segi perangkat lunak, terdapat beberapa *library* yang dibutuhkan sebagai bagian utama eksekusi model, diantaranya:

1. *OpenCV* versi 4.1.1
2. *PyTorch* versi 1.7.0
3. *Torchvision* versi 0.8.1
4. *Numpy* versi 1.18.5
5. *Matplotlib* versi 3.2.2
6. *Pillow* versi 7.1.2
7. *PyYAML* versi 5.3.1
8. *Requests* versi 2.23.0
9. *Scipy* versi 1.4.1

10. *Tqdm* versi 4.64.0

11. *Protobuf* versi 3.20.1

Model kecerdasan buatan yang dilatih dan diuji akan dijalankan menggunakan sebuah komputer. Spesifikasi dari komputer yang digunakan untuk membentuk model tercantum pada Tabel 3.1.

Tabel 3.1 Spesifikasi komputer

<i>Central Processor Unit</i>	Intel i5-13600K @ 3.50 GHz (20 CPUs)
<i>Graphics Processing Unit</i>	NVIDIA GeForce RTX 3070
<i>Memory</i>	32 GB DDR5
<i>Operating System</i>	EndeavourOS Arch Linux
<i>Python Version</i>	3.8.16

3.2. Pengumpulan Data

Pengumpulan data yang akan digunakan dalam proses pelatihan dan pengujian varian model YOLOv7 diambil dari hasil tangkapan kamera yang terpasang pada kapal. Hasil tangkapan akan berbentuk sebuah berkas video yang tersimpan pada memori penyimpanan kamera yang terpasang pada kapal.

Jika data tiap jenis ikan dari berkas video masih belum mencukupi, kekurangan data akan diambil dari *dataset* Fishnet yang bertujuan untuk membuat rasio data antar kelas pada *dataset* seimbang. Data yang diambil berupa sebuah citra yang akan disesuaikan dengan jenis ikan yang belum memiliki data yang cukup di mana data yang terdapat pada *dataset* Fishnet sudah dilabeli sesuai dengan kelas objeknya sehingga data yang diambil hanya data yang sesuai dengan kelas yang ingin dideteksi oleh model. Contoh citra yang terdapat pada *dataset* Fishnet tertera pada Gambar 3.2.

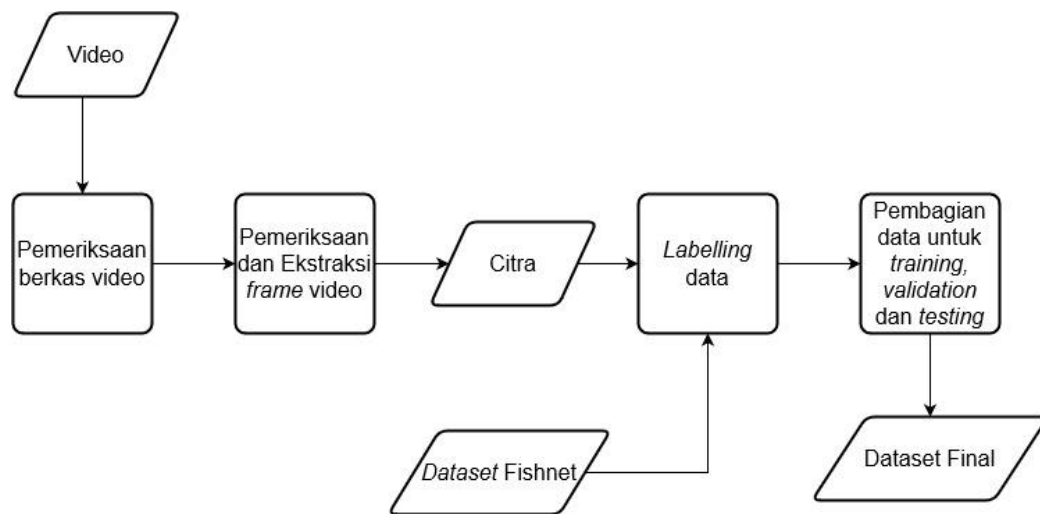
Nantinya, data tersebut akan digabungkan dengan data yang berasal dari berkas video hasil rekaman. Setelah digabungkan, data akan diteruskan menuju tahap *preprocessing*. Data yang berbentuk video akan diekstrak terlebih dahulu tiap *frame*-nya untuk dilihat objek yang terdapat pada *frame* tersebut dan akan dilabeli dengan *polygon annotation*, sedangkan data dari *dataset* Fishnet akan dilabeli ulang menggunakan *polygon annotation*.



Gambar 3.2 Contoh citra dari *dataset* Fishnet [17]

3.3. *Preprocessing Data*

Data yang telah dikumpulkan perlu disesuaikan terlebih dahulu agar dapat digunakan sebagai data masukan dalam proses pelatihan maupun pengujian melalui tahap *preprocessing*. Terdapat beberapa proses yang dilakukan pada tahap *preprocessing*. Jika data berbentuk sebuah berkas video, berkas tersebut akan diperiksa apakah mengalami *corrupt* atau tidak serta apakah pada berkas tersebut terdapat objek yang ingin dideteksi atau tidak. Jika melewati tahapan ini, berkas video tersebut akan diekstrak tiap *frame*-nya untuk mencari *frame* yang terdiri dari objek yang ingin dideteksi dan menghilangkan yang tidak terdiri dari objek yang ingin dideteksi. Kemudian, *frame* tersebut akan digabungkan dengan data dari *dataset* Fishnet untuk nantinya diberikan label menggunakan *polygon annotation*. Jenis anotasi tersebut digunakan agar *dataset* dapat dilatih menggunakan metode *semantic segmentation* sebagai salah satu skenario pengujian model yang akan dilakukan. Setelah selesai dilabeli, data tersebut akan dibagi menjadi 3 bagian, yaitu bagian *training*, *validation* dan *testing*. Tahap *preprocessing* diilustrasikan secara visual pada Gambar 3.3.



Gambar 3.3 Tahapan *pre-processing* data

Tahap awal pemeriksaan berkas video dilakukan secara manual, sedangkan proses ekstraksi *frame* video dapat dilakukan menggunakan sebuah program. Secara umum, berkas video yang akan diekstrak memiliki jumlah FPS tertinggi di angka 24 FPS sehingga sebagai contoh, jika video tersebut memiliki durasi 5 menit, terdapat 7200 *frame* yang harus diekstrak. Oleh karena itu, dapat digunakan sebuah program untuk mempersingkat waktu ekstraksi yang dilakukan. Cuplikan kode yang dapat digunakan untuk mengekstraksi *frame* pada sebuah video tertera pada Gambar 3.4.

Setelah seluruh *frame* terekstraksi, data akan berubah menjadi sebuah citra dan perlu dilabeli terlebih dahulu agar dapat dilatih untuk membentuk sebuah model *machine learning*. Proses pelabelan data dilakukan menggunakan Label Studio. Label Studio merupakan salah satu *tools* bersifat *open-source* yang dapat digunakan untuk melakukan anotasi data. Label Studio dapat digunakan untuk melakukan anotasi data baik dengan label berbentuk *bounding box* maupun berbentuk sebuah *masks*. Pemasangan Label Studio sendiri dapat dilakukan menggunakan *localhost*. Untuk memasang Label Studio, diperlukan aplikasi *Anaconda* untuk membentuk *virtual environment* pada komputer. Setelah memasang *anaconda*, kita perlu membentuk *virtual environment* untuk Label Studio dan mengaktifkan *environment* tersebut. Perintah yang dapat digunakan untuk membuat *virtual environment* pada Label Studio tertera pada Gambar 3.5.

```

# load the video clip
video_clip = VideoFileClip(video_file)
# make a folder by the name of the video file
filename, _ = os.path.splitext(video_file)
filename += "-moviepy"
if not os.path.isdir(filename):
    os.mkdir(filename)
# if the SAVING_FRAMES_PER_SECOND is above video FPS, then set it to FPS (as max)
saving_frames_per_second = min(video_clip.fps, SAVING_FRAMES_PER_SECOND)
# if SAVING_FRAMES_PER_SECOND is set to 0, step is 1/fps, else
1/SAVING_FRAMES_PER_SECOND
step = 1 / video_clip.fps if saving_frames_per_second == 0 else 1 /
saving_frames_per_second
# iterate over each possible frame
for current_duration in np.arange(0, video_clip.duration, step):
# format the file name and save it
    frame_duration_formatted = format_timedelta(timedelta(seconds=current_duration))
    # frame_filename = os.path.join(filename, f"frame{frame_duration_formatted}.jpg")
    frame_filename = os.path.join(filename,
f"{os.path.splitext(video_file)}{frame_duration_formatted}.jpg")
    # save the frame with the current duration
    video_clip.save_frame(frame_filename, current_duration)

```

Gambar 3.4 Kode untuk mengekstrak *frame* pada berkas video

```

conda create --name/-n <environment-name>
conda activate <environment-name>

```

Gambar 3.5 Perintah untuk membuat dan mengaktifkan *virtual environment* pada *anaconda* [19]

Perintah pertama yang bertujuan untuk membentuk *virtual environment* juga dapat menentukan versi *python* yang digunakan. Setelah membentuk *virtual environment*, kita dapat melakukan instalasi Label Studio dan menjalankan Label Studio untuk melakukan anotasi. Perintah yang dapat digunakan untuk memasang Label Studio dan menjalankannya tertera pada Gambar 3.6.

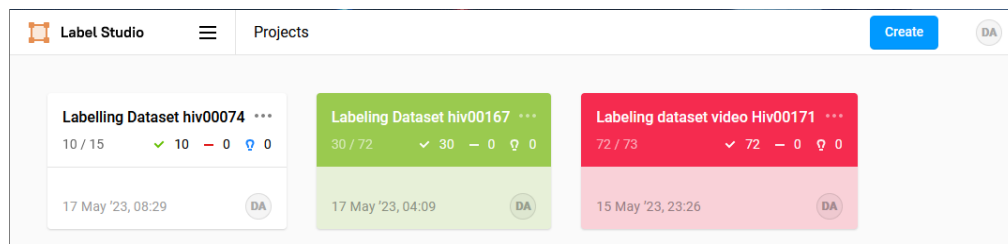
```

pip -u install label-studio
label-studio

```

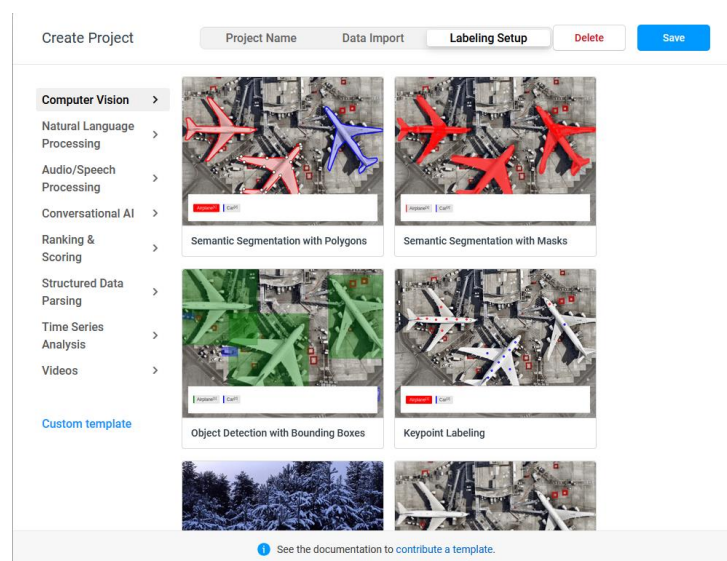
Gambar 3.6 Perintah untuk memasang dan mengaktifkan Label Studio [19]

Setelah diaktifkan, diperlukan sebuah akun untuk bisa melakukan anotasi data pada Label Studio. Setelah membuat akun dan *login*, Label Studio akan diarahkan menuju halaman utama seperti yang tertera pada Gambar 3.7.



Gambar 3.7 Halaman utama Label Studio

Untuk melakukan anotasi data, diperlukan sebuah *projects* untuk bisa menyimpan data-data yang akan dianotasi. Pembuatan *projects* dapat dilakukan dengan menekan tombol *create* pada halaman utama. Kemudian, kita perlu memberikan nama *project*, memasukkan data yang akan dianotasi dan menentukan kelas dari objek yang akan dianotasi serta menentukan metode anotasi yang akan diterapkan pada data seperti yang tertera pada Gambar 3.8.



Gambar 3.8 Tampilan UI untuk metode anotasi yang dapat digunakan

Setelah selesai membuat *projects*, anotasi data sudah dapat dilakukan. Karena data akan dianotasi agar dapat dilatih menggunakan *semantic segmentation*, maka metode anotasi yang akan dipilih adalah *semantic segmentation with polygons* untuk membuat anotasi data berbentuk sebuah *polygon*. *Polygon annotation* merupakan metode pemberian anotasi pada sebuah data dengan mencari titik tepi dari objek yang akan dianotasi. Hal ini bertujuan untuk menghasilkan label

dengan bentuk yang sama dengan objek. Proses pemberian anotasi pada data menggunakan *polygon annotation* dengan Label Studio dapat dilihat pada Gambar 3.9 – 3.11.



Gambar 3.9 Contoh citra yang belum diberikan label



Gambar 3.10 Pemberian *Polygon Annotation* pada citra



Gambar 3.11 Hasil akhir label *Polygon Annotation* pada citra

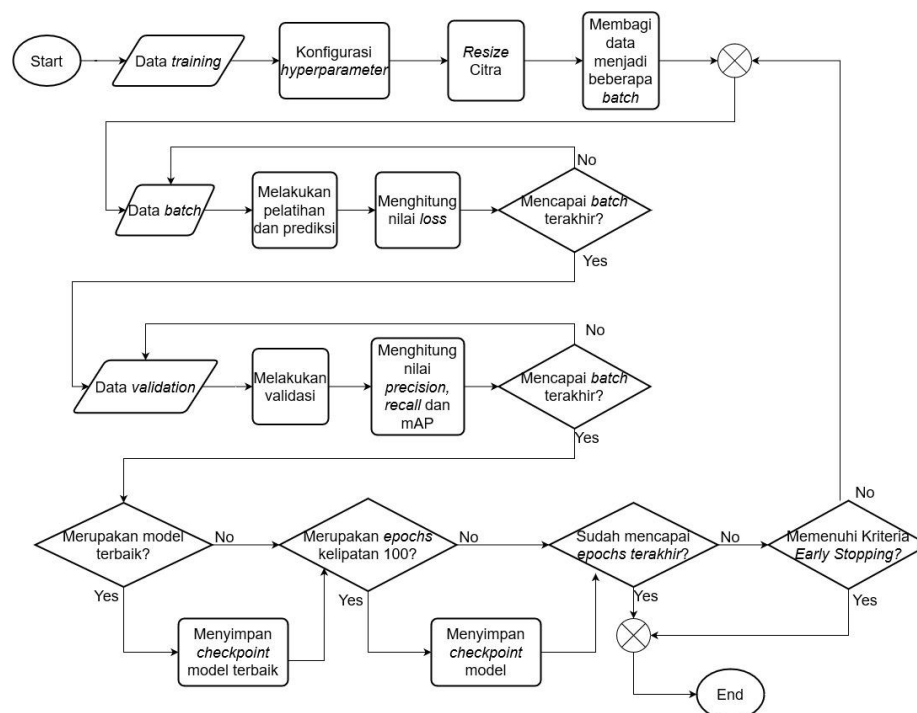
Pada Gambar 3.9, terlihat bahwa pada *frame* tersebut terdapat dua objek yang ingin dideteksi oleh model, yaitu objek pada kelas ‘human’ dan ‘tuna’. Kemudian, kita dapat melakukan anotasi data dengan memilih kelas dari objek dan memberikan titik tepi dari objek seperti pada Gambar 3.10. Pemberian titik tepi perlu dilakukan secara perlahan agar hasil yang didapatkan memiliki bentuk yang mirip dengan objek yang ingin dideteksi tanpa memasukkan piksel yang tidak termasuk sebagai objek. Hasil akhir dari anotasi data dapat dilihat pada Gambar 3.11 di mana label tersebut memiliki bentuk yang mirip dengan orang yang sedang berada pada kapal.

Setelah semua citra telah dilabeli, citra pada *projects* tersebut akan diekspor untuk mendapatkan label hasil dari anotasi yang dilakukan pada citra sesuai dengan format *dataset* yang dibutuhkan. YOLO sendiri memiliki format *dataset* berupa berkas dengan ekstensi ‘.txt’ yang memiliki format label berupa <kelas-objek> <x> <y> <width> <height> untuk setiap label pada citra. <x> dan <y> merupakan koordinat titik tengah dari label, sedangkan <width> dan <height> merupakan ukuran dari label. Namun, karena menggunakan metode *polygon annotation*, format label akan menjadi <kelas-objek> <x0> <y0> <x1> <y1> <xn> <yn> yang merepresentasikan koordinat titik tepi pada label di mana n akan menyesuaikan dengan jumlah titik tepi yang diberikan pada objek. Nilai yang ada pada label tersebut berada pada rentang 0 sampai 1 dikarenakan nilai didalamnya sudah dinormalisasi. Data-data tersebut akan dibagi menjadi 3 bagian, yaitu

training yang akan digunakan sebagai data untuk melatih model, *validation* yang akan digunakan sebagai data untuk mendapatkan performa model dari data *training* serta *testing* yang akan digunakan sebagai data untuk mendapatkan performa model pada data yang belum pernah dilihat pada model.

3.4. Rancangan Proses Pelatihan Model

Setelah data dikumpulkan dan melalui tahap *preprocessing* untuk disesuaikan dengan kebutuhan, data tersebut akan digunakan pada proses pelatihan model. Data tersebut akan digunakan untuk membentuk model yang dapat melakukan pendeteksian pada objek yang akan dideteksi. Secara umum, pelatihan model dilakukan dengan membagi data menjadi beberapa *batch* yang kemudian akan diekstrak fitur nya dan dihitung nilai *loss* dari model tersebut yang merepresentasikan perbedaan antara hasil prediksi dengan *ground truth*. Kemudian, model tersebut akan divalidasi dengan *dataset* bagian *validation* untuk diketahui performa pendeteksiannya seperti akurasi pendeteksian dan kecepatan pendeteksian. Rancangan proses pelatihan model sistem pendeteksian jenis ikan dapat digambarkan secara visual seperti pada Gambar 3.12.



Gambar 3.12 Skema pelatihan model YOLOv7

Skema pelatihan model sistem pendeteksian jenis ikan dilakukan menggunakan data pada *dataset* bagian *training*. Sebelum melakukan pelatihan model, terdapat beberapa parameter yang dapat dikonfigurasi untuk mengoptimalkan model yang disebut sebagai *hyperparameter*. Beberapa *hyperparameter* yang umum disesuaikan seperti *learning rate*, *momentum* dan parameter augmentasi seperti *translate*, *degrees*, *shear*, dan lain-lain. Setelah konfigurasi, citra yang digunakan akan di-*resize* menyesuaikan dengan parameter ukuran citra yang digunakan untuk pelatihan model. Setelah di-*resize*, data akan dibagi menjadi beberapa *batch* yang akan disesuaikan dengan varian model yang akan dilatih untuk menyesuaikan kapabilitas dari komputer pelatihan. Hal tersebut dikarenakan ukuran *batch* yang lebih besar dapat mempercepat proses pelatihan, tetapi membutuhkan memori GPU yang lebih besar untuk memproses *batch* tersebut secara bersamaan, begitu pula sebaliknya.

Kemudian, *batch* data yang telah terbagi akan dilatih menggunakan varian model YOLOv7. Secara umum, model YOLOv7 memiliki bagian *backbone* dan *head* yang sama di mana *backbone* model YOLOv7 menggunakan ELAN (*efficient layer aggregation networks*) untuk mengekstraksi fitur dari citra. *Backbone* tersebut dihubungkan dengan *cross stage partial networks* sebagai *neck* untuk dihubungkan dengan bagian *head* yang akan menentukan hasil deteksi. Arsitektur tiap varian model dapat dilihat pada Gambar 2.12, Gambar 2.16 dan Gambar 2.21. Kemudian, akan dihitung nilai *loss* dari model pada *batch* data tersebut untuk melihat perbandingan hasil pendeteksian dengan *ground truth*. Selama belum mencapai *batch* terakhir dari data bagian *training*, proses tersebut akan terus dilakukan hingga *batch* terakhir data *training*.

Setelah menyelesaikan pelatihan model, model yang terbentuk akan divalidasi untuk diuji dengan data dari *dataset* bagian *validation*. Data tersebut akan digunakan untuk mengalkulasi performa model dalam melakukan pendeteksian seperti nilai *precision*, *recall* dan mAP. Data yang divalidasi juga dibagi menjadi beberapa *batch* sesuai dengan ukuran *batch* yang digunakan ketika pelatihan model. Proses validasi akan terus dilakukan hingga *batch* data *validation* terakhir. Setelah mendapatkan nilai performa model, *weights* dari model akan diperbaharui dan akan

dibandingkan. Jika *weights* tersebut merupakan yang terbaik, *weights* tersebut akan disimpan sebagai model terbaik. Model yang dilatih pada iterasi kelipatan 100 juga akan disimpan. Proses pelatihan model akan berhenti ketika memenuhi salah satu dari dua kondisi, yaitu ketika pelatihan model sudah mencapai *epochs* terakhir atau pelatihan model sudah memenuhi kriteria *early stopping* di mana *early stopping* dapat dipicu ketika tidak ada perkembangan dari *weights* model sebanyak jumlah *patience* yang ditetapkan.

Pelatihan model sendiri akan dilakukan pada dua bentuk label yang berbeda, yaitu bentuk *polygon annotation* dan *bounding box*. Hal tersebut dilakukan untuk membandingkan performa pendeteksian ketika dilatih menggunakan masing-masing bentuk label. Karena data sudah dianotasi menggunakan *polygon annotation*, label data tersebut perlu dikonversi agar bisa menjadi label berbentuk *bounding box*. Data berbentuk *bounding box* terdiri dari 4 komponen, yaitu titik tengah objek pada sumbu x dan y, tinggi objek dan lebar objek. Oleh karena itu, label dari tiap citra akan diperiksa untuk mencari nilai maksimum serta minimum objek pada sumbu x dan y. Cuplikan kode yang dapat digunakan untuk mengonversi data *polygon annotation* tertera pada Gambar 3.13.

```
new_labels = []
for label in labels:
    label = label.strip().split()
    object_class = label[0]
    vertices = label[1:]

    x_values = [float(vertices[i]) for i in range(0, len(vertices), 2)]
    y_values = [float(vertices[i+1]) for i in range(0, len(vertices), 2)]

    # Calculate the minimum bounding box that encompasses the polygon vertices
    xmin = min(x_values)
    xmax = max(x_values)
    ymin = min(y_values)
    ymax = max(y_values)

    x, y, w, h = (xmin + xmax) / 2.0, (ymin + ymax) / 2.0, xmax - xmin, ymax - ymin

    new_labels.append(f"{object_class} {x} {y} {w} {h}")
    save_dir = os.path.join(data_dir, "result")
    with open(os.path.join(save_dir, img_file_id + ".txt"), 'w') as f:
        f.write('\n'.join(new_labels))
```

Gambar 3.13 Kode untuk mengonversi *dataset* dari *Polygon Annotation* menjadi *Bounding Box*

Konversi dilakukan dengan membaca isi dari label yang terdiri dari rincian dari label yang terdapat pada citra seperti kelas objek dan koordinat sumbu x dan y titik tepi anotasi. Data koordinat akan diproses untuk dicari nilai maksimum dan minimumnya yang akan digunakan untuk membentuk data *bounding box*. Kemudian, data *bounding box* akan disimpan pada berkas baru sehingga tidak akan menghilangkan data label yang sebelumnya.

3.5. Rancangan Proses Pengujian Model

Setelah melakukan pelatihan model, model yang terbentuk akan diteruskan ke tahap pengujian model dengan melakukan prediksi data pada *dataset* bagian *testing* yang tidak termasuk dalam data pada bagian *training* dan *validation*. Pengujian sendiri dilakukan untuk menentukan apakah parameter yang digunakan pada pelatihan model merupakan parameter yang bisa menghasilkan performa terbaik. Pengujian model sendiri dilakukan pada beberapa parameter yang mungkin dapat mempengaruhi performa dari model yang terbentuk.

Secara umum, skema pengujian model dilakukan melalui 5 tahapan. Tahapan pertama adalah pengujian bentuk data. Bentuk data yang dimaksud adalah bagaimana bentuk label yang diberikan pada objek yang akan dideteksi. Hal ini dapat mempengaruhi performa model dikarenakan dengan bentuk data yang lebih presisi, hasil yang didapatkan juga dapat semakin baik. Bentuk data dengan performa terbaik akan digunakan pada pengujian berikutnya, yaitu pengujian metode pelatihan model. Pengujian ini bertujuan untuk mencari apakah dengan menggunakan metode yang lebih kompleks dapat menghasilkan performa yang lebih baik dibandingkan dengan metode yang lebih umum digunakan. Metode pelatihan terbaik akan digunakan pada pengujian berikutnya, yaitu pengujian ukuran citra pelatihan model. Pengujian ini bertujuan untuk mengetahui apakah ukuran citra yang digunakan dapat mempengaruhi performa model. Ukuran citra dengan performa terbaik serta varian model YOLOv7 dengan performa terbaik akan digunakan pada pengujian berikutnya, yaitu pengujian jumlah *epochs* dan *patience*. Pengujian bertujuan untuk apakah dengan menggunakan jumlah *epochs* yang lebih tinggi serta *patience* sebagai variabel kontrol untuk *early stopping* bisa

meningkatkan performa model secara keseluruhan. Model dengan performa terbaik akan diuji pada Jetson Nano sebagai salah satu contoh perangkat yang dapat digunakan ketika diterapkan pada dunia nyata.

Beberapa kriteria utama yang digunakan untuk melakukan perbandingan varian model YOLOv7 seperti mAP, jumlah FPS dan F1-score. Nilai mAP menggambarkan bagaimana model tersebut melakukan pendeteksian pada berkas citra atau video di mana semakin tinggi nilai mAP, semakin baik pula akurasi pendeteksian model. Kemudian, Jumlah FPS menggambarkan berapa banyak *frame* yang dapat diproses model dalam 1 detik. Jumlah FPS yang tinggi membuat model tersebut cocok untuk digunakan sebuah sistem *real-time* meskipun memiliki nilai mAP yang baik. Lalu, nilai F1-Score menggambarkan harmonisasi antara nilai *precision* dan *recall* dari sebuah kelas. Nilai F1-score yang tinggi menggambarkan model dengan probabilitas yang kecil untuk melakukan kesalahan pendeteksian karena nilai *precision* dan *recall* nya yang seimbang.

3.6. Rancangan Proses Implementasi Model pada Jetson Nano

Setelah mendapatkan model yang terbaik dari hasil pengujian, model tersebut akan diimplementasikan pada Jetson Nano untuk mendapatkan gambaran performa dari model ketika diterapkan pada *edge devices* sebagai perangkat yang akan digunakan pada kehidupan nyata. Latar belakang digunakannya Jetson Nano sebagai perangkat pengujian adalah Jetson Nano merupakan produk buatan Nvidia di mana Nvidia memiliki CUDA dan CUDNN, yaitu platform yang digunakan untuk mengaktifkan GPU sebagai perangkat utama untuk mengeksekusi dan membentuk model *machine learning*. Secara *default*, Jetson Nano sudah terpasang CUDA dan CUDNN didalamnya sehingga tidak perlu melakukan konfigurasi khusus untuk memasang platform tersebut.

Implementasi model YOLOv7 pada Jetson Nano dilakukan dengan memasang *library* utama yang digunakan untuk mengeksekusi model YOLOv7 seperti *PyTorch*, *Torchvision*, *Matplotlib*, *Numpy*, *Pillow*, dan lain-lain. Jetson Nano sendiri memiliki RAM yang kecil dan tidak cukup untuk mengeksekusi model YOLOv7 sehingga dibutuhkan RAM tambahan agar model tersebut dapat

dieksekusi. Karena Jetson Nano menggunakan sistem operasi ubuntu, hal tersebut dapat dilakukan menggunakan *Swap memory*.

Swap memory adalah fitur pada sistem operasi *linux* yang digunakan ketika perangkat membutuhkan sumber daya untuk mengeksekusi *process* pada perangkat, tetapi RAM pada perangkat tidak memiliki ruang yang tersisa. Linux akan mengambil ruang yang tidak digunakan pada memori penyimpanan untuk digunakan sebagai RAM tambahan sehingga *process* baru dapat dieksekusi[20]. *Swap memory* dapat dibentuk menggunakan berbagai cara seperti *Swap partition* dan *Swap file*. Karena tidak akan digunakan secara permanen, *Swap memory* akan dibentuk menggunakan *Swap file*. Terdapat beberapa langkah yang diperlukan untuk membuat sebuah *Swap file*. Sebelum membuat *Swap file*, perlu dipastikan bahwa di perangkat tersebut masih belum memiliki ruang khusus untuk *Swap memory*. Perintah yang dapat digunakan untuk mengetahui perangkat belum memiliki ruang khusus untuk *Swap memory* dapat dilihat pada Gambar 3.14.

```
sudo swapon --show
free -h
```

Gambar 3.14 Perintah untuk mencari *Swap Memory* [21]

Perintah pertama berfungsi untuk mencari tahu apakah sudah terdapat *Swap memory* yang terpasang pada perangkat, sedangkan perintah kedua berfungsi untuk memastikan tidak ada *Swap memory* yang sedang aktif pada perangkat. Kemudian, setelah memastikan tidak ada *swap memory* yang terpasang ataupun aktif pada perangkat, kita bisa memeriksa apakah terdapat ruang yang dapat digunakan sebagai *swap memory* pada memori penyimpanan. Perintah yang dapat digunakan untuk mengetahui ruang yang dapat digunakan pada memori penyimpanan dapat dilihat pada Gambar 3.15.

```
df -h
```

Gambar 3.15 Perintah untuk mencari ruang pada memori penyimpanan [21]

Setelah memastikan terdapat ruang pada memori penyimpanan yang dapat digunakan sebagai *Swap memory*, kita dapat membuat *Swap file*. Perintah yang

dapat digunakan untuk membuat *Swap file* pada *ubuntu* dapat dilihat pada Gambar 3.16.

```
sudo fallocate -l size directory
ls -lh directory
```

Gambar 3.16 Perintah untuk membuat *Swap File* [21]

Perintah pertama berfungsi untuk membentuk *swap file* pada perangkat. Kita perlu memasukkan ukuran dari *swap file* yang akan digunakan dan direktori *swap file* tersimpan pada perangkat. Kemudian, perintah kedua berfungsi untuk memastikan apakah *swap file* memiliki ukuran yang sesuai dengan perintah pertama. Kemudian, *swap file* perlu kita aktifkan sebagai sebuah *swap space* agar dapat digunakan sebagai *swap memory*. Perintah untuk mengaktifkan *swap file* menjadi sebuah *swap memory* dapat dilihat pada Gambar 3.17.

```
sudo chmod 600 directory
ls -lh directory
sudo mkswap directory
sudo swapon directory
```

Gambar 3.17 Perintah untuk mengaktifkan *Swap File* menjadi *Swap Memory* [21]

Perintah pertama digunakan untuk mengubah *permission* dari direktori *swap file* yang dibuat agar hanya bisa diakses pengguna dengan *permission* administrator atau *root* pada perangkat sehingga *swap file* tersebut tidak dapat dieksploitasi oleh pengguna tanpa *permission* administrator. Perintah kedua digunakan untuk memastikan apakah direktori sudah dalam *permission* administrator. Kemudian, perintah ketiga berfungsi untuk membuat direktori *swap file* menjadi *swap space* sehingga dapat digunakan sebagai *swap memory*. Lalu, perintah keempat berfungsi untuk mengaktifkan *swap file* menjadi *swap memory* pada perangkat.

Setelah *swap memory* aktif, model tersebut akan diuji performa pendeteksiannya dari segi akurasi dan kecepatan pendeteksian. Model akan dimuat pada Jetson Nano untuk melakukan proses *inference* pada berkas video. Terdapat beberapa berkas video yang akan di*inference* oleh model. Berkas video yang akan di*inference* tidak termasuk dalam berkas video yang *frame*-nya diekstrak untuk

dijadikan sebagai *dataset* pembentukan model. Proses *inference* dilakukan pada tiap *frame* yang dimiliki oleh berkas video tersebut yang hasilnya berupa nilai kecepatan pendeteksian model. Setelah proses *inference* selesai, berkas video yang digunakan akan diekstrak tiap *frame*-nya dan beberapa *frame*-nya akan dilabeli dan dijadikan data pengujian. Tujuan dilabelinya data-data tersebut adalah agar dapat mengalkulasi akurasi pendeteksian model ketika dimuat pada Jetson Nano. Data pada video tersebut akan diberikan label dengan bentuk yang sama dengan *dataset* pembentukan model, yaitu *polygon annotation*.

BAB 4

IMPLEMENTASI DAN ANALISIS SISTEM DETEKSI MULTI OBJEK BERBASIS YOLOV7

4.1. Implementasi Sistem

Untuk membentuk sistem pendeteksian jenis ikan, diperlukan proses pengembangan model *machine learning*. Model *machine learning* dikembangkan menggunakan algoritma YOLOv7 menggunakan komputer dengan spesifikasi yang tertera pada Tabel 3.1. Perangkat yang digunakan sudah terpasang CUDA dan CUDNN, yaitu *library* dari Nvidia yang berfungsi untuk mengaktifasi GPU dalam pembentukan model *machine learning* sehingga proses pembentukan model dapat berjalan lebih cepat dengan memanfaatkan *core* pada GPU serta CPU secara bersamaan.

Proses pembentukan model *machine learning* akan dilakukan menggunakan *framework PyTorch*. Terdapat 3 varian model YOLOv7 yang akan dilatih untuk dibandingkan sebelum dipasang pada sistem, yaitu YOLOv7-Tiny, YOLOv7 dan YOLOv7-X. Model yang akan dipasang pada sistem akan ditentukan melalui beberapa skenario pengujian.

Pengujian model *machine learning* akan dilakukan pada dua perangkat, yaitu komputer dengan spesifikasi yang tertera pada Tabel 3.1 dan Jetson Nano. Pengujian pada Jetson Nano dilakukan untuk mendapatkan gambaran performa dari model ketika diaplikasikan pada perangkat yang akan terpasang pada kapal untuk melakukan pendeteksian. Rincian spesifikasi Jetson Nano untuk pengujian tertera pada Tabel 4.1.

Tabel 4.1 Spesifikasi Jetson Nano untuk pengujian model

<i>Central Processor Unit</i>	Quad-Core ARM® Cortex®-A57 MPCore processor (4 CPUs)
<i>Graphics Processing Unit</i>	NVIDIA Maxwell™ architecture
<i>Memory</i>	4 GB 64-bit LPDDR4
<i>Operating System</i>	Ubuntu 18.04
<i>Python Version</i>	3.6.9

4.2. Dataset Pengujian

Pengujian model dilakukan menggunakan *dataset* berupa citra yang telah melewati proses pengambilan data dan *labelling*. Pengambilan data dilakukan menggunakan dua metode, yaitu pengambilan data dengan mengekstrak *frame* video hasil tangkapan kamera dengan objek yang ingin dideteksi serta mengambil dari *dataset* Fishnet. Data tersebut akan diberikan label menggunakan Label Studio dengan memprioritaskan citra yang terdapat objek ikan didalamnya sebagai objek utama pendeteksian.

Setelah melalui proses *labelling*, citra tersebut akan dibagi menjadi tiga bagian untuk proses pelatihan dan pengujian model, yaitu bagian *training*, *validation* dan *testing*. Hal ini bertujuan untuk memastikan model yang sudah terbentuk tidak mengalami *overfitting* pada *dataset training* sehingga ketika terdapat data baru yang digunakan, model tersebut masih dapat mendeteksi dan mengklasifikasikan data tersebut. Secara keseluruhan, jumlah citra yang digunakan sebagai *dataset* pelatihan dan pengujian model adalah 4280 citra yang terdiri dari 6 kelas objek.

Dataset tersebut dibagi dengan rasio 7:3 untuk pelatihan dan pengujian model dengan data untuk pengujian model dibagi kembali dengan rasio 1:1 [22]. Rasio yang digunakan merupakan salah satu rasio yang umum dalam pembentukan model di mana jumlah data untuk *training* membutuhkan data yang banyak untuk mempelajari data yang akan dideteksi. Namun, meskipun sebagian besar jumlah data akan digunakan untuk pelatihan, jumlah data untuk pengujian juga tidak boleh terlalu sedikit agar dapat mencari konfigurasi paling optimal untuk model yang akan terbentuk. Rincian dari persebaran citra serta jumlah data untuk tiap kelas terdapat pada Tabel 4.2 dan Tabel 4.3.

Tabel 4.2 Persebaran citra pada *dataset* yang dikembangkan

Nama Bagian	Jumlah Citra
<i>Training</i>	2.999
<i>Validation</i>	641
<i>Testing</i>	640
Total	4.280

Tabel 4.3 Jumlah data tiap kelas pada *dataset* yang dikembangkan

Kelas	Jumlah Data		
	Training	Validation	Testing
<i>human</i>	7423	1566	1553
<i>tuna</i>	4734	1071	967
<i>Skipjack tuna</i>	262	58	42
<i>Tongkol</i>	447	98	94
<i>Squid</i>	564	93	121
<i>unknown</i>	2631	560	586
Total	16061	3446	3363

4.3. Skenario Pengujian

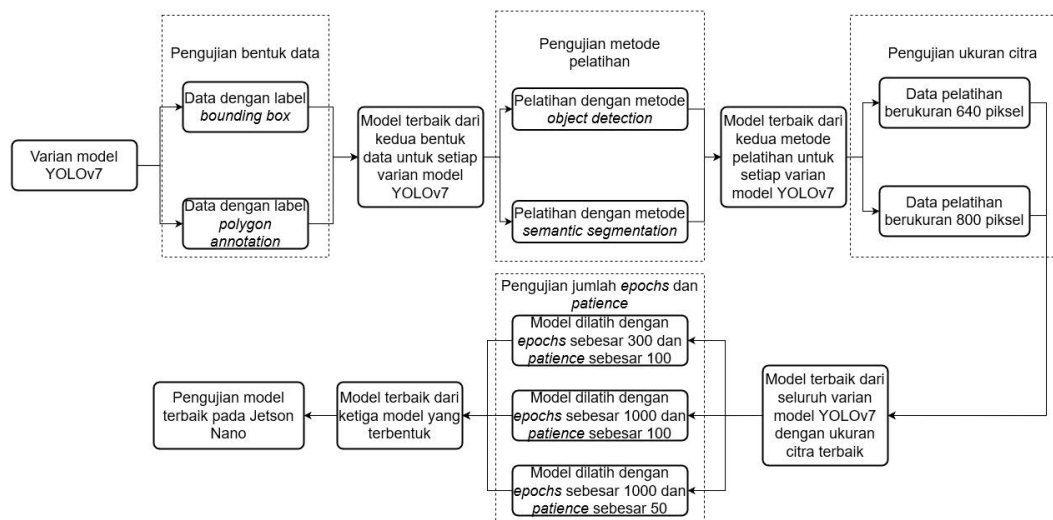
Dalam penelitian ini, ketiga varian model YOLOv7 yang dibandingkan akan diuji dengan beberapa skenario pengujian untuk mendapatkan model dengan kualitas pendeteksian dan performa yang terbaik. Pengujian dilakukan menggunakan *dataset* yang tertera pada Sub-bab 4.2 dengan jumlah data yang sama sehingga ketiga varian tersebut dapat dibandingkan dengan skenario pengujian yang sama. *Dataset* yang akan digunakan dalam pengujian adalah *dataset* yang termasuk pada bagian ‘testing’.

Pada penelitian ini, pengujian dilakukan pada tahap pelatihan model dan implementasi pada Jetson Nano sebagai perangkat yang digunakan di dunia nyata. Tahap pelatihan model akan dilakukan pada tiap varian model YOLOv7 yang dibandingkan untuk mendapatkan kualitas dan performa tiap varian dalam melakukan pendeteksian. Pelatihan yang dilakukan pada tiap varian akan menggunakan *batch size* yang berbeda dikarenakan kompleksitas arsitektur yang berbeda sehingga pelatihan model dapat dilakukan tanpa melebihi kapabilitas perangkat yang digunakan. Pengujian pada saat pelatihan model dilakukan dengan beberapa komponen pengujian diantaranya adalah bentuk *dataset*, teknik pelatihan model serta parameter yang digunakan dalam pelatihan seperti ukuran gambar dan jumlah *epochs*. Skenario pengujian secara visual dapat digambarkan seperti pada Gambar 4.1.

Kualitas dari hasil pelatihan model yang terbentuk akan ditentukan menggunakan mAP dan *F1-Score*. Kedua kriteria tersebut umum digunakan dalam menentukan kualitas model deteksi objek karena perhitungannya yang dapat merepresentasikan akurasi model secara menyeluruh dengan tidak hanya mencakup

nilai *true positive* saja, tetapi juga mencakup nilai *false positive* dan *false negative* sebagai nilai yang merepresentasikan kesalahan model dalam melakukan pendeteksian. Kemudian, pengujian kecepatan pendeteksian dilakukan menggunakan *inference time*. *Inference time* adalah satuan untuk menghitung waktu yang dibutuhkan model deteksi objek dalam melakukan pendeteksian. *Inference time* berkaitan dengan jumlah FPS yang model tersebut dapat hasilkan sehingga semakin kecil *inference time* dari model, semakin banyak jumlah FPS yang dapat dihasilkan dan diproses.

Setelah pengujian pada saat pelatihan model, model yang terbaik akan diuji pada Jetson Nano sebagai perangkat yang akan digunakan ketika diimplementasikan di dunia nyata. Hal ini bertujuan untuk membandingkan apakah terdapat penurunan baik secara kualitas maupun performa model dalam melakukan pendeteksian ketika perangkat yang digunakan memiliki proses komputasi yang lebih rendah dibandingkan perangkat untuk pelatihan model.



Gambar 4.1 Rincian skenario pengujian

4.4. Pengujian Model

Berdasarkan skenario pengujian yang telah dijelaskan pada Sub-bab 4.3, berikut adalah hasil dari tiap pengujian yang dilakukan:

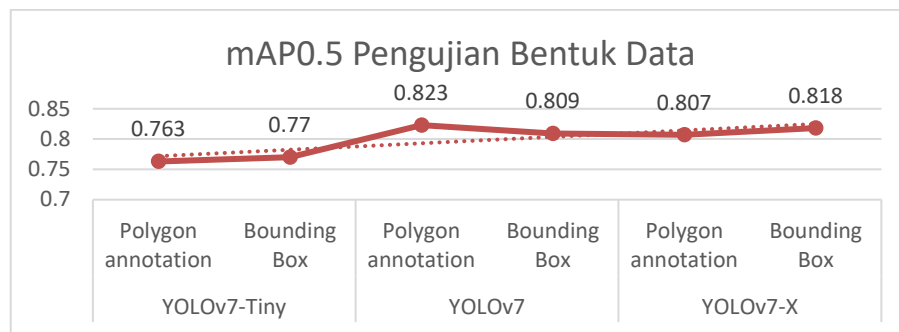
4.4.1. Pengujian Pengaruh Data Berbentuk *Polygon annotation* Terhadap Akurasi dan Jumlah FPS

Pada pengujian ini, tiap varian model YOLOv7 akan memasuki tahap pelatihan model dengan bentuk data yang berbeda. Hal ini bertujuan untuk mengetahui apakah bentuk data yang terdapat pada *dataset* dapat memiliki pengaruh yang signifikan terhadap nilai akurasi yang dimiliki oleh model. Bentuk data yang akan dibandingkan adalah data yang berbentuk *polygon annotation* dengan data yang berbentuk *bounding box*.

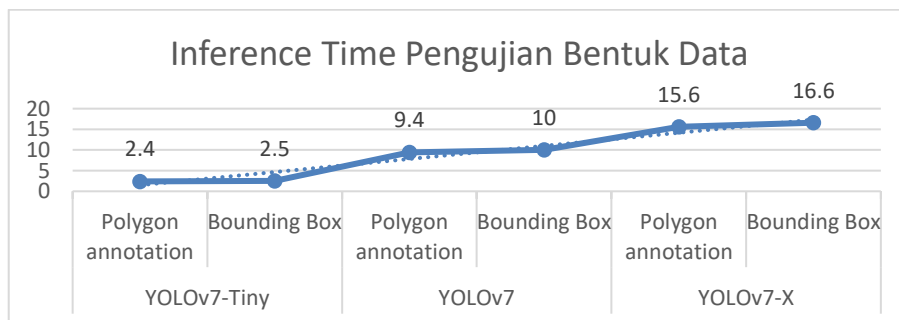
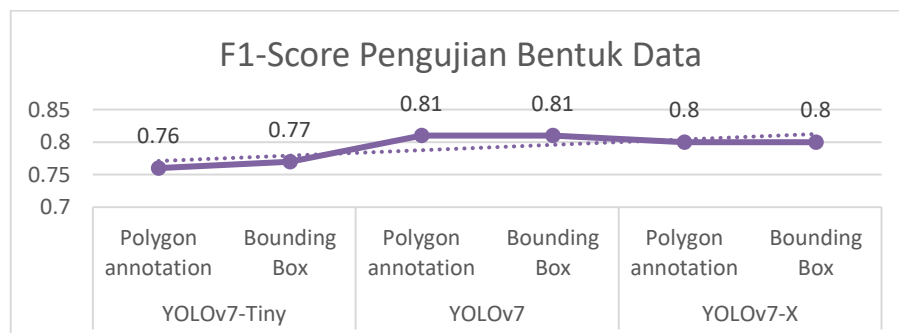
Pelatihan model akan menggunakan citra berukuran 640-piksel dari *dataset training* dengan *learning rate* sebesar 1×10^{-2} , diiterasi sebanyak 300 kali dengan *optimizers* SGD with momentum dikarenakan SGD menjadi *optimizers* yang paling optimal untuk membentuk model *computer vision* disebabkan oleh hasil yang lebih tergeneralisasi dibandingkan jenis *optimizers* lain ditambah dengan adanya momentum yang membuat proses konvergensi dapat terbentuk lebih cepat. *Batch size* yang digunakan akan menyesuaikan dengan kapabilitas dari perangkat pelatihan untuk setiap varian model. Kemudian, model yang terbentuk akan diuji menggunakan *dataset* bagian *testing* dengan *batch size* 1. Pengujian akan menggunakan ukuran citra yang sama dengan pelatihan model dengan IoU *Threshold* sebesar 0,6 dan *confidence Threshold* sebesar 0,001. Nilai IoU dan *confidence threshold* yang digunakan merupakan nilai *default* konfigurasi pengujian model pada YOLOv7.

Tabel 4.4 Hasil pengujian model dengan *dataset* berbentuk *Polygon annotation* dan *Bounding Box*

No	Varian Model	Bentuk Data	Batch Size pelatihan	mAP0.5	mAP0.5:0.95	F1-Score	Jumlah FPS	Inference Time (ms)
1	YOLOv7-Tiny	<i>Polygon annotation</i>	32	0,763	0,449	0,76	416,7	2,4
2		<i>Bounding Box</i>		0,77	0,455	0,77	400	2,5
3	YOLOv7	<i>Polygon annotation</i>	7	0,823	0,53	0,81	106,4	9,4
4		<i>Bounding Box</i>		0,809	0,525	0,81	100	10
5	YOLOv7-X	<i>Polygon annotation</i>	5	0,807	0,528	0,8	64,1	15,6
6		<i>Bounding Box</i>		0,818	0,533	0,8	60,2	16,6



Gambar 4.2 Grafik mAP0.5 pengujian bentuk data pada varian model

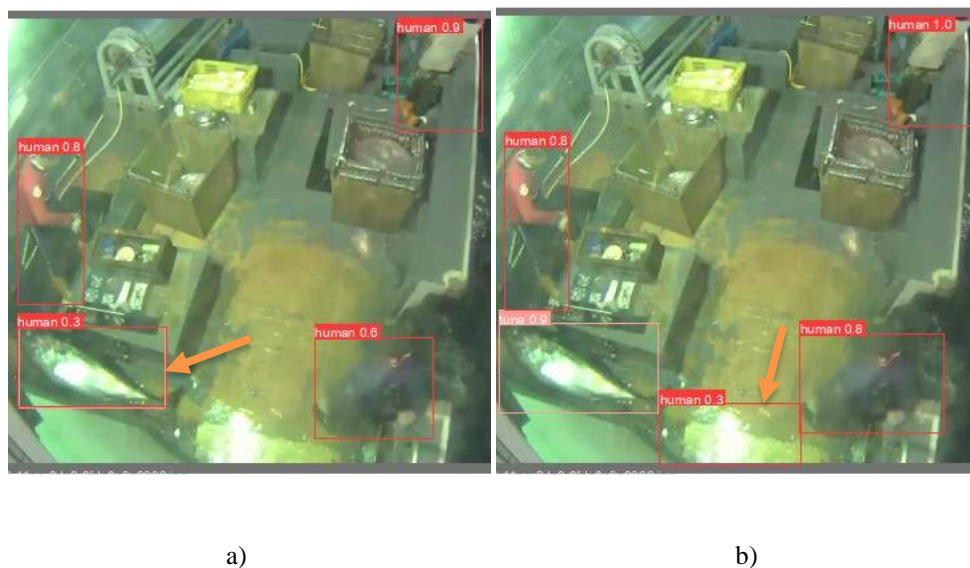
Gambar 4.3 Grafik *inference time* pengujian bentuk data pada varian modelGambar 4.4 Grafik *F1-score* pengujian bentuk data pada varian model

Dari hasil pada Tabel 4.4, dapat terlihat bahwasanya baik data berbentuk *polygon annotation* maupun *bounding box* memiliki hasil yang kurang lebih mirip. Dari segi akurasi, tidak ada perbedaan yang signifikan antara model yang dilatih dengan data berbentuk *polygon annotation* dengan *bounding box*. Selisih dari nilai mAP dan *F1-Score* untuk setiap varian model yang dilatih juga sangat kecil yang tidak mencapai 2% sehingga secara hasil pendeteksian, kedua model ini akan menghasilkan pendeteksian yang mirip. Jika dilihat pada Gambar 4.2, terlihat bahwa semakin kompleks arsitektur yang digunakan varian model, semakin tinggi

nilai mAP yang didapatkan dan hal tersebut juga berlaku pada nilai *F1-score*. Akan tetapi, jika diteliti lebih lanjut pada hasil pengujian, terdapat perbedaan hasil pendeteksian antara model yang dilatih dengan data *polygon annotation* dengan data *bounding box*.



Gambar 4.5 Contoh citra pengujian dengan label yang benar



Gambar 4.6 Hasil pendeteksian model YOLOv7 yang dilatih dengan data a) *Bounding box* dan b) *Polygon annotation*

Sebagai contoh, pada Gambar 4.5 – 4.6, terlihat bahwasanya model YOLOv7 yang dilatih dengan data *bounding box* mengidentifikasi kelas ‘tuna’ sebagai kelas ‘human’ yang termasuk dalam *false negative*, sedangkan model yang dilatih dengan data *polygon annotation* dapat memprediksi dengan tepat ‘tuna’ tersebut, tetapi

memiliki satu nilai *false positive* pada kelas ‘human’. Dari gambar tersebut, terlihat bahwa kedua model tersebut sama-sama melakukan kesalahan dalam pendeteksian objek, tetapi model yang dilatih dengan data *polygon annotation* masih dapat mengidentifikasi seluruh objek yang seharusnya terdeteksi pada citra, sedangkan model yang dilatih dengan data *bounding box* tidak bisa mengidentifikasi seluruh objek yang seharusnya terdeteksi.

Kemudian, dari segi kecepatan pendeteksian, model yang dilatih dengan data *polygon annotation* mendeteksi lebih cepat dibandingkan dengan model yang dilatih dengan data *bounding box*. Nilai *inference time* model yang dilatih dengan data *polygon annotation* dapat lebih cepat hingga 1 ms sehingga terdapat peningkatan juga pada jumlah FPS hingga 16 FPS. Jika dilihat pada Gambar 4.3, semakin kompleks arsitektur yang digunakan oleh varian model, semakin lambat kecepatan pendeteksian yang didapatkan. Faktor yang mungkin menyebabkan hal tersebut berkaitan dengan bagaimana anotasi data yang terbentuk di mana anotasi data *polygon annotation* dilakukan secara manual, sedangkan anotasi data *bounding box* adalah hasil konversi anotasi data *polygon annotation* menggunakan sebuah program seperti yang tertera pada Gambar 3.13. Karena label *bounding box* yang digunakan terbentuk dari hasil konversi, ditambah dengan secara bentuk label tersebut tidak lebih presisi dibandingkan *polygon annotation*, kecepatan pendeteksian mengalami penurunan.

4.4.2. Pengujian Perbandingan Metode *Object Detection* dengan *Semantic Segmentation* Terhadap Akurasi dan Jumlah FPS

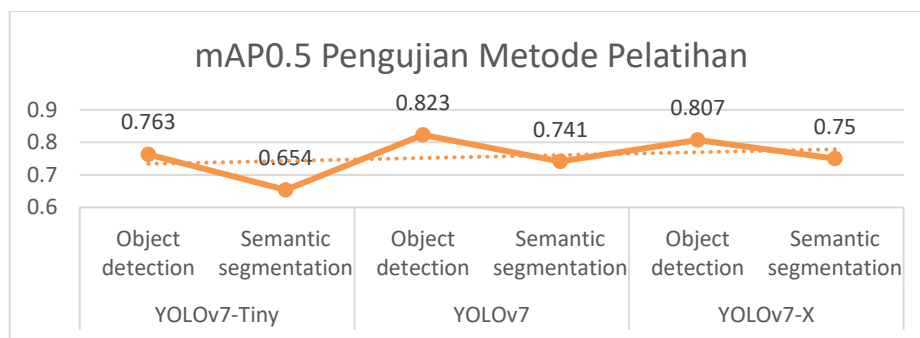
Pengujian berikutnya dilakukan dengan membandingkan metode *object detection* dan *semantic segmentation* yang digunakan dalam pelatihan model. Berdasarkan hasil pengujian sebelumnya, data berbentuk *polygon annotation* memiliki beberapa keunggulan dibandingkan dengan data berbentuk *bounding box*. Oleh karena itu, pengujian ini bertujuan untuk membandingkan performa pendeteksian model yang dilatih dengan dua metode yang berbeda menggunakan data berbentuk *polygon annotation*. *Semantic segmentation* merupakan pengembangan dari metode *object detection* di mana model akan melakukan

pendeteksian terhadap objek secara presisi dengan mencari titik tepi dari objek sehingga hasil pendeteksian akan berbentuk seperti objek tersebut. Oleh karena itu, *semantic segmentation* umum digunakan pada model yang membutuhkan tingkat presisi pendeteksian yang tinggi seperti pada sebuah kendaraan dengan fitur *autonomous driving*.

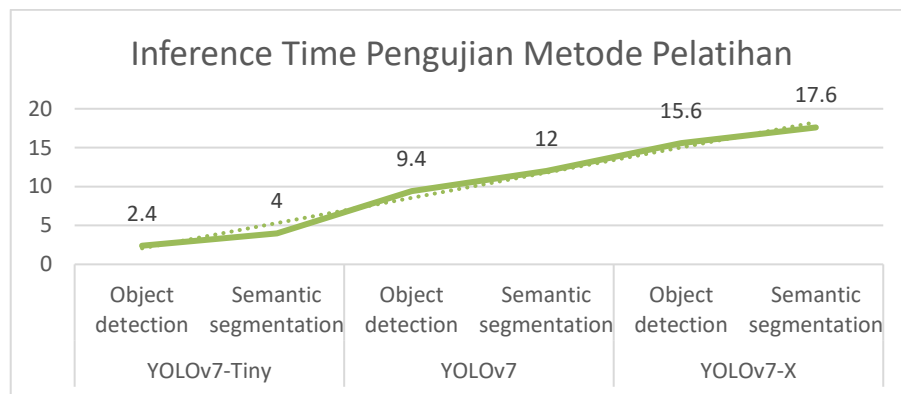
Proses pengujian dilakukan dengan melatih kembali ketiga varian model YOLOv7 menggunakan *dataset polygon annotation* bagian *training* dengan masing-masing metode pembentukan model. Pelatihan model juga akan menggunakan parameter yang sama dengan pengujian pertama kecuali pada bagian *batch size* yang menyesuaikan dengan varian model dan metode pembentukan agar tidak melebihi kapabilitas perangkat pelatihan. Kemudian, model yang terbentuk akan diuji dengan *dataset* bagian *testing* untuk mendapatkan performa pendeteksian masing-masing model menggunakan parameter pengujian yang sama dengan pengujian pertama.

Tabel 4.5 Hasil pengujian model dengan metode *Object Detection* dan *Semantic Segmentation*

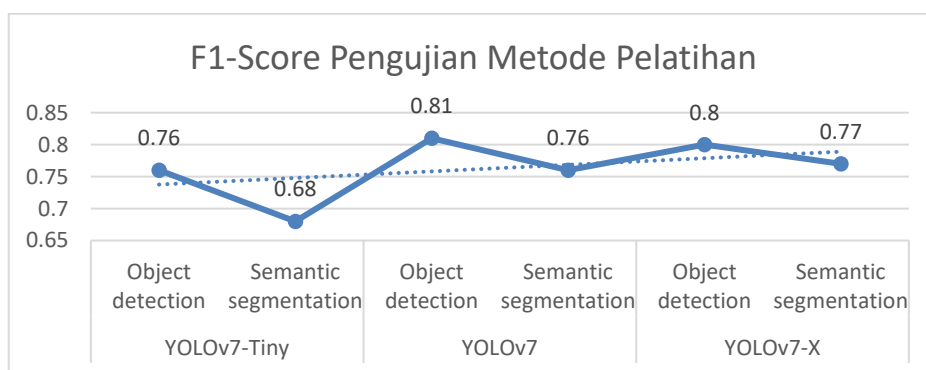
No	Varian Model	Metode Pembentukan	Batch Size pelatihan	mAP0.5	mAP0.5:0.95	F1-Score	Jumlah FPS	Inference Time (ms)
1	YOLOv7-Tiny	<i>Object Detection</i>	32	0,763	0,449	0,76	416,7	2,4
2		<i>Semantic Segmentation</i>	20	0,654	0,297	0,68	250	4
3	YOLOv7	<i>Object Detection</i>	7	0,823	0,53	0,81	106,4	9,4
4		<i>Semantic Segmentation</i>	5	0,741	0,353	0,76	83,3	12
5	YOLOv7-X	<i>Object Detection</i>	5	0,807	0,528	0,8	64,1	15,6
6		<i>Semantic Segmentation</i>	4	0,75	0,365	0,77	56,8	17,6



Gambar 4.7 Grafik mAP pengujian metode pelatihan pada varian model



Gambar 4.8 Grafik *inference time* pengujian metode pelatihan pada varian model



Gambar 4.9 Grafik *F1-score* pengujian metode pelatihan pada varian model

Dari hasil pada Tabel 4.5, berdasarkan performa model, dapat terlihat bahwa *object detection* mengalami peningkatan nilai mAP0.5 sebesar 7 – 15% dan nilai mAP0.5:0.95 sebesar 45 – 55%. Selain itu, model dengan metode *object detection* juga memiliki nilai *F1-Score* yang lebih baik sebesar 3 – 11%. Sama seperti pengujian sebelumnya, jika dilihat pada Gambar 4.7 dan 4.9, semakin kompleks arsitektur yang digunakan varian model, semakin tinggi nilai mAP dan *F1-score* yang diperoleh oleh model. Faktor yang paling berpengaruh dalam perbedaan tersebut adalah bagaimana *dataset* yang digunakan diproses oleh kedua metode tersebut. *Dataset* yang digunakan pada pelatihan model dengan metode *object detection* dan *semantic segmentation* memiliki bentuk data berupa *polygon annotation*, tetapi metode *object detection* hanya akan menampilkan *bounding box* hasil pendeteksian, sedangkan metode *semantic segmentation* akan menampilkan *bounding box* dan hasil segmentasi yang dilakukan model pada data sehingga

dengan salah satu pola ‘unknown’ pada *dataset* bagian *training* sehingga pola tersebut dapat terdeteksi sebagai ‘unknown’.

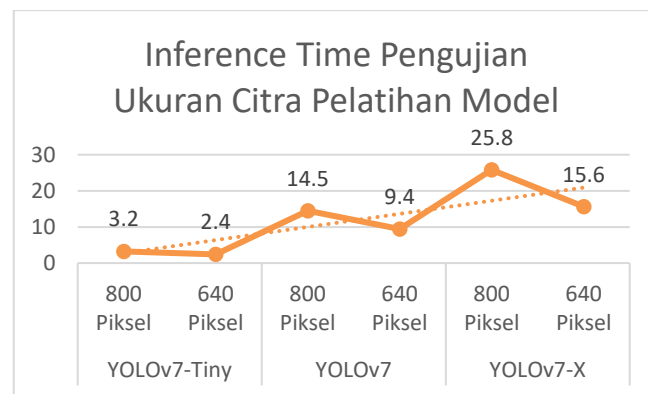
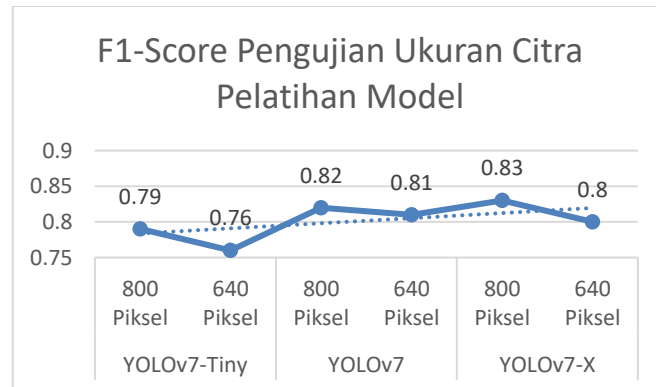
Kemudian, jika dilihat dari segi kecepatan pendeteksian, model *object detection* bisa memiliki nilai *inference time* yang lebih cepat 1,6 – 2,6 ms sehingga meningkatkan jumlah FPS hingga 166 FPS dibandingkan model *semantic segmentation*. Sama seperti pengujian sebelumnya, jika dilihat pada Gambar 4.8, kecepatan pendeteksian akan semakin melambat dengan semakin kompleks arsitektur yang digunakan varian model. Faktor yang menyebabkan hal tersebut juga berkaitan dengan pemrosesan *dataset* yang digunakan. Dikarenakan *semantic segmentation* membutuhkan tingkat presisi yang tinggi dalam melakukan pendeteksian, dibutuhkan waktu yang lebih lama untuk mencari pola yang mirip dengan objek pada *dataset* dan memastikan bahwa pola tersebut merupakan objek yang ingin dideteksi dibandingkan dengan *object detection* yang hanya perlu menyesuaikan *bounding box* dari objek yang ingin dideteksi.

4.4.3. Pengujian Pengaruh Ukuran Citra Pelatihan Model Terhadap Akurasi dan Jumlah FPS

Pengujian berikutnya dilakukan dengan membandingkan ukuran citra yang digunakan untuk pelatihan model. Pengujian ini bertujuan untuk mengetahui apakah ukuran citra yang digunakan pada pelatihan model dapat berpengaruh terhadap model yang terbentuk. Pengujian dilakukan dengan melakukan pelatihan model menggunakan ukuran citra yang berbeda, yaitu ukuran 640 piksel dan 800 piksel untuk setiap varian model. Ukuran citra 800 piksel dipilih karena perangkat yang digunakan untuk pelatihan model akan gagal membentuk model akibat kekurangan sumber daya yang dibutuhkan untuk melatih model menggunakan ukuran citra yang lebih besar dari 800 piksel. Bagian *dataset* dan parameter yang digunakan sama seperti pengujian sebelumnya terkecuali pada *batch size* yang juga menyesuaikan dengan kapabilitas perangkat. Kemudian, pengujian akan dilakukan menggunakan parameter yang sama dengan pengujian sebelumnya terkecuali pada ukuran citra yang menyesuaikan dengan ukuran yang digunakan saat pelatihan model. Model dengan hasil terbaik akan digunakan pada pengujian berikutnya.

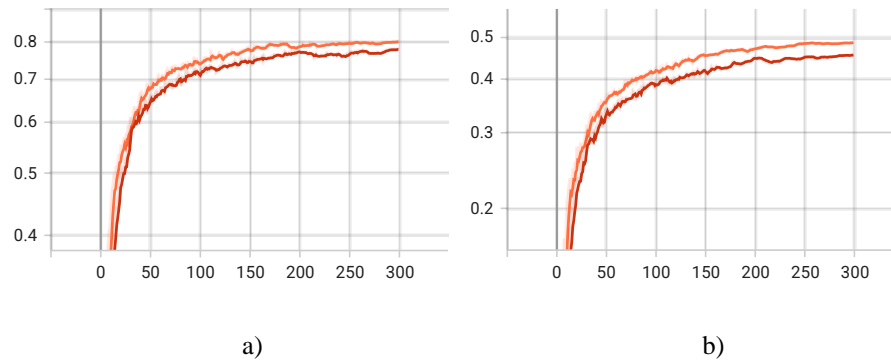
Tabel 4.6 Hasil pengujian model dengan ukuran citra 640 piksel dan 800 piksel

No	Varian Model	Ukuran Citra	Batch Size pelatihan	mAP0.5	mAP0.5 :0.95	F1-Score	Jumlah FPS	Inference Time (ms)	Ukuran berkas (MB)
1	YOLOv7-Tiny	640	32	0,763	0,449	0,76	416,7	2,4	12
2		800	19	0,789	0,48	0,79	312,5	3,2	
3	YOLOv7	640	7	0,823	0,53	0,81	106,4	9,4	72
4		800	4	0,843	0,557	0,82	69	14,5	
5	YOLOv7-X	640	5	0,807	0,528	0,8	64,1	15,6	139
6		800	3	0,852	0,571	0,83	38,8	25,8	

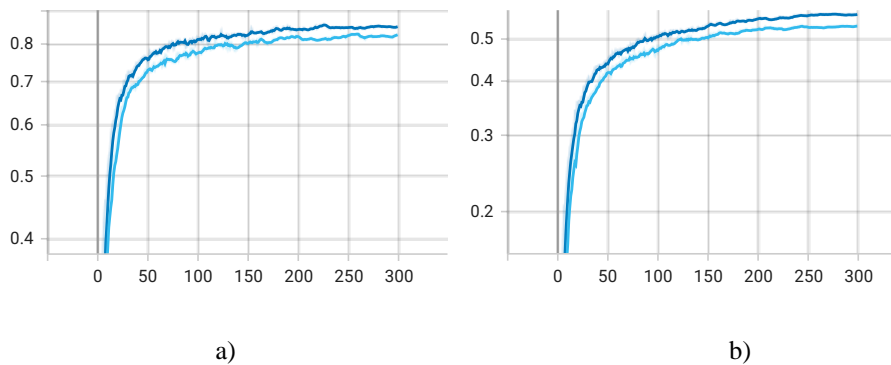
Gambar 4.12 Grafik *inference time* pengujian ukuran citra pelatihan model pada varian modelGambar 4.13 Grafik *F1-score* pengujian ukuran citra pelatihan model pada varian model

Dari hasil pada Tabel 4.6, terlihat bahwasanya masing-masing ukuran citra memiliki keunggulannya masing-masing. Dari segi performa, terlihat bahwa dengan menaikkan ukuran citra menjadi 800 piksel pada pelatihan model, nilai mAP dan *F1-Score* dari model dapat meningkat walaupun tidak secara signifikan. Hal tersebut terjadi pada ketiga varian model yang mengalami peningkatan mAP dan *F1-Score* dalam rentang 2 – 8%. Pada Gambar 4.13, terlihat bahwa model yang dilatih dengan ukuran citra 800 piksel akan memiliki akurasi yang lebih baik

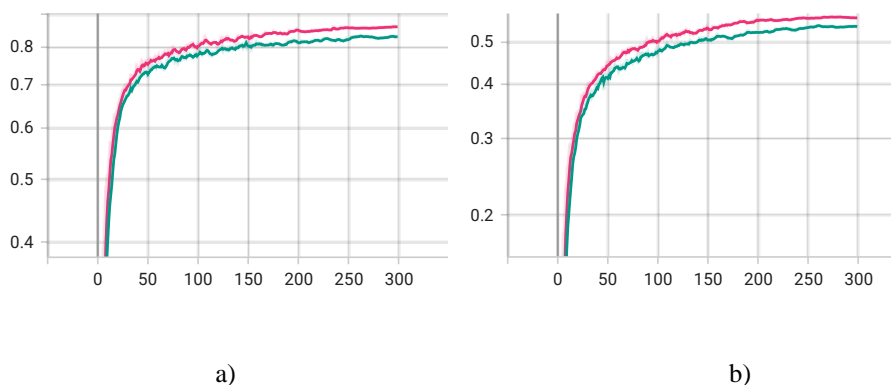
dibandingkan ukuran 640 piksel. Jika dilihat pada tren peningkatan mAP, model dengan ukuran citra 800 piksel selalu memiliki nilai mAP yang lebih baik dibandingkan model dengan ukuran citra 640 piksel pada setiap iterasinya seperti yang tertera pada Gambar 4.14 – 4.16.



Gambar 4.14 Perbandingan tren peningkatan nilai a) mAP0.5 dan b) mAP0.5:0.95 YOLOv7-Tiny dengan ukuran citra 640 piksel (Merah) dan 800 piksel (Oranye)



Gambar 4.15 Perbandingan tren peningkatan nilai a) mAP0.5 dan b) mAP0.5:0.95 YOLOv7 dengan ukuran citra 640 piksel (Biru Muda) dan 800 piksel (Biru Tua)

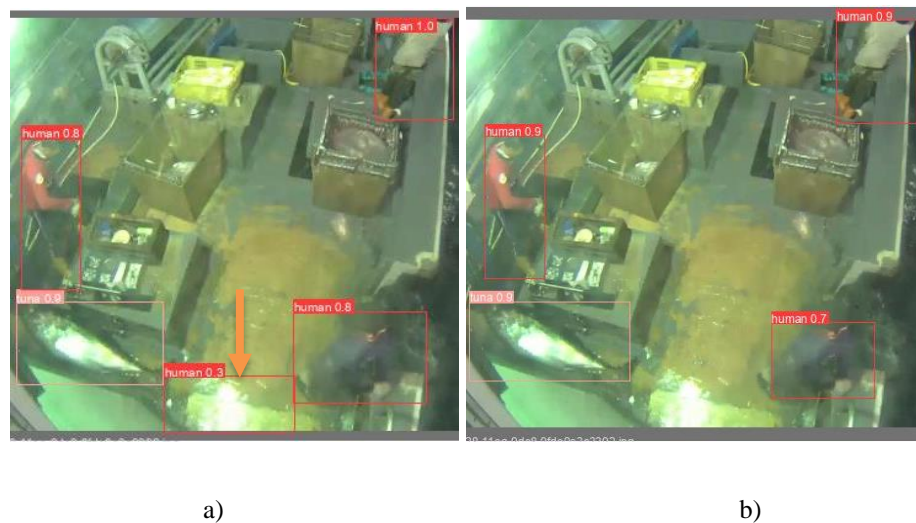


Gambar 4.16 Perbandingan tren peningkatan nilai a) mAP0.5 dan b) mAP0.5:0.95 YOLOv7-X dengan ukuran citra 640 piksel (Hijau) dan 800 piksel (Merah Muda)

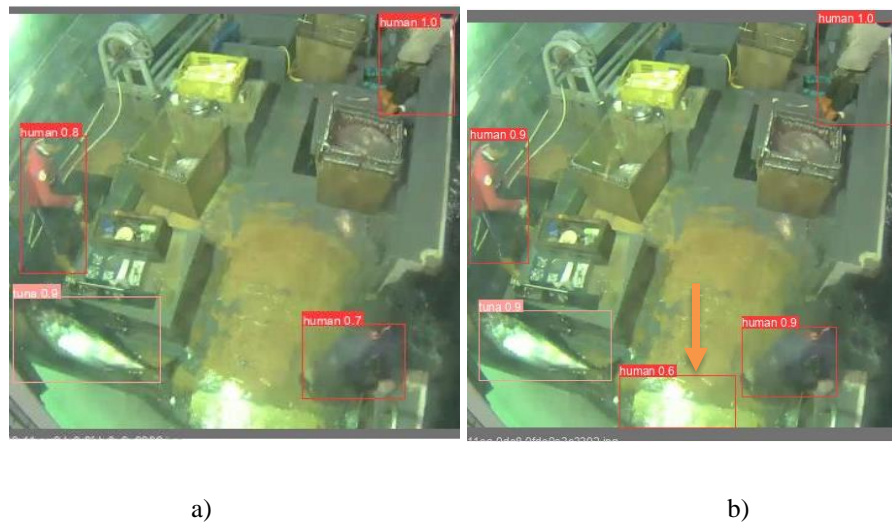
Terlihat pada Gambar 4.14 – 4.16 yang merupakan grafik mAP untuk setiap iterasi, ketiga varian model yang dilatih menggunakan ukuran citra berukuran 800 piksel selalu memiliki nilai mAP yang lebih baik dibandingkan dengan menggunakan ukuran citra 640 piksel baik pada mAP0.5 dan mAP0.5:0.95.



Gambar 4.17 Hasil pengujian model YOLOv7-Tiny dengan ukuran citra a) 640 piksel dan b) 800 piksel



Gambar 4.18 Hasil pendeteksian model YOLOv7 dengan ukuran citra a) 640 piksel dan b) 800 piksel



Gambar 4.19 Hasil pendeteksian model YOLOv7-X dengan ukuran citra a) 640 piksel dan b) 800 piksel

Kemudian, berdasarkan hasil pengujian pada Gambar 4.17 – 4.19 dengan Gambar 4.1 sebagai acuan label yang benar, dapat terlihat bahwa model yang dilatih menggunakan citra berukuran 800 piksel lebih mungkin untuk tidak salah mendeteksi dibandingkan dengan citra berukuran 640 piksel. Sebagai contoh, pada model YOLOv7-Tiny, terdapat 3 pendeteksian yang termasuk dalam *false positive* pada model dengan ukuran citra 640 piksel, sedangkan hanya terdapat 1 pendeteksian yang termasuk dalam *false positive* pada model dengan ukuran citra 800 piksel. Hal tersebut juga terjadi pada model YOLOv7 di mana model dengan ukuran citra 800 piksel tidak melakukan kesalahan dalam pendeteksian, sedangkan model dengan ukuran citra 640 piksel memiliki satu pendeteksian yang termasuk sebagai *false positive*. Hanya pada model YOLOv7-X hasil yang didapatkan memiliki perbedaan.

Jika dianalisis, ukuran citra yang lebih besar membuat lebih banyak objek yang terlihat dan objek tersebut juga lebih mudah untuk dideteksi dikarenakan jumlah piksel yang lebih banyak. Hal tersebut berkaitan dengan nilai mAP yang lebih tinggi di mana nilai *precision* dan *recall* dari model dengan ukuran citra yang lebih besar akan lebih tinggi karena objek yang dideteksi akan lebih mudah untuk terdeteksi. Selain itu, model dengan ukuran citra yang lebih besar juga memiliki nilai *loss* yang lebih rendah dikarenakan lebih banyak fitur dari data yang berhasil

diekstrak dan digunakan dalam pendeteksian sehingga kemungkinan model salah mendeteksi akan berkurang.

Dari segi kecepatan pendeteksian, terlihat bahwa model yang terbentuk dengan ukuran citra 640 piksel memiliki nilai *inference time* yang lebih cepat sebesar 0,8 – 5,2 ms sehingga meningkatkan jumlah FPS hingga 104,2 FPS dibandingkan dengan 800 piksel. Jika dianalisis pada Gambar 4.12, semakin kompleks arsitektur yang digunakan serta semakin besar ukuran citra yang digunakan, semakin lambat pula kecepatan pendeteksian yang didapatkan. Hal tersebut dapat terjadi dikarenakan piksel yang diproses pada ukuran citra 640 piksel lebih sedikit dibandingkan ukuran citra 800 piksel sehingga kompleksitas model dalam melakukan pendeteksian akan lebih rendah. Oleh karena itu, ukuran citra yang lebih kecil dapat meningkatkan kecepatan pendeteksian dan jumlah FPS.

Dikarenakan model yang dibutuhkan adalah model dengan tingkat akurasi yang tinggi, akurasi dari model lebih diprioritaskan dibandingkan kecepatan pendeteksian. Oleh karena itu, varian model YOLOv7 dengan ukuran citra 800 piksel akan dipilih sebagai model yang digunakan pada pengujian berikutnya dikarenakan memiliki akurasi sebesar 84,3% dan jumlah FPS sebanyak 69 FPS di mana secara performa, varian model YOLOv7 dengan ukuran citra 800 piksel cukup baik dari segi akurasi pendeteksian dan kecepatan pendeteksian. Selain itu, varian tersebut memiliki ukuran berkas 72 MB yang tidak terlalu besar untuk ukuran model sehingga dapat disimpan pada memori penyimpanan ukuran yang tidak besar. Selain itu, berdasarkan hasil pengujian, varian tersebut tidak melakukan kesalahan dalam melakukan pendeteksian.

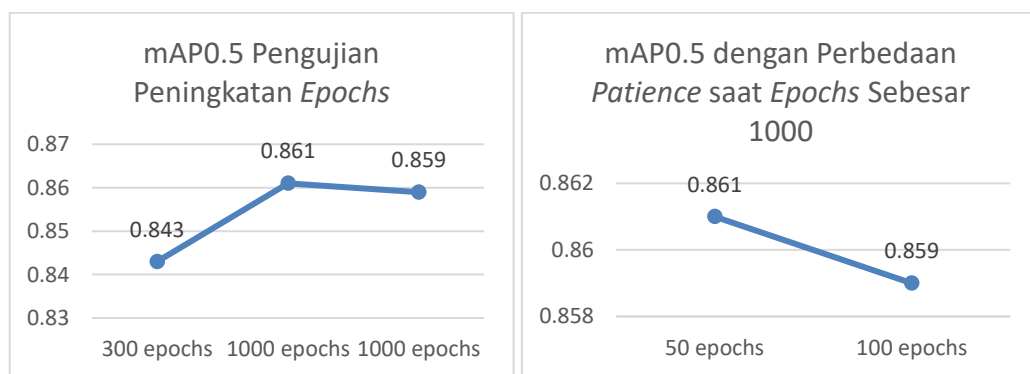
4.4.4. Pengujian Pengaruh Penambahan Epochs dalam Memicu *Early Stopping* Terhadap Akurasi dan Jumlah FPS

Pengujian berikutnya dilakukan dengan menambah jumlah *epochs* sebagai jumlah iterasi yang dilakukan dalam pelatihan model. Dengan menambah jumlah *epochs*, terdapat kemungkinan model tersebut memicu *early stopping*, yaitu sebuah algoritma yang digunakan untuk mencari jumlah *epochs* paling optimal untuk mencegah terjadinya *overfitting* pada model [13]. Oleh karena itu, pengujian ini

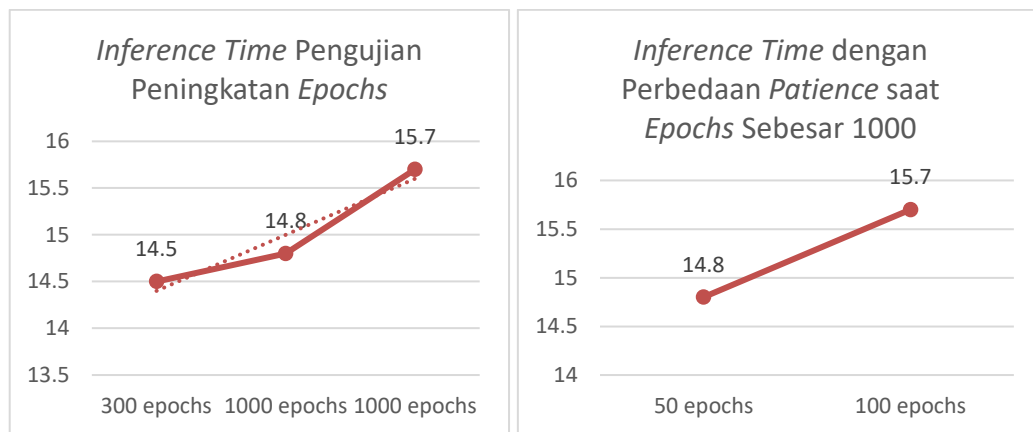
bertujuan untuk mengetahui apakah dengan terpicunya *early stopping*, model yang terbentuk memiliki akurasi yang lebih baik dan tidak mengalami *overfitting*. Pengujian akan dilakukan dengan varian model terbaik hasil pengujian sebelumnya, yaitu YOLOv7 menggunakan ukuran citra 800. Model tersebut dilatih menggunakan jumlah *epochs* sebesar 300 dengan *patience* sebesar 100 yang merupakan nilai *patience default* dari YOLOv7. Nantinya, model tersebut akan dilatih dengan jumlah *epochs* yang ditingkatkan menjadi 1000 dengan *patience* 50 dan 100 sebagai parameter untuk jumlah *epochs* yang dibutuhkan untuk menghentikan pelatihan model ketika tidak terjadi peningkatan. *Batch size* yang digunakan juga menyesuaikan dengan *batch size* yang digunakan model pada pengujian sebelumnya, yaitu 4.

Tabel 4.7 Hasil pengujian model dengan peningkatan *epochs*

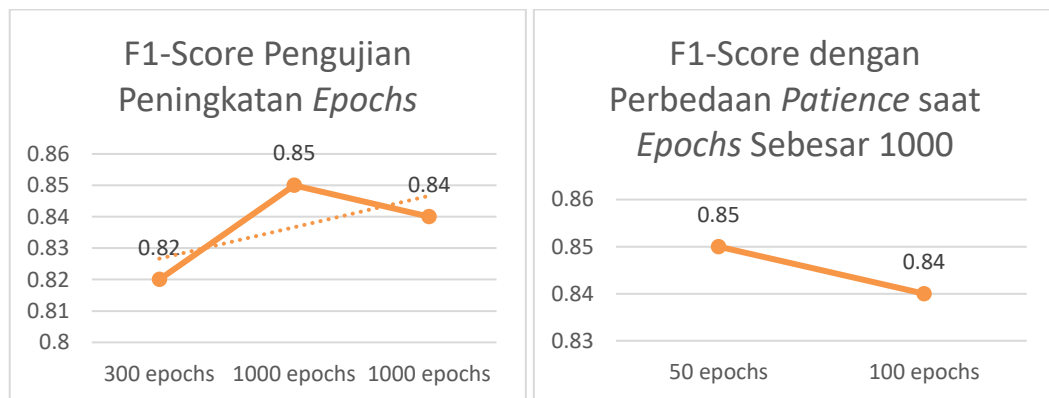
No	Jumlah Epochs	Patience	mAP0.5	mAP0.5:0.95	F1-Score	Jumlah FPS	Inference Time (ms)	Epochs Terbaik
1	300	100	0,843	0,557	0,82	69	14,5	-
2	1000	50	0,861	0,573	0,85	67,6	14,8	644/1000
3	1000	100	0,859	0,569	0,84	63,7	15,7	641/1000



Gambar 4.20 Grafik mAP0.5 pengujian peningkatan epochs dan perbedaan *patience* saat *epochs* sebesar 1000 pada varian model YOLOv7



Gambar 4.21 Grafik *inference time* pengujian peningkatan epochs dan perbedaan patience saat epochs sebesar 1000 pada varian model YOLOv7

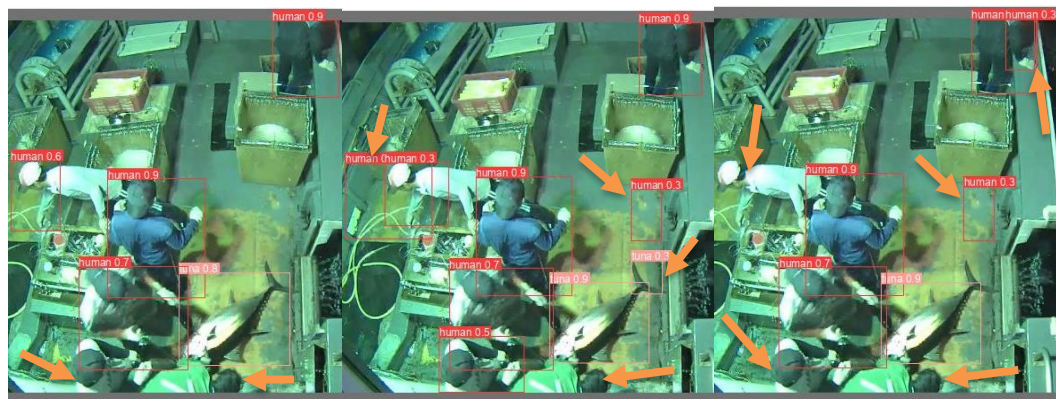


Gambar 4.22 Grafik *F1-score* pengujian peningkatan epochs dan perbedaan patience saat epochs sebesar 1000 pada varian model YOLOv7

Berdasarkan hasil pengujian pada Tabel 4.7, terlihat bahwasanya peningkatan jumlah *epochs* memiliki pengaruh yang kurang lebih mirip dengan peningkatan ukuran citra. Akurasi dari model tidak mengalami peningkatan yang signifikan di mana hanya terjadi peningkatan sekitar 3% dibandingkan model yang dilatih dengan 300 *epochs*. Hal tersebut bisa saja terjadi dikarenakan model yang terbentuk sudah bisa mengidentifikasi kelas yang ingin dideteksi sehingga peningkatan akurasi hanya akan menambah *node* pada *neural networks* dan menambah kemungkinan untuk teridentifikasi sebagai kelas tertentu. Peningkatan akurasi tersebut juga diikuti dengan bertambahnya pendeteksian yang termasuk dalam *false positive* dan *false negative*.



Gambar 4.23 Contoh citra pada pengujian dengan label yang benar



a)

b)

c)

Gambar 4.24 Hasil pengujian pada model YOLOv7 dengan a) *epochs* 300, b) *epochs* 1000 dengan *patience* 50, c) *epochs* 1000 dengan *patience* 100

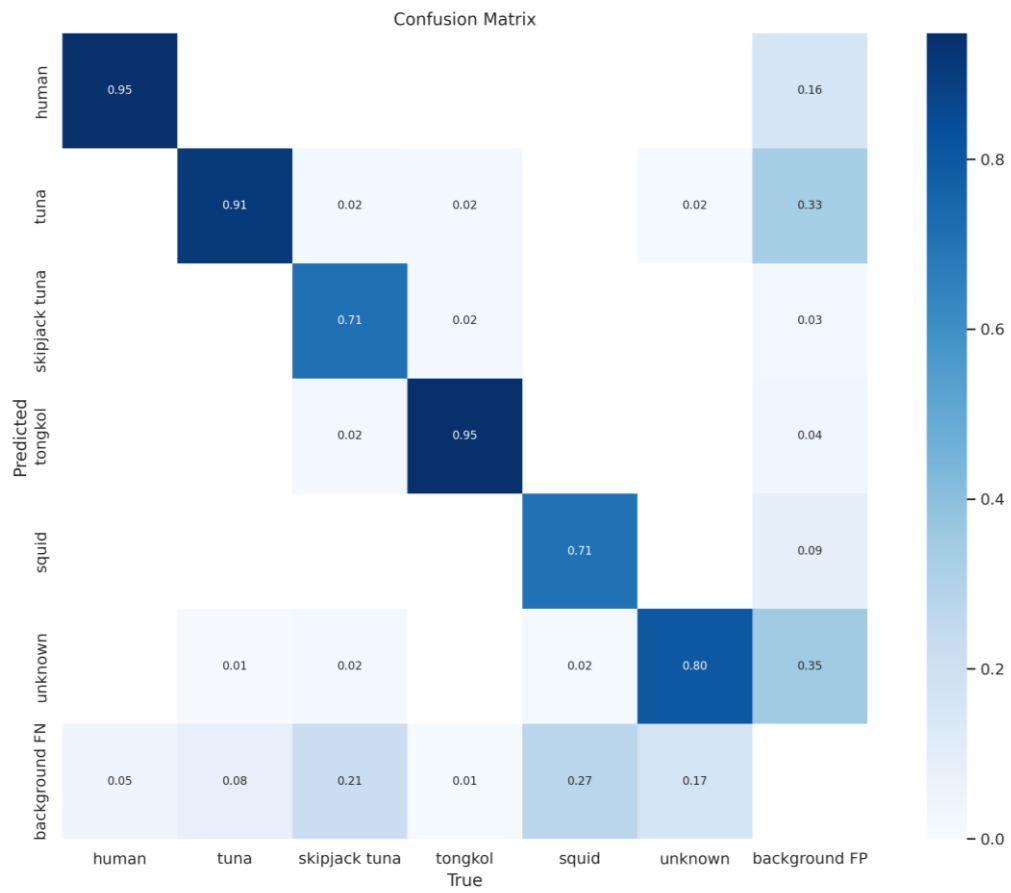
Sebagai contoh, pada Gambar 4.23 – 4.24, jika dianalisis, model YOLOv7 dengan *epochs* 300 memiliki 2 kesalahan pendeteksian yang termasuk dalam *false negative*. Kemudian, model YOLOv7 dengan *epochs* 1000 dan *patience* 50 memiliki 4 kesalahan pendeteksian yang terbagi menjadi 3 *false positive* dan 1 *false negative*. Lalu, model YOLOv7 dengan *epochs* 1000 dan *patience* 100 memiliki 5 kesalahan pendeteksian yang terbagi menjadi 3 *false negative* dan 2 *false positive*. Dari ketiga hasil tersebut, terlihat bahwasanya model dengan *epochs* yang lebih besar rentan untuk melakukan kesalahan. Hal tersebut bisa saja terjadi dikarenakan model

tersebut sudah hampir memasuki fase *overfitting* yang membuat model tidak tergeneralisir dalam melakukan pendeteksian.

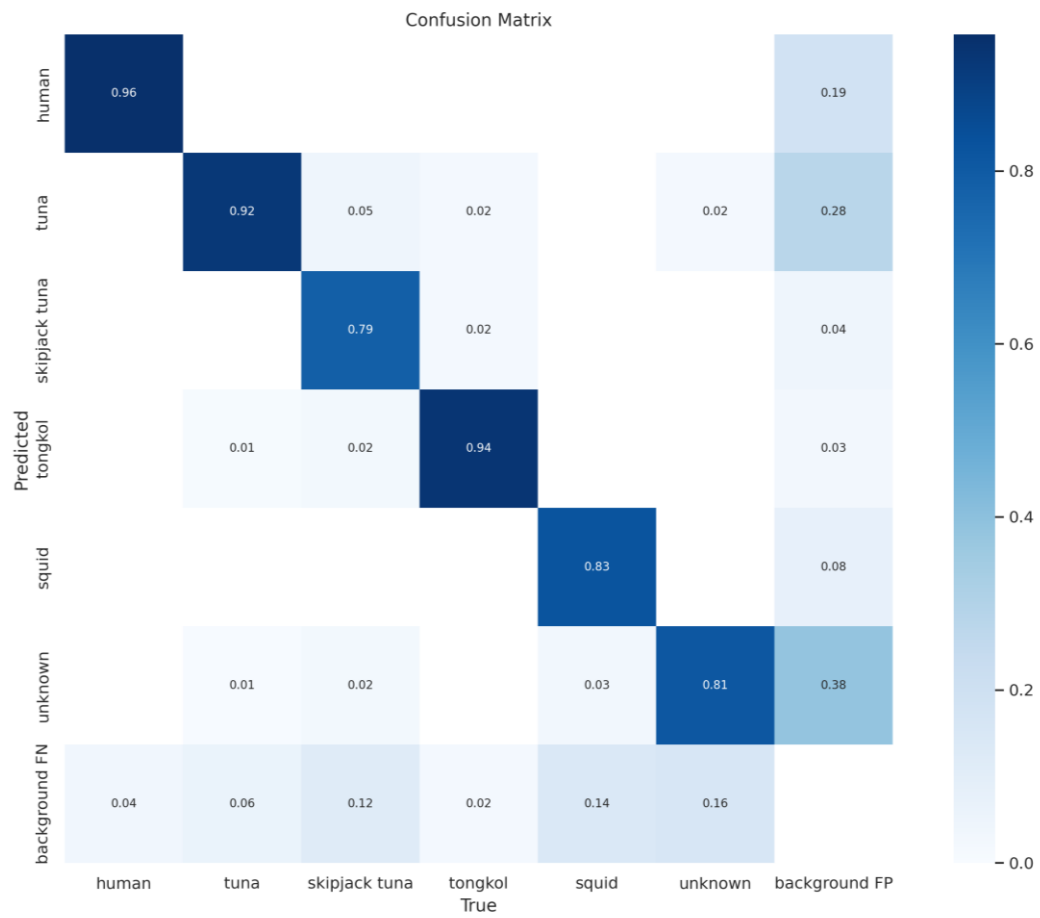
Namun, jika dianalisis lebih lanjut, ketiga model tersebut berhasil mendeteksi kelas ‘tuna’ dengan tepat meskipun terdapat *false positive* pada model YOLOv7 dengan *epochs* sebesar 1000 dan *patience* sebesar 50. Oleh karena itu, ketiga model ini masih dapat mengidentifikasi kelas ‘tuna’ dengan baik. Untuk mencari tahu bagaimana akurasi model dalam melakukan pendeteksian pada kelas yang termasuk sebagai ikan, perlu membandingkan bagaimana *F1-Score* dari tiap model ketika melakukan pendeteksian pada kelas yang termasuk dalam ikan sebagai tujuan utama model dibentuk.

Tabel 4.8 Perbandingan *F1-score* model YOLOv7 dalam mendeteksi kelas yang termasuk ikan

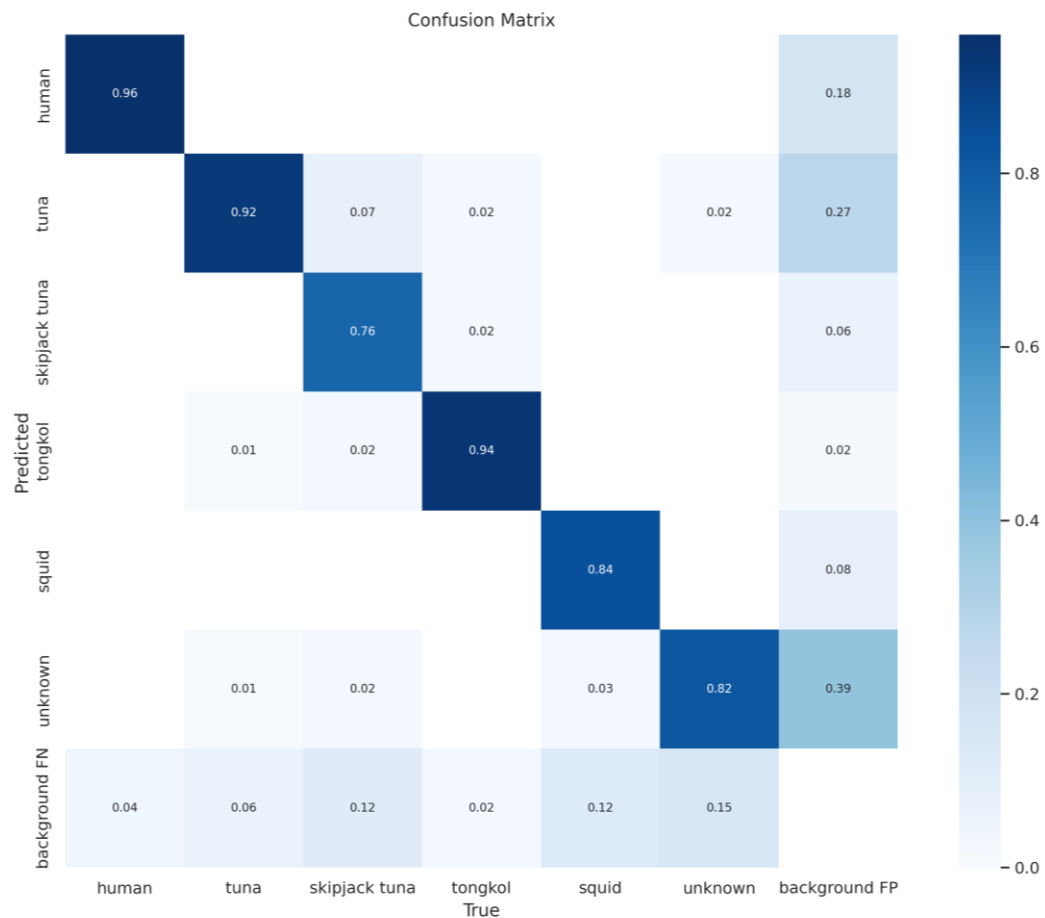
No	Kelas	Metrics	YOLOv7 <i>epochs</i> =300	YOLOv7 <i>epochs</i> =1000, <i>patience</i> =50	YOLOv7 <i>epochs</i> =1000, <i>patience</i> =100
1	Tuna	<i>Precision</i>	0,887	0,903	0,904
		<i>Recall</i>	0,893	0,915	0,908
		<i>F1-Score</i>	0,890	0,909	0,906
2	Skipjack Tuna	<i>Precision</i>	0,772	0,785	0,768
		<i>Recall</i>	0,646	0,738	0,738
		<i>F1-Score</i>	0,703	0,761	0,753
3	Tongkol	<i>Precision</i>	0,935	0,951	0,956
		<i>Recall</i>	0,936	0,936	0,936
		<i>F1-Score</i>	0,935	0,943	0,946
4	Squid	<i>Precision</i>	0,848	0,825	0,830
		<i>Recall</i>	0,529	0,661	0,628
		<i>F1-Score</i>	0,652	0,734	0,715
5	Unknown	<i>Precision</i>	0,799	0,779	0,788
		<i>Recall</i>	0,751	0,777	0,771
		<i>F1-Score</i>	0,774	0,778	0,779



Gambar 4.25 *Confusion matrix* hasil pengujian model YOLOv7 dengan *epochs* sebesar 300



Gambar 4.26 *Confusion matrix* hasil pengujian model YOLOv7 dengan *epochs* sebesar 1000 dengan *patience* sebesar 50



Gambar 4.27 *Confusion matrix* hasil pengujian model YOLOv7 dengan *epochs* sebesar 1000 dengan *patience* 100

Dari hasil pada Tabel 4.8, terlihat bahwa model dengan *epochs* yang lebih besar memiliki nilai *F1-Score* yang lebih tinggi dalam mendeteksi kelas yang termasuk sebagai ikan. Hal tersebut disebabkan oleh nilai *recall* dari model dengan *epochs* yang lebih besar cenderung lebih besar dibandingkan dengan jumlah *epochs* yang lebih kecil sehingga model tersebut lebih kecil kemungkinannya melakukan pendeteksian yang termasuk sebagai *false negative* seperti yang terlihat pada Gambar 4.25 – 4.27. Secara keseluruhan, model dengan *epochs* sebesar 1000 dan *patience* sebesar 50 adalah model yang terbaik dibandingkan model lain karena terdapat 3 dari 5 kelas yang memiliki nilai *F1-Score* tertinggi pada model tersebut, yaitu pada kelas tuna, ‘skipjack tuna’ dan ‘squid’ meskipun nilai *F1-score* dari kelas ‘skipjack tuna’ dan ‘squid’ tidak terlalu tinggi dikarenakan jumlah data yang digunakan memiliki jumlah yang sedikit dibandingkan data pada kelas lain.

Dari segi kecepatan pendeteksian, penambahan jumlah *epochs* membuat model memiliki kecepatan pendeteksian yang lebih lambat sekitar 2 - 7%. Penurunan nilai *inference time* dapat terjadi dikarenakan kompleksitas dari *neural networks* yang terbentuk pada model dengan makin banyak *node* yang membuat probabilitas dari setiap kelas juga bertambah sehingga dapat menambah waktu yang dibutuhkan untuk mengklasifikasikan objek yang terdeteksi. Model dengan *epochs* sebesar 1000 dan *patience* sebesar 50 sebagai model yang terbaik dari segi akurasi tidak mengalami penurunan kecepatan yang cukup signifikan saat melakukan pendeteksian. Oleh karena itu, model ini menjadi model yang terbaik dari pengujian dan akan digunakan pada pengujian berikutnya.

4.4.5. Pengujian Pendeteksian Model dengan Jetson Nano

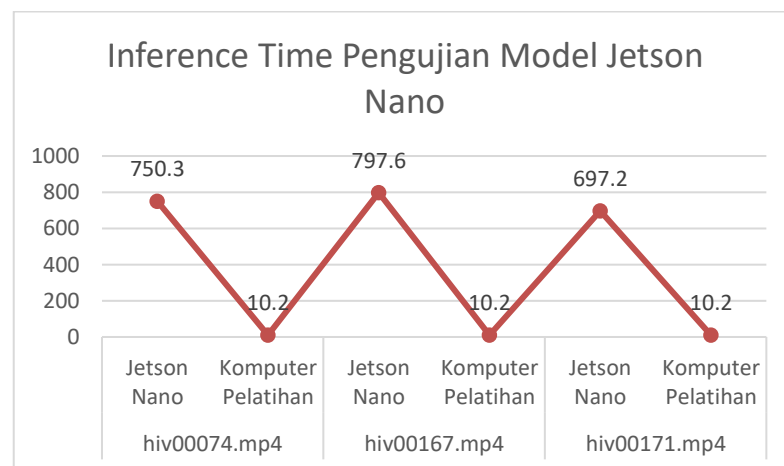
Pengujian terakhir dilakukan dengan menguji model yang terbaik berdasarkan hasil pengujian-pengujian sebelumnya di Jetson Nano. Pengujian ini bertujuan untuk mendapatkan perbandingan hasil pendeteksian antara perangkat yang digunakan untuk pelatihan model dengan perangkat yang dapat digunakan pada dunia nyata.

Pengujian akan dilakukan menggunakan sebuah video yang tidak termasuk dalam *dataset* pengujian dengan *frame rate* 24 FPS berukuran 640 piksel. Ukuran video tidak menggunakan ukuran 800 piksel sebagai ukuran citra terbaik dikarenakan Jetson Nano membutuhkan waktu yang melebihi batas waktu untuk melakukan pendeteksian pada bagian *non-maximum suppression*, yaitu 0,3 detik yang sudah terpasang pada fungsi *non-maximum suppression*. Oleh karena itu, ukuran citra 640 piksel akan digunakan agar pendeteksian pada *non-maximum suppression* tidak melebihi batas waktu. Video tersebut terdiri dari beberapa objek yang termasuk dalam kelas pada pelatihan model. Model akan dimuat pada Jetson Nano dan akan melakukan inferensi pada video tersebut untuk mendeteksi objek yang terdapat pada video dengan parameter *IoU threshold* sebesar 0,45 dan *confidence threshold* sebesar 0,25 yang merupakan bagian dari konfigurasi *default inference* model dari YOLOv7. Kemudian, video tersebut akan diekstrak tiap *frame*-nya untuk nantinya beberapa *frame* akan digunakan sebagai *dataset* baru dan

digunakan untuk menguji apakah akurasi dari model pada Jetson Nano dan perangkat pelatihan model memiliki hasil yang sama. Pengujian akurasi model pada Jetson Nano menggunakan parameter yang sama dengan parameter pada pengujian-pengujian sebelumnya.

Tabel 4.9 Hasil pengujian model pada Jetson Nano dan komputer pelatihan

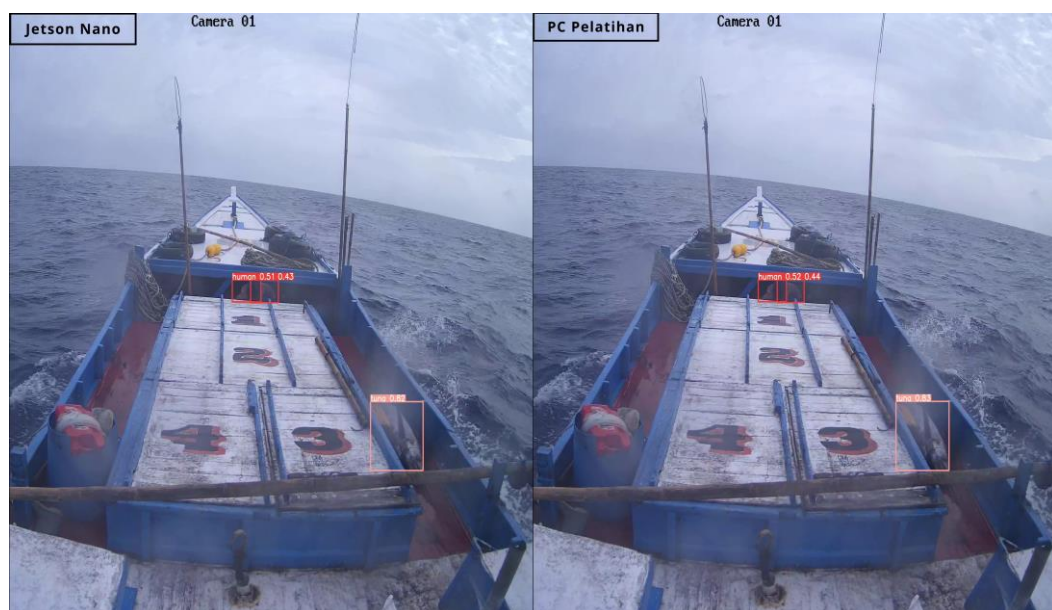
No	Berkas Video	Perangkat	Jumlah FPS	<i>Inference Time</i> (ms)	Ukuran Video
1	hiv00074.mp4	Jetson Nano	1,3	750,3	384 x 640
2		Komputer Pelatihan	98	10,2	
3	hiv00167.mp4	Jetson Nano	1,3	797,6	
4		Komputer Pelatihan	98	10,2	
5	hiv00171.mp4	Jetson Nano	1,4	697,2	
6		Komputer Pelatihan	98	10,2	



Gambar 4.28 Grafik *inference time* pengujian model pada Jetson Nano

Berdasarkan hasil pengujian pada Tabel 4.9, dapat terlihat bahwasanya nilai *inference time* model dalam mendeteksi pada Jetson Nano mengalami penurunan yang signifikan dibandingkan dengan *inference time* pada komputer pelatihan. Kecepatan pendeteksian pada Jetson Nano dapat mengalami penurunan sebesar 687 – 787 ms dibandingkan kecepatan pendeteksian pada komputer pelatihan. Hal tersebut dapat terjadi karena beberapa faktor, salah satunya adalah perbedaan proses komputasi yang dapat dilakukan oleh Jetson Nano dan komputer pelatihan. Jetson Nano memiliki spesifikasi yang jauh lebih rendah dibandingkan dengan komputer pelatihan seperti CPU, GPU dan RAM. Menurunnya spesifikasi perangkat membuat proses komputasi yang dapat dilakukan Jetson Nano tidak bisa menyamai proses komputasi yang dapat dilakukan oleh komputer pelatihan. Selain itu, varian

model YOLOv7 memiliki arsitektur yang cukup kompleks jika dibandingkan dengan varian YOLOv7 yang lebih rendah seperti YOLOv7-Tiny sehingga *trainable parameters* model memiliki jumlah yang lebih banyak. Oleh karena itu, dengan spesifikasi yang tertera pada Tabel 4.1, Jetson Nano tidak memiliki spesifikasi yang dapat memproses seluruh parameter dengan cepat sehingga kecepatan pendeteksian pada Jetson Nano akan menurun. Akan tetapi, jika dianalisis lebih lanjut pada video hasil *inference*, tidak ada *frame* video yang hilang meskipun kecepatan pendeteksian mengalami penurunan.



Gambar 4.29 Contoh *frame* hasil *inference* pada video hiv00167.mp4

Sebagai contoh, Gambar 4.29 merupakan salah satu *frame* dari berkas video hiv00167.mp4. Dapat dilihat bahwa hasil *inference* model pada video masih menampilkan *frame* dengan objek deteksi yang sama meskipun kecepatan pendeteksian mengalami penurunan. Hal tersebut dapat terjadi dikarenakan *inference* akan dilakukan pada semua *frame* yang terdapat pada video sehingga hasil akhirnya akan memiliki durasi dan *frame rate* yang sama meskipun kecepatan pendeteksian mengalami penurunan.

Kemudian, berdasarkan akurasi pendeteksian, model yang dipasang pada Jetson Nano dan komputer pelatihan tidak mengalami perbedaan. Dari hasil pada

Tabel 4.10, dapat terlihat bahwa model tersebut memiliki nilai mAP dan $F1-Score$ yang sama meskipun menggunakan perangkat dengan spesifikasi yang sangat berbeda.

Tabel 4.10 Hasil pengujian akurasi model pada Jetson Nano dan komputer pelatihan

No	Perangkat	Ukuran Citra	$mAP_{0.5}$	$mAP_{0.5:0.95}$	$F1-Score$
1	Jetson Nano	640	0,669	0,47	0,68
2	Komputer Pelatihan		0,669	0,47	0,68



Gambar 4.30 Citra hasil ekstraksi *frame* video pengujian dengan label yang benar



a)

b)

Gambar 4.31 Hasil pengujian akurasi pendeteksian pada a) Jetson Nano dan b) Komputer pelatihan

Sebagai contoh, pada Gambar 4.30, terlihat bahwa citra memiliki dua label, yaitu kelas 'tuna' dan 'human'. Pada Gambar 4.31, hasil pengujian model memiliki hasil yang sama di mana model pada Jetson Nano dan komputer pelatihan berhasil

mendeteksi kedua objek yang seharusnya terdeteksi dengan label yang tepat ditambah dengan dua kesalahan *false positive* pada kelas ‘unknown’ dan ‘human’. Hal tersebut dapat terjadi karena model yang digunakan memiliki jumlah *trainable parameters* yang sama sehingga model tersebut akan memiliki kapabilitas pendeteksian yang sama meskipun dipasang pada perangkat dengan spesifikasi yang sangat berbeda. Oleh karena itu, dapat disimpulkan bahwa penggunaan perangkat Jetson Nano hanya akan menurunkan kecepatan pendeteksian tanpa mengurangi akurasi pendeteksian.

4.5. Dampak Terhadap Lingkungan, Masyarakat dan/atau Bidang Lain

Jika ditinjau dari aspek lingkungan, masyarakat atau bidang lain, pengembangan sistem deteksi multi objek untuk pendeteksian jenis ikan berbasis YOLOv7 memberikan beberapa dampak pada beberapa aspek. Dampak utama dari pengembangan sistem deteksi ini adalah dapat mencegah terjadinya tindak *illegal, unreported and unregulated (IUU) fishing* sesuai dengan latar belakang pengembangan sistem. Dengan sistem ini, hasil tangkapan dari nelayan dapat dipantau sehingga jumlah ikan dapat diketahui dan diverifikasi ketika tiba di pelabuhan. Jika jumlah ikan yang dilaporkan tidak sesuai dengan hasil pemantauan, dapat terindikasi bahwa nelayan melakukan tindak IUU *fishing* dengan memeriksa kembali hasil pemantauan.

Kemudian, dengan berkurangnya tindak IUU *fishing*, pendapatan devisa negara Indonesia dapat bertambah dari sektor perikanan. Hal tersebut berkaitan dengan bertambahnya jumlah ikan yang dapat diekspor ke berbagai negara jika penjualan tidak dilakukan secara ilegal sehingga sektor perikanan dapat dikembangkan oleh pemerintah Indonesia. Selain itu, ekosistem laut Indonesia dapat terjaga dari kerusakan karena IUU *fishing*. Tindak IUU *fishing* sangat umum dilakukan dengan menggunakan peralatan yang tidak diperbolehkan untuk melakukan penangkapan ikan seperti jaring pukat harimau atau bom ikan agar mendapatkan banyak hasil tangkapan. Oleh karena itu, dengan berkurangnya penggunaan peralatan yang tidak diperbolehkan, ekosistem kelautan Indonesia akan tetap terjaga dari kerusakan atau kepunahan spesies.

Namun, pengembangan sistem pendeteksian jenis ikan ini masih belum sempurna. Terdapat beberapa kelemahan dari hasil pengembangan sistem yang telah dilakukan seperti model yang cukup berat untuk dieksekusi pada perangkat *edge devices* seperti mini-PC atau mikrokontroler karena membutuhkan proses komputasi yang cukup besar untuk mendapatkan kecepatan pendeteksian yang cepat. Selain itu, *dataset* yang digunakan juga masih terbatas pada 6 kelas sehingga dibutuhkan kelas tambahan untuk bisa mengidentifikasi objek yang mungkin terdeteksi. Rasio data antar kelas juga masih belum seimbang sehingga model lebih cenderung berhasil melakukan pendeteksian secara tepat pada kelas-kelas tertentu.

BAB 5

KESIMPULAN

5.1. Kesimpulan

Berdasarkan penelitian yang telah dilakukan, terdapat beberapa hal yang dapat disimpulkan terkait pengembangan sistem pendeteksian jenis ikan secara multi objek menggunakan algoritma deteksi objek YOLOv7, yaitu:

1. Pengembangan sistem deteksi multi objek untuk pendeteksian jenis ikan dengan model YOLOv7 berhasil dilakukan dengan *dataset* yang dirancang secara personal dengan jumlah gambar sebanyak 4280 mencakup 22.870 label dan 6 kelas.
2. Bentuk data *polygon annotation* tidak memiliki peningkatan yang signifikan pada akurasi pendeteksian, tetapi dapat meningkatkan kecepatan pendeteksian model YOLOv7 sebesar 0,1 hingga 1 ms dibandingkan bentuk data *bounding box* yang dapat meningkatkan jumlah FPS hingga 16 FPS.
3. Metode *object detection* dapat meningkatkan akurasi pendeteksian sebesar 45-55% dan meningkatkan kecepatan pendeteksian sebesar 1,6 – 2,6 ms sehingga meningkatkan jumlah FPS hingga 166 FPS dibandingkan metode *semantic segmentation*.
4. Menggunakan ukuran citra yang lebih besar pada pelatihan model dapat meningkatkan akurasi pendeteksian sebesar 2 – 8% pada model YOLOv7 dengan menurunkan kecepatan pendeteksian sebesar 0,8 – 5,2 ms sehingga menurunkan jumlah FPS hingga 104.2 FPS.
5. Meningkatkan jumlah *epochs* yang digunakan dapat mengurangi jumlah pendeteksian *false negative* model YOLOv7 dengan peningkatan akurasi sebesar 3% dan penurunan kecepatan pendeteksian sebesar 0,3 – 1,2 ms.
6. Berdasarkan skenario pengujian 1 sampai 4, Varian YOLOv7 menjadi varian model terbaik dengan akurasi pendeteksian hingga 86,1%, kecepatan pendeteksian hingga 14,5 ms dan jumlah FPS sebesar 69 FPS. Hal tersebut didapatkan dengan bentuk data *polygon annotation*, metode pelatihan *object*

detection, ukuran citra sebesar 800 piksel dan jumlah *epochs* sebesar 1000 dengan *patience* sebesar 50.

7. Pengimplementasian sistem pendeteksian jenis ikan secara multi objek berhasil dilakukan pada Jetson Nano dengan penurunan kecepatan pendeteksian sebesar 687 – 787 ms sehingga menurunkan jumlah FPS hingga 96 FPS dibandingkan dengan perangkat pelatihan, tetapi dengan akurasi pendeteksian yang sama.

5.2. Saran

Setelah menyelesaikan seluruh rangkaian penelitian, terdapat beberapa saran yang dapat diberikan untuk penelitian lebih lanjut, yakni:

1. Menambah jumlah kelas dari *dataset* yang digunakan sehingga lebih banyak jenis objek yang dapat terdeteksi oleh model dan menyeimbangkan rasio dari data untuk setiap kelas pada *dataset* agar pendeteksian tidak cenderung baik hanya pada kelas tertentu.
2. Menggunakan varian model YOLOv7 yang lebih rendah seperti YOLOv7-Tiny jika akan dikembangkan pada sebuah Jetson nano atau menggunakan perangkat *edge devices* yang lebih kencang dibandingkan Jetson Nano.
3. Mengembangkan model pendeteksian menggunakan metode *semantic segmentation* agar perhitungan jumlah ikan yang terdeteksi bisa lebih presisi dan menambah penggunaan dari model pendeteksian.

DAFTAR REFERENSI

- [1] “Fish; fillets, frozen exports by country |2021.”
<https://wits.worldbank.org/trade/comtrade/en/country/ALL/year/2021/trade-flow/Exports/partner/WLD/product/030420> (accessed Nov. 20, 2022).
- [2] “Tuna export company and exporters in Indonesia - Tridge.”
<https://www.tridge.com/intelligences/atlantic-bluefin-tuna/ID/export>
 (accessed Nov. 21, 2022).
- [3] “KKP Tangkap 167 Kapal Pelaku Illegal Fishing Selama 2021 Halaman all
 - Kompas.com.”
<https://money.kompas.com/read/2021/12/13/173905726/kkp-tangkap-167-kapal-pelaku-illegal-fishing-selama-2021?page=all> (accessed Nov. 21, 2022).
- [4] “IUU Fishing as an Evolving Threat to Southeast Asia’s Maritime Security | Asia Maritime Transparency Initiative.” <https://amti.csis.org/iuu-fishing-as-an-evolving-threat-to-southeast-asias-maritime-security/> (accessed Nov. 21, 2022).
- [5] “Indonesia sinks 8 Malaysian vessels for illegal fishing.”
<https://www.aa.com.tr/en/asia-pacific/indonesia-sinks-8-malaysian-vessels-for-illegal-fishing/2181821> (accessed Nov. 21, 2022).
- [6] “What is Computer Vision? | IBM.” <https://www.ibm.com/id-en/topics/computer-vision> (accessed Dec. 05, 2022).
- [7] Z. Zou, Z. Shi, Y. Guo, J. Ye, and S. Member, “Object Detection in 20 Years: A Survey,” May 2019, doi: 10.48550/arxiv.1905.05055.
- [8] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao, “YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors”.
- [9] C. J. L. Chan, E. J. A. Reyes, N. B. Linsangan, and R. A. Juanatas, “Real-time Detection of Aquarium Fish Species Using YOLOv4-tiny on Raspberry

- Pi 4,” in *4th IEEE International Conference on Artificial Intelligence in Engineering and Technology, IICAIET 2022*, Institute of Electrical and Electronics Engineers Inc., 2022. doi: 10.1109/IICAIET55139.2022.9936790.
- [10] N. Radha, R. Swathika, and P. S. Shreya, “Automatic Fish Detection in Underwater Videos using Machine Learning,” in *6th International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud), I-SMAC 2022 - Proceedings*, Institute of Electrical and Electronics Engineers Inc., 2022, pp. 587–592. doi: 10.1109/I-SMAC55078.2022.9987363.
- [11] V. Lakshmanan, M. Görner, and R. Gillard, *Practical Machine Learning for Computer Vision End-to-End Machine Learning for Images*.
- [12] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*.
- [13] M. Elgendy, O’Reilly for Higher Education (Firm), and an O. M. Company. Safari, *Deep Learning for Vision Systems*.
- [14] R. Szeliski, *Computer Vision: Algorithms and Applications 2nd Edition*. 2021. [Online]. Available: <https://szeliski.org/Book>,
- [15] “Mean Average Precision (mAP) Explained: Everything You Need to Know.” <https://www.v7labs.com/blog/mean-average-precision> (accessed Jun. 01, 2023).
- [16] “Getting Started with YOLO v4 - MATLAB & Simulink.” <https://www.mathworks.com/help/vision/ug/getting-started-with-yolo-v4.html> (accessed May 25, 2023).
- [17] “Description.” <https://www.fishnet.ai/description> (accessed Dec. 16, 2022).
- [18] “Polygon Annotation | Humans in the Loop.” <https://humansintheloop.org/services/polygon-annotation/> (accessed May 18, 2023).

- [19] “Label Studio Documentation — Install and Upgrade Label Studio.”
<https://labelstud.io/guide/install.html#Install-with-Anaconda> (accessed Jun. 01, 2023).
- [20] “Chapter 15. Swap Space Red Hat Enterprise Linux 7 | Red Hat Customer Portal.”
https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/storage_administration_guide/ch-swapspace (accessed May 22, 2023).
- [21] “How To Add Swap Space on Ubuntu 18.04 | DigitalOcean.”
<https://www.digitalocean.com/community/tutorials/how-to-add-swap-space-on-ubuntu-18-04> (accessed May 22, 2023).
- [22] “Train Test Validation Split: How To & Best Practices [2023].”
<https://www.v7labs.com/blog/train-validation-test-set> (accessed Jun. 10, 2023).