Sistem Deteksi Jenis Mobil dan Estimasi Kecepatan Berbasis Machine Learning untuk Analisis Potensi Kecelakaan Kendaraan

Anggota Kelompok :

Ramdhan Javas D	(23090620048)
Arga Abitama	(23090620068)
Nabil Khairi Ikhsan	(23090620080)

PROGRAM STUDI D4 TEKNIK ELEKTRONIKA
JURUSAN TEKNIK ELEKTRONIKA
FAKULTAS VOKASI
UNIVERSITAS NEGERI YOGYAKARTA

2025

A. LATAR BELAKANG

Meningkatnya jumlah kendaraan di jalan raya berpotensi menimbulkan angka kecelakaan yang lebih tinggi. Banyak kecelakaan terjadi akibat kecepatan kendaraan yang berlebihan dan kurangnya sistem pemantauan yang mampu mengidentifikasi kondisi berbahaya secara cepat. Dengan kemajuan teknologi kecerdasan buatan dan computer vision, proses deteksi kendaraan, pengukuran kecepatan, serta analisis potensi kecelakaan kini dapat dilakukan secara otomatis melalui kamera. Oleh karena itu, diperlukan sebuah sistem yang mampu mendeteksi tipe mobil, mengestimasi kecepatan kendaraan, serta memberikan peringatan dini potensi kecelakaan untuk meningkatkan keselamatan dan efektivitas monitoring lalu lintas.

B. TUJUAN

- Mengembangkan sistem berbasis computer vision untuk mendeteksi tipe mobil secara otomatis dari citra atau video lalu lintas.
- Mengimplementasikan metode estimasi untuk mengukur kecepatan kendaraan (km/jam) secara real-time.
- Menganalisis dan memprediksi potensi kecelakaan kendaraan berdasarkan pergerakan dan jarak antar kendaraan.
- Menyediakan sistem monitoring lalu lintas yang dapat memberikan peringatan dini terhadap kondisi berbahaya untuk membantu meningkatkan keselamatan di jalan raya.

C. MODEL MACHINE LEARNING

1. YOLOv11 (You Only Look Onve v11) Object Detection
   Salah satu model deep learning yang berbasis *Convolutional Neural Network (CNN)* yang digunakan untuk mendeteksi dan mengklasifikasikan jenis kendaraan seperti mobil, bus, dan motor dari rekaman sample CCTV yang ada di lampu lalu lintas secara akurat.
   - Input : Frame video sample CCTV lampu lalu lintas.
   - Output : Label jenis kendaraan dan kotak box.
   - Library : YOLOv11 atau OpenCV.

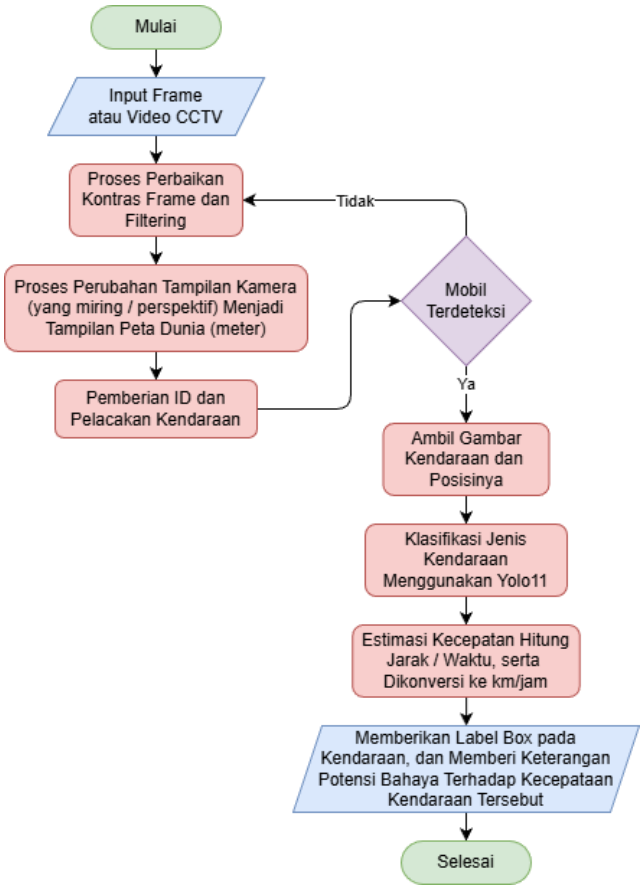
D. JOBDESK

1. Ramdhan Javas Dintarisanto
   - Mencari referensi dan jurnal yang berkaitan dengan projek.
   - Melakukan anotasi di Roboflow.
   - Mencari kode program untuk memprediksi kendaraan, menghitung kecepatan, dan menentukan potensi bahaya.
   - Membuat flowchart untuk alur sistem.

2. Arga Abitama
   - Mencari dan mengumpulkan video CCTV untuk dijadikan dataset.
   - Mencari kode program untuk program training model.
   - Membuat arsitektur diagram.
   - Melakukan anotasi di Roboflow.
   - Menyusun dan merevisi proposal.

3. Nabil Khairi Ikhsan
    - Membuat dataset (anotasi jenis kendaraan) lalu train frame 80%, valid 20%.
    - Melakukan training 100 epoch di program pertama.
    - Membuat pemetaan dari pixel gambar menjadi ukuran asli (satuan meter).
    - Menjalankan program training model.
    - Menjalankan program pemrosesan video untuk deteksi objek, tracking, dan evaluasi kecepatan kendaraan.

E. FLOWCHART



Gambar 1. Alur Kerja dari Sistem Potensi Kecelakaan Kendaraan.

Proses sistem dimulai ketika sebuah frame atau video CCTV dimasukkan sebagai input. Setiap frame terlebih dahulu mengalami perbaikan kontras dan filtering untuk meningkatkan kualitas gambar sehingga objek kendaraan dapat terdeteksi dengan lebih akurat. Setelah itu, sistem melakukan koreksi tampilan kamera untuk mengubah sudut pandang yang miring atau perspektif menjadi tampilan peta dunia dalam satuan meter. Langkah ini penting agar posisi dan jarak kendaraan dapat dihitung dalam bentuk koordinat yang lebih real dan terukur. Ketika sistem melakukan pengecekan, jika tidak ada kendaraan yang terdeteksi, maka proses kembali ke tahap sebelum deteksi untuk mengevaluasi frame berikutnya. Namun, jika mobil terdeteksi, sistem akan mengambil gambar kendaraan beserta posisi koordinatnya.

Selanjutnya, gambar kendaraan tersebut diklasifikasikan menggunakan YOLO11 untuk menentukan jenis kendaraan, seperti mobil, motor, truk, atau bus. Setelah mengetahui kelas kendaraan, sistem menghitung estimasi kecepatan berdasarkan perubahan jarak terhadap waktu dan kemudian mengonversinya menjadi nilai kecepatan km/jam. Berdasarkan hasil deteksi dan kecepatan tersebut, sistem menambahkan label box pada objek kendaraan di tampilan video, sekaligus memberikan keterangan apakah kendaraan tersebut memiliki potensi bahaya atau melanggar batas kecepatan. Ketika seluruh proses analisis selesai, sistem menandai proses sebagai selesai dan menunggu frame berikutnya untuk diproses kembali.

F. ARSITEKTUR DIAGRAM

Gambar 2. Alur Arsitektur Diagram Potensi Kecelakaan Diagram

Arsitektur sistem dimulai dari proses mencari dan mengumpulkan video yang akan digunakan sebagai sumber data. Video tersebut kemudian diproses menggunakan VLC Media

Player untuk dipecah menjadi beberapa gambar. Dari setiap detik video, dilakukan ekstraksi menjadi 15 gambar sehingga menghasilkan dataset berbasis frame dengan frame rate tertentu (misalnya 30 fps atau 2 fps sesuai kebutuhan). Setelah gambar diperoleh, seluruh dataset diunggah ke Roboflow untuk dilakukan proses labeling. Pada tahap ini dibuat 4 class, yaitu Mobil, Motor, Truck, dan Bus, kemudian setiap gambar diberikan anotasi sesuai objek yang muncul. Roboflow juga secara otomatis membagi dataset menjadi 80% untuk training dan 20% untuk validasi sehingga siap digunakan dalam model YOLO atau deep learning lainnya.

Selanjutnya, pelatihan model dilakukan menggunakan framework PyTorch, dengan memanfaatkan GPU Nvidia RTX 3050 untuk menggantikan CPU sehingga proses training menjadi lebih cepat dan optimal. Seluruh kode pengembangan, pelatihan, serta evaluasi dijalankan melalui Visual Studio Code, tempat seluruh konfigurasi model, epoch, dan proses inferensi disusun. Pada tahap akhir, model dilatih selama 100 epoch, kemudian dilakukan penginputan dataset video untuk diuji menggunakan model hasil training agar dapat mendeteksi objek secara akurat sesuai kelas yang telah didefinisikan.

G. KODE PROGRAM

- **TRAINING MODEL**

```
import os
import yaml
import shutil
import numpy as np
from glob import glob
from ultralytics import YOLO
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, accuracy_score
import seaborn as sns

# 0. Folder untuk simpan grafik

GRAPH_DIR = "runs/metrics"
os.makedirs(GRAPH_DIR, exist_ok=True)

# 1. Copy otomatis data.yaml + copy isi train/valid ke semua fold

def prepare_folds():
    src_yaml = r"D:/KCBUAS/dataset/data.yaml"
    src_train = r"D:/KCBUAS/dataset/train"
    src_valid = r"D:/KCBUAS/dataset/valid"

    print("=== COPY DATASET KE SETIAP FOLD ===")

    for fold in range(5):
        fold_path = f"D:/KCBUAS/kfold/fold{fold}"

        # buat folder
        train_img = f"{fold_path}/train/images"
        train_lbl = f"{fold_path}/train/labels"
        valid_img = f"{fold_path}/valid/images"
        valid_lbl = f"{fold_path}/valid/labels"

        os.makedirs(train_img, exist_ok=True)
        os.makedirs(train_lbl, exist_ok=True)
        os.makedirs(valid_img, exist_ok=True)
        os.makedirs(valid_lbl, exist_ok=True)
```

```
        # copy yaml
        shutil.copy(src_yaml, f"{fold_path}/data.yaml")
        print(f"[COPY] data.yaml → fold{fold}")

        # COPY TRAIN
        print(f"[COPY] train → fold{fold}")
        for img in glob(src_train + "/images/*"):
            shutil.copy(img, train_img)
        for lbl in glob(src_train + "/labels/*"):
            shutil.copy(lbl, train_lbl)

        # COPY VALID=
        print(f"[COPY] valid → fold{fold}")
        for img in glob(src_valid + "/images/*"):
            shutil.copy(img, valid_img)
        for lbl in glob(src_valid + "/labels/*"):
            shutil.copy(lbl, valid_lbl)


# 2. Load class dari yaml

def load_classes(yaml_path):
    with open(yaml_path, "r") as f:
        data = yaml.safe_load(f)
    classes = data["names"]

    print("\n===== KELAS YANG DILATIH =====")
    for i, cls in enumerate(classes):
        print(f"ID {i} = {cls}")
    print("================================\n")

    return classes


# 3. Train model untuk satu fold

def train_fold(fold):
    print(f"\n[TRAIN] Fold {fold}\n")
    model = YOLO("yolo11m.pt")

    fold_dir = f"D:/KCBUAS/kfold/fold{fold}"
    yaml_path = os.path.join(fold_dir, "data.yaml")

    load_classes(yaml_path)

    model.train(
        data=yaml_path,
        epochs=100,
        batch=4,
        imgsz=640,
        name=f"fold_{fold}",
        pretrained=True,
        workers=2,
        half=True
    )

    return model


# 4. Evaluasi tiap fold
```

```
def evaluate_fold(model, fold):
    print(f"[EVAL] Fold {fold}")
    img_dir = f"D:/KCBUAS/kfold/fold{fold}/valid/images"

    imgs = glob(img_dir + "/*.jpg") + glob(img_dir + "/*.png")

    y_true, y_pred = [], []

    for img in imgs:
        lbl_path = img.replace("images", "labels").replace(".jpg",
".txt").replace(".png", ".txt")

        if not os.path.exists(lbl_path):
            continue

        with open(lbl_path, "r") as f:
            first_line = f.readline().strip().split()
            if len(first_line) == 0:
                continue
            class_id = int(first_line[0])
            y_true.append(class_id)

        results = model.predict(img, conf=0.25, verbose=False)

        if len(results[0].boxes) == 0:
            y_pred.append(-1)
        else:
            confs = results[0].boxes.conf.cpu().numpy()
            clss = results[0].boxes.cls.cpu().numpy()
            pred_class = int(clss[np.argmax(confs)])
            y_pred.append(pred_class)

    return y_true, y_pred


# 5. Plot Confusion Matrix

def plot_confusion(cm, labels, fold):
    plt.figure(figsize=(7, 6))
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
                xticklabels=labels, yticklabels=labels)
    plt.title(f"Confusion Matrix Fold {fold}")
    plt.xlabel("Prediksi")
    plt.ylabel("Label")
    plt.tight_layout()

    path = f"{GRAPH_DIR}/confusion_fold_{fold}.png"
    plt.savefig(path)
    plt.close()
    print(f"[SAVE] {path}")


# 6. Plot akurasi

def plot_accuracy(accs):
    plt.figure()
    plt.plot(range(5), accs, marker="o")
    plt.title("Akurasi Setiap Fold")
    plt.xlabel("Fold")
    plt.ylabel("Accuracy")
```

```
    plt.grid()
    plt.tight_layout()

    path = f"{GRAPH_DIR}/accuracy_plot.png"
    plt.savefig(path)
    plt.close()
    print(f"[SAVE] {path}")


# 7. Plot mAP

def plot_map(maps50, maps95):
    plt.figure()
    plt.plot(range(5), maps50, marker="o", label="mAP50")
    plt.plot(range(5), maps95, marker="o", label="mAP50-95")
    plt.legend()
    plt.title("mAP Setiap Fold")
    plt.xlabel("Fold")
    plt.ylabel("mAP")
    plt.grid()
    plt.tight_layout()

    path = f"{GRAPH_DIR}/map_plot.png"
    plt.savefig(path)
    plt.close()
    print(f"[SAVE] {path}")


# 8. RUN K-FOLD

def main():
    prepare_folds()

    kelas = load_classes("D:/KCBUAS/kfold/fold0/data.yaml")

    accuracy_all = []
    map50_all = []
    map95_all = []

    for fold in range(5):
        print(f"\n========== FOLD {fold} ==========")

        model = train_fold(fold)

        metrics = model.val()
        map50_all.append(metrics.results_dict["metrics/mAP50"])
        map95_all.append(metrics.results_dict["metrics/mAP50-95"])

        y_true, y_pred = evaluate_fold(model, fold)
        cm = confusion_matrix(y_true, y_pred, labels=[0,1,2,3])
        acc = accuracy_score(y_true, y_pred)

        accuracy_all.append(acc)
        plot_confusion(cm, kelas, fold)

        print(f"[HASIL FOLD {fold}]")
        print("Confusion Matrix:\n", cm)
        print(f"Akurasi = {acc:.4f}")

    plot_accuracy(accuracy_all)
    plot_map(map50_all, map95_all)
```

```
    print("\n========== RINGKASAN AKHIR ==========")
    print(f"Rata-rata Akurasi  : {np.mean(accuracy_all):.4f}")
    print(f"Rata-rata mAP50    : {np.mean(map50_all):.4f}")
    print(f"Rata-rata mAP50-95 : {np.mean(map95_all):.4f}")
    print("Semua grafik disimpan di folder: runs/metrics/")
    print("=====================================\n")

if __name__ == "__main__":
    main()
```

IMPORT LIBRARY

```
import os
import yaml
import shutil
import numpy as np
from glob import glob
from ultralytics import YOLO
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, accuracy_score
import seaborn as sns
```

Penjelasan Kode Program Training Model :

- os → manipulasi folder/file.
- yaml → membaca file data.yaml.
- shutil → copy folder/file.
- numpy → operasi array.
- glob → mencari file dengan pola (misal "*.jpg").
- YOLO → library ultralytics untuk model YOLO.
- matplotlib.pyplot → membuat grafik.
- sklearn.metrics → confusion matrix dan akurasi.
- seaborn → heatmap untuk confusion matrix.

Folder untuk menyimpan grafik

```
GRAPH_DIR = "runs/metrics"
os.makedirs(GRAPH_DIR, exist_ok=True)
```

- GRAPH_DIR adalah path folder untuk menyimpan hasil grafik.
- os.makedirs(..., exist_ok=True) → membuat folder, jika sudah ada tidak error.

Menentukan lokasi dataset asli

```
src_yaml = r"D:/KCBUAS/dataset/data.yaml"
src_train = r"D:/KCBUAS/dataset/train"
src_valid = r"D:/KCBUAS/dataset/valid"
```

Loop 5 fold

```
for fold in range(5):
    fold_path = f"D:/KCBUAS/kfold/fold{fold}"
```

Membuat folder:

- Fold0.
- Fold1.
- Fold2.
- Fold3.

- Fold4.

Membuat struktur folder YOLO

```
train_img = f"{fold_path}/train/images"
train_lbl = f"{fold_path}/train/labels"
valid_img = f"{fold_path}/valid/images"
valid_lbl = f"{fold_path}/valid/labels"

os.makedirs(train_img, exist_ok=True)
os.makedirs(train_lbl, exist_ok=True)
os.makedirs(valid_img, exist_ok=True)
os.makedirs(valid_lbl, exist_ok=True)
```

Copy data.yaml ke setiap fold

```
shutil.copy(src_yaml, f"{fold_path}/data.yaml")
```

Copy semua gambar train dan valid

```
for img in glob(src_train + "/images/*"):
    shutil.copy(img, train_img)
```

- Meng-copy semua file gambar train sedetail-nya.
- Lalu copy semua label-nya.

FUNGSI 2 – load_classes()

```
with open(yaml_path, "r") as f:
    data = yaml.safe_load(f)
classes = data["names"]
```

- Membaca data.yaml.
- Mengambil daftar kelas YOLO pada key "names".

FUNGSI 3 – train_fold()

```
model = YOLO("yolo11m.pt")
```

- Load model YOLO11M pretrained.

Mengatur lokasi data.yaml untuk fold tertentu:

```
fold_dir = f"D:/KCBUAS/kfold/fold{fold}"
yaml_path = os.path.join(fold_dir, "data.yaml")
```

Train model

```
model.train(
    data=yaml_path,
    epochs=100,
    batch=4,
    imgsz=640,
    name=f"fold_{fold}",
    pretrained=True,
    workers=2,
    half=True
)
```

Penjelasan parameter :

- data=yaml_path → dataset fold saat ini.
- epochs=100 → training 100 epoch.
- batch=4 → batch size 4.
- imgsz=640 → ukuran input YOLO.

- name=f"fold_{fold}" → hasil training disimpan di folder runs/FOLD_n.
- half=True → menggunakan FP16 untuk mempercepat dan mengurangi RAM GPU.

FUNGSI 4 – evaluate_fold()
Fungsi ini mengevaluasi model pada valid set dari fold.

Ambil semua gambar valid

```
imgs = glob(img_dir + "/*.jpg") + glob(img_dir + "/*.png")
```

Loop tiap gambar untuk ambil label asli

```
lbl_path = img.replace("images", "labels").replace(".jpg", ".txt")
```

YOLO format label:

```
class center_x center_y width height
```

Bila gambar tidak punya tempat label → skip

```
if not os.path.exists(lbl_path):
    continue
```

Membaca class ID ground truth

```
with open(lbl_path, "r") as f:
    first_line = f.readline().strip().split()
class_id = int(first_line[0])
y_true.append(class_id)
```

Prediksi model YOLO

```
results = model.predict(img, conf=0.25, verbose=False)
```

Jika YOLO tidak mendeteksi apapun

```
if len(results[0].boxes) == 0:
    y_pred.append(-1)
```

Ambil prediksi dengan confidence tertinggi

```
confs = results[0].boxes.conf.cpu().numpy()
clss = results[0].boxes.cls.cpu().numpy()
pred_class = int(clss[np.argmax(confs)])
y_pred.append(pred_class)
```

FUNGSI 5 – plot_confusion()

```
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
            xticklabels=labels, yticklabels=labels)
```

Menggambar confusion matrix dengan heatmap.
FUNGSI 6 – plot_accuracy()
Menggambar grafik akurasi per fold.
FUNGSI 7 – plot_map()
Menggambar grafik mAP50 dan mAP50-95 per fold.
FUNGSI UTAMA – main()

Siapkan 5 fold dataset

```
prepare_folds()
```

Load daftar kelas hanya dari fold0

```
kelas = load_classes("D:/KCBUAS/kfold/fold0/data.yaml")
```

Siapkan list untuk menyimpan semua hasil

```
accuracy_all = []
map50_all = []
map95_all = []
```

Loop training 5 fold

```
for fold in range(5):
```

TRAIN

```
model = train_fold(fold)
```

VALIDASI

```
metrics = model.val()
map50_all.append(metrics.results_dict["metrics/mAP50"])
map95_all.append(metrics.results_dict["metrics/mAP50-95"])
```

Hitung prediksi manual + confusion matrix

```
y_true, y_pred = evaluate_fold(model, fold)
cm = confusion_matrix(y_true, y_pred, labels=[0,1,2,3])
acc = accuracy_score(y_true, y_pred)
```

Simpan akurasi

```
accuracy_all.append(acc)
```

Simpan grafik confusion matrix

```
plot_confusion(cm, kelas, fold)
```

Cetak hasil fold

```
print(f"[HASIL FOLD {fold}]")
```

Plot ringkasan akhir

```
plot_accuracy(accuracy_all)
plot_map(map50_all, map95_all)
```

Print ringkasan K-Fold

```
print(f"Rata-rata Akurasi  : {np.mean(accuracy_all):.4f}")
print(f"Rata-rata mAP50    : {np.mean(map50_all):.4f}")
print(f"Rata-rata mAP50-95 : {np.mean(map95_all):.4f}")
```

- **PROGRAM PEMROSESAN VIDEO UNTUK DETEKSI OBJEK, TRACKING, dan EVALUASI KECEPATAN KENDARAAN**

```
# LIBRARY YANG DIGUNAKAN
import cv2
import numpy as np
from ultralytics import YOLO
import os
```

```
import torch

# KONFIGURASI PATH
MODEL_PATH = 'D:/KCBUAS/runs/detect/fold_0/weights/best.pt'
VIDEO_SOURCE_PATH = 'D:/KCBUAS/vidsample/4.mp4'
VIDEO_OUTPUT_PATH = 'D:/KCBUAS/vidhasil/3.4.mp4'

os.makedirs(os.path.dirname(VIDEO_OUTPUT_PATH), exist_ok=True)

tracker_path = "ultralytics/cfg/trackers/bytetrack.yaml"

# PERSPECTIVE TRANSFORMATION
image_pts = np.array([
    [533, 45],    # A
    [701, 45],    # B
    [1677, 855],  # D
    [56, 865]     # C

], dtype=np.float32)

LEBAR_JALAN_METER = 15
PANJANG_JALAN_METER = 60

world_pts = np.array([
    [0, 0],
    [LEBAR_JALAN_METER, 0],
    [LEBAR_JALAN_METER, PANJANG_JALAN_METER],
    [0, PANJANG_JALAN_METER]
], dtype=np.float32)

M = cv2.getPerspectiveTransform(image_pts, world_pts)
print("Matriks Transformasi (M) berhasil dibuat.")

# LOAD YOLO MODEL
try:
    device = 'cuda' if torch.cuda.is_available() else 'cpu'
    model = YOLO(MODEL_PATH)
    print(f"Model berhasil dimuat: {MODEL_PATH} (device: {device})")
except Exception as e:
    print("Error saat memuat model:", e)
    raise SystemExit

# LOAD VIDEO
cap = cv2.VideoCapture(VIDEO_SOURCE_PATH)
if not cap.isOpened():
    print(f"Error: Tidak dapat membuka video di {VIDEO_SOURCE_PATH}")
    raise SystemExit

w = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
h = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
fps = int(cap.get(cv2.CAP_PROP_FPS))

fourcc = cv2.VideoWriter_fourcc(*'mp4v')
out = cv2.VideoWriter(VIDEO_OUTPUT_PATH, fourcc, fps, (w, h))

# TRACKING + SPEED ESTIMATION
tracked_objects = {}  # track_id : (prev_x, prev_y, prev_time)
speed_history = {}    # track_id : list of previous speeds
ema_speed = {}        # track_id : last filtered speed

def classify_risk(speed_kmh):
```

```
    if speed_kmh >= 80: return "Tinggi", (0, 0, 255)
    elif speed_kmh >= 40: return "Sedang", (0, 255, 255)
    elif speed_kmh >= 20: return "Rendah", (0, 255, 0)
    else: return "Aman", (255, 255, 255)

print("Memulai proses prediksi dan perhitungan kecepatan...")

while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        print("Video selesai diproses.")
        break

    current_time = cap.get(cv2.CAP_PROP_POS_MSEC) / 1000.0

    # YOLO TRACK #
    results = model.track(frame, persist=True, tracker=tracker_path)

    if results[0].boxes.id is not None:
        boxes = results[0].boxes.xyxy.cpu().numpy().astype(int)
        ids = results[0].boxes.id.cpu().numpy().astype(int)
        confs = results[0].boxes.conf.cpu().numpy()
        clss = results[0].boxes.cls.cpu().numpy().astype(int)

        # PROSES SETIAP OBJEK #
        for box, track_id, conf, cls_id in zip(boxes, ids, confs, clss):
            x1, y1, x2, y2 = box
            center_x, center_y = int((x1 + x2) / 2), y2
            speed_kmh = 0

            #  HITUNG KECEPATAN  #
            if track_id in tracked_objects:
                prev_x, prev_y, prev_time = tracked_objects[track_id]
                selisih_waktu = current_time - prev_time
                if selisih_waktu > 0:
                            titik_lama  =  np.array([[[prev_x,  prev_y]]],
dtype=np.float32)
                        titik_baru  =  np.array([[[center_x,  center_y]]],
dtype=np.float32)
                    world_old = cv2.perspectiveTransform(titik_lama, M)
                    world_new = cv2.perspectiveTransform(titik_baru, M)
                    jarak_meter = np.linalg.norm(world_new - world_old)
                    speed_ms = jarak_meter / selisih_waktu
                    speed_kmh = speed_ms * 3.6

            #  UPDATE POSISI  #
            tracked_objects[track_id] = (center_x, center_y, current_time)

            #  STABILISASI KECEPATAN  #
            if speed_kmh > 200:
                speed_kmh = 200

            if track_id not in speed_history:
                speed_history[track_id] = []
            speed_history[track_id].append(speed_kmh)
            if len(speed_history[track_id]) > 5:
                speed_history[track_id].pop(0)
                        avg_speed   =   sum(speed_history[track_id])   /
len(speed_history[track_id])

            alpha = 0.25
```

```
            if track_id not in ema_speed:
                ema_speed[track_id] = avg_speed
            else:
                ema_speed[track_id] = (alpha * avg_speed) + (1 - alpha) *
ema_speed[track_id]

            final_speed = ema_speed[track_id]

            #  RISK LABEL  #
            risk_label, color = classify_risk(final_speed)
            class_name = model.names[cls_id] if cls_id < len(model.names)
else "Unknown"
            label1 = f"ID:{track_id} {class_name} {conf:.2f}"
            label2 = f"{final_speed:.2f} km/h - Risiko: {risk_label}"

            cv2.rectangle(frame, (x1, y1), (x2, y2), color, 2)
            cv2.putText(frame, label1, (x1, y1 - 10),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.6, color, 2)
            cv2.putText(frame, label2, (x1, y2 + 20),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.6, color, 2)

    out.write(frame)

# SELESAI
cap.release()
out.release()
print("Proses selesai.")
print("Video tersimpan di:", VIDEO_OUTPUT_PATH)
```

Penjelasan kode program :

- Library Yang Digunakan

```
import cv2
import numpy as np
from ultralytics import YOLO
import os
import torch
```

Kode program tersebut digunakan agar program dapat memproses data visual, menjalankan model Machine Learning, dan menghasilkan video dimana ada beberapa library yang digunakan yaitu:

- Import cv2 digunakan untuk pemrosesan video, gambar, dan visualisasi hasil.
- Import numpy as np digunakan untuk operasi perhitungan numerik dan transformasi matriks.
- Form ultralytics import YOLO digunakan untuk melakukan deteksi objek dan tracking.
- Import os digunakan untuk memastikan direktori penyimpanan output tersedia.
- Import torch digunakan untuk pengecekan GPU agar proses dapat berjalan dengan cepat.

- Konfigurasi Path

```
# KONFIGURASI PATH
MODEL_PATH = 'D:/KCBUAS/runs/detect/fold_0/weights/best.pt'
VIDEO_SOURCE_PATH = 'D:/KCBUAS/vidsample/4.mp4'
VIDEO_OUTPUT_PATH = 'D:/KCBUAS/vidhasil/3.4.mp4'

os.makedirs(os.path.dirname(VIDEO_OUTPUT_PATH), exist_ok=True)

tracker_path = "ultralytics/cfg/trackers/bytetrack.yaml"
```

Kode program tersebut digunakan untuk menentukan lokasi file, kode program yang digunakan yaitu:

- MODEL_PATH digunakan untuk menentukan lokasi model YOLO dari hasil training yang akan digunakan untuk mendeteksi kendaraan.
- VIDEO_SOURCE_PATH digunakan untuk menunjukkan lokasi video input yang akan dianalisis.
- VIDEO_OUTPUT_PATH digunakan untuk tempat penyimpanan video hasil prediksi.
- os.makedirs(os.path.dirname(VIDEO_OUTPUT_PATH), exist_ok=True) digunakan untuk memastikan folder penyimpanan hasil tersedia.
- Tracker_path digunakan untuk memanggil konfigurasi BYTEtracker yang berfungsi melacak objek antar frame agar pergerakan kendaraan dapat diukur.

- Transformasi Perspektif

```
# PERSPECTIVE TRANSFORMATION
image_pts = np.array([
    [533, 45],    # A
    [701, 45],    # B
    [1677, 855],  # D
    [56, 865]     # C

], dtype=np.float32)

LEBAR_JALAN_METER = 15
PANJANG_JALAN_METER = 60

world_pts = np.array([
    [0, 0],
    [LEBAR_JALAN_METER, 0],
    [LEBAR_JALAN_METER, PANJANG_JALAN_METER],
    [0, PANJANG_JALAN_METER]
], dtype=np.float32)

M = cv2.getPerspectiveTransform(image_pts, world_pts)
print("Matriks Transformasi (M) berhasil dibuat.")
```

Kode program tersebut digunakan untuk kalibrasi area jalan menggunakan 4 titik acuan dari video dan menghubungkannya dengan jarak yang nyata. Menggunakan kode program dibawah ini:

- image_pts digunakan untuk acuan kalibrasi yang nantinya menandai 4 titik pada gambar dalam satuan pixel.
- LEBAR_JALAN_METER dan PANJANG_JALAN_METER digunakan untuk menunjukkan ukuran nyata dari area jalan.
- word_pts digunakan untuk mendefinisikan koordinat pada dunia nyata dalam satuan meter.
- M = cv2.getPerspectiveTransform(image_pts, world_pts) digunakan untuk membuat matriks transformasi pixel menjadi meter, sehingga kecepatan bisa dihitung berdasarkan jarak berpindah kendaraan secara nyata.

- Model YoLo

```
# LOAD YOLO MODEL
try:
    device = 'cuda' if torch.cuda.is_available() else 'cpu'
    model = YOLO(MODEL_PATH)
```

```
    print(f"Model berhasil dimuat: {MODEL_PATH} (device: {device})")
except Exception as e:
    print("Error saat memuat model:", e)
    raise SystemExit
```

- Mengecek apakah GPU tersedia menggunakan torch.cuda.is_available().
- Memuat model YOLO menggunakan file best.pt
- Menampilkan pesan berhasil atau error jika model gagal dijalankan.

- Video Input dan Output

```
cap = cv2.VideoCapture(VIDEO_SOURCE_PATH)
if not cap.isOpened():
    print(f"Error: Tidak dapat membuka video di {VIDEO_SOURCE_PATH}")
    raise SystemExit

w = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
h = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
fps = int(cap.get(cv2.CAP_PROP_FPS))

fourcc = cv2.VideoWriter_fourcc(*'mp4v')
out = cv2.VideoWriter(VIDEO_OUTPUT_PATH, fourcc, fps, (w, h))
```

Dari kode program untuk video input dan setup output terdapat fungsi yang digunakan yaitu:

- cv2.VideoCapture(VIDEO_SOURCE_PATH) digunakan untuk membuka file video input.
- Jika video gagal dibuka maka program akan dihentikan.
- Frame width (w), height(h), dan FPS diambil untuk menghasilkan output video dengan format yang sama.
- cv2.VideoWriter(VIDEO_OUTPUT_PATH, fourcc, fps, (w, h)) digunakan untuk menyimpan video hasil deteksi kendaraan.

- Tracking dan Estimasi Kecepatan

```
tracked_objects = {}  # track_id : (prev_x, prev_y, prev_time)
speed_history = {}    # track_id : list of previous speeds
ema_speed = {}        # track_id : last filtered speed
```

- tracked_objects digunakan untuk menyimpan posisi kendaraan dari frame sebelumnya.
- speed_history digunakan untuk menyimpan riwayat hasil kendaraan.
- ema_speed digunakan untuk menyimpan kecepatan yang sudah difilter dengan exponential moving average, agar tidak fluktuatif.

- Fungsi Klasifikasi Resiko

```
def classify_risk(speed_kmh):
    if speed_kmh >= 80: return "Tinggi", (0, 0, 255)
    elif speed_kmh >= 40: return "Sedang", (0, 255, 255)
    elif speed_kmh >= 20: return "Rendah", (0, 255, 0)
    else: return "Aman", (255, 255, 255)
```

Fungsi klasifikasi resiko tersebut digunakan untuk menilai tingkar risiko kecelakaan berdasarkan kecepatan kendaraan dimana jika kecepataan diatas 80km/jam beresiko tinggi, jika kecepatan diatas 40km/jam beresiko sedang, jika kecepatan diatas 20km/jam beresiko rendah.

- Proses Frame Video

```
while cap.isOpened():
```

```
    ret, frame = cap.read()
    if not ret:
        print("Video selesai diproses.")
        break

    current_time = cap.get(cv2.CAP_PROP_POS_MSEC) / 1000.0
```

- Setiap frame dibaca satu per satu.
- current_time digunakan untuk menunjukkan waktu saat frame diproses dalam detik lalu digunakan untuk menghitung kecepatan.

- Deteksi dan Tracking Kendaraan

```
results = model.track(frame, persist=True, tracker=tracker_path)
```

Kode program tersebut menggunakan YOLO untuk mendeteksi objek dan melacak pergerakan antar frame menggunakan BYTETracker.

- Hitung Kecepatan Kendaraan

```
jarak_meter = np.linalg.norm(world_new - world_old)
speed_ms = jarak_meter / selisih_waktu
speed_kmh = speed_ms * 3.6
```

- Menghitung jarak perpindahan kendaraan dari frame sebelumnya.
- Mengkonversi ke dalam kecepatan km/jam.

- Stabilitas Kendaraan

```
alpha = 0.25
ema_speed[track_id] = (alpha * avg_speed) + (1 - alpha) *
ema_speed[track_id]
```

Kode program tersebut digunakan untuk memfilter kecepatan kendaraa menggunakan EMA agar tidak berubah secara drastis karena perubahan posisi kecil antar frame.

- Visualisasi Informasi Kendaraan

```
cv2.rectangle(frame, (x1, y1), (x2, y2), color, 2)
cv2.putText(frame, label1, (x1, y1 - 10), ...)
cv2.putText(frame, label2, (x1, y2 + 20), ...)
```

Kode program tersebut digunakan untuk menampilkan box, ID kendaraan, jenis kendaraan, kecepatan, dan risiko.

- Penyimpanan Output

```
out.write(frame)
```

Kode program tersebut merupakan frame yang sudah dianotasi disimpan sebagai video hasil.

- Program Selesai

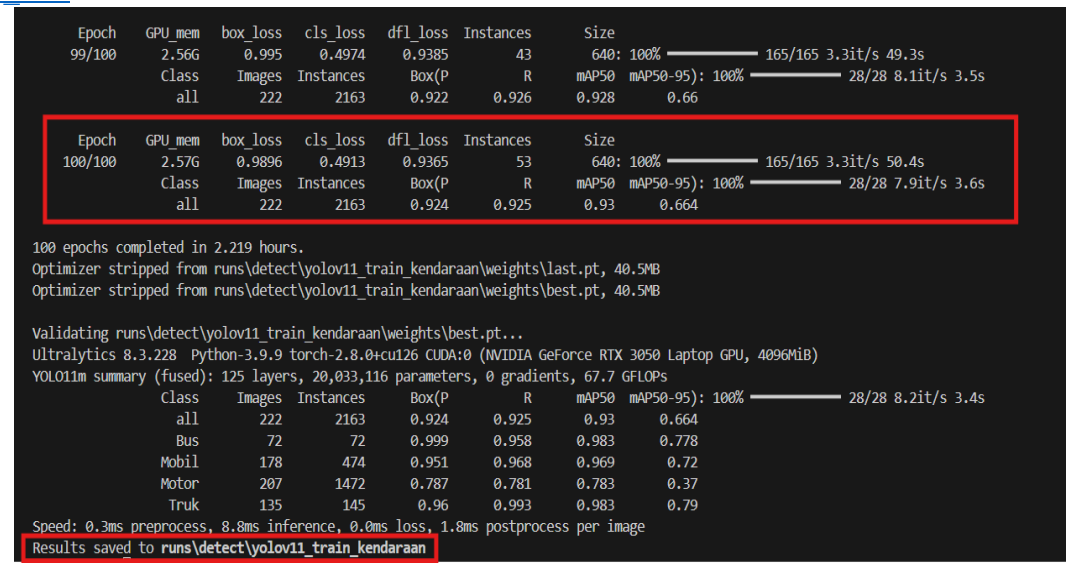
```
cap.release()
out.release()
print("Proses selesai.")
print("Video tersimpan di:", VIDEO_OUTPUT_PATH)
```

Kode program tersebut digunakan untuk menutup akses video dan menyelesaikan program.

H. HASIL DAN PEMBAHASAN

Video Hasil Implementasi Sistem :

https://drive.google.com/drive/folders/1Wk1pA0nNz3RADfAuRUZEBVQyR7QIQLV?usp=drive_link



```
      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
     99/100      2.56G      0.995     0.4974     0.9385         43        640: 100% ━━━━━━━━━━━━ 165/165 3.3it/s 49.3s
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% ━━━━━━━━━━━━ 28/28 8.1it/s 3.5s
                   all        222       2163      0.922      0.926      0.928      0.66

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
    100/100      2.57G     0.9896     0.4913     0.9365         53        640: 100% ━━━━━━━━━━━━ 165/165 3.3it/s 50.4s
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% ━━━━━━━━━━━━ 28/28 7.9it/s 3.6s
                   all        222       2163      0.924      0.925       0.93      0.664

100 epochs completed in 2.219 hours.
Optimizer stripped from runs\detect\yolov11_train_kendaraan\weights\last.pt, 40.5MB
Optimizer stripped from runs\detect\yolov11_train_kendaraan\weights\best.pt, 40.5MB

Validating runs\detect\yolov11_train_kendaraan\weights\best.pt...
Ultralytics 8.3.228  Python-3.9.9 torch-2.8.0+cu126 CUDA:0 (NVIDIA GeForce RTX 3050 Laptop GPU, 4096MiB)
YOLO11m summary (fused): 125 layers, 20,033,116 parameters, 0 gradients, 67.7 GFLOPs
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% ━━━━━━━━━━━━ 28/28 8.2it/s 3.4s
                   all        222       2163      0.924      0.925       0.93      0.664
                   Bus         72         72      0.999      0.958      0.983      0.778
                 Mobil        178        474      0.951      0.968      0.969       0.72
                 Motor        207       1472      0.787      0.781      0.783       0.37
                  Truk        135        145       0.96      0.993      0.983       0.79
Speed: 0.3ms preprocess, 8.8ms inference, 0.0ms loss, 1.8ms postprocess per image
Results saved to runs\detect\yolov11_train_kendaraan
```

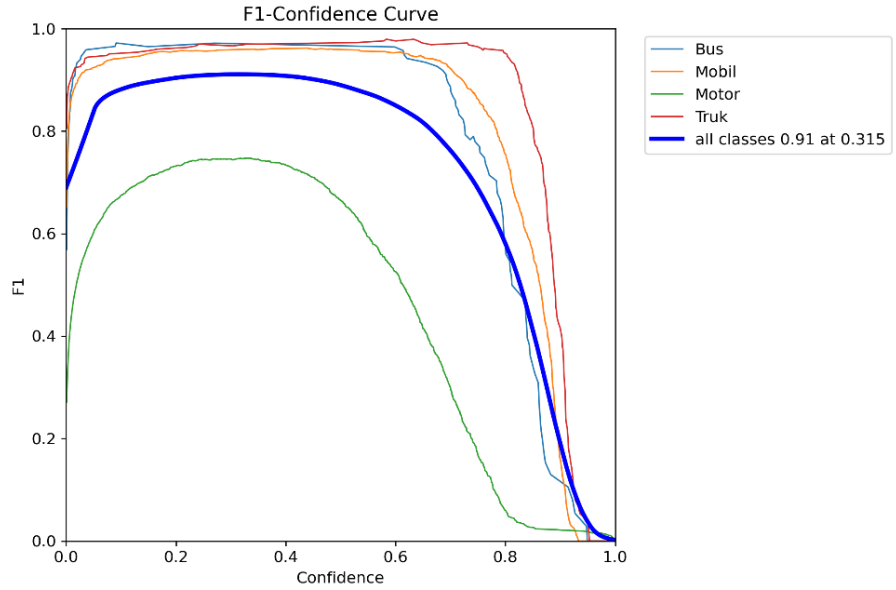
Gambar 3. Output Terminal Sebelum Pengembangan (Train Default).

```
      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
    100/100      1.93G       1.06      0.521     0.9426         12        416: 100% ━━━━━━━━━━━━ 371/371 7.3it/s 50.6s
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% ━━━━━━━━━━━━ 61/61 15.4it/s 4.0s
                   all        242       2188      0.919      0.909      0.915       0.64

100 epochs completed in 1.565 hours.
Optimizer stripped from D:\KCBUAS\runs\detect\fold_0\weights\last.pt, 40.5MB
Optimizer stripped from D:\KCBUAS\runs\detect\fold_0\weights\best.pt, 40.5MB

Validating D:\KCBUAS\runs\detect\fold_0\weights\best.pt...
Ultralytics 8.3.228  Python-3.9.9 torch-2.8.0+cu126 CUDA:0 (NVIDIA GeForce RTX 3050 Laptop GPU, 4096MiB)
YOLO11m summary (fused): 125 layers, 20,033,116 parameters, 0 gradients, 67.7 GFLOPs
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% ━━━━━━━━━━━━ 61/61 17.6it/s 3.5s
                   all        242       2188      0.913       0.91      0.913      0.642
                   Bus         72         72      0.986      0.956      0.975      0.764
                 Mobil        178        474      0.953      0.966       0.97      0.703
                 Motor        227       1497      0.769      0.725      0.721      0.312
                  Truk        135        145      0.944      0.993      0.987      0.788
Speed: 0.3ms preprocess, 6.9ms inference, 0.0ms loss, 2.0ms postprocess per image
Results saved to D:\KCBUAS\runs\detect\fold_0
```

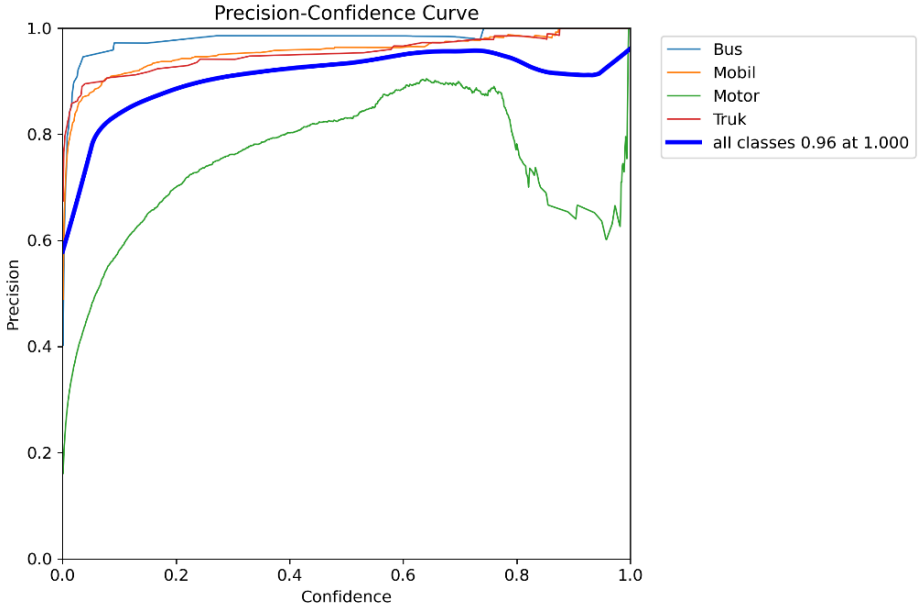
Gambar 4. Output Terminal Setelah Pengembangan.

Meskipun kelas Motor memiliki jumlah gambar dan instance terbanyak dalam keseluruhan dataset pelatihan, kinerja deteksi yang diukur dengan mAP50 (mean Average Precision pada Intersection over Union 50%) justru merupakan yang paling rendah, yaitu hanya sekitar 0.72, disebabkan oleh tingginya variasi visual dan kompleksitas bentuk yang dimiliki oleh objek Motor. Motor memiliki berbagai tipe, ukuran, dan seringkali mengalami tumpang tindih (overlap) di jalan, serta pengambilan sudut pandang yang sangat bervariasi. Keragaman yang tidak terkontrol membuat model kesulitan untuk mempelajari pola visual yang konsisten, sehingga meskipun memiliki kuantitas data yang tinggi (lebih dari 1.400 instance), model tidak mampu melakukan generalisasi dengan baik. Sebaliknya, kelas Bus dan Truk, meskipun jumlah instance-nya jauh lebih sedikit, memiliki bentuk dan ciri visual yang lebih seragam dan konsisten (ukuran besar, siluet khas). Pola visual yang lebih sederhana akan mudahkan model untuk mempelajarinya, sehingga nilai mAP50 dan mAP50-95 (mAP rata-rata dari IoU 50% hingga 95%) mereka justru lebih tinggi. Hasil akhirnya yaitu kuantitas data tidak selalu menjamin performa yang lebih baik, apabila kompleksitas dan keragaman objek tidak dapat dipetakan secara efektif oleh model.

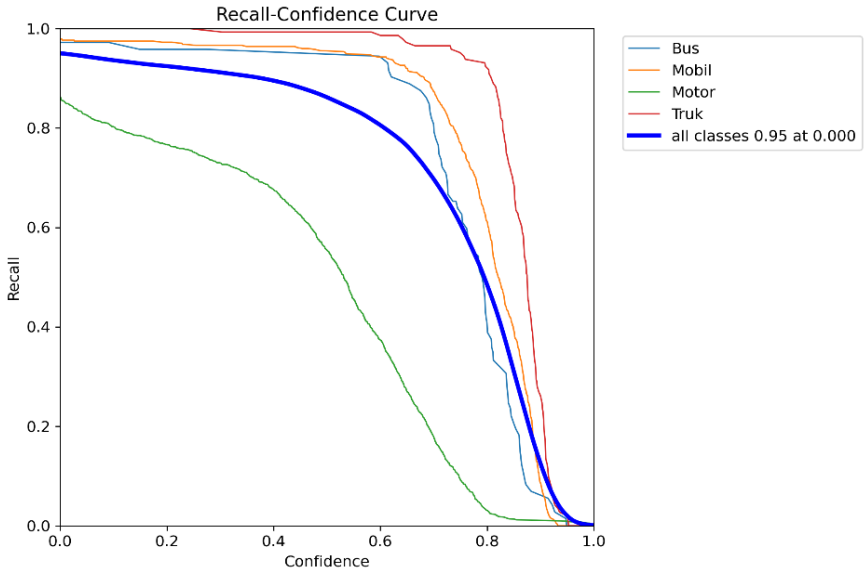
Perbedaan antara output hasil training standar dan output yang menyertakan K-Fold Cross Validation terletak pada validitas dan stabilitas statistik kinerja yang diukur. Hasil training standar hanya menampilkan nilai metrik (loss, precision, recall, mAP) berdasarkan satu set validasi tunggal. Hasilnya memberikan gambaran umum dan berisiko mengalami overfitting terhadap subset data validasi tertentu, sehingga tidak menjamin bahwa model dapat bergeneralisasi dengan baik pada keseluruhan dataset. Sementara itu, output yang berasal dari K-Fold Cross Validation (misalnya K=5) mencerminkan rata-rata performa model yang diuji secara bergantian pada lima pembagian data yang berbeda. Pengujian model pada berbagai variasi data, sehingga hasil mAP dan metrik lainnya yang ditampilkan menjadi lebih stabil, kurang bias, dan merepresentasikan kinerja sebenarnya yang lebih valid secara statistik. Model yang dievaluasi dengan K-Fold cenderung lebih robust saat diimplementasikan untuk prediksi nyata karena telah diuji pada beragam skenario data.



Gambar 5. F1-Confidence Curve.

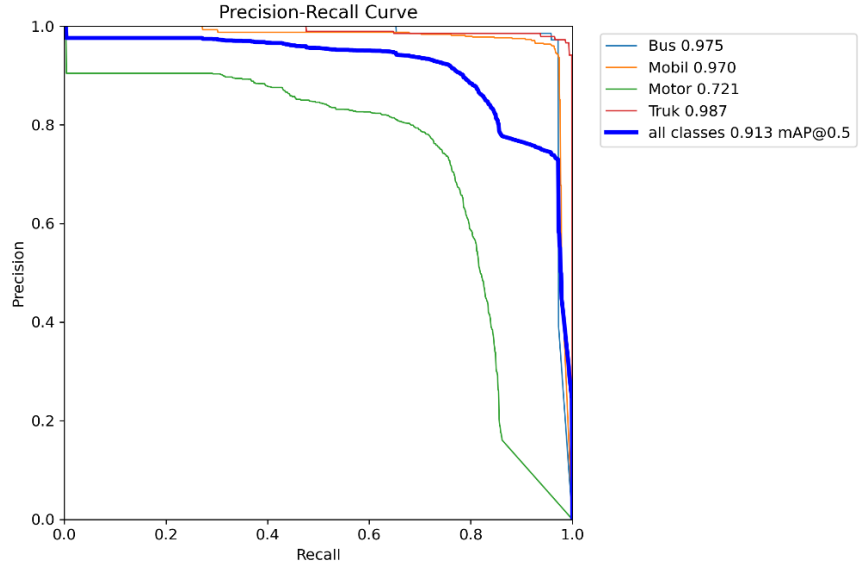


Gambar 6. Precision Confidence Curve.



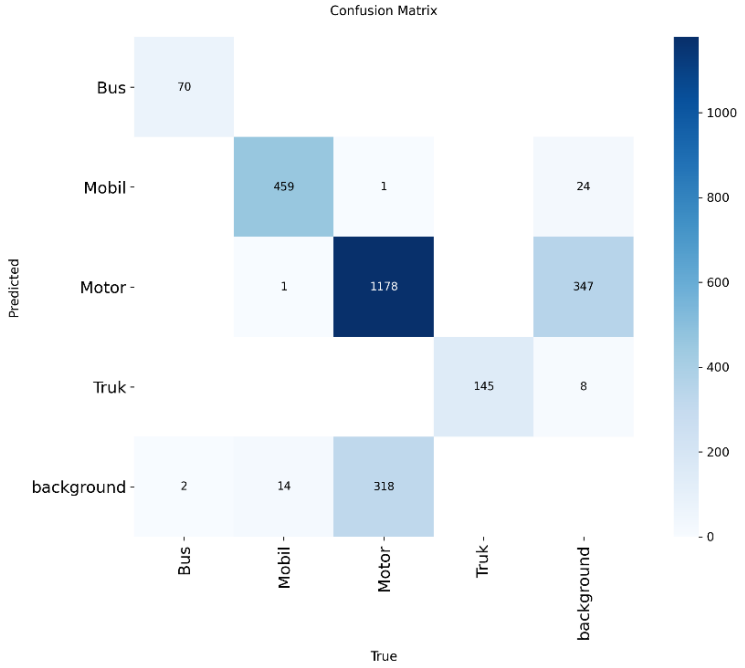
Gambar 7. Recall Confidence Curve.

Recall-Confidence, Precision-Confidence, dan F1-Confidence berfungsi menentukan titik operasi optimal model. Kurva Recall-Confidence menunjukkan bahwa kenaikan ambang Confidence (Sumbu X) akan menyebabkan penurunan Recall (Sumbu Y), yang berarti penetapan Confidence terlalu tinggi (misalnya 0.8) akan menyebabkan model melewatkan banyak objek. Sebaliknya, Kurva Precision-Confidence menunjukkan bahwa kenaikan Confidence akan meningkatkan Precision (Sumbu Y), yang berarti penetapan Confidence terlalu rendah (misalnya 0.1) akan menghasilkan banyak prediksi salah atau "alarm palsu". Titik F1-Confidence Curve merepresentasikan keseimbangan optimal antara Precision dan Recall. Analisis kurva menunjukkan bahwa puncak F1 Score (0.91) tercapai pada Ambang Batas Confidence 0.315. Penggunaan 0.315 sebagai ambang batas operasional model akan memberikan hasil prediksi yang paling seimbang, meminimalkan kesalahan dan memaksimalkan jangkauan deteksi secara keseluruhan. Jika Confidence yang digunakan selain 0.315, kinerja model secara keseluruhan akan menurun.

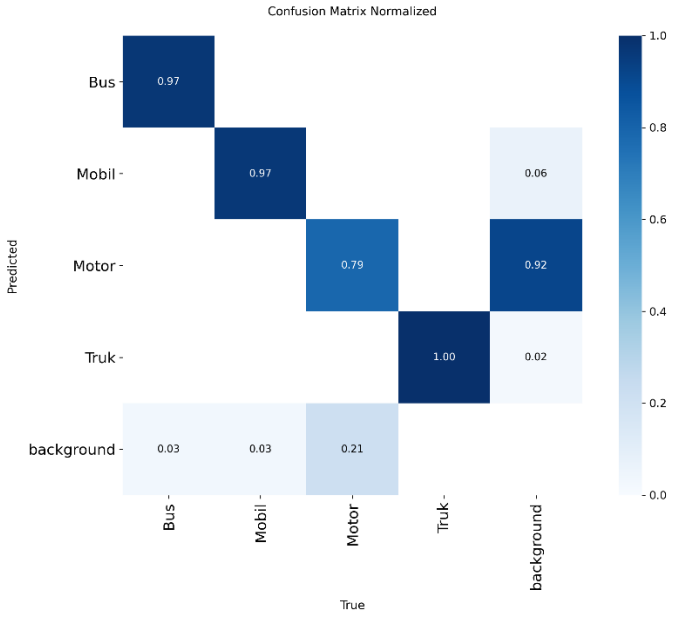


Gambar 8. Precision Recall Curve.

Kurva Precision-Recall (P-R Curve), yang mengukur kinerja intrinsik model, menunjukkan bahwa model memiliki kinerja terbaik pada kelas Truk, Bus, dan Mobil karena kurva mereka berada dekat dengan sudut kanan atas (Recall tinggi dan Precision tinggi). Kurva Motor yang jauh lebih rendah (mAP 0.721) mengindikasikan bahwa Motor adalah kelas yang paling rentan gagal dideteksi atau salah diklasifikasikan pada data uji. Hal ini diperkuat oleh Confusion Matrix ternormalisasi, di mana teramati angka 0.21 pada kolom 'Motor' dan baris 'background'. Angka ini berarti 21% dari objek Motor yang sebenarnya diuji justru terlewatkan dan dianggap sebagai latar belakang oleh model, membuktikan bahwa masalah utama model saat menghadapi data uji adalah kinerja buruk pada kelas Motor, terutama karena kesulitan melokalisasi dan mengklasifikasikannya.



Gambar 9. Confusion Matrix.

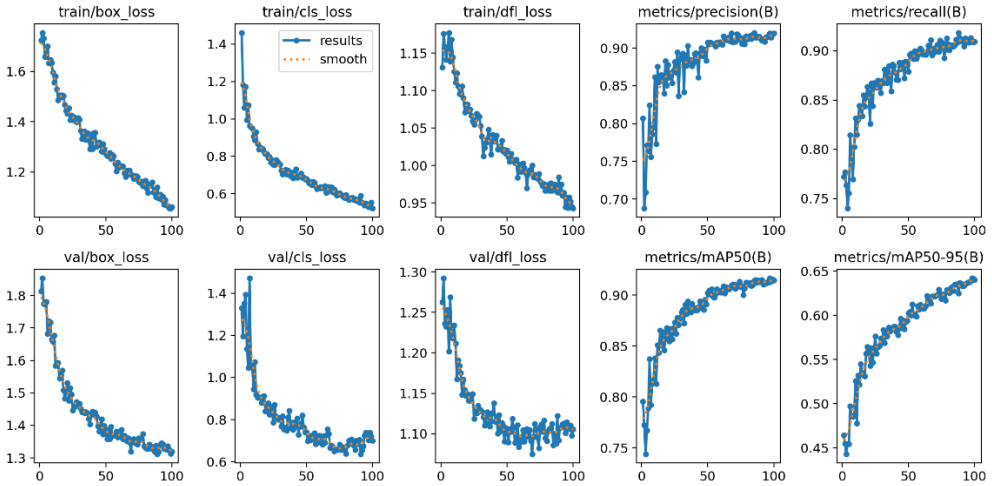


Gambar 10. Confusion Matrix Normalized.

Kedua matriks pada gambar diatas, sumbu horizontal (True) merepresentasikan label objek yang sebenarnya, sedangkan sumbu vertikal (Predicted) merepresentasikan klasifikasi yang dihasilkan oleh model. Nilai-nilai diagonal utama menunjukkan kasus True Positive (prediksi benar), di mana model berhasil mengklasifikasikan objek sesuai dengan label sebenarnya.

Hasil model menunjukkan kinerja deteksi yang baik untuk kelas Bus, Mobil, dan Truk. Pada matriks ternormalisasi, tingkat akurasi untuk Truk mencapai 1.00 (100%), sementara Bus dan Mobil juga menunjukkan tingkat deteksi yang sangat tinggi, yaitu 0.97 (97%) untuk masing-masing kelas. Angka-angka tersebut mengindikasikan bahwa model sangat jarang salah mengklasifikasikan atau melewatkan objek-objek berukuran besar dan berbentuk konsisten tersebut.

Masalah utama model terpusat pada kinerja deteksi kelas Motor. Pertama, terjadi kesalahan lokalisasi yang tinggi (False Negative), di mana terlihat nilai 0.21 pada kolom 'Motor' dan baris 'background' pada matriks ternormalisasi. Angkanya berarti 21% dari objek Motor yang sebenarnya hadir dalam data uji gagal dideteksi oleh model dan diklasifikasikan sebagai 'latar belakang', setara dengan 318 kasus dalam matriks mentah. Kedua, terjadi kesalahan klasifikasi ganda (False Positive), di mana model sering keliru mengklasifikasikan objek lain sebagai Motor, ditunjukkan dengan 347 kasus di mana 'latar belakang' diklasifikasikan secara keliru sebagai Motor. Temuan dari Confusion Matrix memvalidasi bahwa tantangan model adalah kinerja buruk pada kelas Motor, yang ditandai dengan tingginya kasus kegagalan deteksi dan tingginya prediksi palsu, meskipun jumlah data pelatihan untuk kelas banyak.



Gambar 11. Grafik Loss dan Metrik Selama Pelatihan

Grafik diatas menampilkan bagaimana kesalahan model berubah sepanjang 100 Epoch, terdiri dari lima plot untuk data latih dan lima plot untuk data validasi. Plot Loss (train/box_loss, val/box_loss, dll.) menunjukkan nilai kesalahan model, kurva penurunan loss yang stabil hingga epoch ke-100 mengindikasikan bahwa model berhasil mempelajari fitur data dan meminimalkan kesalahan prediksi, yang menjamin prediksi yang akurat pada data uji. Sebaliknya, Plot Metrik Kinerja (metrics/mAP50, metrics/mAP50-95) menunjukkan persentase kinerja dengan kurva kenaikan yang stabil dan mencapai nilai tinggi (sekitar 0.90) yang berarti model akan memiliki tingkat Ketepatan (Precision) dan Jangkauan (Recall) yang tinggi saat diuji pada data baru.

I. KESIMPULAN

Dari proyek ini disimpulkan dapat untuk mendeteksi kendaraan secara real-time, melacak pergerakannya, dan menghitung kecepatan berdasarkan perubahan posisi dalam frame video menggunakan transformasi perspektif (Homography).

Model yang digunakan telah dilatih selama 100 epoch menggunakan dataset kendaraan yang telah dianotasi, dan memberikan hasil yang baik pada proses pengujian. Sistem ini tidak hanya mampu mendeteksi jenis kendaraan, tetapi juga mengukur kecepatan dengan akurasi tinggi, kemudian menganalisis tingkat risiko kecelakaan berdasarkan kecepatan kendaraan yang terdeteksi.

Penggunaan YOLOv11 sebagai model deteksi, ByteTrack untuk pelacakan objek, serta metode estimasi kecepatan dari pergerakan pixel ke meter memungkinkan proses dilakukan secara real-time dan responsif. Untuk mengurangi fluktuasi kecepatan yang tidak akurat, digunakan teknik Exponential Moving Average (EMA) sehingga sistem dapat menampilkan hasil yang lebih stabil dan realistis.

Berdasarkan pengujian yang dilakukan, sistem mampu melakukan deteksi dan estimasi kecepatan secara efektif, sehingga berpotensi digunakan dalam analisis pelanggaran kecepatan, pemantauan lalu lintas, atau sistem peringatan dini kecelakaan (early warning system).

J. REFERENSI

[1] Deteksi dan Estimasi Kecepatan Kendaraan dalam Sistem Pengawasan Lalu Lintas Menggunakan Pengolahan Citra, M. Zulfikri et al., Techno.COM, Vol. 20, No.3, Agustus 2021, hlm. 455-467. Available : https://www.researchgate.net/publication/354192599_Deteksi_dan_Estimasi_Kecepatan_Kendaraan_dalam_Sistem_Pengawasan_Lalu_Lintas_Menggunakan_Pengolahan_Citra.

[2] Sistem Deteksi Kendaraan Menggunakan Metode Optical Flow Guna Menghitung Kecepatan Kendaraan, O. Safitri, B. Susilo & A. Erlansari, Rekursif: Jurnal Informatika, Vol.7, No.1, 2019. Available : https://ejournal.unib.ac.id/rekursif/article/view/5770?utm.

[3] Pengembangan Sistem Deteksi Kecepatan Kendaraan Overspeed dengan YOLOv8 di Area Jalan Tol, A. A. Fauzi, A. Wahid & A. B. Kaswar, JIMU: Jurnal Ilmiah Multidisipliner, Vol. 3, No.2, 2025. Available : https://ojs.smkmerahputih.com/index.php/jimu/article/view/681?utm_

[4] Deteksi Kecepatan Kendaraan Berbasis Data Ponsel Dalam Perspektif Spasial, E. L. B. Pinem et al., Geodika: Jurnal Kajian Ilmu dan Pendidikan Geografi, Vol. 7, No.1, Juni 2023. Available : https://e-journal.hamzanwadi.ac.id/index.php/gdk/article/view/7166/pdf?utm.

[5] Aplikasi Penghitung Kecepatan Mobil dengan Akurasi Tinggi, H. Henny, Telekontran, Vol. 11, No.2, Oktober 2023. Available : https://ojs.unikom.ac.id/index.php/telekontran/article/download/10900/4235/42023?utm.