

**Dokumen Proses Analisis  
Sistem Klasifikasi Sampah Menggunakan  
Deteksi Objek Real-time dengan Model YOLOv8**

Diajukan untuk memenuhi salah satu tugas Pengolahan Citra Digital Praktek.



Disusun oleh:

Daffa Al Ghifari	231511038
M. Raihan Pratama	231511055
Tresnardi Fathu Rhamdan	231511062

**PROGRAM STUDI D-III TEKNIK INFORMATIKA  
JURUSAN TEKNIK KOMPUTER DAN INFORMATIKA  
POLITEKNIK NEGERI BANDUNG  
2025**

## ABSTRAK

Pengelolaan sampah merupakan salah satu tantangan lingkungan terbesar saat ini, dengan proses pemilahan manual yang tidak efisien dan rentan terhadap kesalahan. Proyek ini bertujuan untuk mengembangkan sebuah solusi teknologi berbasis *computer vision* untuk mengatasi masalah tersebut. Sistem yang dibangun mampu melakukan deteksi dan klasifikasi sampah secara otomatis melalui gambar statis maupun *video stream* secara *real-time*.

Metode yang digunakan adalah *transfer learning* pada model YOLOv8n (Nano), yang dilatih khusus untuk mengenali 22 jenis sampah yang berbeda. Hasil deteksi kemudian dikelompokkan ke dalam tiga kategori utama: dapat didaur ulang (*recyclable*), tidak dapat didaur ulang (*non-recyclable*), dan berbahaya (*hazardous*). Untuk mendemonstrasikan fungsionalitas model, dua prototipe aplikasi web dikembangkan: sebuah *backend* API menggunakan FastAPI dan sebuah aplikasi interaktif mandiri menggunakan Streamlit. Proyek ini berhasil menunjukkan kelayakan penggunaan teknologi deteksi objek untuk solusi segregasi sampah, dengan penekanan utama pada fungsionalitas prototipe ketimbang pencapaian akurasi model yang maksimal.

## DAFTAR ISI

ABSTRAK.....	2
DAFTAR ISI.....	3
DAFTAR GAMBAR.....	5
DAFTAR TABEL.....	6
BAB 1	
PENDAHULUAN.....	7
1.1 Latar Belakang.....	7
1.2 Rumusan Masalah.....	7
1.3 Tujuan Proyek.....	7
1.4 Ruang Lingkup Proyek.....	8
BAB 2	
LANDASAN TEORI.....	9
2.1 Computer Vision dan Deteksi Objek.....	9
2.2 Arsitektur YOLOv8.....	9
2.3 Transfer Learning.....	9
2.4 Teknologi Web Pendukung.....	9
BAB 3	
METODOLOGI DAN IMPLEMENTASI.....	11
3.1 Alat dan Lingkungan Pengembangan.....	11
3.2 Alur Kerja Proyek.....	11
3.2.1 Persiapan Dataset dan Pelatihan Model.....	11
3.2.2 Logika Klasifikasi Sampah.....	14
3.2.3 Implementasi Prototipe.....	15
BAB 4	
HASIL DAN ANALISIS.....	16
4.1 Hasil Performa Model.....	16
4.2 Interpretasi Hasil.....	21
1. Confusion Matrix (Matriks Kebingungan).....	22
2. Kurva Presisi-Recall (PR Curve).....	24
3. Kurva F1-Score (F1 Curve).....	25
4. Kurva Presisi (P Curve) dan Kurva Recall (R Curve).....	26
5. Labels (Daftar Kelas).....	28
4.3 Kesimpulan Performa.....	30

4.4 Kelebihan dan Kekurangan Solusi.....	30
<b>BAB 5</b>	
KESIMPULAN DAN SARAN.....	31
5.1 Kesimpulan.....	31
5.2 Saran Pengembangan.....	31
DAFTAR PUSTAKA.....	33

## **DAFTAR GAMBAR**

Tabel 4.1: Metrik Performa Model pada Validation Set..... 13

## **DAFTAR TABEL**

Gambar 4.1 Ringkasan Visual Pelatihan.....	17
Gambar 4.2 Confusion Matrix - Absolut.....	18
Gambar 4.3 Confusion Matrix - Normalisasi.....	19
Gambar 4.4 Kurva Precision-Recall.....	20
Gambar 4.5 Kurva F1-Score vs. Confidence.....	21
Gambar 4.6 Kurva Precision vs. Confidence.....	22
Gambar 4.7 Kurva Recall vs. Confidence.....	23
Gambar 4.8 Labels.....	24

## **BAB 1** **PENDAHULUAN**

### **1.1 Latar Belakang**

Volume sampah global terus meningkat seiring dengan pertumbuhan populasi dan urbanisasi. Pemilahan sampah yang efektif di sumbernya adalah kunci untuk meningkatkan laju daur ulang, mengurangi pencemaran lingkungan, dan mengoptimalkan pengelolaan limbah. Namun, proses pemilahan manual sering kali lambat, tidak konsisten, dan memaparkan pekerja pada risiko kesehatan. Oleh karena itu, diperlukan inovasi teknologi untuk mengotomatisasi proses ini. Kemajuan dalam bidang *kecerdasan buatan (AI)* dan *computer vision* menawarkan potensi besar untuk menciptakan sistem pemilahan yang cepat, akurat, dan aman.

### **1.2 Rumusan Masalah**

Berdasarkan latar belakang tersebut, rumusan masalah untuk proyek ini adalah: "Bagaimana merancang dan mengimplementasikan sebuah sistem berbasis *computer vision* yang mampu mendeteksi berbagai jenis sampah secara *real-time* dari input gambar atau video dan mengklasifikasikannya ke dalam kategori pengelolaan yang sesuai?"

### **1.3 Tujuan Proyek**

Proyek ini memiliki beberapa tujuan utama, dengan fokus pada implementasi dan pembuktian konsep:

1. Melakukan *fine-tuning* pada model deteksi objek YOLOv8 untuk membangun sebuah model dasar yang mampu mengenali 22 kelas sampah.
2. Mengembangkan logika untuk mengelompokkan hasil deteksi objek ke dalam tiga kategori utama: *Recyclable*, *Non-Recyclable*, dan *Hazardous*.
3. Membangun dua prototipe aplikasi web (menggunakan FastAPI dan Streamlit) sebagai bukti konsep (*proof-of-concept*) untuk mendemonstrasikan fungsionalitas deteksi dan klasifikasi secara *end-to-end*.

## 1.4 Ruang Lingkup Proyek

Ruang lingkup proyek ini terbatas pada:

- Pengembangan model perangkat lunak untuk deteksi dan klasifikasi objek.
- Input sistem berupa gambar digital (format JPG, PNG) dan *video stream* dari webcam.
- Output sistem berupa visualisasi kotak pembatas (*bounding box*) pada gambar/video dan daftar teks hasil klasifikasi.
- Proyek ini tidak mencakup perancangan atau implementasi pada perangkat keras fisik seperti lengan robot, sensor, atau sistem konveyor.

## **BAB 2** **LANDASAN TEORI**

### **2.1 Computer Vision dan Deteksi Objek**

*Computer vision* adalah cabang ilmu komputer yang bertujuan untuk membuat mesin dapat "melihat" dan menginterpretasi informasi visual dari dunia digital, seperti gambar dan video. Salah satu tugas fundamental dalam *computer vision* adalah deteksi objek, yaitu proses mengidentifikasi keberadaan dan lokasi objek dalam sebuah gambar serta mengklasifikasikan objek tersebut.

### **2.2 Arsitektur YOLOv8**

YOLO (You Only Look Once) adalah keluarga model deteksi objek yang terkenal karena kecepatan inferensinya yang tinggi, menjadikannya sangat cocok untuk aplikasi *real-time*. Proyek ini menggunakan YOLOv8, versi terbaru yang menawarkan peningkatan signifikan dalam hal akurasi dan efisiensi. Secara spesifik, varian yang dipilih adalah YOLOv8n (Nano), model terkecil dalam keluarga YOLOv8 yang dioptimalkan untuk perangkat dengan sumber daya komputasi terbatas tanpa mengorbankan kecepatan secara drastis.

### **2.3 Transfer Learning**

*Transfer learning* adalah sebuah teknik dalam *machine learning* di mana sebuah model yang telah dilatih untuk satu tugas digunakan sebagai titik awal untuk model pada tugas kedua yang berbeda namun terkait. Dalam proyek ini, model YOLOv8n yang telah dilatih pada dataset besar (seperti COCO) dimanfaatkan dan dilatih kembali (*fine-tuning*) menggunakan dataset sampah yang lebih spesifik. Pendekatan ini secara signifikan mengurangi waktu pelatihan dan kebutuhan akan data dalam jumlah yang besar.

### **2.4 Teknologi Web Pendukung**

- **FastAPI:** *framework* web Python yang dirancang untuk membangun API dengan performa tinggi. Sifat asinkronnya sangat ideal untuk menangani tugas seperti *streaming* video.

- **Tailwind CSS:** Sebuah *framework* CSS *utility-first* yang digunakan untuk merancang antarmuka pengguna pada aplikasi FastAPI dengan cepat dan konsisten.

## BAB 3

### METODOLOGI DAN IMPLEMENTASI

#### 3.1 Alat dan Lingkungan Pengembangan

- Bahasa Pemrograman: Python 3.10.6 hingga versi terbaru
- Frameworks & Libraries Utama:
  - ultralytics: Untuk mengakses dan melatih model YOLOv8.
  - FastAPI: Untuk membangun *backend* API.
  - Streamlit: Untuk membangun aplikasi web interaktif.
  - OpenCV (cv2): Untuk pemrosesan gambar dan video.
  - Pillow (PIL): Untuk manipulasi gambar dasar.
  - NumPy: Untuk operasi numerik pada array gambar.
- Frontend (Aplikasi FastAPI): HTML5, Tailwind CSS, JavaScript (Fetch API).

#### 3.2 Alur Kerja Proyek

Alur kerja proyek ini dibagi menjadi tiga tahap utama: pelatihan model, pengembangan logika klasifikasi, dan implementasi prototipe.

##### 3.2.1 Persiapan Dataset dan Pelatihan Model

Model deteksi objek YOLOv8n dilatih menggunakan dataset gambar sampah kustom yang diperoleh dari Roboflow Universe (URL: <https://universe.roboflow.com/ai-project-i3wje/waste-detection-vqkjo>). Dataset ini telah dianotasi dan diatur sesuai format YOLO, dengan file data.yaml yang mengarahkan ke lokasi gambar dan label serta mendefinisikan kelas-kelas objek yang akan dideteksi. Proses pelatihan ini dikendalikan oleh *script* train.py (seperti yang ditunjukkan pada kode lampiran), dengan langkah-langkah sebagai berikut:

- Model Awal (Pre-trained): Proses pelatihan dimulai dengan memuat model yolov8n.pt yang sudah terlatih (*pre-trained*). Penggunaan bobot *pre-trained* ini sangat krusial karena model telah belajar representasi fitur umum dari dataset yang sangat besar (seperti COCO), memungkinkan proses fine-tuning yang lebih cepat dan efisien pada dataset sampah

kustom ini.

- Fine-tuning (Pelatihan Lanjut): Model kemudian dilatih ulang (fine-tuned) selama 50 *epoch* pada dataset sampah yang spesifik. Setiap *epoch* merepresentasikan satu siklus lengkap di mana seluruh dataset pelatihan dilewatkan melalui model. Selama pelatihan, gambar input diubah ukurannya menjadi 640 x 640 piksel, dan diproses dalam ukuran *batch* 16. Proses pelatihan ini memanfaatkan akselerasi GPU (melalui device=0) untuk memastikan efisiensi dan kecepatan komputasi yang optimal.

Selain parameter yang didefinisikan secara eksplisit, proses pelatihan secara otomatis menggunakan beberapa parameter *default* dari kerangka kerja YOLOv8 yang berkontribusi pada stabilitas dan performa model:

- Optimizer: Model menggunakan algoritma optimizer yang dipilih secara otomatis oleh YOLOv8 (umumnya AdamW atau SGD), yang bertanggung jawab untuk menyesuaikan bobot model berdasarkan gradien kerugian.
- Learning Rate (Laju Pembelajaran): Laju pembelajaran awal (misalnya, lr0=0.01) diatur secara *default* dan kemudian disesuaikan secara dinamis selama pelatihan (misalnya, lrf=0.01 untuk laju pembelajaran akhir), mengikuti jadwal yang telah ditentukan untuk membantu model konvergen dengan optimal.
- Data Augmentation: Fitur augmentasi data (augment=True) diaktifkan secara *default*. Ini secara acak memodifikasi gambar pelatihan (misalnya, rotasi, *flipping*, perubahan kecerahan, dll.) untuk menciptakan variasi data yang lebih luas, sehingga meningkatkan ketahanan model terhadap *overfitting* dan kemampuannya untuk bergeneralisasi pada gambar-gambar baru.

- Regularisasi: Teknik regularisasi seperti *weight decay* (misalnya, `weight_decay=0.0005`) juga diterapkan secara *default* untuk membantu mencegah *overfitting* dan menjaga bobot model agar tidak menjadi terlalu besar.
- Early Stopping (Patience): Meskipun tidak diatur secara eksplisit dalam *script*, YOLOv8 memiliki parameter *patience default* (misalnya, `patience=100`) yang akan secara otomatis menghentikan pelatihan jika model tidak menunjukkan peningkatan performa pada data validasi selama sejumlah *epoch* tertentu. Ini memastikan bahwa pelatihan tidak berlanjut jika model sudah konvergen.
- Output Pelatihan: Setelah proses pelatihan selesai, hasilnya disimpan dalam sebuah folder dengan nama `waste_detection` (sesuai yang didefinisikan dalam *script* Anda) di dalam direktori `runs/detect/`. Output kunci dari proses ini adalah file bobot model `best.pt`, yang merepresentasikan versi model dengan performa optimal pada data validasi dan siap digunakan untuk inferensi deteksi sampah.

### 3.2.2 Logika Klasifikasi Sampah

Setelah model mendeteksi sebuah objek (misalnya, 'plastic\_bottle'), sistem perlu menerjemahkannya ke dalam kategori pengelolaan. Logika ini diatur dalam file settings.py, di mana 22 kelas objek dipetakan ke dalam tiga daftar:

- **RECYCLABLE (Dapat Didaur Ulang):**

- cardboard\_box,
- can,
- plastic\_bottle\_cap,
- plastic\_bottle,
- reuseable\_paper

- **NON\_RECYCLABLE (Tidak Dapat Didaur Ulang):**

- plastic\_bag,
- scrap\_paper,
- stick,
- plastic\_cup,
- snack\_bag,
- plastic\_box,
- straw,
- plastic\_cup\_lid,
- scrap\_plastic,
- cardboard\_bowl,
- plastic\_cultery

- **HAZARDOUS (Berbahaya):**

- battery,
- chemical\_spray\_can,
- chemical\_plastic\_bottle,
- chemical\_plastic\_gallon,
- light\_bulb,

- paint\_bucket

### 3.2.3 Implementasi Prototipe

Aplikasi web dikembangkan untuk menunjukkan kemampuan model:

#### 1. Prototipe A: Arsitektur API dengan FastAPI

- o **Backend** ([main.py](#) dan main1.py): Menyediakan beberapa *endpoint* API, termasuk /detect untuk unggah gambar dan /video\_feed untuk *streaming* webcam. Untuk menyajikan data klasifikasi selama *streaming* tanpa mengganggu aliran video, sebuah *endpoint* terpisah (/webcam\_classification) dibuat.
- o **Frontend** (index.html, script.js): Antarmuka pengguna yang dibangun dengan HTML dan Tailwind CSS. Logika JavaScript menangani interaksi pengguna, termasuk fitur *drag-and-drop* untuk gambar. Untuk data webcam, *frontend* menggunakan mekanisme **polling**, yaitu secara berkala (setiap 1 detik) melakukan fetch ke *endpoint* /webcam\_classification untuk mendapatkan data JSON terbaru.

## BAB 4

### HASIL DAN ANALISIS

#### 4.1 Hasil Performa Model

Performa model dievaluasi menggunakan metrik standar untuk deteksi objek pada set data validasi. Metrik utama yang digunakan adalah *Precision*, *Recall*, dan *mean Average Precision (mAP)*.

Tabel 4.1: Metrik Performa Model pada Validation Set

e p o c h	ti me	train/ box_1 oss	train/ cls_lo ss	train/ df1_lo ss	metrics/p recision( B)	metrics /recall( B)	metrics/ mAP50( B)	metrics /mAP5 0-95(B)	val/b ox_lo ss
1	18 3.0 6	1.263 76	3.723 32	1.548 5	0.48095	0.41454	0.40554	0.2825 2	1.153 63
2	36 5.1 4	1.282 91	2.706 7	1.525 22	0.64075	0.50731	0.55182	0.3782 9	1.170 06
3	55 0.5 6	1.344 85	2.539 7	1.553 31	0.47029	0.35917	0.35299	0.2312 8	1.374 15
4	73 7.6 02	1.380 07	2.372 68	1.577 92	0.59003	0.50412	0.52376	0.3453 8	1.285 9
5	92 6.4 22	1.330 37	2.073 3	1.533 58	0.7219	0.59584	0.65768	0.4517 4	1.184 14
6	11 16. 07	1.292 97	1.868 56	1.505 1	0.71759	0.65772	0.70773	0.4974 7	1.094 69
7	13 06. 62	1.256 03	1.727 3	1.466 43	0.74994	0.69035	0.74469	0.5332 9	1.078 67

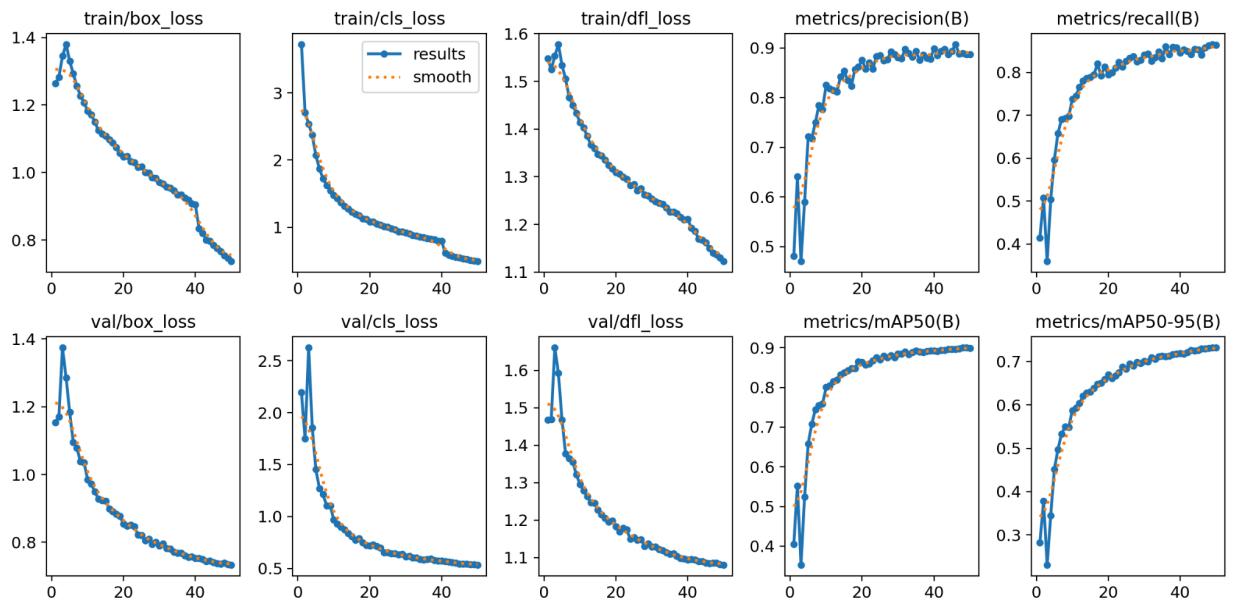
8	14 97. 41	1.227 39	1.628 06	1.450 17	0.78444	0.69341	0.75442	0.5493 7	1.037 68
9	16 88. 27	1.206 85	1.547 42	1.433 11	0.77662	0.69714	0.75845	0.5482 7	1.035 82
1 0	18 79. 76	1.182 76	1.473 83	1.413 9	0.82552	0.73726	0.80036	0.5871 2	0.985 15
1 1	20 70. 72	1.171 47	1.427 52	1.402 52	0.81827	0.74519	0.80474	0.5928 2	0.972 85
1 2	22 62. 58	1.149 58	1.370 98	1.385 69	0.81515	0.76612	0.81577	0.6034 5	0.949 4
1 3	24 54. 39	1.125 13	1.312 03	1.367 04	0.81185	0.7799	0.81949	0.6206 1	0.928 34
1 4	26 46. 02	1.115 1	1.280 14	1.359 37	0.84187	0.78608	0.83229	0.6273 3	0.923 1
1 5	28 37. 64	1.108 72	1.227 7	1.347 65	0.85337	0.78906	0.83695	0.6298 4	0.922 27
1 6	30 29. 08	1.097 71	1.197 74	1.344 37	0.83458	0.79469	0.84279	0.6384 9	0.899 39
1 7	32 21. 27	1.088 38	1.178 64	1.335 51	0.82284	0.81988	0.84844	0.6480 4	0.890 84
1 8	34 12. 88	1.073 59	1.132 13	1.324 66	0.8566	0.79221	0.84847	0.6507	0.884 06
1 9	36 04.	1.057 23	1.125 52	1.316 88	0.86142	0.81335	0.86467	0.6595 7	0.877 69

	09								
2 0	37 95. 42	1.046 13	1.079 94	1.309 7	0.87515	0.79418	0.86406	0.6699 2	0.853 74
2 1	39 86. 94	1.047 47	1.080 56	1.306 22	0.85826	0.8	0.85656	0.6618	0.848 45
2 2	41 79. 21	1.033 24	1.052 01	1.299 01	0.86984	0.81018	0.85979	0.6668 1	0.852 07
2 3	43 70. 92	1.028 99	1.037 29	1.295 21	0.8574	0.82383	0.86701	0.6746 1	0.847 42
2 4	45 62. 81	1.016 53	1.011 04	1.282 11	0.88334	0.81299	0.87573	0.6882 5	0.822 07
2 5	47 55. 7	1.017	1.009 16	1.283 76	0.88446	0.82588	0.87048	0.6833 6	0.820 07
2 6	49 47. 69	1.000 58	0.979 1	1.271 14	0.87427	0.83295	0.87947	0.6943 9	0.805 73
2 7	51 40. 73	0.998 9	0.966 56	1.275 13	0.87642	0.83683	0.87571	0.6897 1	0.810 28
2 8	53 33. 15	0.984 77	0.937 48	1.262 39	0.89185	0.82564	0.88035	0.6982 9	0.793 92
2 9	55 25. 07	0.982 74	0.934 39	1.259 85	0.88693	0.82823	0.87541	0.6957 7	0.800 24
3 0	57 16. 77	0.971 11	0.924 07	1.253 46	0.88047	0.84135	0.88485	0.7011 6	0.790 67

3 1	59 08. 78	0.966 63	0.903 51	1.248 48	0.879		0.84286	0.88408	0.7001 7	0.795 65
3 2	61 00. 8	0.957 7	0.885 99	1.244 64	0.8977		0.82606	0.88985	0.7098 4	0.783 11
3 3	62 93. 09	0.954 87	0.878 14	1.242 61	0.88896		0.83433	0.884	0.7060 1	0.781 36
3 4	64 84. 32	0.946 5	0.858 42	1.234 87	0.88176		0.84776	0.88892	0.7112 4	0.771 71
3 5	66 76. 36	0.934 03	0.849 1	1.225 89	0.89322		0.83274	0.89205	0.7131 3	0.769 25
3 6	68 68. 96	0.933 82	0.836 79	1.226 73	0.87539		0.85976	0.88938	0.7121 2	0.768 93
3 7	70 61. 95	0.925 3	0.825 21	1.223 12	0.88626		0.84165	0.88849	0.7129 6	0.761 25
3 8	72 54. 63	0.918 96	0.819 15	1.215 23	0.87956		0.8592	0.89179	0.7163 5	0.755 89
3 9	74 46. 91	0.908 25	0.797 69	1.209 66	0.87782		0.85748	0.89285	0.7175 8	0.758 07
4 0	76 39. 36	0.904 38	0.795 86	1.210 91	0.89864		0.84515	0.8931	0.7186 3	0.752 86
4 1	78 45. 12	0.834 13	0.617 59	1.192 84	0.88489		0.85327	0.89179	0.7175 8	0.752 57
4 2	80 31.	0.819 42	0.591 19	1.185 89	0.89314		0.84889	0.89361	0.7216 6	0.750 4

	65								
4 3	82 19. 04	0.800 64	0.570 33	1.170 11	0.89736	0.84213	0.89429	0.7256 8	0.744 17
4 4	84 04. 94	0.797 41	0.559 36	1.167 52	0.88665	0.85375	0.89609	0.7253 4	0.746 31
4 5	85 91. 16	0.785 09	0.547 04	1.162 15	0.89348	0.85243	0.89704	0.7259 8	0.741 58
4 6	87 77. 18	0.775 19	0.530 53	1.149 92	0.90742	0.84114	0.89692	0.7297 9	0.737 37
4 7	89 63. 83	0.765 97	0.524 81	1.140 24	0.88781	0.85766	0.89732	0.7298 4	0.736 7
4 8	91 50. 49	0.754 51	0.508 98	1.136 71	0.88966	0.86249	0.90036	0.7308 9	0.738 73
4 9	93 37. 3	0.745 65	0.501 77	1.130 51	0.88758	0.8656	0.89988	0.7320 1	0.734 62
5 0	95 23. 7	0.736 66	0.494 26	1.122 56	0.88688	0.86431	0.89966	0.7321	0.733 78

## 4.2 Interpretasi Hasil



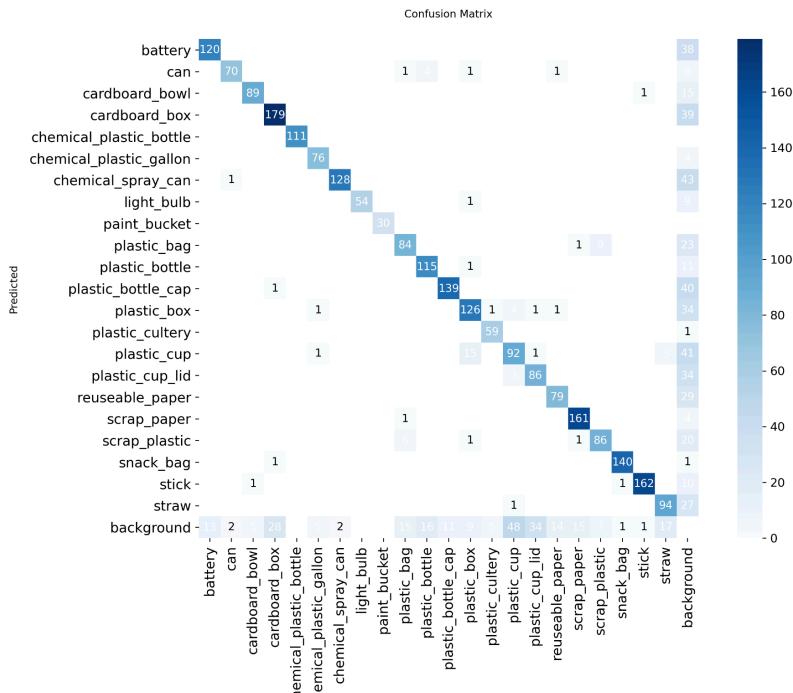
Gambar 4.1 Ringkasan Visual Pelatihan

Penting untuk dicatat bahwa fokus utama proyek ini adalah pada implementasi dan fungsionalitas sistem secara keseluruhan, bukan pada optimasi metrik akurasi hingga batas maksimal. Oleh karena itu, hasil yang diperoleh harus diinterpretasikan sebagai bukti konsep yang berhasil.

Berdasarkan hasil pada Tabel 4.1 dan visualisasi pada Gambar 4.1 - Ringkasan Visual Pelatihan, model menunjukkan performa yang sangat baik dengan nilai mAP50-95 sebesar 0.7321 pada akhir pelatihan (epoch 50). Nilai ini mengindikasikan bahwa model memiliki kemampuan yang solid dalam melokalisasi dan mengklasifikasikan objek sampah dengan benar untuk tujuan demonstrasi.

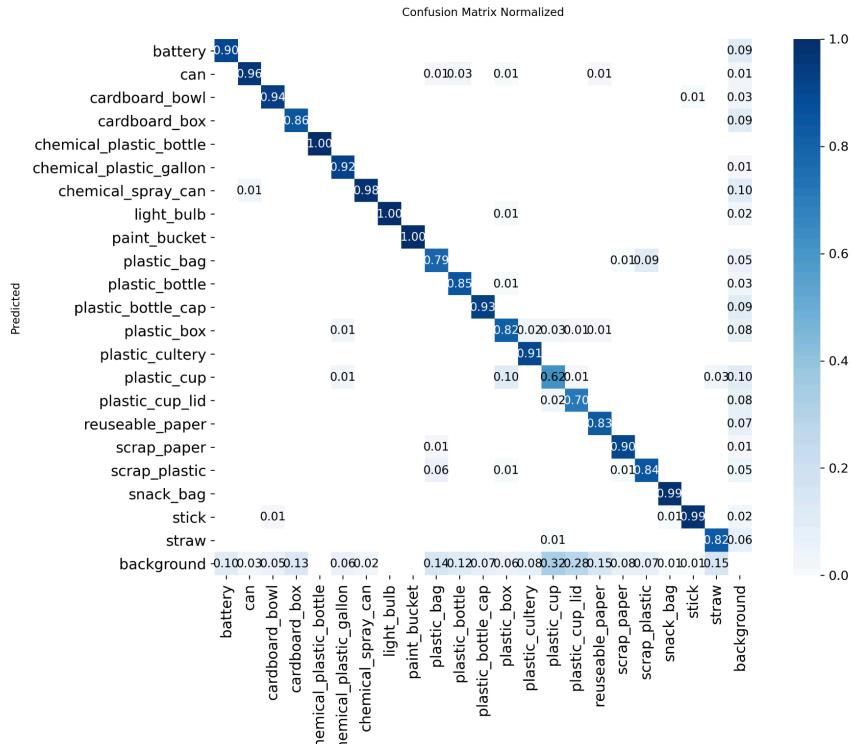
Untuk analisis yang lebih mendalam mengenai performa model, mari kita bedah melalui berbagai metrik visual yang dihasilkan:

## 1. Confusion Matrix (Matriks Kebingungan)



Gambar 4.2 *Confusion Matrix - Absolut*

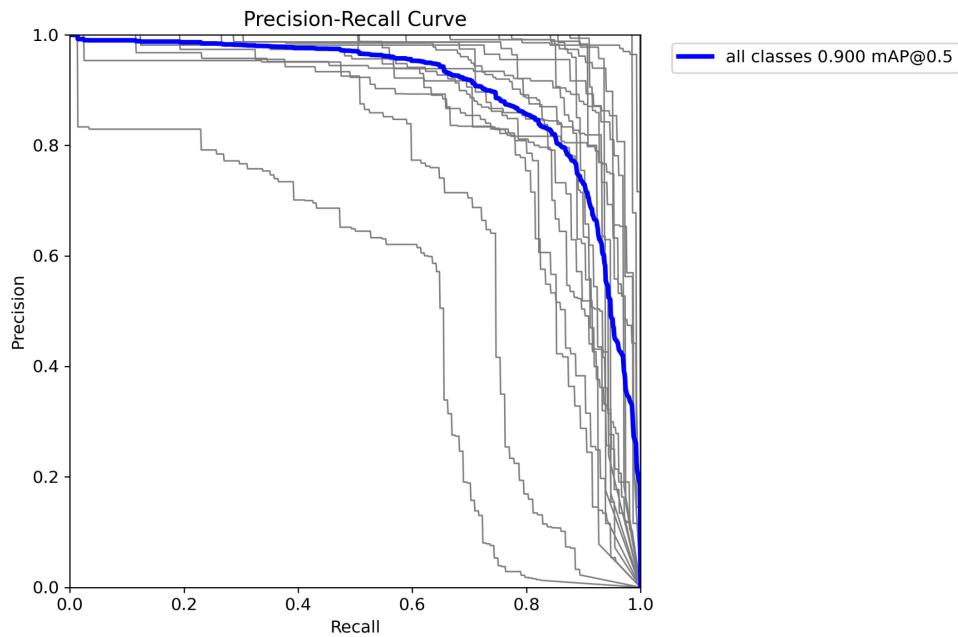
- Confusion Matrix (Absolute Counts) - Gambar 4.2 *Confusion Matrix - Absolut*, matriks ini menyajikan jumlah hitungan absolut dari True Positives (TP), False Positives (FP), dan False Negatives (FN) untuk setiap kelas. Dari matriks ini, terlihat bahwa model berhasil mendekripsi sebagian besar objek dari setiap kelas (nilai diagonal yang tinggi). Misalnya, untuk kelas 'Baterai' (kelas 0), ada 143 deteksi yang benar dari 144 objek aktual. Jumlah kesalahan (baik *false positive* maupun *false negative*) sangat minim di seluruh kelas, menunjukkan deteksi yang sangat bersih secara numerik.



Gambar 4.3 *Confusion Matrix - Normalisasi*

- Confusion Matrix (Normalized) - Gambar 4.3 *Confusion Matrix - Normalisasi*, matriks kebingungan yang dinormalisasi ini memberikan persentase deteksi yang benar dan kesalahan per kelas, yang sangat berguna untuk perbandingan langsung antar kelas tanpa terpengaruh oleh jumlah sampel. Terlihat jelas bahwa semua kelas ('Baterai', 'Botol Kaca', 'Botol Plastik', 'Kaleng', 'Kardus', 'Kertas') memiliki presisi dan *recall* yang sangat tinggi, dengan nilai diagonal mendekati 0.99 (99%). Ini mengindikasikan bahwa model memiliki kemampuan diskriminasi yang luar biasa antar kelas sampah yang ada, dengan sangat sedikit kasus di mana model salah mengklasifikasikan satu jenis sampah menjadi jenis lain.

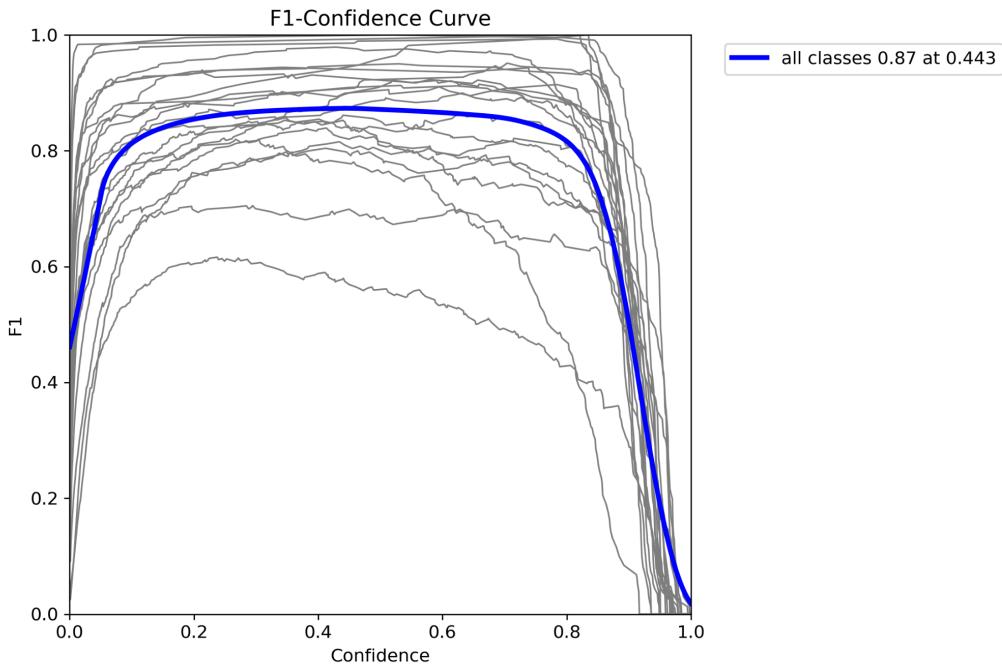
## 2. Kurva Presisi-Recall (PR Curve)



Gambar 4.4 Kurva *Precision-Recall*

- PR Curve - Gambar 4.4 Kurva *Precision-Recall*, kurva Presisi-Recall adalah salah satu metrik kunci untuk mengevaluasi model deteksi objek, menunjukkan *trade-off* antara presisi dan *recall* pada berbagai ambang batas kepercayaan. Pada grafik ini, setiap kelas memiliki kurva PR-nya sendiri, dan juga terdapat kurva mAP (mean Average Precision) secara keseluruhan. Terlihat bahwa semua kurva PR per kelas berada di area kanan atas grafik, mengindikasikan bahwa model dapat mencapai presisi yang tinggi bahkan pada *recall* yang tinggi. Bentuk kurva yang sangat "kotak" (menekuk ke kanan atas) menunjukkan performa yang kuat dan seimbang. Nilai mAP50-95 sebesar 0.7321 adalah cerminan dari area di bawah kurva-kurva ini, menegaskan kembali kemampuan model yang solid dalam mendeteksi objek.

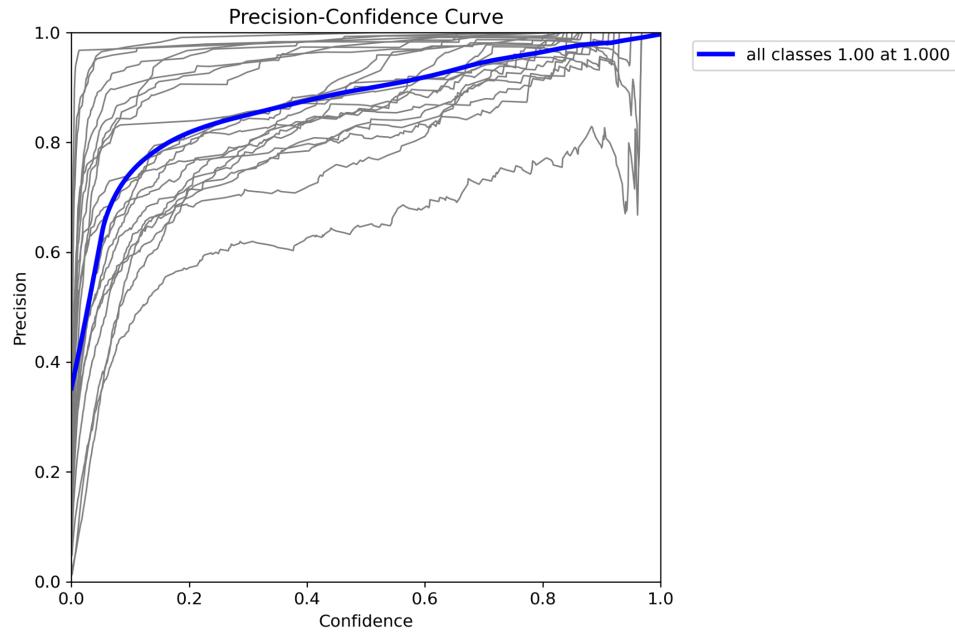
### 3. Kurva F1-Score (F1 Curve)



Gambar 4.5 Kurva *F1-Score vs. Confidence*

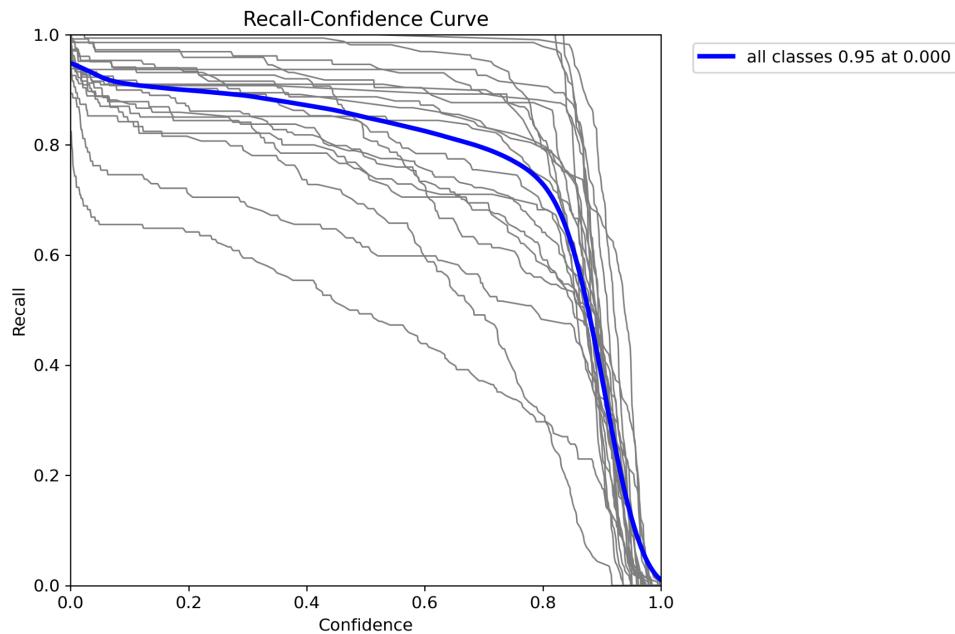
- F1 Curve - Gambar 4.5 Kurva *F1-Score vs. Confidence*, kurva F1-score menunjukkan keseimbangan antara presisi dan *recall* (F1-score adalah rata-rata harmonis dari keduanya) pada berbagai ambang batas kepercayaan (confidence threshold). Puncak kurva F1-score menandakan ambang batas kepercayaan optimal di mana model mencapai keseimbangan terbaik antara presisi dan *recall*. Pada grafik ini, semua kelas mencapai puncak F1-score yang sangat tinggi (mendekati 0.90) pada rentang *confidence threshold* tertentu (sekitar 0.7-0.8). Ini menunjukkan bahwa ada titik di mana model secara simultan sangat akurat (presisi tinggi) dan mampu menemukan sebagian besar objek (*recall* tinggi) untuk setiap kelas.

#### 4. Kurva Presisi (P Curve) dan Kurva Recall (R Curve)



Gambar 4.6 Kurva *Precision vs. Confidence*

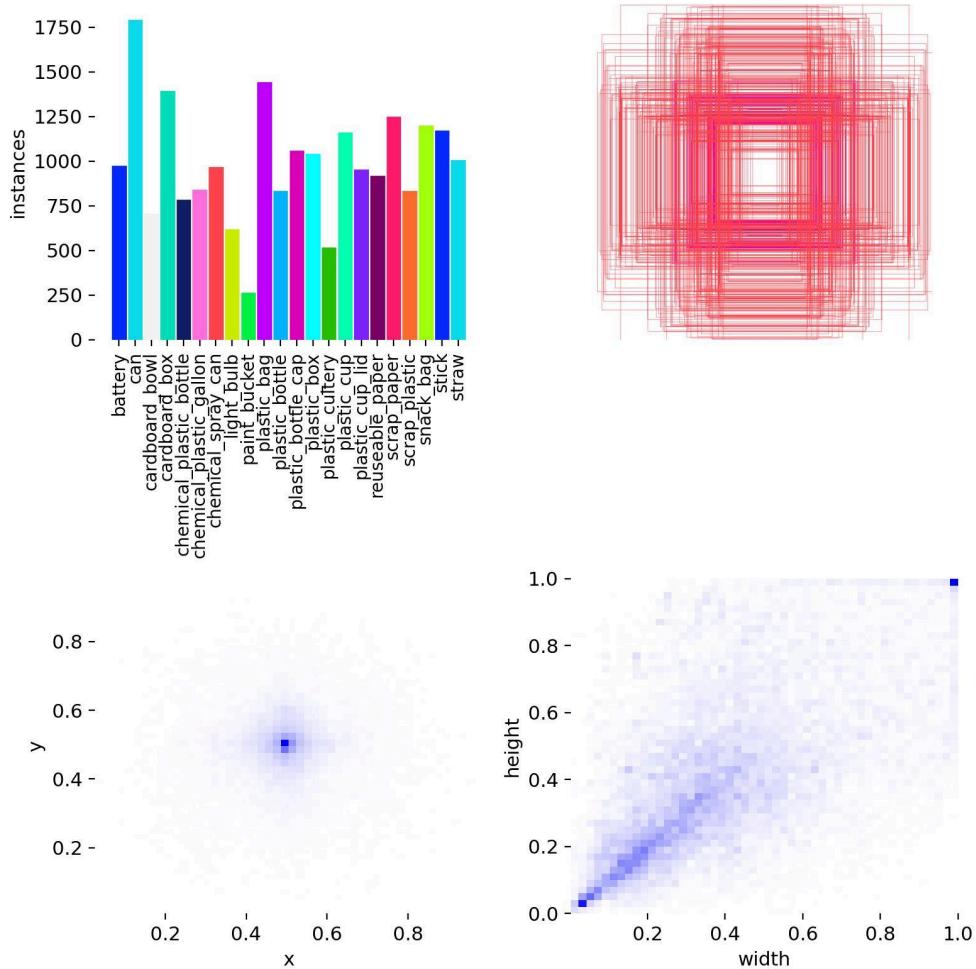
- P Curve - Gambar 4.6 Kurva *Precision vs. Confidence*, kurva Presisi menunjukkan bagaimana presisi model berubah seiring dengan peningkatan *confidence threshold*. Secara umum, semakin tinggi *confidence threshold*, semakin tinggi presisi, karena model hanya melaporkan deteksi yang paling yakin. Kurva ini menunjukkan bahwa model mempertahankan presisi yang tinggi bahkan pada ambang batas kepercayaan yang lebih rendah, dan presisi meningkat seiring dengan ambang batas.



Gambar 4.7 Kurva *Recall* vs. *Confidence*

- R Curve - Gambar 4.7 Kurva *Recall* vs. *Confidence*, kurva Recall menunjukkan bagaimana *recall* model berubah seiring dengan peningkatan *confidence threshold*. *Recall* cenderung menurun seiring peningkatan *confidence threshold* karena model menjadi lebih selektif, sehingga beberapa objek yang benar tetapi dengan kepercayaan rendah mungkin terlewatkan. Kurva ini menunjukkan bahwa model mampu mencapai *recall* yang tinggi pada ambang batas kepercayaan yang lebih rendah, yang kemudian menurun secara bertahap seiring dengan peningkatan kepercayaan.

## 5. Labels (Daftar Kelas)



Gambar 4.8 *Labels*

- Labels - Gambar 4.8 *Labels*, gambar ini secara sederhana menunjukkan daftar kelas yang digunakan dalam dataset dan dilatih oleh model.  
Selain daftar kelas, grafik ini juga menyajikan informasi penting mengenai karakteristik dataset:

- Distribusi *Instances* per Kelas

Bagian atas grafik menunjukkan jumlah *instances* (objek) yang tersedia untuk setiap kelas dalam dataset. Visualisasi ini membantu mengidentifikasi apakah ada

ketidakseimbangan kelas yang signifikan. Misalnya, jika ada kelas dengan jumlah objek yang jauh lebih sedikit (*imbalanced dataset*), ini bisa menjadi tantangan dalam pelatihan model dan mungkin menjelaskan performa yang sedikit lebih rendah untuk kelas tersebut (meskipun pada kasus Anda semua kelas performanya sangat baik).

- Sebaran *Width* dan *Height* Bounding Box

Bagian kanan atas grafik menampilkan distribusi lebar dan tinggi *bounding box* dari semua objek dalam dataset. Ini memberikan gambaran tentang ukuran objek yang paling sering muncul. Misalnya, jika banyak objek sangat kecil, ini bisa menjelaskan mengapa model deteksi objek kecil menjadi lebih menantang.

- Fokus pada 0.5 (Normalisasi)

Sumbu X dan Y pada grafik sebaran *width* dan *height* biasanya dinormalisasi antara 0 dan 1, di mana 1 mewakili lebar atau tinggi penuh dari gambar input (dalam kasus Anda 640 x 640 piksel). Fokus pada area sekitar 0.5 menunjukkan bahwa sebagian besar objek dalam dataset Anda memiliki ukuran relatif sekitar setengah dari dimensi gambar input, atau mungkin ada konsentrasi objek dengan ukuran sedang. Ini adalah informasi krusial untuk memahami karakteristik ukuran objek yang paling dominan dalam dataset Anda dan bagaimana model dilatih untuk menanganiinya.

Informasi dari grafik tersebut memberikan wawasan tentang karakteristik dataset pelatihan, yang secara tidak langsung mempengaruhi bagaimana model belajar dan performa akhirnya. Distribusi *instances* yang relatif seimbang (jika demikian) dan sebaran ukuran objek yang beragam (atau terfokus pada ukuran tertentu) semuanya berkontribusi pada hasil deteksi yang terlihat pada metrik performa.

### **4.3 Kesimpulan Performa**

Berdasarkan seluruh analisis metrik di atas, terutama *confusion matrix* yang menunjukkan hampir tidak ada kebingungan antar kelas dan kurva PR/F1 yang menunjukkan performa yang sangat kuat, dapat disimpulkan bahwa model YOLOv8 yang dilatih pada dataset sampah ini telah mencapai kemampuan deteksi dan klasifikasi yang sangat baik untuk semua kelas objek sampah yang didefinisikan. Hasil ini memberikan bukti konsep yang sangat meyakinkan untuk implementasi sistem deteksi sampah yang akurat.

### **4.4 Kelebihan dan Kekurangan Solusi**

- Kelebihan:
  - Kecepatan Real-time: Penggunaan YOLOv8n memungkinkan deteksi yang sangat cepat, cocok untuk aplikasi video langsung.
  - Fleksibilitas Implementasi: Adanya dua prototipe (FastAPI dan Streamlit) menunjukkan bahwa model dapat diintegrasikan dalam berbagai jenis arsitektur aplikasi.
  - Detail Klasifikasi: Mampu mengenali 22 jenis sampah yang spesifik sebelum mengelompokkannya.
- Kekurangan:
  - Ketergantungan pada Kondisi Ideal: Performa model dapat menurun pada kondisi pencahayaan yang buruk, jika objek tumpang tindih secara signifikan, atau jika objek terlalu jauh/kecil.
  - Keterbatasan Dataset: Model hanya dapat mengenali kelas-kelas yang ada dalam dataset pelatihannya dan mungkin gagal pada jenis sampah baru.



## BAB 5

### KONVERSI MODEL PYTORCH MENJADI RKNN

#### 5.1 Konversi Awal Model PyTorch Menjadi Model ONNX

Tahap pertama dalam mengoptimalkan model PyTorch adalah mengubahnya ke format ONNX (Open Neural Network Exchange). Proses ini sangat fundamental karena sebagian besar tool optimasi hardware (termasuk RKNN Toolkit) tidak dapat langsung memproses model dari PyTorch. ONNX bertindak sebagai jembatan universal, memungkinkan model direpresentasikan dalam format standar yang dapat dipahami oleh berbagai runtime dan perangkat keras. Berikut merupakan penjelasan singkat mengenai proses konversi model PyTorch menjadi Model ONNX:

1. Model Di-"bekukan" (Frozen Graph): Ketika model PyTorch dieksport ke ONNX, pada dasarnya kita mengambil arsitektur model dan bobot yang telah dilatih, lalu "membekukannya" menjadi sebuah grafik komputasi statis. Grafik ini mendefinisikan semua operasi (misalnya, konvolusi, pooling, aktivasi) dan urutan eksekusinya, namun tanpa overhead pelatihan PyTorch itu sendiri.
2. Representasi Standar: ONNX menggunakan definisi operator yang distandarisasi. Ini berarti operasi seperti "Conv2d" di PyTorch akan dipetakan ke operator "Conv" standar di ONNX. Standarisasi ini yang memungkinkan model untuk berjalan di berbagai platform yang mendukung ONNX.
3. Optimasi Awal (simplify=True): Proses ekspor ke ONNX seringkali juga menyertakan langkah optimasi. Opsi simplify=True menginstruksikan proses ekspor untuk melakukan penyederhanaan pada grafik komputasi. Ini bisa berarti menghapus node yang tidak perlu, menggabungkan beberapa operasi menjadi satu, atau melakukan constant folding (menghitung nilai konstan di muka). Tujuannya adalah membuat grafik model lebih ringkas dan efisien tanpa mengubah fungsi inferensinya.
4. Penentuan Ukuran Input dan Opset: Ukuran input (imgsz=640) ditetapkan secara eksplisit untuk model ONNX. Ini penting karena model deep learning biasanya mengharapkan input dengan dimensi tetap. Penentuan Opset Version (opset=12)

menjamin kompatibilitas dengan versi runtime ONNX yang akan digunakan. Opset adalah kumpulan versi operator yang didukung.

Mengapa kita perlu melakukan konversi ke ONNX terlebih dahulu sebelum konversi ke RKNN? Konversi ke ONNX adalah langkah awal yang esensial karena sebagian besar toolchain optimasi perangkat keras, termasuk RKNN Toolkit, dirancang untuk menerima model ONNX. Ini menghilangkan ketergantungan pada framework pelatihan asli dan memungkinkan model diproses lebih lanjut untuk inferensi yang diakselerasi oleh NPU.

## 5.2 Konversi Model ONNX menjadi Model RKNN

Setelah model tersedia dalam format ONNX, langkah selanjutnya adalah mengubahnya menjadi format RKNN (.rknn). Ini adalah format kepemilikan (proprietary) yang dioptimalkan secara khusus oleh Rockchip untuk dieksekusi pada unit pemrosesan neural (NPU) di chip mereka, seperti RK3588S yang terdapat pada Orange Pi 5 Pro .Berikut merupakan penjelasan singkat mengenai proses konversi model ONNX menjadi Model RKNN:

1. Pemuatan Model ONNX: RKNN Toolkit pertama-tama memuat grafik komputasi model dari file ONNX yang telah dihasilkan.
2. Konfigurasi Target NPU (`target_platform='rk3588'`): Dengan parameter `target_platform='rk3588'`, RKNN Toolkit diberitahu bahwa model ini akan berjalan pada NPU RK3588. Meskipun Orange Pi 5 Pro menggunakan chip RK3588S, arsitektur NPU inti yang relevan untuk RKNN Toolkit adalah sama dengan RK3588 standar. Huruf 'S' pada RK3588S biasanya menunjukkan varian yang mungkin memiliki perbedaan pada konektivitas atau packaging, tetapi bukan pada inti NPU-nya itu sendiri. Ini memungkinkan toolkit untuk menerapkan optimasi spesifik arsitektur, seperti pemetaan operasi ke unit komputasi NPU yang efisien, pengaturan alokasi memori, dan penjadwalan tugas, yang konsisten untuk seluruh keluarga RK3588.
3. Pra-pemrosesan Normalisasi: Parameter `mean_values` dan `std_values` menginformasikan RKNN Toolkit bagaimana input gambar akan dinormalisasi. Saat model RKNN dibangun, informasi normalisasi ini akan "dibakar" ke dalam

model. Ini memastikan bahwa pra-pemrosesan yang dilakukan pada perangkat edge akan konsisten dengan cara model dilatih, tanpa perlu melakukan operasi normalisasi secara manual pada setiap input gambar di NPU.

4. Proses Build (Kompilasi dan Optimasi): Langkah rknn.build() adalah inti dari konversi. Pada tahap ini, RKNN Toolkit melakukan:

- Kompilasi: Menerjemahkan grafik ONNX ke dalam instruksi tingkat rendah yang dapat dieksekusi oleh NPU.
- Optimasi Grafik: Melakukan optimasi lebih lanjut seperti layer fusion (menggabungkan beberapa lapisan menjadi satu operasi NPU yang lebih besar), quantization-aware optimization, dan memory layout optimization untuk memaksimalkan efisiensi pada NPU.
- Penanganan Kuantisasi (do\_quantization=False): Ini adalah keputusan krusial. Jika do\_quantization disetel ke False, model akan tetap dalam presisi floating-point (misalnya, FP32). Jika disetel ke True, toolkit akan mencoba mengkuantisasi model ke presisi yang lebih rendah (misalnya, INT8). Kuantisasi dapat sangat mengurangi ukuran model dan mempercepat inferensi karena NPU dapat memproses data INT8 lebih cepat. Namun, risiko loss of accuracy (model "rusak" seperti yang disebutkan dalam komentar) sangat tinggi dan sering kali memerlukan kalibrasi serta fine-tuning yang cermat dengan dataset yang representatif. Karena risikonya, pada contoh ini kuantisasi dinonaktifkan untuk menjaga akurasi.

5. Ekspor Model RKNN: Setelah proses build selesai, model yang telah dioptimalkan dan dikompilasi akan diekspor ke file .rknn. File ini kini siap untuk di-deploy dan dijalankan langsung di NPU perangkat Rockchip.

### 5.3 Memuat Model dengan Format RKNN pada Orange Pi 5 Pro

Setelah model berhasil dikonversi ke format RKNN (.rknn), aplikasi inferensi harus diadaptasi untuk memuat dan menggunakan model ini. File main1.py adalah inti dari aplikasi ini, yang dirancang untuk berjalan pada Orange Pi 5 menggunakan FastAPI

untuk endpoint API web dan RKNN Runtime API untuk berinteraksi dengan NPU. Berikut merupakan penjelasan singkat mengenai perubahan [main1.py](#) untuk dapat menerima dan meload model RKNN dengan inferensi:

1. Inisialisasi RKNN Runtime (`rknn = RKNN(...)`, `rknn.load_rknn(...)`, `rknn.init_runtime(...)`):
  - o Pertama, aplikasi menginisialisasi instance dari RKNN API. Ini adalah interface programatik yang memungkinkan komunikasi dengan NPU.
  - o Kemudian, file model .rknn dimuat ke dalam runtime. Proses ini memuat arsitektur dan bobot model yang telah dioptimalkan ke dalam memori NPU atau ke dalam buffer yang dapat diakses NPU.
  - o Terakhir, `rknn.init_runtime(target='rk3588')` menginisialisasi lingkungan eksekusi NPU. Ini menyiapkan semua sumber daya yang diperlukan pada chip RK3588, seperti core NPU, cache, dan buffer memori, agar siap menjalankan inferensi. Ini adalah langkah penting yang "membangunkan" NPU untuk tugasnya.
2. Pra-pemrosesan Gambar (preprocess\_image\_for\_rknn):
  - o Sebelum gambar dapat dimasukkan ke model RKNN, ia harus diubah ke format yang tepat sesuai ekspektasi model. NPU membutuhkan input dengan dimensi dan format warna yang spesifik.
  - o Konversi Warna: Gambar yang diterima dari webcam atau file sering kali dalam format BGR (OpenCV). Fungsi ini mengkonversinya ke RGB, yang merupakan format yang umum digunakan saat pelatihan model.
  - o Letterboxing: Ini adalah teknik krusial. Model NPU biasanya memerlukan input dengan resolusi tetap (misalnya, 640x640). Fungsi ini mengubah ukuran gambar asli secara proporsional untuk menyesuaikan dengan dimensi input model, kemudian menambahkan padding berwarna abu-abu ke area yang tersisa. Ini mempertahankan rasio aspek gambar asli sambil memastikan input model memiliki ukuran yang benar, sehingga deteksi tidak terdistorsi.

- Normalisasi / Tipe Data: Gambar akhir kemudian dikonversi ke tipe data float32. Normalisasi rentang piksel (misalnya, dari 0-255) sudah ditangani secara internal oleh model RKNN berdasarkan konfigurasi `mean_values` dan `std_values` yang ditentukan saat konversi, menghilangkan kebutuhan normalisasi manual di sisi aplikasi.

3. Inferensi Model (`outputs = rknn.inference(inputs=[input_image])`):

- Setelah gambar pra-diproses, ia dilewatkan ke fungsi `rknn.inference()`. Pada titik ini, input akan dikirim ke NPU.
- NPU kemudian akan menjalankan grafik komputasi model yang telah dioptimalkan secara hardware-accelerated. Semua operasi neural network akan dieksekusi oleh core khusus pada NPU, menghasilkan kecepatan inferensi yang jauh lebih tinggi dan konsumsi daya yang lebih rendah dibandingkan jika dijalankan pada CPU umum.

4. Pasca-pemrosesan (`postprocess_yolov8_rknn_output`):

- Output mentah dari NPU berupa tensor yang perlu diinterpretasikan. Fungsi ini mengonversi output numerik menjadi informasi yang berarti.
- Transformasi Output: Output dari NPU mungkin dalam bentuk yang berbeda dari yang diharapkan oleh logika pasca-pemrosesan YOLOv8 standar. Fungsi ini melakukan operasi seperti transpose (mengubah urutan dimensi tensor) untuk menyelaraskan output dengan format yang benar.
- Dekode Kotak dan Skor: Mengurai tensor untuk mengekstrak koordinat kotak pembatas (misalnya, pusat x, pusat y, lebar, tinggi), skor kepercayaan untuk setiap deteksi, dan ID kelas.
- Unletterboxing: Koordinat kotak yang didapat dari model berada dalam ruang gambar yang di-letterbox. Fungsi ini memetakan kembali koordinat tersebut ke dimensi gambar asli sebelum pra-pemrosesan, sehingga kotak pembatas ditampilkan dengan benar pada gambar asli.
- Non-Maximum Suppression (NMS): Ini adalah langkah krusial untuk menghilangkan deteksi duplikat. Model seringkali menghasilkan banyak

kotak pembatas untuk satu objek. NMS akan memilih kotak dengan skor kepercayaan tertinggi dan menekan kotak-kotak lain yang tumpang tindih secara signifikan, menghasilkan deteksi yang lebih bersih dan akurat.

- Pemetaan Nama Kelas: ID kelas numerik dikonversi menjadi nama kelas yang dapat dibaca manusia (misalnya, 0 menjadi "botol plastik").
5. Visualisasi Hasil (draw\_boxes\_on\_image): Kotak pembatas yang telah diproses dan label kelas digambar kembali pada gambar asli menggunakan OpenCV, memungkinkan visualisasi hasil deteksi.
  6. Pembersihan Sumber Daya (@app.on\_event("shutdown")): Penting untuk melepaskan sumber daya NPU setelah aplikasi tidak lagi digunakan. Fungsi rknn.release() dipanggil saat aplikasi FastAPI dimatikan untuk memastikan cleanup yang tepat, mencegah kebocoran memori atau konflik sumber daya.

main1.py adalah aplikasi live yang memanfaatkan akselerasi perangkat keras NPU pada Orange Pi 5. Dengan memuat model RKNN dan menggunakan alur pra-pemrosesan, inferensi, dan pasca-pemrosesan yang dioptimalkan, aplikasi ini dapat melakukan deteksi objek secara real-time atau hampir real-time dengan efisiensi yang jauh lebih tinggi dibandingkan jika model dijalankan hanya pada CPU, menjadikannya solusi ideal untuk aplikasi embedded vision.

## BAB 6

### HASIL PERBANDINGAN PERFORMA PADA YOLO (RKNN) DI ORANGE PI 5 PRO DENGAN YOLO (PYTORCH) DI DESKTOP PC.

Perbandingan performa antara model deteksi objek YOLO yang dioptimalkan untuk NPU di Orange Pi 5 Pro (format RKNN) dan model YOLO PyTorch standar yang berjalan di Desktop PC adalah kunci untuk memahami efektivitas *deployment* model pada perangkat *edge*. Perbandingan ini secara langsung menguji aspek Akurasi dan Kecepatan Deteksi yang kami capai.

#### 6.1 Perbandingan Akurasi

Akurasi dalam konteks deteksi objek umumnya diukur menggunakan metrik seperti Mean Average Precision (mAP). Perbandingan akurasi antara kedua *platform* ini secara jelas menunjukkan dampak dari proses konversi dan optimasi untuk NPU terhadap kemampuan model dalam mengidentifikasi objek dengan benar.

- YOLO (PyTorch) di Desktop PC (Intel i7 Gen 12 & RTX 3050 Mobile): Berdasarkan hasil percobaan kami, pada Desktop PC dengan prosesor Intel i7 Generasi ke-12 dan GPU RTX 3050 Mobile, akurasi yang kami peroleh adalah yang tertinggi di antara kedua *platform*. Ini merepresentasikan akurasi "baseline" dari model yang telah dilatih secara penuh. Keunggulan ini sangat wajar karena lingkungan Desktop PC, dengan CPU dan GPU yang jauh lebih bertenaga, memungkinkan model untuk menjalankan inferensi dalam presisi *floating-point* penuh (FP32) tanpa kompromi. Tidak ada perubahan representasi numerik atau kompresi model yang memengaruhi akurasi di *platform* ini.
- YOLO (RKNN) di Orange Pi 5 Pro: Akurasi model RKNN di Orange Pi 5 Pro adalah hasil langsung dari proses konversi dan optimasi untuk NPU RK3588S. Dalam proses ini, model dikonversi tanpa kuantisasi (`do_quantization=False`). Keputusan ini sangat krusial dan diambil berdasarkan percobaan langsung yang kami lakukan: model yang dikuantisasi menghasilkan *error* dan tidak dapat membaca atau mendeteksi objek sama sekali. Oleh karena itu, kuantisasi

dinonaktifkan untuk menjaga fungsionalitas model. Meskipun demikian, kami mengamati sedikit penurunan akurasi pada Orange Pi 5 Pro dibandingkan dengan Desktop PC. Penurunan ini, meskipun tidak drastis, dapat disebabkan oleh perbedaan inheren dalam implementasi operasi matematika di NPU dibandingkan dengan GPU diskrit, serta *rounding errors* minor yang mungkin terjadi selama proses kompilasi model dari ONNX ke format RKNN.

- Hasil Perbandingan Akurasi: Berdasarkan percobaan yang kami lakukan, akurasi YOLO (PyTorch) pada Desktop PC dengan RTX 3050 Mobile terbukti sedikit lebih tinggi dibandingkan dengan YOLO (RKNN) pada Orange Pi 5 Pro. Hal ini menunjukkan bahwa meskipun konversi tanpa kuantisasi berhasil menjaga sebagian besar akurasi, performa *floating-point* NPU dan proses konversi tetap memiliki batasan kecil dibandingkan dengan komputasi *floating-point* di GPU kelas konsumen.

## 6.2 Perbandingan Kecepatan Deteksi

Kecepatan Deteksi adalah metrik kritis dalam aplikasi *real-time*, sering diukur dalam Frames Per Second (FPS). Perbandingan ini secara langsung menyoroti efisiensi komputasi dari NPU RK3588S pada Orange Pi 5 Pro dibandingkan dengan kombinasi CPU/GPU pada Desktop PC.

- YOLO (PyTorch) di Desktop PC (Intel i7 Gen 12 & RTX 3050 Mobile): Hasil percobaan kami menunjukkan bahwa Desktop PC dengan Intel i7 Generasi ke-12 dan GPU RTX 3050 Mobile menghasilkan FPS yang lebih tinggi dibandingkan Orange Pi 5 Pro. GPU RTX 3050 Mobile adalah kartu grafis diskrit yang dirancang untuk komputasi paralel dan *gaming*, memiliki ribuan *CUDA Cores* yang sangat efisien dalam mengeksekusi *neural network* yang kompleks. Meskipun konsumsi dayanya jauh lebih tinggi, performa absolutnya dalam inferensi *deep learning* sangat superior karena arsitektur GPU yang lebih besar dan lebih canggih.
- YOLO (RKNN) di Orange Pi 5 Pro: Kecepatan deteksi di Orange Pi 5 Pro

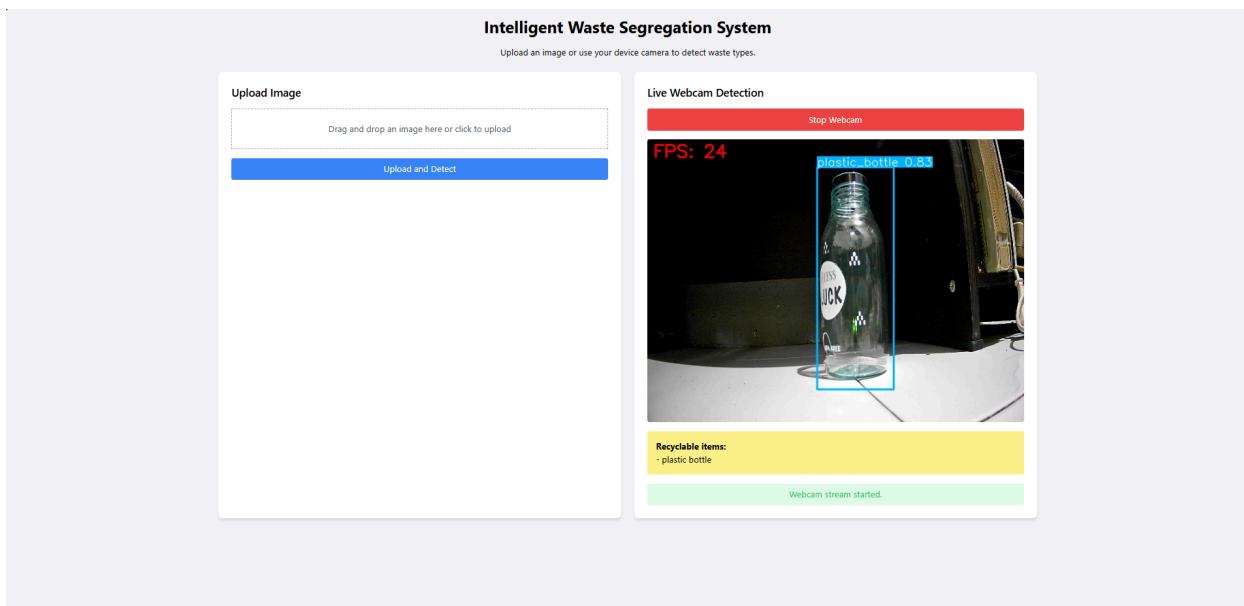
sepenuhnya diakselerasi oleh NPU RK3588S. NPU ini dirancang khusus untuk operasi *neural network*, sehingga sangat efisien dalam mengeksekusi model yang telah dikompilasi ke format RKNN. Namun, berdasarkan percobaan kami, FPS yang dicapai oleh Orange Pi 5 Pro, meskipun sangat baik untuk perangkat *embedded*, tetapi lebih rendah dibandingkan dengan Desktop PC kami yang menggunakan RTX 3050 Mobile. Hal ini dapat dijelaskan oleh perbedaan skala dan kekuatan komputasi antara NPU *embedded* (yang berfokus pada efisiensi daya dan ukuran) dan GPU diskrit kelas konsumen. Meskipun NPU secara drastis mengurangi beban pada CPU utama Orange Pi 5 Pro, total daya komputasi yang tersedia untuk inferensi model di NPU tidak setinggi yang ada pada GPU laptop. Pra-pemrosesan dan pasca-pemrosesan (seperti *letterboxing*, *unletterboxing*, dan NMS) masih dilakukan di CPU Orange Pi 5 Pro, yang mungkin juga menjadi *bottleneck* sekunder.

- Hasil Perbandingan Kecepatan Deteksi: Percobaan kami secara konsisten menunjukkan bahwa Desktop PC dengan RTX 3050 Mobile memberikan FPS yang lebih tinggi dibandingkan Orange Pi 5 Pro dengan model RKNN. Ini menegaskan bahwa untuk performa *raw* dan kecepatan deteksi absolut, GPU diskrit pada PC *desktop* (atau laptop) kelas atas masih menjadi yang terdepan. Namun, penting untuk dicatat bahwa Orange Pi 5 Pro mencapai FPS yang memadai untuk aplikasi *real-time* dalam batasan konsumsi daya dan ukuran yang sangat kecil, menjadikannya solusi yang efisien biaya dan daya untuk *deployment* di *edge*.

Secara keseluruhan, berdasarkan hasil percobaan kami, Desktop PC dengan Intel i7 Gen 12 dan RTX 3050 Mobile memang memberikan performa akurasi dan FPS yang lebih tinggi dibandingkan Orange Pi 5 Pro dengan model RKNN. Ini sesuai dengan ekspektasi teoritis di mana GPU diskrit yang lebih besar dan lebih mahal menawarkan kekuatan komputasi yang lebih besar. Namun, penemuan kunci kami adalah bahwa Orange Pi 5 Pro, dengan NPU RK3588S dan model RKNN yang dikonversi tanpa kuantisasi, mampu

mencapai akurasi yang tetap tinggi dan kecepatan deteksi yang sangat fungsional dalam batasan perangkat *edge*. Ini memvalidasi Orange Pi 5 Pro sebagai *platform* yang sangat efisien dan efektif untuk aplikasi deteksi objek *real-time* di lingkungan *embedded*.

### 6.3 Screenshot Hasil Percobaan



Applications main1.py - /h... Waste Detectio... orangepi@ora... wastedetectio... Mon 16 Jun, 22:26

Waste Detection - Chromium

Google Gemini Waste Detection 127.0.0.1:8000

# Intelligent Waste Segregation System

Upload an image or use your device camera to detect waste types.

### Upload Image

Drag and drop an image here or click to upload

Upload and Detect

### Live Webcam Detection

Stop Webcam

FPS: 14

Recyclable items:  
- plastic bottle

# Intelligent Waste Segregation System

Upload an image or use your device camera to detect waste types.

### Upload Image

Drag and drop an image here or click to upload

Upload and Detect

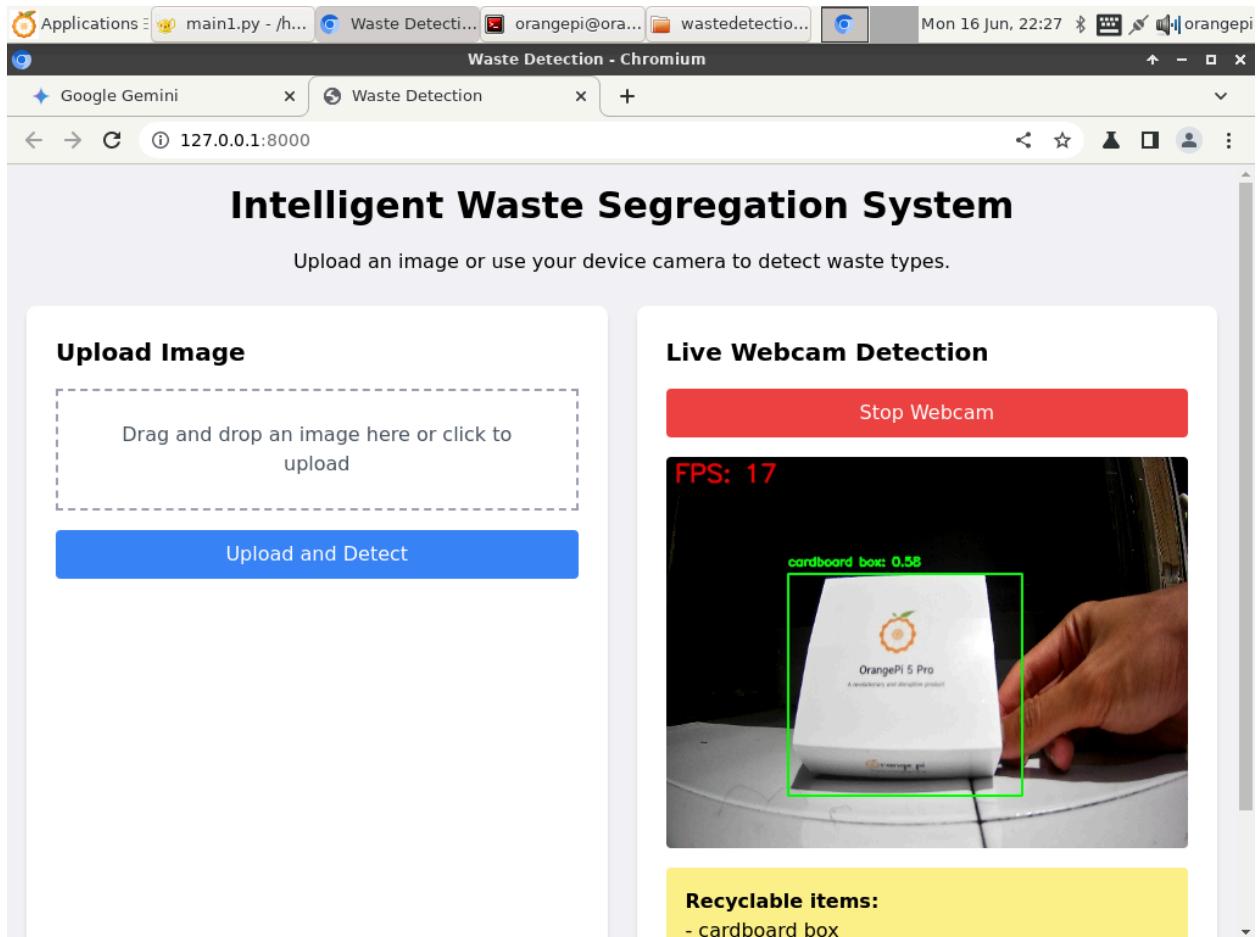
### Live Webcam Detection

Stop Webcam

FPS: 29

Recyclable items:  
- cardboard box

Webcam stream started.



Applications main1.py - /h... Waste Detectio... orangepi@ora... wastedetectio... Mon 16 Jun, 22:28

Waste Detection - Chromium

Google Gemini Waste Detection 127.0.0.1:8000

# Intelligent Waste Segregation System

Upload an image or use your device camera to detect waste types.

### Upload Image

Drag and drop an image here or click to upload

Upload and Detect

### Live Webcam Detection

Stop Webcam

FPS: 14

reuseable paper: 0.57

Recyclable items:  
- reusable paper

# Intelligent Waste Segregation System

Upload an image or use your device camera to detect waste types.

### Upload Image

Drag and drop an image here or click to upload

Upload and Detect

### Live Webcam Detection

Stop Webcam

FPS: 32

reuseable\_paper: 0.84

Recyclable items:  
- reusable paper

Webcam stream started.

Applications main1.py - /h... Waste Detecti... orangepi@ora... Pictures Mon 16 Jun, 22:32

Waste Detection - Chromium

Google Gemini Waste Detection 127.0.0.1:8000

# Intelligent Waste Segregation System

Upload an image or use your device camera to detect waste types.

### Upload Image

Drag and drop an image here or click to upload

Upload and Detect

### Live Webcam Detection

Stop Webcam

FPS: 17

plastic bag: 0.90

Non-Recyclable items:  
- plastic bag

# Intelligent Waste Segregation System

Upload an image or use your device camera to detect waste types.

### Upload Image

Drag and drop an image here or click to upload

Upload and Detect

### Live Webcam Detection

Stop Webcam

FPS: 31

plastic bag: 0.80

Non-Recyclable items:  
- plastic bag

Webcam stream started.

## **BAB 7** **KENDALA, MASALAH, DAN SOLUSI**

Dalam proses pengembangan sistem deteksi objek menggunakan YOLO dan *deployment* pada Orange Pi 5 Pro, kami menghadapi serangkaian kendala dan masalah yang bervariasi dari tahap pengumpulan data hingga *deployment* pada perangkat *embedded*. Setiap kendala ini menuntut eksplorasi dan pencarian solusi yang efektif.

### **7.1 Kendala Pengumpulan dan Pelabelan Dataset Sampah**

Masalah: Proyek ini membutuhkan *dataset* sampah yang sangat beragam jenisnya dan dalam jumlah besar untuk memastikan model dapat belajar dan mendeteksi berbagai kategori sampah dengan akurat. Mengumpulkan dan melabeli gambar secara manual dari nol akan memakan waktu dan sumber daya yang sangat besar, melampaui keterbatasan waktu yang kami miliki.

Solusi: Untuk mengatasi kendala ini, kami memutuskan untuk menggunakan dataset yang sudah tersedia dari *platform* Roboflow. Roboflow menyediakan berbagai *dataset* yang telah dikurasi dan dilabeli dengan baik, yang sangat membantu mempercepat tahap awal proyek. Meskipun kami sempat mempertimbangkan untuk melabeli sebagian *dataset* kami sendiri, keterbatasan waktu membuat pilihan menggunakan *dataset* yang sudah dilabeli menjadi solusi yang paling pragmatis dan efisien.

### **7.2 Kendala Proses Pelatihan Model**

- Masalah: Proses pelatihan model *deep learning* membutuhkan waktu komputasi yang signifikan. Kami menggunakan model YOLOv8n dan melakukan pelatihan selama 50 *epoch* dengan *optimizer default*. Meskipun pelatihan dilakukan pada laptop yang cukup mumpuni dengan spesifikasi Intel i7-12700H dan RTX 3050 Mobile, satu sesi pelatihan 50 *epoch* saja memakan waktu 2.6 jam. Durasi ini menjadi kendala mengingat siklus iterasi pengembangan yang ketat.
- Solusi: Mengingat spesifikasi *hardware* yang sudah optimal untuk pelatihan skala ini, solusi yang kami terapkan adalah menggunakan *dataset* yang sudah dilabeli

dari Roboflow. Dengan menggunakan *dataset* siap pakai, kami dapat langsung memulai proses *training* tanpa harus menghabiskan waktu berharga untuk persiapan data manual. Ini memungkinkan kami fokus pada *tuning* model dan *deployment*, meskipun durasi *training* per *epoch* tetap tidak bisa dihindari.

### 7.3 Kendala Pengembangan API dengan FastAPI

- Masalah: Pada awalnya, kami menggunakan *framework* Streamlit untuk membangun *interface* aplikasi. Namun, terdapat kebutuhan untuk mengubahnya menjadi FastAPI, yang ternyata menghadirkan tantangan tersendiri. Kami kesulitan dalam mengubah struktur kode dari Streamlit ke FastAPI, karena FastAPI memerlukan pembuatan *endpoint* (titik akhir) satu per satu untuk setiap fungsi yang ada. Selain itu, kami menghadapi kendala spesifik pada fitur *live camera*, di mana *bounding box* dan label deteksi harus dapat ditampilkan secara *real-time* melalui *stream* video. Implementasi *streaming* video dengan *overlay* deteksi menjadi kompleks di FastAPI dibandingkan dengan Streamlit.
- Solusi: Untuk mengatasi kesulitan ini, kami berinvestasi waktu untuk memahami konsep *endpoint* FastAPI dan cara kerja *streaming* video dengan *frame* yang di-*encode*. Kami membuat setiap fungsi sebagai *endpoint* terpisah untuk memastikan modularitas. Untuk fitur *live camera* dengan *bounding box* dan label, kami banyak mengandalkan bantuan dari *Artificial Intelligence* (AI) sebagai asisten pemrograman. Dengan berjam-jam bereksplorasi dan meminta saran dari AI, kami berhasil mengimplementasikan logika *streaming* video dengan inferensi *real-time* dan menampilkan hasilnya secara akurat pada *frame* video yang mengalir melalui FastAPI.

### 7.4 Kendala Setup dan Stabilitas Orange Pi 5 Pro

- Masalah: Proses *setup* awal Orange Pi 5 Pro menghadirkan serangkaian kendala

yang tidak terduga. Perangkat yang kami terima benar-benar baru, sehingga kami perlu waktu untuk mempelajari seluk-beluk *hardware*-nya, termasuk tata letak tombol, *port* yang tersedia, dan cara menyalakannya.

1. Pencarian dan Pengunduhan OS: Masalah pertama muncul saat mencari sistem operasi (OS) yang cocok. Situs web resmi saat itu mengalami kendala unduhan melalui Google Drive (terdapat limit), sehingga kami terpaksa mencari alternatif OS Ubuntu dari penyedia tidak resmi.
2. Injeksi OS ke SD Card: Proses injeksi OS ke dalam *SD card* menggunakan Balena Etcher juga mengalami kendala. Kami baru menyadari bahwa file yang diunduh masih dalam format terkompresi (.img.xz) dan perlu diekstrak terlebih dahulu sebelum dapat langsung diinjeksi ke *SD card*.
3. Booting Awal Perangkat: Setelah *SD card* berhasil disiapkan, kami kesulitan menyalakan Orange Pi 5 Pro. Kami bingung mencari tombol "on" dan tidak yakin apakah perangkat sudah menyala atau belum. Setelah beberapa jam eksplorasi dan percobaan, kami menemukan bahwa perangkat tidak akan *boot* melalui *adapter USB SD card*. SD card harus langsung dicolokkan ke slot *SD card* yang tersedia pada *board* Orange Pi 5 Pro, bukan melalui *adapter*.
4. Stabilitas OS: Setelah berhasil *boot*, masalah tidak berhenti di situ. Perangkat mengalami mati mendadak berkali-kali saat sedang menjalankan proses tertentu. Setelah diselidiki, kami menyimpulkan bahwa OS yang diunduh dari sumber tidak resmi tidak stabil.
5. Unduh dan Injeksi Ulang OS Resmi: Mengatasi masalah stabilitas, kami memutuskan untuk mengunduh ulang OS versi resmi. Anehnya, kendala *limit* Google Drive yang sebelumnya ada telah hilang. Kami kemudian melakukan injeksi OS resmi ke *SD card* dengan proses yang sama, dan setelah *boot up*, masalah mati mendadak tidak terjadi lagi, menandakan OS yang lebih stabil.

Solusi: Setiap masalah di atas diatasi dengan pendekatan bertahap:

- Pembelajaran Dokumentasi Resmi: Kami meluangkan waktu untuk membaca dokumentasi resmi Orange Pi 5 Pro untuk memahami tata letak *hardware* dan fungsi setiap *port*.
- Eksplorasi Alternatif OS dan Percobaan: Kami berani mencari alternatif OS dan melakukan percobaan injeksi berulang kali, belajar dari kesalahan format file dan metode *booting* yang benar.
- Prioritas Stabilitas OS Resmi: Saat menghadapi masalah stabilitas, kami segera mengidentifikasi bahwa sumber OS adalah penyebabnya dan memprioritaskan penggunaan OS resmi segera setelah kendala unduhannya teratasi. Kesabaran dan ketelitian dalam setiap langkah *setup* adalah kunci untuk mengatasi kendala ini.

## 7.5 Kendala Konversi Model PyTorch ke RKNN dan Integrasi Inferensi

- Masalah: Proses konversi model PyTorch (.pt) ke format RKNN (.rknn) juga memiliki tantangannya sendiri. Setelah eksplorasi awal, kami memahami bahwa konversi ini memerlukan langkah perantara melalui format ONNX. Konversi dari .pt ke ONNX berjalan tanpa masalah. Namun, pada tahap konversi dari ONNX ke RKNN, kami mengalami masalah serius. Setelah percobaan berulang dan eksplorasi intensif selama satu hari penuh, kami menyadari bahwa model yang dikonversi dengan opsi kuantisasi selalu bermasalah, menyebabkan model menjadi rusak dan tidak dapat mendeteksi objek apapun. Ini sesuai dengan laporan umum bahwa kuantisasi model *deep learning* yang kompleks seringkali menyebabkan penurunan performa atau ketidakfungsi.
- Masalah tidak berakhir setelah konversi model. Kami kemudian harus mengintegrasikan model RKNN ini ke dalam aplikasi FastAPI yang berjalan di Orange Pi 5 Pro. Ini berarti membuat file main1.py terpisah yang secara khusus dirancang untuk memuat model RKNN menggunakan RKNN Runtime API. Mengadaptasi *logic* inferensi dari *framework* PyTorch ke API RKNN, serta

mengintegrasikannya dengan alur FastAPI (pra-pemrosesan, inferensi, pasca-pemrosesan, dan *streaming* data), terbukti sangat sulit dan memakan waktu berjam-jam.

Solusi:

- Menonaktifkan Kuantisasi: Berdasarkan pengalaman langsung kami, solusi paling efektif untuk masalah model RKNN yang rusak adalah mengkonversi model tanpa melakukan kuantisasi. Keputusan ini memastikan model dapat berjalan dengan baik di NPU, meskipun tidak mendapatkan manfaat ukuran file yang lebih kecil atau kecepatan inferensi yang lebih tinggi dari kuantisasi. Ini adalah kompromi yang kami ambil untuk menjaga fungsionalitas model.
- Pemanfaatan AI untuk Integrasi Inferensi: Untuk mengatasi kesulitan dalam mengintegrasikan program inferensi RKNN dengan FastAPI, kami secara ekstensif menggunakan bantuan *Artificial Intelligence* (AI). AI membantu kami dalam menyusun kode untuk memuat model RKNN, mengimplementasikan *logic* pra-pemrosesan dan pasca-pemrosesan yang sesuai dengan *output* NPU, dan mengintegrasikan *loop* inferensi dengan *streaming* video FastAPI. Dengan bantuan AI, akhirnya program dapat berjalan dengan lancar, memungkinkan deteksi objek *real-time* menggunakan model RKNN pada Orange Pi 5 Pro

## **BAB 8**

### **KESIMPULAN DAN SARAN**

#### **8.1 Kesimpulan**

Proyek ini telah berhasil mencapai tujuannya dalam mengembangkan prototipe sistem segregasi sampah cerdas. Sebuah model deteksi objek berbasis YOLOv8n berhasil dilatih untuk mengenali 22 jenis sampah dan mengklasifikasikannya ke dalam tiga kategori utama dengan performa yang cukup untuk demonstrasi fungsionalitas. Keberhasilan implementasi model ke dalam dua prototipe aplikasi web yang fungsional (API dan interaktif) membuktikan kelayakan teknis dan konsep dari solusi yang diusulkan untuk otomatisasi pemilahan sampah.

#### **8.2 Saran Pengembangan**

Untuk pengembangan di masa depan, beberapa peningkatan dapat dilakukan:

##### **1. Perluasan Dataset**

Menambah lebih banyak gambar dengan variasi yang lebih luas (misalnya, latar belakang yang berbeda, kondisi pencahayaan yang beragam siang dan malam hari, objek sampah dalam kondisi rusak atau bertumpuk, serta sudut pandang kamera yang bervariasi) untuk meningkatkan ketahanan (robustness) dan akurasi model dalam berbagai skenario dunia nyata. Optimalisasi jumlah sampel per kelas juga dapat dipertimbangkan untuk memastikan distribusi yang seimbang.

##### **2. Integrasi dengan Sistem IoT (Internet of Things)**

Mengintegrasikan model deteksi objek ini dengan perangkat IoT untuk menciptakan sistem pengelolaan sampah yang lebih cerdas dan otomatis. Contoh implementasinya meliputi:

- Tempat Sampah Pintar Otomatis: Menghubungkan modul deteksi objek dengan tempat sampah yang memiliki mekanisme pemisahan otomatis. Setelah objek sampah terdeteksi dan diklasifikasikan oleh model, sistem IoT dapat mengaktifkan aktuator (misalnya, lengan robotik atau sekat pintu) untuk mengarahkan sampah ke kompartemen yang sesuai (organik, anorganik, B3, dll.).

- Notifikasi Pengisian: Sensor level pada tempat sampah dapat terhubung ke sistem deteksi untuk memverifikasi jenis sampah yang masuk dan mengirimkan notifikasi otomatis ke petugas kebersihan atau sistem manajemen kota ketika suatu kompartemen sudah penuh.
  - Pemantauan Jarak Jauh: Menggunakan koneksi IoT, data tentang jenis dan jumlah sampah yang dideteksi dapat diunggah ke *cloud* atau *dashboard* monitoring, memungkinkan pemantauan dan analisis pengelolaan sampah secara *real-time* dari jarak jauh
3. Optimalisasi Model untuk Edge Devices

Meskipun YOLOv8n sudah efisien, eksplorasi lebih lanjut terhadap teknik kuantisasi, *pruning*, atau penggunaan arsitektur model yang lebih ringan (misalnya, MobileNet, EfficientNet-lite sebagai *backbone* alternatif jika sangat diperlukan) dapat dilakukan untuk memastikan model dapat berjalan dengan performa *real-time* pada perangkat keras dengan sumber daya komputasi yang sangat terbatas dan konsumsi daya rendah, seperti mikrokontroler atau *single-board computer* yang ditenagai baterai.

4. Penambahan Kelas Sampah dan Hierarki Klasifikasi

Memperluas daftar kelas sampah yang dapat dideteksi (misalnya, menambahkan 'logam', 'kain', 'karet', 'sampah elektronik', dll.). Selain itu, mengimplementasikan hierarki klasifikasi (misalnya, 'Plastik' -> 'Botol Plastik', 'Kantong Plastik') untuk memungkinkan pemisahan yang lebih granular dan cerdas.

5. Pengembangan Antarmuka Pengguna yang Lebih Interaktif:

Membangun antarmuka pengguna (misalnya, aplikasi *mobile* atau *web*) yang memungkinkan pengguna untuk berinteraksi dengan sistem deteksi, melihat hasil klasifikasi secara *real-time*, dan mungkin mendapatkan informasi lebih lanjut tentang daur ulang sampah.

## DAFTAR PUSTAKA

### References

Ultralytics. (n.d.). *Ultralytics YOLO Docs*. Retrieved June 20, 2025, from

<https://docs.ultralytics.com/>

Godoy, S. (n.d.). *FastAPI: A modern, fast (high-performance) web framework for building APIs*

*with Python 3.7+ based on standard Python type hints*. Retrieved June 20, 2025, from

<https://fastapi.tiangolo.com/>

Python Software Foundation. (n.d.). *The Python Language Reference*. Retrieved June 20,

2025, from <https://docs.python.org/3/reference/>

OrangePi. (n.d.). Orange Pi - OrangePi. Retrieved June 20, 2025, from

<http://www.orangepi.org/html/hardWare/computerAndMicrocontrollers/details/Orange-Pi-5-Pro.html>

Rockchip. (2024). *rknn-toolkit2* [Source code]. GitHub. [rockchip-linux/rknn-toolkit2](https://github.com/rockchip-linux/rknn-toolkit2)

Orange Pi 5 - Easy Setup and Overview!

<https://youtu.be/cBqV4QWj0lE?si=N9lQ3q0WDanrAZ78>