APLIKASI COMPUTER VISION DALAM PEMBUATAN PROTOTIPE DETEKSI KENDARAAN DAN LAHAN PARKIR MENGGUNAKAN LIBRARY YOLOv8

TUGAS AKHIR

STEVEN NATA

NIM : 312010011

PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI DAN DESAIN
UNIVERSITAS MA CHUNG
MALANG
2024

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

DAFTAR PUSTAKA

Zadeh, N. R. N., & Cruz, J. C. D. (2016). *Smart urban parking detection system*. https://doi.org/10.1109/iccsce.2016.7893601

Angelidou, M. (2014). Smart city policies: A spatial approach. *Cities*, *41*, S3–S11. https://doi.org/10.1016/j.cities.2014.06.007

Shinde, P. P., & Shah, S. (2018). *A Review of Machine Learning and Deep Learning Applications*. https://doi.org/10.1109/iccubea.2018.8697857

Weinstein, B. G. (2017). A computer vision for animal ecology. *Journal of Animal Ecology*, *87*(3), 533–545. https://doi.org/10.1111/1365-2656.12780

Wu, Y. C., & Feng, J. W. (2017). Development and Application of Artificial Neural Network. *Wireless Personal Communications*, *102*(2), 1645–1656. https://doi.org/10.1007/s11277-017-5224-x

Andreas, Gidion, et al. "Analisa Dampak Lalu Lintas Akibat Keterbatasan Lahan pada Ruang Parkir Pasar Tradisional di Kota Semarang." *Jurnal Karya Teknik Sipil S1 Undip*, vol. 6, no. 1, 2017, pp. 438-449.

Classification: True vs. False and Positive vs. Negative. (n.d.). Google for Developers. https://developers.google.com/machine-learning/crash-course/classification/true-false-positive-negative

Lin, Q., Ye, G., Wang, J., & Liu, H. (2021). RoboFlow: a Data-centric Workflow Management System for Developing AI-enhanced Robots. In *Blue Sky Papers, 5th Conference on Robot Learning (CoRL 2021), London, UK*. https://sites.google.com/u.northwestern.edu/roboflow

Pandey, A. K. (2024, June 13). SiLU (Sigmoid Linear Unit) activation function - Abhishek Kumar Pandey - Medium. *Medium*. https://medium.com/@akp83540/silu-sigmoid-linear-unit-activation-function-d9b6845f0c81

Raza, M. (2024, January 8). Yolo V8: A Deep Dive Into Its Advanced Functions and New Features. *Medium*. https://medium.com/@mujtabaraza194/yolo-v8-a-deep-dive-into-its-advanced-functions-and-new-features-f008599fe604

Setiawan, R. (2022, January 18). *Apa itu Arduino? Pahami Lebih Mendalam*. Dicoding Blog. https://www.dicoding.com/blog/apa-itu-arduino/

Torres, J. (2024, January 13). *What is YOLOv8? Exploring its Cutting-Edge Features - YOLOv8*. YOLOv8. https://yolov8.org/what-is-yolov8/

Ultralytics. (n.d.). *Train,val and test sets · Issue #3490 · ultralytics/ultralytics*. GitHub. https://github.com/ultralytics/ultralytics/issues/3490

LAMPIRAN

A. File YOLOimage

```
001    import math
002    import cv2
003    import cvzone
004    import pickle
005    import numpy as np
006    from ultralytics import YOLO
007    import serial
008    import time
009
010    # --------------- Variables -------------------
011    p1 = 0
012    p2 = 3
013    p3 = 2
014    roadCam = 1
015    x, y = 400, 400
016    color = (0, 200, 0)
017    count = 40
018    avanow1 = 6
019    avanow2 = 6
020    avanow3 = 4
021
022    sign = " WAIT ! "
023    polyroad = 'Road.p'
024    obj_road = open(polyroad, 'rb')
025    roi_road = pickle.load(obj_road)
026    obj_road.close()
027
028
029    statusPalang1 = False
030    statusPalang2 = False  # assign status palang parkiran
031    statusPalang3 = False
032
033    polygonKeluar1 = 'parkiran1_out.p'
034    polygonKeluar2 = 'parkiran2_out.p'  # assign semua polygons
   Keluar parkiran
035    polygonKeluar3 = 'parkiran3_out.p'
036
037    polygonParkir1 = 'parkiran1.p'
038    polygonParkir2 = 'parkiran2.p'  # assign semua polygons
   parkiran
039    polygonParkir3 = 'parkiran3.p'
040
041    model_path = "D:\YOLO TA\model\model very
   accurate\\best.pt"
042
043
044    confidence = 0.5
045
046    class_names = ["SEDAN", "SUV", "TRUCK"]
047
048    file_obj_parkir1 = open(polygonParkir1, 'rb')
049    file_obj_parkir2 = open(polygonParkir2, 'rb')
050    file_obj_parkir3 = open(polygonParkir3, 'rb')
```

```
051
052    roisParkir1 = pickle.load(file_obj_parkir1)
053    roisParkir2 = pickle.load(file_obj_parkir2)
054    roisParkir3 = pickle.load(file_obj_parkir3)
055
056    file_obj_keluar1 = open(polygonKeluar1, 'rb')
057    file_obj_keluar2 = open(polygonKeluar2, 'rb')
058    file_obj_keluar3 = open(polygonKeluar3, 'rb')
059
060    roisKeluar1 = pickle.load(file_obj_keluar1)
061    roisKeluar2 = pickle.load(file_obj_keluar2)
062    roisKeluar3 = pickle.load(file_obj_keluar3)  # OPEN POLYGON
   FILE
063
064    file_obj_keluar1.close()
065    file_obj_keluar2.close()
066    file_obj_keluar3.close()
067
068    file_obj_parkir1.close()
069    file_obj_parkir2.close()
070    file_obj_parkir3.close()
071
072    arduino = serial.Serial(port='COM8', baudrate=9600,
   timeout=.1)  # connect to arduino
073
074    cam_width, cam_height = 1280, 720  # CAM RES
075
076    cap1 = cv2.VideoCapture(p1)  # WEB CAM ASSIGN
077    cap2 = cv2.VideoCapture(p2)
078    cap3 = cv2.VideoCapture(p3)
079    cap4 = cv2.VideoCapture(roadCam)
080
081    cap2.set(3, cam_width)
082    cap2.set(4, cam_height)
083    cap3.set(3, cam_width)
084    cap3.set(4, cam_height)  # SET ALL FRAME SIZE
085    cap1.set(3, cam_width)
086    cap1.set(4, cam_height)
087    cap4.set(3, cam_width)
088    cap4.set(4, cam_height)
089
090
091    model = YOLO(model_path)  # SELECT MODEL USED
092
093    def count_Spaces(ava_spaces, _object_list,
   _parking_spaces):
094        for parking_space in _parking_spaces:
095            ret = 0
096            empty = True
097
098            # Convert polygon to numpy array and reshape
099            parking_space = np.array(parking_space,
   np.int32).reshape((-1, 1, 2))
0100
0101            # Check if any car is present in this polygon
0102            for obj in _object_list:
0103                car_center = obj["center"]
0104                result = cv2.pointPolygonTest(parking_space,
   car_center, False)
```

```
0105                if result > 0:
0106                    empty = False
0107                    ret += 1
0108                    break
0109            if not empty:
0110                ava_spaces -= ret
0111        return ava_spaces
0112
0113    def overlay_polygons(_image, _object_list, _parking_spaces,
    _draw_occupied=False):
0114        overlay = _image.copy()
0115        global is_empty
0116        for parking_space in _parking_spaces:
0117            is_empty = True
0118
0119            # Convert polygon to numpy array and reshape
0120            parking_space = np.array(parking_space,
    np.int32).reshape((-1, 1, 2))
0121
0122            # Check if any car is present in this polygon
0123            for obj in _object_list:
0124                car_center = obj["center"]
0125                result = cv2.pointPolygonTest(parking_space,
    car_center, False)
0126                if result > 0:
0127                    is_empty = False
0128                    break
0129
0130            if is_empty:
0131                cv2.fillPoly(overlay, [parking_space], (0, 255,
    0))  # Green for empty space
0132            if not is_empty and _draw_occupied:
0133                cv2.fillPoly(overlay, [parking_space], (0, 0,
    255))  # Red for occupied space
0134
0135        cv2.addWeighted(overlay, 0.35, _image, 0.65, 0, _image)
0136        return is_empty
0137
0138    def get_object_list_yolo(_model, _img, _class_names,
    _confidence=0.5, draw=True):
0139        _results = _model(_img, stream=False, verbose=False)
0140        _object_list = []
0141        for r in _results:
0142            boxes = r.boxes
0143            for box in boxes:
0144                conf = math.ceil((box.conf[0] * 100)) / 100
0145                if conf > _confidence:
0146                    x1, y1, x2, y2 = box.xyxy[0]
0147                    x1, y1, x2, y2 = int(x1), int(y1), int(x2),
    int(y2)
0148                    w, h = x2 - x1, y2 - y1
0149                    center = x1 + (w // 2), y1 + (h // 2)
0150                    class_name = _class_names[int(box.cls[0])]
0151
0152                    _object_list.append({"bbox": (x1, y1, w,
    h),
0153                                         "center": center,
0154                                         "conf": conf,
0155                                         "class": class_name})
```

```
0156
0157                    if draw:
0158                        cvzone.cornerRect(_img, (x1, y1, w, h))
0159                        cvzone.putTextRect(_img, f'{class_name}
   {conf}',
0160                                           (max(0, x1), max(35,
   y1)), scale=2, thickness=1)
0161        return _object_list
0162
0163    def write(x):
0164        arduino.write(bytes(x, 'utf-8'))
0165        time.sleep(0.05)
0166
0167
0168    cv2.namedWindow("SUV"), cv2.namedWindow("SEDAN"),
   cv2.namedWindow("TRUCK"), cv2.namedWindow("camRoad")  # NAMING
   ALL WINDOW TO MOVE IT
0169    cv2.moveWindow("SUV", 0, 100), cv2.moveWindow("SEDAN", 482,
   100), cv2.moveWindow("TRUCK", 964, 100),
   cv2.moveWindow("camRoad", 964, 0) #MOVE ALL WINDOWS TO CERTAIN
   COORDINATE
0170
0171
0172    def frameDetectCar():
0173        global center1,center2,center3
0174        global center
0175        global statusPalang1
0176        global statusPalang2
0177        global statusPalang3
0178        global color
0179        global count
0180        global sign
0181        park_spaces = 6
0182        park_spaces_truck = 4
0183        reset = 40
0184        global avanow1
0185        global avanow2
0186        global avanow3
0187        success1, img1 = cap1.read()
0188        success1, img2 = cap2.read()  # READING CAMERA ASSIGNED
0189        success1, img3 = cap3.read()
0190        success1, img4 = cap4.read()
0191
0192        object_list1 = get_object_list_yolo(model, img1,
   class_names, confidence, draw=True)
0193        object_list2 = get_object_list_yolo(model, img2,
   class_names, confidence, draw=True)  # CAMERA FRAME GET
   DETECTED WITH YOLO
0194        object_list3 = get_object_list_yolo(model, img3,
   class_names, confidence, draw=True)
0195        object_list4 = get_object_list_yolo(model, img4,
   class_names, confidence, draw=True)
0196
0197        ava1 = count_Spaces(park_spaces, object_list1,
   roisParkir1)
0198        ava2 = count_Spaces(park_spaces, object_list2,
   roisParkir2)
0199        ava3 = count_Spaces(park_spaces_truck, object_list3,
   roisParkir3)
```

```
0200
0201      # ---------------------------------- PARKIRAN KELUAR --
   ------------------------------------------------------------
   ---------
0202
0203      empty_status1 = overlay_polygons(img1, object_list1,
   roisKeluar1, _draw_occupied=True)
0204      empty_status2 = overlay_polygons(img2, object_list2,
   roisKeluar2, _draw_occupied=True)  # status polygons keluar
0205      empty_status3 = overlay_polygons(img3, object_list3,
   roisKeluar3, _draw_occupied=True)
0206
0207      keluar_space1 = np.array(roisKeluar1,
   np.int32).reshape(
0208          (-1, 1, 2))  # to detect center point polygon
   keluar(convert polygon to array)
0209      keluar_space2 = np.array(roisKeluar2,
   np.int32).reshape((-1, 1, 2))
0210      keluar_space3 = np.array(roisKeluar3,
   np.int32).reshape((-1, 1, 2))
0211
0212      # ------------------------------------------------------
   -------------
0213      # SUV
0214      if ((len(object_list1) != 0) and (empty_status1 ==
   False)):
0215          # Check if any car is present in this polygon
0216          for obj in object_list1:
0217              car_center = obj["center"]
0218              result = cv2.pointPolygonTest(keluar_space1,
   car_center, False)
0219              if result > 0:
0220                  center1 = obj
0221
0222          # Palang 1
0223          if ((center1["class"] == "SUV") and (statusPalang1
   == False)):
0224              # print("open")
0225              # print(center1["class"])
0226              write("2")
0227              statusPalang1 = True
0228              avanow1 = ava1
0229          elif((center1["class"] != "SUV") and (statusPalang1
   == True)):
0230              write("3")
0231
0232      # ------------------------------------------------------
   -------------
0233
0234      # SEDAN
0235      if ((len(object_list2) != 0) and (empty_status2 ==
   False)):
0236
0237          # Check if any car is present in this polygon
0238          for obj in object_list2:
0239              car_center = obj["center"]
0240              result = cv2.pointPolygonTest(keluar_space2,
   car_center, False)
0241              if result > 0:
```

```
0242                    center2 = obj
0243
0244            # Palang 2
0245            if ((center2["class"] == "SEDAN") and
   (statusPalang2 == False)):
0246                # print("open")
0247                # print(center2["class"])
0248                write("4")
0249                statusPalang2 = True
0250                avanow2 = ava2
0251            elif((center2["class"] != "SEDAN") and
   (statusPalang2 == True)):
0252                write("5")
0253
0254
0255        # ------------------------------------------------------
   -------------
0256        # TRUCK
0257        if ((len(object_list3) != 0) and (empty_status3 ==
   False)):
0258            # Check if any car is present in this polygon
0259            for obj in object_list3:
0260                car_center = obj["center"]
0261                result = cv2.pointPolygonTest(keluar_space3,
   car_center, False)
0262                if result > 0:
0263                    center3 = obj
0264
0265            # Palang 3
0266            if ((center3["class"] == "TRUCK") and
   (statusPalang3 == False)):
0267                # print("open")
0268                # print(center3["class"])
0269                write("6")
0270                statusPalang3 = True
0271                avanow3 = ava3
0272            elif((center3["class"] != "TRUCK") and
   (statusPalang3 == True)):
0273                write("7")
0274
0275        # ---------------------------------- PARKIRAN --------
   ------------------------------------------------------------
   -----
0276
0277        overlay_polygons(img1, object_list1, roisParkir1,
   _draw_occupied=True)  # Parkiran
0278        overlay_polygons(img2, object_list2, roisParkir2,
   _draw_occupied=True)
0279        overlay_polygons(img3, object_list3, roisParkir3,
   _draw_occupied=True)
0280
0281        # 1 SUV
0282
0283        if ava1 == 0:
0284            color = (0, 0, 255)  # Red for no available spaces
0285        else:
0286            color = (0, 200, 0)  # Green for available spaces
0287        cvzone.putTextRect(img1, f"Available:
   {ava1}/{str(park_spaces)}", (20, 50), colorR=color)
```

```
0288
0289        # 2 SEDAN
0290
0291        if ava2 == 0:
0292            color = (0, 0, 255)  # Red for no available spaces
0293        else:
0294            color = (0, 200, 0)  # Green for available spaces
0295        cvzone.putTextRect(img2, f"Available:
   {ava2}/{str(park_spaces)}", (20, 50), colorR=color)
0296
0297        # 3 TRUCK
0298
0299        if ava3 == 0:
0300            color = (0, 0, 255)  # Red for no available spaces
0301        else:
0302            color = (0, 200, 0)  # Green for available spaces
0303        cvzone.putTextRect(img3, f"Available:
   {ava3}/{str(park_spaces_truck)}", (20, 50), colorR=color)
0304
0305        # ------------------------------- ROAD ---------------
   ----------------------------------------------------------------
   -------
0306
0307        empty_status_road = overlay_polygons(img4,
   object_list4, roi_road, _draw_occupied=True)
0308
0309        parking_space = np.array(roi_road, np.int32).reshape((-
   1, 1, 2))
0310
0311
0312        if ((len(object_list4) != 0) and (empty_status_road ==
   False)):
0313
0314            # Check if any car is present in this polygon
0315            for obj in object_list4:
0316                car_center = obj["center"]
0317                result = cv2.pointPolygonTest(parking_space,
   car_center, False)
0318                if result > 0:
0319                    center = obj
0320
0321
0322            #Palang 1
0323            if ((center["class"] == "SUV") and (statusPalang1
   == False)):
0324                print("open")
0325                print(center["class"])
0326                write("2")
0327                statusPalang1 = True
0328                count = reset
0329                sign = "Go SUV"
0330                avanow1 = ava1
0331
0332            # Palang 2
0333            if ((center["class"] == "SEDAN") and (statusPalang2
   == False)):
0334                # print("open")
0335                # print(center["class"])
0336                write("4")
```

```
0337                statusPalang2 = True
0338                count = reset
0339                sign = "Go SEDAN"
0340                avanow2 = ava2
0341
0342
0343            # Palang 3
0344            if ((center["class"] == "TRUCK") and (statusPalang3
   == False)):
0345                print("open")
0346                print(center["class"])
0347                write("6")
0348                statusPalang3 = True
0349                count = reset
0350                sign = "Go TRUCK"
0351                avanow3 = ava3
0352
0353        cvzone.putTextRect(img4, f" {sign} ", (20, 50),
   colorR=(0, 255, 0))
0354
0355    #--------------------------------- CLOSE ALL PALANG AFTER
   SPECIFIC AMOUNT OF TIME ----------------------------------------
   -----------
0356
0357        if(count == 0):
0358            if (statusPalang1 == True):
0359                print("close")
0360                write("3")
0361                statusPalang1 = False
0362            if (statusPalang2 == True):
0363                print("close")
0364                write("5")
0365                statusPalang2 = False
0366            if (statusPalang3 == True):
0367                print("close")
0368                write("7")
0369                statusPalang3 = False
0370            sign = " WAIT ! "
0371            count = reset
0372        count-=1
0373    # --------------------------------- CLOSE ALL PALANG WHEN
   CAR GOT IN ----------------------------------------------------
0374        #----- SUV ------
0375
0376        if(statusPalang1 == True):
0377            if (ava1 < avanow1):
0378                write("3")
0379                statusPalang1 = False
0380                sign = "WAIT! ... "
0381        if(statusPalang2 == True):
0382            if (ava2 < avanow2):
0383                write("5")
0384                statusPalang2 = False
0385                sign = "WAIT! ... "
0386        if(statusPalang3 == True):
0387            if (ava3 < avanow3):
0388                write("7")
0389                statusPalang3 = False
0390                sign = "WAIT! ... "
```

```
0391        print("s1 ",ava1,"dan",avanow1)
0392        print("s2 ", ava2, "dan", avanow2)
0393        print("s3 ", ava3, "dan", avanow3)
0394        #-------------------------------------------------------
    ------------------------------------------------------------
    -------
0395        # cv2.imshow("camRoad",img4)
0396        # cv2.imshow("Image1", img1)
0397        # cv2.imshow("Image2", img2)  # SHOW CAMERA FRAME full
    720 p
0398        # cv2.imshow("Image3", img3)
0399
0400        cv2.imshow("camRoad",cv2.resize(img4, (640, 360)))
0401        cv2.imshow("SUV"    , cv2.resize(img1, (640, 360)))
0402        cv2.imshow("SEDAN"  , cv2.resize(img2, (640, 360)))  #
    SHOW CAMERA FRAME AND RESIZE TO 480x272
0403        cv2.imshow("TRUCK"  , cv2.resize(img3, (640, 360)))
0404
0405        cv2.waitKey(1)
0406
0407
0408    while True:
0409        frameDetectCar()
```

B. File takeImageInterval dan get_roi parking space

```
001    import cv2
002
003
004
005    CAR_COUNT = "1"
006
007    cam_width, cam_height = 1280, 720
008    path = "D:\\YOLO
   TA\\Imagedataset\\NullImage\\"+CAR_COUNT+"_"
009    count = 1
010    cam_port = 0
011    cam = cv2.VideoCapture(cam_port)
012    cam.set(3,cam_width)
013    cam.set(4,cam_height)
014
015    def save(path, varImg):
016
017        cv2.imwrite(path,varImg)
018    while True:
019        result, image = cam.read()
020        cv2.imshow("Image", cv2.resize(image,(854,480)))
021
022        if result:
023            save(path +"_"+ str(count) + ".jpg", image)
024            cv2.waitKey(2000)
025            print(".....SAVED!!")
026            count += 1
```

```
001    # Import necessary libraries
002    import cv2  # OpenCV library for computer vision tasks
003    import numpy as np
004    import pickle  #
005
006    # Open a connection to the webcam
007    cap = cv2.VideoCapture(3)  # For Webcam
008
009    # Set the width and height of the webcam frame
010    width, height = 1280, 720
011    cap.set(3, width)
012    cap.set(4, height)
013
014
015    # Total number of parking spaces to mark
016    totalSpaces = 1
017
018    # List to store all polygons (each representing a parking
   space)
019    polygons = []
020
021    # Temporary list to store the four points of the current
   polygon being marked
022    current_polygon = []
023
024    # Counter to keep track of how many polygons have been
   created
025    counter = 0
```

```
026
027    # Function to handle mouse events (used to mark points for
   polygons)
028    def mousePoints(event, x, y, flags, params):
029        global counter, current_polygon
030
031        # If left mouse button is clicked
032        if event == cv2.EVENT_LBUTTONDOWN:
033            # Append the clicked point (x, y) to the
   current_polygon list
034            current_polygon.append((x, y))
035
036            # If we have collected four points for one polygon
037            if len(current_polygon) == 4:
038                polygons.append(current_polygon)  # Add the
   polygon to the list
039                current_polygon = []  # Reset for the next
   polygon
040                counter += 1  # Increment the counter
041                print(polygons)  # Print the collected polygons
042
043    # Main loop for capturing video and marking parking spaces
044    while True:
045        # Read a frame from the webcam
046        success, img = cap.read()
047
048        # If we have collected all 12 polygons, then save and
   exit the loop
049        if counter == totalSpaces:
050            fileObj = open('Road.p', 'wb')
051            pickle.dump(polygons, fileObj)  # Save the polygons
   to a file
052            fileObj.close()
053            print("Saved all polygon points.")
054            break
055
056        # Draw the collected polygons on the image
057        for polygon in polygons:
058            cv2.polylines(img, [np.array(polygon)],
   isClosed=True, color=(0, 255, 0), thickness=2)
059
060        # Display the image with marked polygons
061        cv2.imshow("Original Image", img)
062
063        # Set the mouse callback function for marking points
064        cv2.setMouseCallback("Original Image", mousePoints)
065
066        # Wait for a key press (1 ms delay)
067        cv2.waitKey(1)
```

APLIKASI COMPUTER VISION DALAM PEMBUATAN PROTOTIPE DETEKSI KENDARAAN DAN LAHAN PARKIR MENGGUNAKAN LIBRARY YOLOv8

Steven Nata[1], Windra Swastika[2], Romy B. Widodo[3]

Universitas Ma Chung

Abstrak

Peningkatan penggunaan kendaraan pribadi yang pesat telah memperburuk tantangan dalam mengelola ketersediaan lahan parkir. Fasilitas parkir yang dikelola dengan buruk sering kali menyebabkan kesulitan bagi pemilik kendaraan, yang berdampak pada kemacetan lalu lintas dan parkir yang tidak teratur.

Untuk mengatasi masalah ini, studi ini mengeksplorasi penerapan teknik Computer Vision, khususnya menggunakan model You Only Look Once (YOLO), untuk meningkatkan manajemen lahan parkir dengan mendeteksi jenis kendaraan dan mengidentifikasi tempat parkir yang tersedia secara akurat. Model YOLO dilatih dan diintegrasikan dengan sistem yang terdiri dari empat kamera, masing-masing memantau area tertentu.

Model ini menunjukkan kinerja yang kuat, mencapai nilai F1-score antara 0,6 hingga 0,83, dengan nilai kepercayaan antara 0,7 dan 0,8. Pendekatan ini menawarkan solusi yang menjanjikan untuk meningkatkan efisiensi manajemen lahan parkir, yang berpotensi mengurangi masalah terkait lalu lintas di daerah perkotaan.

Kata kunci: *Computer Vision*, *Deep Learning*, Lahan parkir, YOLO

APPLICATION OF COMPUTER VISION IN DEVELOPING A PROTOTYPE FOR VEHICLE AND PARKING SPACE DETECTION USING THE YOLOv8 LIBRARY

Steven Nata[1], Windra Swastika[2], Romy B. Widodo[3]

Universitas Ma Chung

Abstract

The rapid increase in private vehicle usage has exacerbated the challenge of managing available parking spaces. Poorly managed parking facilities often result in difficulties for vehicle owners, contributing to traffic congestion and disorganized parking.

To address these issues, this study explores the application of Computer Vision techniques, specifically utilizing the You Only Look Once (YOLO) model, to enhance parking space management by accurately detecting vehicle types and identifying available parking spots. The YOLO model was trained and integrated with a system of four cameras, each monitoring a designated area.

The model demonstrated robust performance, achieving an F1-score ranging from 0.6 to 0.83, with confidence values between 0.7 and 0.8. This approach presents a promising solution for improving the efficiency of parking space management, potentially alleviating traffic-related problems in urban areas.

Keywords: Computer Vision, Deep Learning, Parking Spaces , YOLO

BAB I
PENDAHULUAN

1.1 Latar Belakang

Pada era modern ini, penggunaan kendaraan pribadi meningkat sangat pesat. Faktor utama terjadinya hal ini adalah mobilitas masyarakat yang semakin tinggi serta semakin mudah nya memiliki kendaraan pribadi. Namun, peningkatan jumlah kendaraan yang besar ini memunculkan masalah baru yang cukup signifikan, yaitu ketersediaan lahan parkir yang memadai.

Lahan parkir yang ada, sering kali tidak dikelola dengan baik, hal ini menyebabkan para pengendaraan mengalami kesulitan dalam mencari lahan parkir yang kosong, sehingga masalah – masalah seperti banyaknya parkir liar dan kemacetan lalu lintas sering terjadi. Kondisi ini sangat mengganggu kelancaran lalu lintas serta mengurangi kenyamanan dan keamanan bagi pengendara.

Menurut Andre dan Ismiyanti (2017), peran parkir dalam sistem transportasi kota adalah bahwa tempat parkir menjadi suatu rangkaian kompleks yang saling mendukung dengan pola jaringan jalan dan karakteristik guna lahan tertentu, dimana tempat inilah yang akan membedakan antara tempat bergerak dan tempat untuk pemberhenti.

Penelitian oleh Zadeh dan Dela (2016) berusaha untuk memecahkan masalah tersebut dengan membangun sistem deteksi parkir dengan menggunakan sensor ultrasonik untuk mendeteksi banyaknya kendaraan yang ada di dalam lahan parkir. Dalam artikel tersebut, sistem digunakan pada beberapa lahan parkir dimana sistem ini hanya menghitung jumlah kendaraan yang masuk dan keluar, sehingga pada sistem ini informasi mengenai ketersediaan lahan parkir hanya terbatas pada jumlah kendaraan yang ada di dalam, sehingga pengoprasian lahan parkir masih sama seperti lahan parkir konvensional dimana pengendara tetap harus mencari sendiri lahan parkir yang masih kosong.

Untuk mengatasi masalah dari pengelolaan lahan parkir, penelitian kali ini akan membantu para pengendara dalam mencari lahan parkir dengan efisien, letak tempat parkir yang tersedia akan ditunjukan, sehingga meningkatkan mobilitas penduduk dan kelancaran lalu lintas. Sistem ini memanfaatkan *machine learning*

dan *computer vision* dalam mengidentifikasi data – data yang diperlukan untuk menyediakan informasi yang akurat kepada pengguna mulai dari jumlah tempat yang tersedia serta lokasi tempat yang dapat digunakan.

Pada penelitian ini prototipe yang dibuat akan menggunakan *Computer Vision* dan arsitektur YOLOv8, arsitektur ini sering digunakan untuk melakukan *Object Detection.* Arsitektur ini dapat melakukan deteksi objek dengan jumlah data *training* yang relatif tidak banyak.

1.2 Identifikasi Masalah

Hampir seluruh lahan parkir yang tersedia tidak dikelola dengan baik. Susahnya mencari lahan parkir menyebabkan para pengendara harus menghabiskan banyak waktu, sehingga lahan parkir seperti ini sangat tidak efisien dan menyebabkan berbagai masalah lalu lintas.

1.3 Batasan Masalah

Batasan masalah dalam pengerjaan sistem deteksi lahan parkir:

1. Pembuatan sistem prototipe berbasis Python dengan memanfaatkan YOLO sebagai arsitekturnya
2. Lahan parkir yang digunakan berupa prototipe / maket
3. Menggunakan 3 klasifikasi kendaraan (*SEDAN, SUV, TRUCK* (*mini truck*))
4. Penyediaan informasi berupa teks
5. Prototipe dapat diintegrasikan ke dalam 3 lahan parkir

1.4 Rumusan Masalah

Berdasarkan pemaparan pada latar belakang dan batasan masalah, dapat disimpulkan bahwa dalam penelitian ini, pembuatan sistem pada prototipe akan berbasis python dengan memanfaatkan YOLO untuk menjadi model *neural network* dalam mendeteksi dan mengklasifikasi miniatur kendaraan dengan akurat. Program python juga akan mengakomodir untuk fungsi dari arduino dan penyediaan informasi untuk pengguna

1.5 Tujuan Penelitian

Tujuan dari pembuatan prototipe ini untuk menjadi sistem yang bisa diandalkan dalam pengelolaan lahan parkir dengan memanfaatkan model *neural network* untuk mendeteksi kendaraan dan mengklasifikasikanya sehingga lahan parkir mampu untuk bekerja secara otonom dengan mendeteksi tiap kendaraan yang ada dengan akurat, serta memberikan informasi mengenai ketersediaan lahan parkir kepada pengguna.

1.6 Manfaat Pengenelitian

1. Manfaat bagi prodi yaitu:
    a. Sebagai bahan pembelajaran bagi mahasiswa lain yang sedang melakukan studi dengan materi yang serupa
    b. Bahan banding bagi studi yang serupa
2. Manfaat bagi peneliti yaitu:
    a. Publikasi ilmiah pada jurnal
    b. Syarat kelulusan

1.7 Luaran Penelitian

Luaran dari penelitian ini adalah sebuah model *Neural Network* yang telah dikembangkan serta prototipe dari lahan parkir yang memanfaatkan model tersebut sebagai fungsi deteksi objeknya. Model ini akan menentukan lahan parkir mana yang sesuai dengan klasifikasi dari objek.

1.8 Sistematika Penulisan

Sistematika penulisan ini adalah sebagai berikut:

Bab I : Pendahuluan

Bab pendahuluan terdiri dari latar belakang, identifikasi masalah, batasan masalah, rumusan masalah, tujuan penelitian, luaran penelitian, dan sistematika penulisan.

Bab II : Tinjauan Pustaka

Bab tinjauan pustaka terdiri dari penguraian teori tentang *computer vision*, *deep learning*, *artificial neural network*, *convolutional neural network*, pemrosesan data, *smart city*, arduino, *mikrokontroler*, pin, konektor, yolov8, roboflow, dan google colaboratory.

Bab III : Analisis dan Perancangan Sistem

Bab analisis dan perancangan sistem terdiri dari penguraian detail mengenai analisis kebutuhan dari perangkat keras dan perangkat lunak, pengumpulan data, desain dan perancangan sistem, *preprocessing*, *anotation*, *data filtering*, *data splitting*, model *training*, integrasi arduino, integrasi pada prototipe, uji coba sistem, pengujian model, dan pengujian fungsi perangkat arduino.

Bab IV : Hasil dan Pembahasan

Bab hasil dan pembagasan berisi penguraian dari hasil pembuatan *data set*, hasil *training* model, pembuatan program python dan arduino, dan pengujian model.

Bab V : Kesimpulan dan Saran

Bab kesimpulan dan saran berisi kesimpulan dari penelitian yang telah dilakukan dan saran untuk sistem dan model dapat lebih akurat dalam mengklasifikasikan kendaraan

BAB II
TINJAUAN PUSTAKA

2.1 *Computer Vision*

Computer vision is a form of image-based computer science *that uses pixel values to infer image content* (Weinstein dalam LeCun, 2015). *Computer vision* adalah suatu bidang kecerdasan buatan yang menggunakan *machine learning* untuk mengajarkan sebuah komputer untuk mendapatkan informasi yang berguna dari citra *digital*. Jenis *machine learning* yang digunakan dalam *computer vision* yaitu *deep learning* dengan jenis model *convolutional neural network* (CNN)

2.1.1 *Deep Learning*

Deep Learning merupakan suatu bagian dari *Machine Learning. Deep learning* menggunakan *Artificial Neural Network* sebagai metodenya untuk mengekstraksi dan mempelajari fitur pada data, sehingga memungkinkan sebuah mesin untuk bisa melakukan tugas yang biasanya hanya bisa dilakukan oleh manusia, seperti pengenalan dan pemrosesan citra, pemrosesan bahasa dan sistem otomisasi (*Self-Driving Cars*).

Deep Learning uses a cascade of multiple layers of nonlinear processing units for feature extraction and transformation. The lower layers close to the data input learn simple features, while higher layers learn more complex features derived from lower layer features. (Shinde, P. P., & Shah, S., 2018)

2.1.2 *Artificial Neural Network*

Artificial Neural network merupakan suatu model komputasi yang menyerupai otak manusia. *Artificial Neural network* adalah suatu model komputasi yang terdiri dari jumlah *nodes* (*Neurons*) yang besar yang terhubung satu dengan yang lain (Wu dan Feng, 2017).

Arsitektur *Artificial Neural Network* terdiri dari tiga lapisan utama yaitu lapisan *input* (*input layer*), lapisan tersembunyi (*hidden layer*), dan lapisan *output* (*output layer*).

1. Lapisan *input* (*input layer*)

   Lapisan ini merupakan tahap awal dimana fitur dari data akan diambil dan diberikan nilai (*value*) yang nantinya akan diteruskan ke lapisan tersembunyi (*hidden layer*).

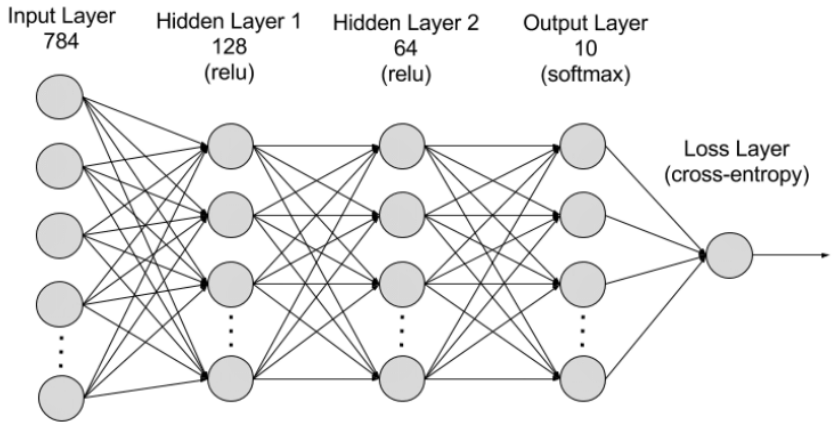
2. Lapisan tersembunyi (*hidden layer*)

   Selanjutnya fitur dengan nilai nya akan diteruskan ke lapisan tersembunyi (*hidden layer*). Lapisan ini merupakan lapisan dengan jumlah *node* yang banyak, tiap *node*-nya memiliki bobot nya (*weight*) masing – masing. Setiap data yang melewati *node* dalam lapisan tersembunyi, akan melalui komputasi yang mengalikan nilai dari data dengan bobot pada tiap *node* tersebut. Selanjutnya nilai luaran akan diterapkan bias dan fungsi aktifasi sebelum diteruskan ke lapisan *output*.

3. Lapisan *output* (*output layer*)

   Terakhir, pada lapisan ini semua nilai yang telah melalui komputasi akan diteruskan ke lapisan *output*. Lapisan ini akan menampung hasil dari nilai tersebut dan akan menjadi hasil akhir dari pemrosesan data.

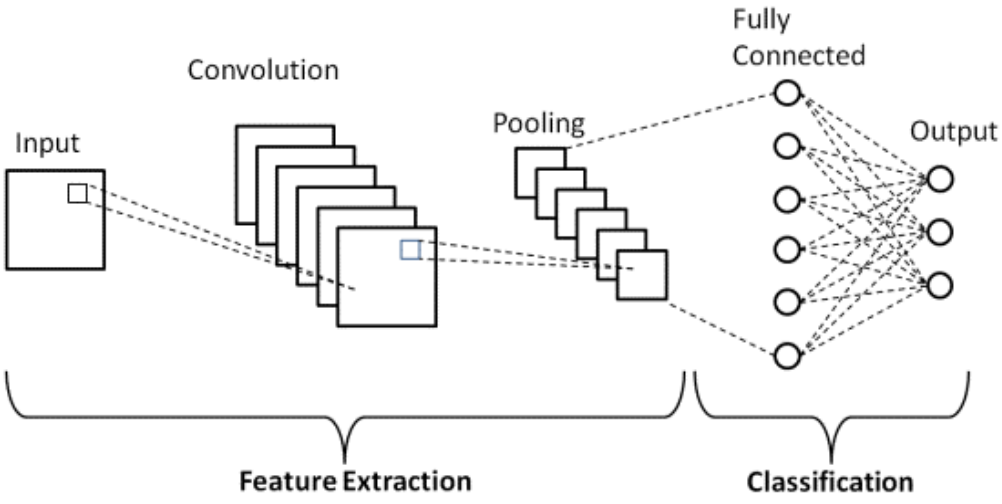
Contoh arsitektur dari *Artificial Neural Network* dapat dilihat pada gambar 2.1. Dari contoh tersebut dapat terlihat bahwa tiap lapisan dari ANN tidak terbatas satu *node* melainkan tiap lapisan dapat terdiri dari banyak *nodes*



Gambar 2.1 Arsitektur ANN

2.1.3 *Convolutional Neural Network* (CNN)

Convolutional Neural Network (CNN) adalah suatu jenis *neural network* yang didesain untuk melakukan proses data dengan struktur *grid* seperti citra digital (citra digital terdiri dari struktur *pixel* yang membentuk *grid*). Arsitektur ini menggantikan lapisan *input* pada arsitektur ANN konvensional dengan lapisan *convolution*, sehingga pada arsitektur ini ekstraksi fitur pada data tidak perlu dilakukan secara manual.



Gambar 2.2 Arsitektur CNN

Gambar 2.2 merupakan arsitektur dari CNN. Arsitektur CNN dibagi menjadi dua bagian yaitu *feature extraction* dan *classification.*

1. *Feature Extraction*

   Bagian ini merupakan bagian dari input dan terdiri dari dua lapisan, yaitu *convolutional layer* dan *pooling layer*. Lapisan pertama yaitu *convolutional layer* merupakan lapisan dimana fitur dari data yang berupa gambar akan diekstrak. Data gambar akan melalui komputasi yang akan membagi gambar menjadi *grid,* data ini akan menjadi *feature map*. Data yang telah melalui komputasi akan diterapkan filter. Jumlah filter yang digunakan akan menentukan kedalaman vektor input. output dari lapisan ini adalah vektor 3D yang menampung seluruh fitur dari data. Lapisan kedua yaitu *pooling layer* atau *downsampling* merupakan bagian dimana jumlah dimensi dari lapisan *convolution* akan dikurangi. Terdapat dua *pooling* yang digunakan pada lapisan ini yaitu *max pooling* dimana nilai tertinggi dari *feature map* akan dipilih, dan *average pooling* dimana jumlah dari nilai rata – rata bidang reseptif akan dipilih.

2. *Classification*

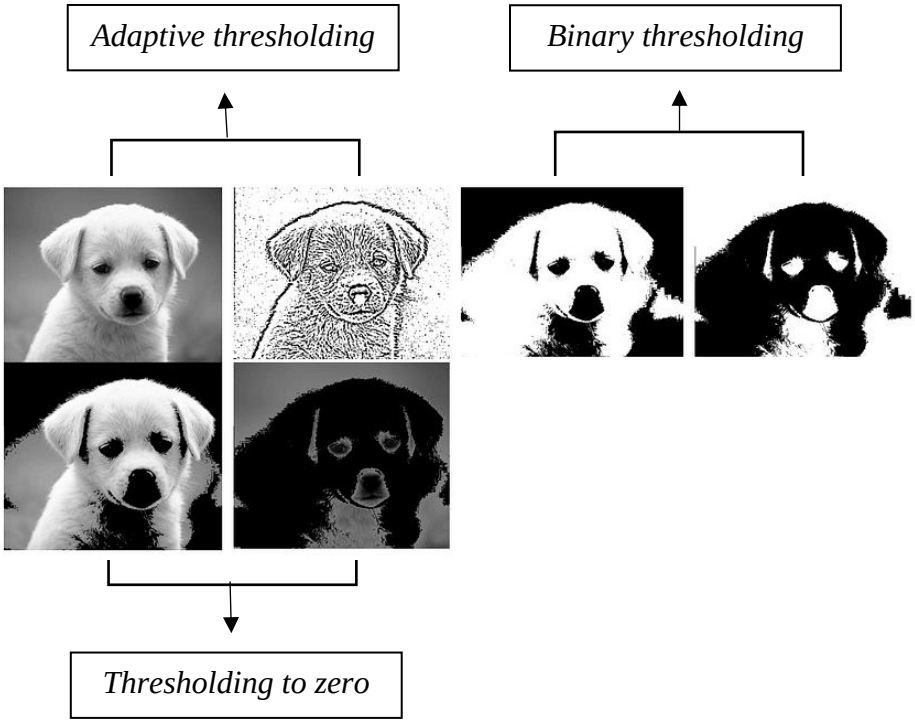
   Bagian kedua merupakan bagian yang sama dengan arsitektur ANN konvensional, namun bagian ini dimulai dengan *hidden layer* (*fully connected*) dan dilanjutkan dengan *output*.

2.1.4 Pemrosesan Data

Dalam pengenelan objek model memiliki beberapa kriteria untuk mempermudah proses ekstraksi fitur. Untuk memenuhi kriteria ini model akan menyesuaikan data yang digunakan. Beberapa contoh penyesuaian yang dilakukan adalah dengan menyesuaikan ukuran gambar ke ukuran yang diinginkan dan melakukan thresholding.

Gambar 2.3 Penyesuaian citra

Gambar 2.3 merupakan contoh data berupa citra yang melalui proses penyesuaian ukuran. Ukuran asli dari citra akan disesuaikan dengan kebutuhan dari model, seluruh resolusi akan diperkecil menjadi 736 x 736. Proses ini dilukakan ke seluruh data yang digunakan agar tidak ada perbedaan format antara satu data dan data lainya.



Gambar 2.4 *Thresholding*

Gambar 2.4 merupakan citra yang telah melalui proses *thresholding. Proses thresholding* merupakan proses yang bertujuan untuk memisahkan

antara objek dan *background*. Proses ini mengubah citra menjadi *grayscale* dari citra *grayscale* tersebut akan diubah kembali menjadi citra biner.

Adaptive thresholding merupakan teknik *thresholding* dengan mensegmentasi citra menjadi area – area kecil yang tidak *overlapping*. Masing – masing area mendapat nilai yang berbeda beda. Metode ini berguna untuk gambar dengan kondisi pencahayaan yang bervariasi atau latar belakang yang tidak seragam.

Binary thresholding merupakan teknik *thresholding* dengan mengubah gambar menjadi hitam putih dengan cara menghitamkan *pixel* dengan warna yang ada di bawah nilai *threshold* dan memutihkan *pixel* dengan warna yang ada di atas nilai *threshold.* Teknik ini sederhana, efisien, dan sering digunakan dalam segmentasi gambar, analisis dokumen, dan pengenalan pola. Namun, teknik ini kurang cocok untuk gambar dengan kondisi pencahayaan yang bervariasi, di mana *thresholding* adaptif lebih sesuai.

Threshold to zero merupakan teknik *thresholding* dengan menghitamkan area jika warna *pixel* di bawah nilai *threshold*. Jika warna *pixel* berada di atas nilai *threshold* maka warna akan dipertahankan. Metode ini membantu menunjukan fitur penting tanpa mengubah gambar menjadi bentuk biner. Teknik ini berguna dalam peningkatan gambar, *preprocessing,* dan pencitraan medis, karena mempertahankan informasi intensitas untuk piksel di atas ambang dan meningkatkan kontras. Namun, pemilihan nilai ambang yang tepat sangat penting untuk hasil yang optimal. Kemudian data akan masuk ke tahap *filtering* tahap ini merupakan tahap dimana data yang dapat dideteksi oleh model akan dipisahkan dengan data yang tidak dapat dideteksi.

2.1.5 *Smart City*

Smart City merupakan sebuah konsep dari pengembangan konsep kota urban dengan pemanfaatan manusia, kolektif, dan teknologi untuk meningkatkan pembangunan dan kemakmuran pada kota (Angelidou, 2014). Konsep ini biasanya terfokus pada infrastruktur kota yang memanfaatkan komputasi digital, pemanfaatan yang dimaksud seperti, implementasi dari *cashless payment,* pemanfaatan *computer* vision untuk mengatur lampu lalu lintas berdasarkan tingkat kepadatan kendaraan pada lalu lintas, integrasi kamera pada jalan raya untuk melakukan tilang *online*, dan lainya. Konsep ini bertunjuan untuk meningkatkan efisiensi dari pengelolaan kota dan meningkatkan produktivitas masyarakat.

Dari penjelasan ini, penelitian akan menekankan pada pemanfaatan teknologi berupa *computer vision* dalam melakukan peningkatan efisiensi dari pengelolaan lahan parkir untuk mempermudah pengendara dalam mencari lahan parkir.

2.2 Arduino

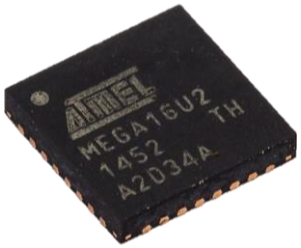
Arduino merupakan sebuah perangkat elektronik yang bersifat *open source* dan sering digunakan untuk merancang dan membuat perangkat elektronik serta software yang mudah untuk digunakan (Setiawan, 2022). Arduino sering kali digunakan dalam pembuatan perangkat elektronik sebagai otak dari perangkat tersebut.

Gambar 2.5 Papan Arduino uno R3

Arduino sendiri merupakan suatu perangkat yang bertugas untuk melakukan proses pada input yang diberikan sesuai dengan *software* yang telah dituliskan data yang telah diproses nantinya dapat dikeluarkan dalam bentuk teks pada terminal ataupun berupa sinyal pada perangkat yang terhubung dengan papan arduino. Arduino memiliki beberapa komponen yang penting di dalamnya:

2.2.1 Mikrokontroler

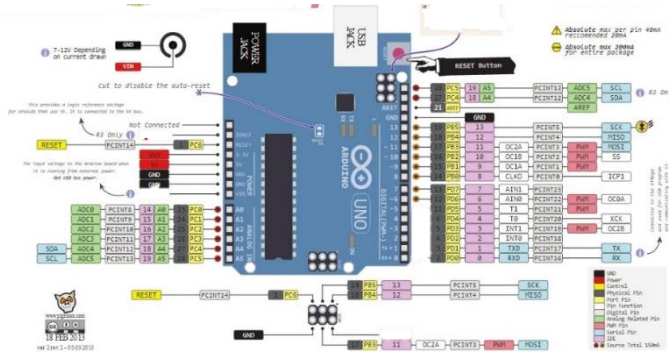
Mikrokontroler adalah chip yang memungkinkan untuk memprogram arduino sehingga mampu untuk memproses output berdasarkan input yang diberikan. Hal ini memungkinkan arduino untuk memproses data untuk memberikan perintah kepada perangkat lain untuk melakukan sebuah perintah. Gambar 2.6 merupakan gambar dari chip mikro kontroller pada arduino.



Gambar 2.6 Chip mikrokontroler

2.2.2 Pin

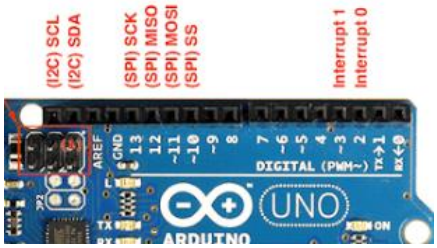
Pin merupakan bagian dari arduino yang penting sebagai penghubung antara perangkat lain dengan arduino. Setiap pin pada arduino memiliki fungsinya masing – masing.



Gambar 2.7 Diagram Pin Arduino

Gambar 2.7 merupakan diagram pin yang terdapat pada arduino, pin dalam arduino dibagi menjadi dua yaitu:

- Pin digital (Dapat menerima dan mengirim sinyal digital), gambar 2.8 merupakan gambar dari pin digital pada arduino
- Pin analog (Hanya dapat menerima sinyal analog), gambar 2.9 merupakan gambar dari pin analog pada Arduino



Gambar 2.8 Pin digital Arduino

Gambar 2.9 Pin analog Arduino

2.2.3 Konektor

Arduino memiliki dua jenis konektor yaitu:

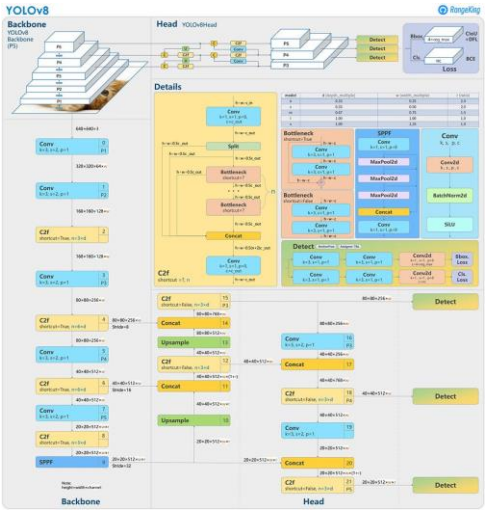
- Power konektor (Untuk menyalurkan daya ke Arduino)
- Serial konektor (Untuk menghubungkan perangkat dengan *port* USB)

Pada penelitian ini arduino digunakan dalam mengontrol servo yang nantinya akan menjadi pintu portal dari lahan parkir.

2.3 YOLOv8

YOLOv8 merupakan *real-time object detection and image segmentation model*. Model YOLO mampu untuk menyeimbangkan antara kecepatan dan akurasi pada arsitekturnya, hal ini menjadikan YOLO sangat cocok dalam pengaplikasian yang membutuhkan deteksi dalam waktu yang cepat, seperti *autonomous driving, surveillance*, dan robotik.

YOLO memiliki kemampuan untuk menghasilkan model yang relatif cepat dan akurat berkat arsitekturnya. Arsitektur pada YOLO terbagi menjadi tiga bagian yaitu *backbone, neck,* dan *head* . Arsitektur YOLO dapat dilihat pada gambar 2.10 (VK, 2023)



Gambar 2.10 Arsitektur YOLOv8

1. *Backbone*

   Bagian pertama dari arsitektur YOLO adalah *Backbone.* Bagian ini merupakan bagian dari YOLO yang menerima dan mengekstrak fitur dari gambar untuk menghasilkan *feature map*. Pada bagian ini gambar akan terbagi menjadi beberapa bagian, dari yang paling besar hingga yang paling detail. Dalam ekstraksi fitur YOLO menggunakan fungsi aktivasi SiLU (Sigmoid Linear Unit). Fungsi ini berguna untuk

menambahkan *non-linearity* pada data yang dapat membantu dalam mengenali *pattern* dari data yang kompleks. Formula untuk SiLU sebagai berikut.

$$f(x) = \sigma(z) = \frac{1}{1 + e^{-z}} \tag{2-1}$$

2. *Neck*

   Bagian ini merupakan bagian yang bertugas untuk menggabungkan *feature map* dari yang terbesar (kurang detail) hingga yang paling detail dan melakukan *feature enhancement*. Komponen utama pada bagian *Neck* adalah FPN (*Feature Pyramid Network*). FPN bertugas untuk menghasilkan *feature map* atau *up-samples feature map* yang lebih detail untuk nantinya akan digabungkan. Bagian ini juga menambahkan fitur berupa *spatial attention mechanisms* yang memberikan bobot ke seluruh bagian *feature map* dengan bobot yang berbeda – beda untuk meningkatkan fitur yang penting agar memudahkan proses deteksi objek dan klasifikasinya.

3. *Head*

   Bagian paling akhir dari YOLO merupakan bagian dimana proses deteksi terjadi. Bagian ini memiliki kemampuan untuk meningkatkan lagi *feature maps* dari bagian sebelumnya namun fungsi utama dari bagian ini adalah melakukan deteksi, memprediksi *bouding box*, memprediksi *class* (*classification*), dan memberikan *confidence score*. Bagian ini biasannya terdiri dari beberapa blok CNN. Dalam proses training *bounding box* yang memiliki Iou (*Intersection over Union*) atau koordinat *bounding box* terdekat dengan tag *bounding box* yang diberikan akan dipilih.

Pada penelitian ini YOLO akan menjadi arsitektur yang akan digunakan dalam melakukan *training* data set untuk melakukan deteksi objek dan klasifikasi dari miniatur kendaraan.

2.4 RoboFlow

RoboFlow merupakan *cloud-based workflow management system* untuk pengembangan yang terfokus pada data dan AI (Lin, et al., 2022). Sistem ini digunakan untuk membagi tugas kepada tim pengembang untuk melakukan proses pembuatan *data set* dengan memberikan *tag* berupa *bounding box*.

Dalam penelitian ini RoboFlow digunakan untuk membuat *data set* sebagai bahan untuk *training* dari model. RoboFlow memberikan *tools* yang sangat mempermudah untuk menggambar *bounding box* pada gambar, *tools* ini juga secara otomatis memberikan *tag* berupa koordinat dari *bounding box* yang telah digambar pada gambar input. Pada tahap terakhir RoboFlow menawarkan untuk memanfaatkan *platform*-nya untuk melakukan training, namun pengguna juga dapat membuat kode *snippet* yang akan mendownload *data set* yang telah dibuat jika pengguna ingin melakukan *training* sendiri.

2.5 Google Colaboratory

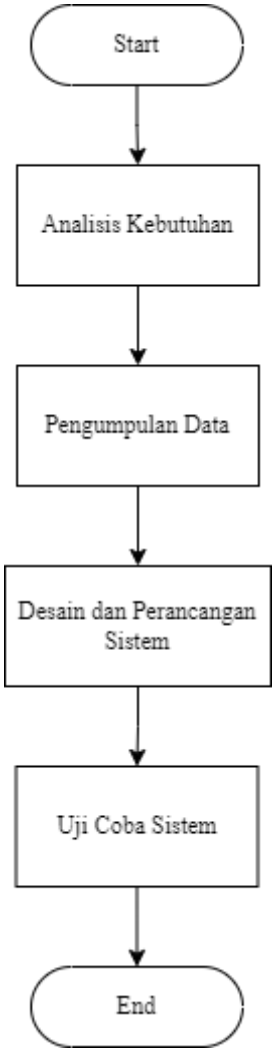
Colab is a hosted Jupyter Notebook service that requires no setup to use and provides free of charge access to computing resources, including GPUs and TPUs (Google, 2024). Penggunaan google colaboratory atau google colab tidak jauh berbeda dengan penggunaan IDE pada umumnya, namun google colab memiliki fitur untuk dapat menjalankan kode per segmen, sehingga cocok untuk pengembangan *machine learning.*

Dalam pengembangan *machine learning*, google colab menyediakan *resource* berupa berbagai macam GPU (Graphic Processing Unit) dan TPU (Tensor Processing Unit). GPU dan TPU memiliki performa yang tidak jauh berbeda namun

TPU lebih dioptimasi untuk melakukan komputasi, namun, GPU yang disediakan pada google colab memiliki *tensor core* sehingga pada dasarnya keduanya sangat efisien dalam melakukan komputasi dalam pengembangan *machine learning.* Google colab membatasi para penggunanya untuk mengakses GPU dan TPU yang tersedia tanpa membeli kuota pemakaian. Namun untuk pilihan tidak berbayar google colab menawarkan NVIDIA T4 GPU sebagai pilihan terbaik.

BAB III
ANALISIS DAN PERANCANGAN SISTEM

Penelitian ini dilakukan untuk pengembangan sistem parkir dalam prototipe kota yang akan dibuat dengan memanfaatkan *computer vision* untuk melakukan deteksi dan klasifikasi dari kendaraan. Proses penelitian dibagi menjadi beberapa tahapan seperti yang terlihat pada gambar 3.1



Gambar 3.1 Tahapan proses penelitian

3.1 Analisis Kebutuhan

Analisis kebutuhan merupakan tahapan yang penting untuk kelancaran proses pengembangan. Proses ini berguna untuk menganalisis kebutuhan peneliti

selama proses penelitian. Proses ini meneliti kebutuhan dalam bentuk perangkat keras dan perangkat lunak yang digunakan. Proses ini dimulai dengan menganalisis detail mengenai penelitian, analisa didapatkan dengan melakukan riset terlebih dahulu dengan tinjauan pustaka dari penelitian ini. Analisis yang didapatkan sebagai berikut.

3.3.1 Perangkat Keras

a. Laptop ASUS VivoBook A416FA-FHD324
    i. CPU : Intel Core i3 – 10110U
    ii. RAM : 4 GB DDR4
    iii. SSD : 256 GB
    iv. Sistem Operasi : Windows 11 Home
    v. GPU : Integrated Intel Graphics
b. Personal Computer Laboratorium KRPA
    i. CPU : Intel Core i7 – 12700F
    ii. RAM : 16 GB DDR4
    iii. SSD : 500 GB
    iv. Sistem Operasi : Windows 11 Pro
    v. GPU : Nvidia RTX 3060
c. Arduino UNO R3
    i. ATMega328P processor
        a. Memory
            i. 32 KB Flash
            ii. 2 KB SRAM
            iii. 1KB EEPROM
    ii. ATMega16U2 Processor (Mikro kontroller)
        a. Memory
            i. 16 KB ISP Flash
            ii. 512 B EEPROM
            iii. 512 B EEPROM
d. AKASO V50X (Kamera)
e. OBS BOT (Kamera)
f. Miniatur Kota

3.3.2 Perangkat Lunak

a. PyCharm
b. Python 3
c. CV2
d. CVZone
    i. Open CV
e. Ultralytics (YOLO)
f. Math
g. Arduino IDE
h. RoboFlow
i. Pickle

3.2 Pengumpulan Data

Tahap pengumpulan data merupakan tahap pembuatan *data set* berupa citra miniatur kendaraan. Miniatur kendaraan yang digunakan dibagi menjadi tiga jenis yaitu sedan, suv, dan *mini truck* atau mobil bak terbuka. Pengumpulan data dilakukan dengan menggunakan *script* berbasis python dan menggunakan kamera AKASO V50X.

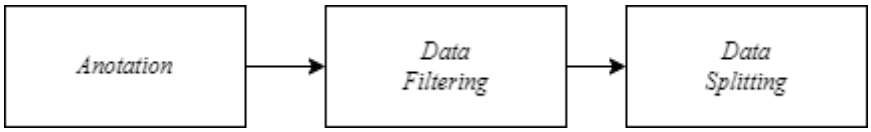
Script yang dijalankan berfungsi untuk mengambil citra dalam interval waktu 1.5 detik. Dalam pengambilan citra posisi miniatur akan selalu diubah setiap interval waktu agar model AI yang dihasilkan lebih akurat. Terdapat tujuh miniatur kendaraan yang digunakan dan diklasifikasikan menjadi tiga jenis, miniatur kendaraan yang digunakan dapat dilihat pada gambar 3.2.

Gambar 3.2 Miniatur kendaraan

3.3 Desain dan Perancangan Sistem

Tahapan ini merupakan tahap proses dari pembuatan sistem. Tahapan ini meliputi proses pembuatan model AI, rangkaian arduino, dan integrasi prototipe atau miniatur perkotaan. Tahap ini terdiri dari empat bagian yaitu *preprocessing*, model *training*, rangkaian arduino, dan integrasi antara prototipe kota dengan model AI dan rangkaian arduino.

3.3.1 *Preprocessing*



Gambar 3.3 Tahap *preprocessing*

Sebelum citra yang diambil dapat dijadikan *data set*, terdapat proses yang akan dilalui proses ini dinamakan *preprocessing*. Dalam proses ini data melewati tiga jenis tahap yaitu *annotation, data filtering* dan *data splitting*. Diagram mengenai tahapan *preprocessing* dapat dilihat pada gambar 3.3. Penjelasan mengenai kedua proses dalam *preprocessing* sebagai berikut.

a. *Anotation*

Proses ini merupakan proses awal dari pembuatan *data set* dengan memberikan *tag* pada citra yang sudah ditangkap. *Tag* yang diberikan berupa *bounding box* pada area yang ingin dideteksi dengan tambahan koordinat dari *bounding box* , klasifikasi dari objek dan jumlah objek. Proses ini memanfaatkan *tools* yang disediakan oleh RoboFlow.

b. *Data filtering*

Proses ini merupakan proses yang terjadi bersamaan dengan proses *anotation*. Proses ini menghapus citra yang dianggap dapat menurunkan akurasi dari model. Citra yang dimaksud adalah, citra yang mengandung objek yang akan dideteksi namun sebagian besar dari objek tersebut terhalang oleh objek lain atau hanya sebagian kecil dari objek yang terlihat pada citra.

c. *Data splitting*

Proses ini merupakan proses terakhir pada *preprocessing.* Pada tahap ini data akan dibagi menjadi tiga bagian yaitu *train set, validation set,* dan *test set* dengan pembagian sebagai berikut *train set 70%, validation set 20%,* dan *test set* 10%. Pembagian data ke *validation set* berfungsi untuk meningkatkan akurasi dari model dengan memberikan data yang tidak bias dan tidak pernah dilihat sebelumnya selama proses training agar model dapat menyesuaikan parameternya. *Test set* berfungsi untuk melakukan tes akurasi pada model dengan data yang tidak terdapat pada proses training

3.3.2 Model *Training*

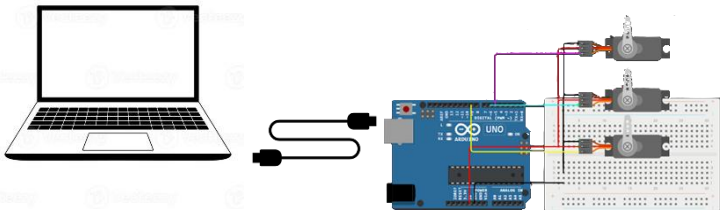
Tahap ini merupakan tahap selanjutnya dimana *data set* sudah siap untuk digunakan dalam *training* model. *Training* model dilakukan menggunakan google colaboratory dengan menggunakan *runtime* T4 GPU. Penggunaan google colaboratory dapat mempercepat proses training karena akses terhadap T4 GPU. Proses training dilakukan dengan epoch sebanyak 50 dan model YOLOv8n. Varian YOLOv8n merupakan varian dari YOLOv8 yang paling kecil dan cepat, penggunaan varian YOLOv8 yang lebih besar tidak diperlukan karena klasifikasi yang digunakan relatif tidak banyak dan performa dengan perangkat keras yang digunakan akan menurun.

3.3.3 Integrasi Arduino

Untuk mengontrol servo, maka arduino harus dikoneksikan dahulu dengan servo melalui beberapa pin. Dalam mengoneksikan servo terdapat beberapa pin yang digunakan yaitu:

- GND : Merupakan pin yang bertanggung jawab untuk ground.
- 5V : 5V adalah pin yang akan memberikan *power* pada papan Arduino.
- Digital ~ 5 , 6 , 9 : Pin ini merupakan pin yang akan menjadi tempat untuk mengirimkan sinyal pada servo.

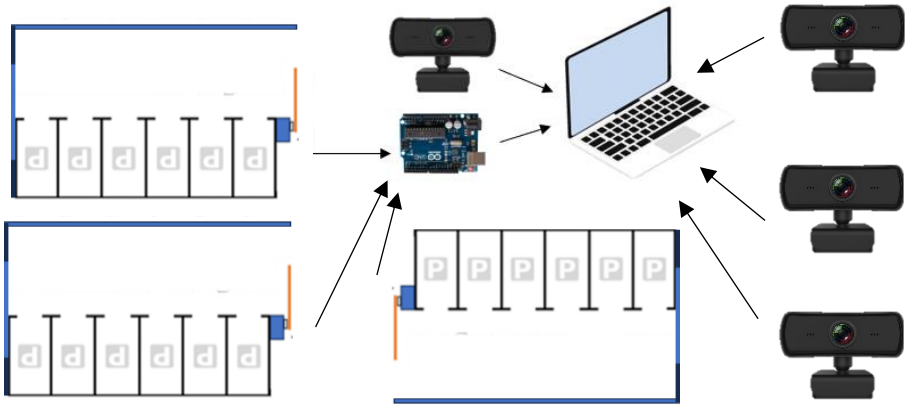
Selanjutnya arduino akan terkoneksi dengan laptop melalui *port* USB. Gambar 3.4 adalah gambar diagram koneksi antara arduino, servo, dan laptop.



Gambar 3.4 Diagram koneksi arduino, servo, dan laptop

3.3.4 Integrasi pada Prototipe

Tahap ini merupakan tahap terakhir dari perancangan sistem. Integrasi antara arduino, *webcam*, servo, dan model *Computer Vision* akan dilakukan pada prototipe. Gambar 3.5 merupakan rancangan prototipe yang akan digunakan.



Gambar 3.5 Rancangan prototipe

Pada rancangan prototipe, *webcam* akan diletakkan di atas lahan parkir menghadap bawah melihat lahan parkir dan kendaraan dari atas agar kendaraan dan lahan parkir dapat terlihat dengan jelas, *webcam* akan terhubung dengan laptop melalui *port* USB, sama halnya dengan arduino juga akan terhubung dengan laptop melalui *port* USB. Servo akan menjadi palang dari lahan parkir yang akan dihubungkan dengan arduino untuk menerima perintah.

Terdapat empat kamera yang diletakkan pada posisi yang berbeda – beda. Tiga kamera pertama diletakkan tepat di atas masing – masing lahan parkir menghadap ke lahan parkir. Untuk satu kamera lainya akan diletakkan di atas jalan di luar lahan parkir menghadap ke ruas jalan, untuk mendeteksi kendaraan yang akan memasuki lahan parkir.

Citra dari kamera akan dikirimkan ke laptop, jika terdeteksi adanya kendaraan yang akan memasuki lahan parkir maka laptop mengirimkan perintah menuju arduino untuk membuka palang.

Deteksi pada kendaraan dilakukan pada area yang telah ditandai dengan polygons. Polygons ini dibuat dengan menyimpan koordinat di titik – titik yang ditentukan.

3.4 Uji Coba Sistem

Tahap terakhir adalah tahap uji coba, tahap ini merupakan tahap dimana hasil dari penelitian akan diuji coba, beberapa pengujian yang dilakukan adalah pengujian model dan pengujian fungsi perangkat arduino.

3.4.1 Pengujian Model

Pengujian pada model yang akan dilakukan adalah *precision, recall* dan F1 – *Score* dari model yang sudah melalui proses *training*. Terdapat empat kemunkinan luaran dari model, yaitu *True Positive* (TP), *False Positive* (FP), *False Negative* (FN),dan *True Negative* (TN). Keempat luaran ini dapat dirangkum menggunakan *confusion matrix*.

Tabel 3.1 Contoh *Confusion Matrix*

Jumlah Sesungguhnya	Jumlah terdeteksi	
	1	0
1	TP	FN
0	FP	TN

Tabel di atas merupakan contoh *confusion matrix* sederhana dengan satu variabel positif dan satu variabel negatif yang dilambangkan dengan nilai 1 sebagai positif atau terdapat objek yang akan dideteksi dan 0 sebagai negatif atau tidak ada objek yang akan dideteksi. Pada contoh tersebut *true positive* (TP) merupakan bagian yang melambangkan bahwa model mendeteksi objek yang sesuai dengan kenyataan. *True negative* (TN) adalah bagian yang melambangkan jika model tidak mendeteksi apapun dan sesungguhnya tidak ada objek yang harus dideteksi. *False negative* (FN) adalah bagian yang melambangkan bahwa model tidak mendeteksi dapat objek meskipun kenyataannya terdapat objek yang ingin dideteksi. *False positive* (FP) merupakan bagian yang melambangkan bahwa model mendeteksi objek walau pada kenyataannya tidak terdapat objek yang akan dideteksi. Namun karena terdapat 3 klasifikasi dari luaran model maka *confusion matrix* yang dihasilkan juga akan berubah. *confusion matrix* untuk tiga klasifikasi dapat dilihat pada tabel 3.2.

Tabel 3.2 *Confusion Matrix* tiga klasifikasi

Jumlah Sesungguhnya	Jumlah Terdeteksi			
	SEDAN	SUV	TRUCK	BACKGROUND
SEDAN	TP	FN	FN	FN
SUV	FN	TP	FN	FN
TRUCK	FN	FN	TP	FN
BACKGROUND	FP	FP	FP	TN

Penambahan variabel background pada *confusion matrix* berfungsi menjadi variabel *negative* pada *matrix*. Variabel tersebut penting untuk menentukan nilai yang akan digunakan untuk melakukan pengujian model.

Selanjutnya perhitungan *precision* dan *recall* akan dilakukan untuk melihat proporsi kemampuan dari model untuk melakukan deteksi TP pada objek. *Precision* merupakan perhitungan yang menggambarkan perbandingan deteksi *positive* yang dibandingkan dengan seluruh data

positive. Sedangkan *Recall* merupakan perhitungan yang menggambarkan perbandingan deteksi *positive* jika dibandingkan dengan seluruh data yang ada. Perhitungan *precision* dan *recall* didefinisikan sebagai berikut

$$Precision = \frac{TP}{TP + FP} \tag{3-1}$$

$$Recall = \frac{TP}{TP + FN} \tag{3-2}$$

F1- *score* merupakan *metrics* yang menggabungkan *precision* dan *recall.* F1 – *score* akan menguji banyaknya jumlah deteksi benar yang dapat dilakukan oleh model. F1- *score* diperoleh dengan menghitung *harmonic mean* dari perhitungan *precision* dan *recall.* Perhitungan F1 – *score* dapat dilakukan dengan definisi sebagai berikut.

$$F1\ score = 2 \times \frac{Precision \times Recall}{Precision \times Recall} \tag{3-3}$$

Perhitungan F1 – *score* akan memberikan gambaran mengenai performa model yang digunakan. Nilai F1 – score berkisar antara 0 hingga 1.0 dimana nilai 0 berarti bahwa model selalu salah dalam mendeteksi objek dan nilai 1.0 berarti model selalu benar dalam mendeteksi objek.

3.4.2 Pengujian Fungsi Perangkat Arduino

Pengujian pada perangkat arduino merupakan pengujian yang cukup simpel dan mudah. Pengujian ini ditujukan untuk mengetahui apakah program yang ada pada laptop mampu untuk mengirimkan perintah serial menuju arduino dan apakah arduino mampu untuk mengirimkan perintah menuju ketiga servo. Pengujian dilakukan dengan mengirimkan *string* menuju arduino yang selanjutnya arduino akan memerintahkan servo untuk bergerak sesuai dengan *string* yang dikirimkan.

BAB IV
HASIL DAN PEMBAHASAN

Pada bab ini akan membahas perancangan yang sudah ditetapkan sebelumnya. Bab ini juga akan membahas tentang hasil pengujian dari penelitian ini. Bab ini akan dibagi menjadi enam bagian yaitu pembuatan *data set*, *training* model, pembuatan program python, pembuatan program arduino, pengujian model serta fungsionalitas dari arduino, dan pembuatan protipe kota.

4.1 Pembuatan *Data Set*

Dalam pembuatan *data set* terdapat empat tahap yaitu pengambilan citra dari miniatur kendaraan, *annotation*, *data filtering,* dan *data splitting*. Keempat tahapan ini penting untuk menghasilkan *data set* yang baik. Pada penelitian ini posisi *bounding box*, jumlah citra, dan kejelasan dari citra yang diambil sangat penting. Penjelasan mendetail mengenai tahapan pembuatan *data set* sebagai berikut.

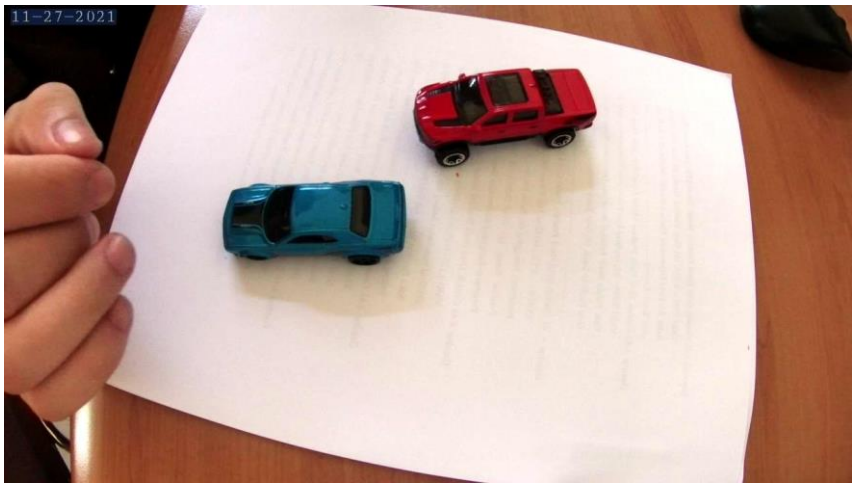
4.1.1 Pengambilan Citra

Pada tahapan pertama pengambilan citra dilakukan menggunakan kamera AKASO V50X dengan bantuan *script* berbasis python. Script tersebut berfungsi untuk mengambil citra dalam interval waktu yang ditentukan.

```
001  import cv2
002
003  CAR_COUNT = "1"
004
005  cam_width, cam_height = 1280, 720
006  path = "D:\\YOLO TA\\Imagedataset\\NullImage\\"+CAR_COUNT+"_"
007  count = 1
008  cam_port = 0
009  cam = cv2.VideoCapture(cam_port)
010  cam.set(3,cam_width)
011  cam.set(4,cam_height)
012
013  def save(path, varImg):
014
015      cv2.imwrite(path,varImg)
016  while True:
017      result, image = cam.read()
018      cv2.imshow("Image", cv2.resize(image,(854,480)))
019
```

```
020       if result:
021           save(path +"_"+ str(count) + ".jpg", image)
022           cv2.waitKey(2000)
023           print(".....SAVED!!")
024           count += 1
```

Potongan kode di atas merupakan kode yang bertugas untuk menangkap citra. Potongan kode `cv2.waitKey(2000)` berfungsi untuk menentukan waktu dalam satuan ms (*mili second*), dalam kode ini citra akan diambil dalam interval 2000 ms atau 2 detik. Bagian kode `save(path +"_"+ str(count) + ".jpg", image)` merupakan bagian yang bertujuan untuk menyimpan citra pada tempat yang sudah ditempatkan pada variabel "path" dan dengan nama sesuai dengan variabel `str(count)`, pada potongan kode yang sama variabel "image" merupakan variabel yang menampung citra yang ditangkap oleh kamera. Kamera yang digunakan didefinisikan dengan variabel cam_port.



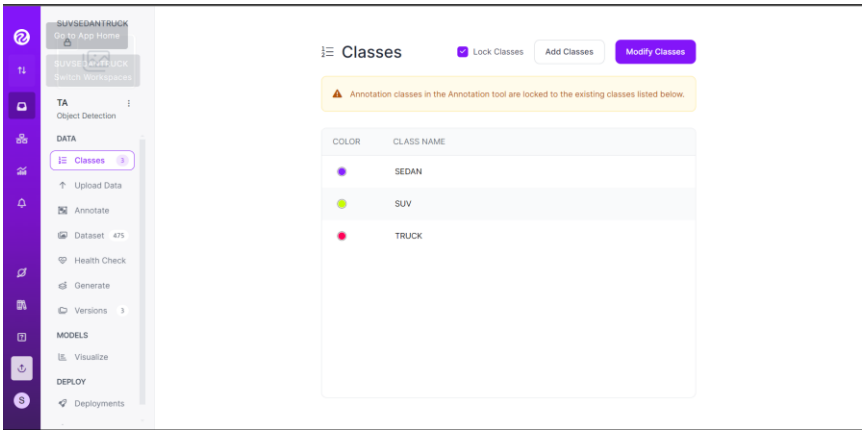
Gambar 4.1 Citra kendaraan

Gambar 4.1 merupakan beberapa citra yang telah diambil. Citra yang diambil mencakup ketiga klasifikasi yang akan digunakan yaitu SEDAN, SUV, dan TRUCK. Total pengambilan citra keseluruhan adalah 504 citra. Peletakan kamera akan

disesuaikan dengan peletakan pada prototipe kota yaitu dari atas mengarah ke bawah.

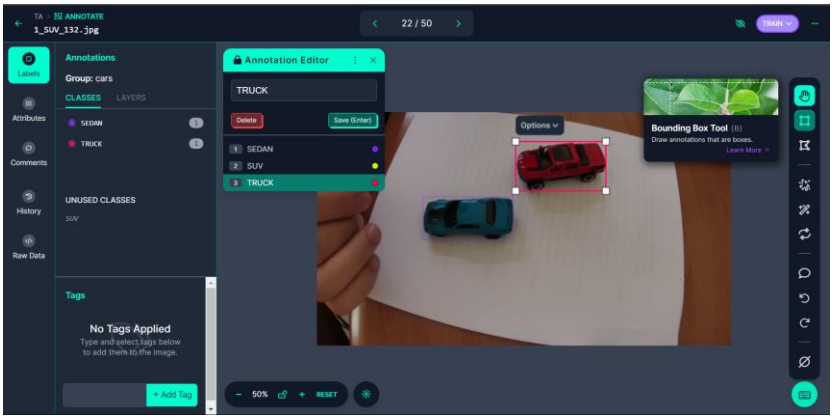
4.1.2 *Anotation*

Tahapan ini merupakan tahap penambahan *tag* berupa *bounding box*, dan koordinat dari *bounding box*. Tahapan ini dilakukan secara manual menggunakan tools yang disediakan oleh roboflow. Pada tahap ini sangat penting untuk memperhatikan area *bounding box* yang diberikan, *bounding box* yang diberikan akan meliputi seluruh objek yang akan dideteksi namun tidak terlalu besar.



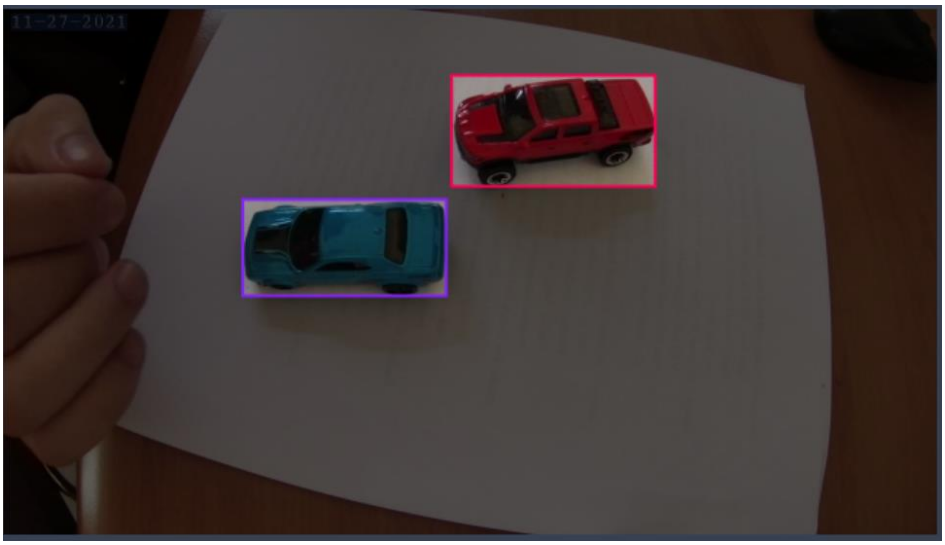
Gambar 4.2 *Interface* penambahan nama kelas

Sebelum proses *annotation*, nama dari kelas yang akan dicantumkan pada objek akan ditambahkan terlebih dulu. Penambahan nama kelas dapat dilihan pada gambar 4.2. Terdapat tiga klasifikasi yang ditambahkan.



Gambar 4.3 *Interface tool* roboflow

Gambar 4.3 merupakan *interface tool* dari roboflow yang digunakan untuk menggambar *bounding box* pada citra. Dalam melakukan anotasi, klasifikasi objek perlu diperhatikan. Setiap warna dari *bounding box* melambangkan klasifikasi yang telah ditentukan sebelumnya. *Tag* berupa koordinat, klasifikasi, dan jumlah objek akan secara otomatis di-*generate* oleh roboflow dalam *file* tersendiri.

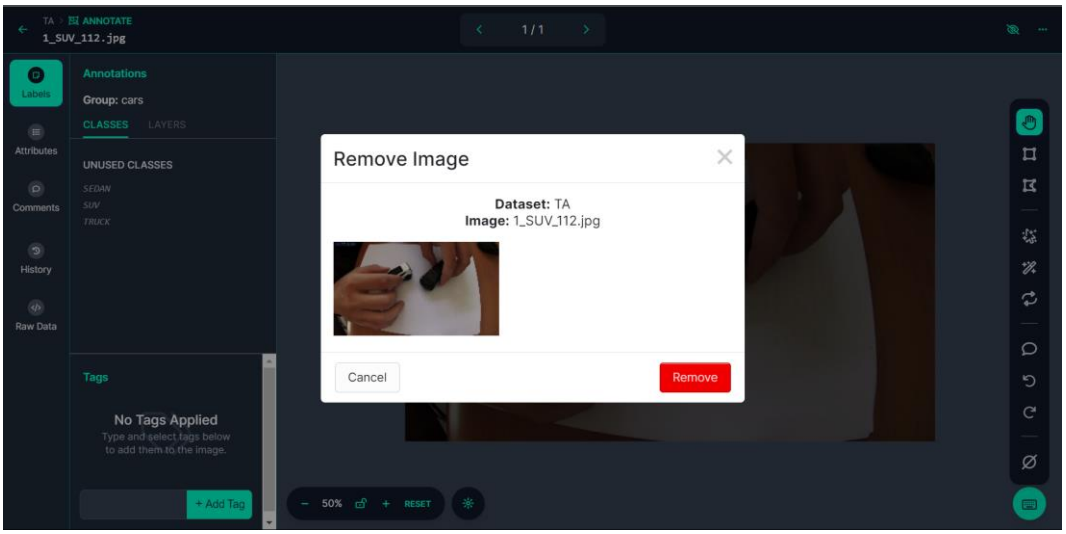


Gambar 4.4 Hasil anotasi citra

Gambar 4.4 adalah hasil dari citra yang telah melalui anotasi. *Bounding box* yang diberikan sesuai dengan besar dari objek.

4.1.3 *Data Filtering*

Tahapan ini merupakan tahapan yang dilakukan untuk membersihkan *data set* dari citra yang tidak baik. Citra yang tidak baik merupakan citra yang mengandung objek yang ingin dideteksi namun, objek tersebut tidak jelas atau bagian dari objek hanya terlihat sedikit. Tahapan *data filtering* ini dilakukan bersamaan dengan tahapan *annotation*. Proses pembersihan dilakukan dengan cara menghapus citra yang tidak baik.



Gambar 4.5 Proses penghapusan citra

Gambar 4.5 merupakan proses penghapusan yang dilakukan dengan menggunakan *tool* yang sama pada halaman yang sama. Pada saat proses *annotation* dilakukan jika ada citra yang dianggap tidak baik maka citra tersebut dapat segera dihapus. Data citra yang tersisa setelah melewati tahapan ini adalah sebanyak 475 citra.

4.1.4 *Data Splitting*

Data splitting merupakan tahap terakhir dalam pembuatan *data set*. Pada tahap ini data yang sudah dibersihkan dan diberi *tag* akan dibagi menjadi tiga bagian yaitu *train set, validation set,* dan

test set. Perbandingan dari pembagian ketiga bagian ini adalah 70% *train set*, 20% *validation set*, dan 10% *test set.* Dari pembagian ini didapatkan jumlah citra pada *train set* sebanyak 333 citra, *validation set* sebanyak 95 citra, dan *test set* sebanyak 47 citra.

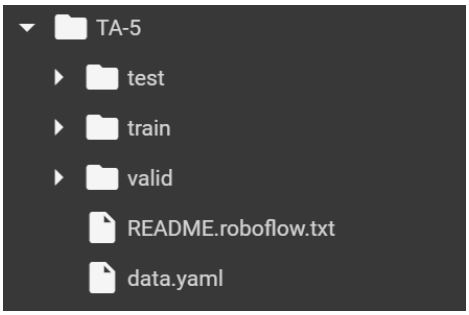
Data splitting penting untuk dilakukan untuk meningkatkan akurasi deteksi dari model yang dihasilkan. *Train set* merupakan data yang akan menjadi data utama yang digunakan dalam proses *training* dengan jumlah citra terbanyak. *Validation set* merupakan data yang digunakan sebagai pembanding dalam proses *training*. Pada proses *training* tiap *epoch* yang selesai maka model akan menggunakan *validation set* untuk mengatur parameter modelnya, dengan begitu model dihadapkan dengan data baru. Hal ini sangat membantu untuk menghindari *overfitting* pada model. *Overfitting* merupakan keadaan dimana *data set* terdiri dari data yang terlalu bagus sehingga jika model dihadapkan pada keadaan dengan keberadaan *noise* maka model tidak mampu untuk mendeteksi objek. *Train set* merupakan data yang digunakan untuk mengukur keakuratan pada model. *Train set* merupakan data yang akan diproses ketika seluruh proses training telah selesai.

4.2 *Training* Model

Pada tahap ini *data set* yang siap untuk digunakan akan melalui tahap training. Sebelum *data set* dapat digunakan, *data set* harus di-*download* terlebih dahulu ke google colab. Roboflow menyediakan fitur untuk meng-*generate snippet* kode yang dapat dijalankan dalam google colab untuk men-*download data set* dari *platform* roboflow dengan format yang diinginkan.

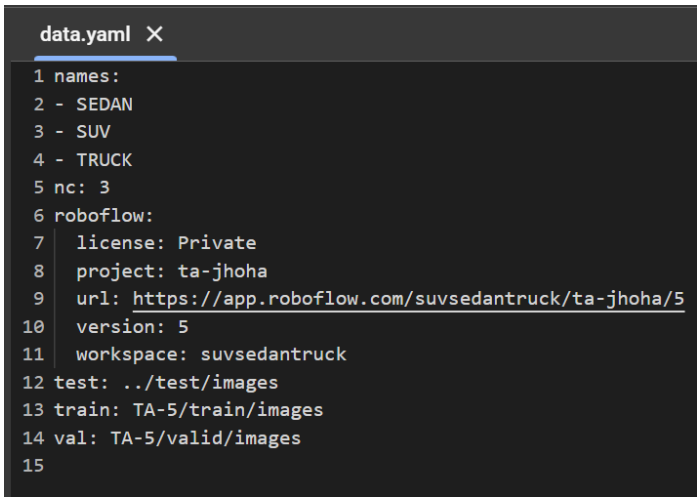
```
001    !pip install roboflow
002
003    from roboflow import Roboflow
004    rf = Roboflow(api_key="lKMj2udZfi3fzg2cS3K2")
005    project = rf.workspace("suvsedantruck").project("ta-jhoha")
006    version = project.version(5)
007    dataset = version.download("yolov8")
```

Potongan kode diatas merupakan kode *snippet* yang dihasilkan oleh roboflow dengan format yolov8. Kode *snippet* ini mengakses *data set* dari akun roboflow dengan nama *workspace* "suvsedantruck" dan kode projek "ta-jhoha".



Gambar 4.6 *Folder data set*

Gambar 4.6 merupakan *folder* yang telah di-*download*. *Folder* yang di-*download* berisi seluruh *data set* yang telah terbagi dengan tambahan file "data.yaml". File tersebut merupakan data yang mengatur konfigurasi dari letak *test set*, *validation set*, dan *train set* serta nama klasifikasi dari objek.



Gambar 4.7 Isi *file* data.yaml

Pada gambar 4.7 menunjukan isi dari *file* data.yaml. Dapat terlihat pada empat baris pertama merupakan nama dari klasifikasi yang telah ditentukan sebelumnya. Pada *file* ini terdapat bagian yang berisi *path* untuk folder test (*test*

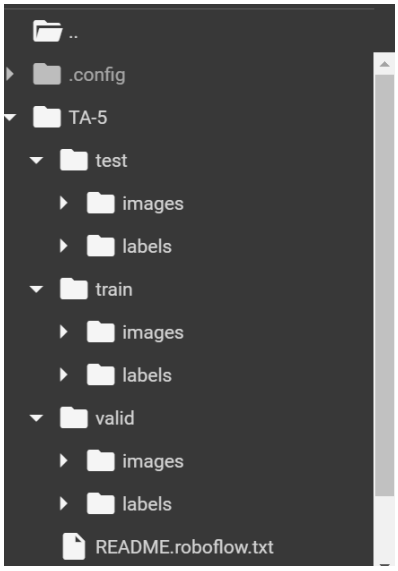
set), train (*train set*), dan val (*validation set*). *Path* yang tercantum akan sedikit disesuaikan dengan *path* yang ada pada google colab.

```
names:
- SEDAN
- SUV
- TRUCK
nc: 3
roboflow:
  license: Private
  project: ta-jhoha
  url: https://app.roboflow.com/suvsedantruck/ta-jhoha/5
  version: 5
  workspace: suvsedantruck
test: /content/TA-5/test
train: /content/TA-5/train
val: /content/TA-5/valid

```

Gambar 4.8 *File* data.yaml setelah penyesuaian

Gambar 4.8 merupakan isi dari *file* data.yaml yang telah disesuaikan dengan *path* folder *data set* pada google colab.



Gambar 4.9 *Folder data set*

Gambar 4.9 merupakan isi dari *Folder data set*. Setiap folder dari *data set* berisi dua jenis *folder* yaitu images dan labels. Images merupakan folder yang berisi citra yang ditangkap sebelumnya sedangkan labels berisi *file tag*.

Gambar 4.10 *File tag* pada citra

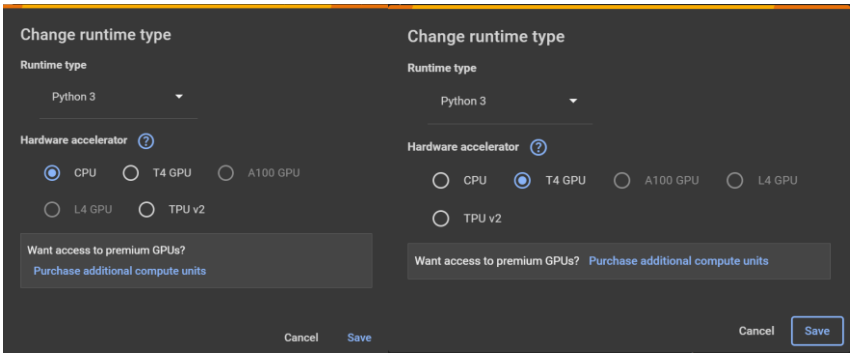
Gambar 4.10 merupakan isi dari *file tag* yang diberikan pada citra. Pada baris pertama, "2" merupakan kode klasifikasi objek yang diikuti oleh 4 baris koordinat. Terdapat dua baris dengan kode klasifikasi yang berbeda, menunjukkan pada citra tersebut terdapat dua objek berbeda.

Sebelum proses berlangsung terdapat satu *library* yang diperlukan untuk mengakses model YOLO. Ultralytics merupakan *library* yang dapat digunakan untuk mengakses model YOLO.

```
001  !pip install ultralytics
002  From ultralytics import YOLO
```

Potongan kode di atas berfungsi untuk meng-*install library* ultralytics dan mengambil komponen YOLO dari library tersebut.

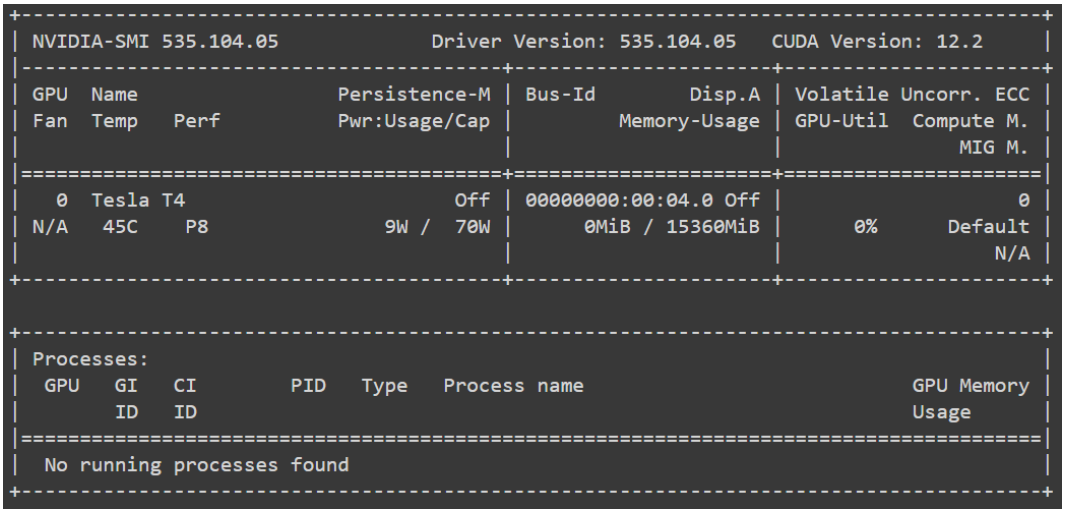
Sebelum proses training dijalankan jenis perangkat keras yang digunakan pada runtime google colab akan dipastikan terlebih dahulu. Perangkat keras terbaik yang dapat dipilih tanpa melakukan pembelian adalah T4 GPU, seperti pada gambar 4.11.



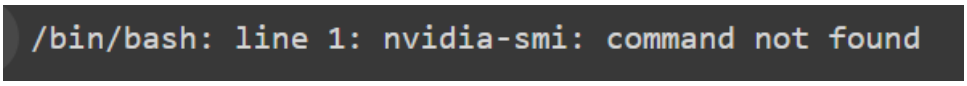
Gambar 4.11 Jenis runtime

Untuk menguji apakah perangkat keras T4 GPU telah terhubung, potongan kode `!nvidia-smi` digunakan untuk melihat spesifikasi dari perangkat keras T4

GPU. Spesifikasi dari T4 GPU dapat dilihat pada gambar 4.12. Jika perangkat keras T4 GPU belum terhubung maka potongan kode `!nvidia-smi` akan mengeluarkan *error code* seperti yang terlihat pada gambar 4.13.



Gambar 4.12 Spesifikasi T4 GPU



Gambar 4.13 Kode error GPU

Setelah seluruh persiapan *data set* dan perangkat keras telah siap maka proses *training* dapat dilakukan. Proses training dapat dilakukan dengan menggunakan potongan kode sebagai berikut.

```
001  Yolo  task=detect  mode=train  data=/content/TA-5/data.yaml
     epochs=50 imgsz=730
```

Pada potongan kode diatas terdapat beberapa parameter yang digunakan oleh model untuk menetapkan alur *training* dan *path* dari *data set*. Parameter task merupakan parameter yang menetapkan tugas pada YOLO. Pada potongan kode

di atas parameter “task” berisi perintah untuk “detect”, perintah ini bertugas untuk menetapkan bahwa model YOLO akan melakukan tugas deteksi objek. Selanjutnya parameter “mode” berisi perintah “train”. Perintah train bertugas untuk menetapkan bahwa model YOLO akan melakukan training pada model. Parameter “data” merupakan parameter yang berisi *path* yang menuju *file* konfigurasi yang telah dijelaskan sebelumnya. Paramater “epochs” merupakan jumlah siklus *training* yang akan dilakukan, tiap *epoch* yang selesai model akan melakukan validasi menggunakan *validation set*. Parameter terakhir yaitu “imgsz” merupaka parameter yang berisi ukuran dari citra.

Luaran dari proses training adalah *folder* berisi model yang telah disesuaikan dengan klasifikasi dan objek yang telah ditentukan sebelumnya serta beberapa file *testing* dari model.

```
50 epochs completed in 0.160 hours.
Optimizer stripped from runs/detect/train2/weights/last.pt, 6.2MB
Optimizer stripped from runs/detect/train2/weights/best.pt, 6.2MB
```

Gambar 4.14 *Path folder* model

Gambar 4.14 menunjukan *path* folder tempat model disimpan. Model “last.pt” merupakan model yang telah melalui proses *validation* dan *testing*, sedangkan “last.pt” merupakan model yang hanya melalui proses *training*.

Proses *training* menggunakan GPU merupakan hal yang penting untuk dilakukan. GPU yang telah disediakan oleh google colab merupakan perangkat yang sangat optimal untuk melakukan proses *training*. Perbedaan waktu yang digunakan sangat signifikan dimana *training* menggunakan GPU membutuhkan waktu 9.6 menit, sebaliknya penggunaan CPU dalam proses *training* menghabiskan waktu sebanyak 5 jam dengan jumlah serta jenis *data set* dan parameter yang sama.

4.3 Pembuatan Program Python

Program python yang dibuat akan meliputi beberapa tugas, beberapa tugas yang akan diliputi adalah mengkoneksikan serta menangkap citra dari keempat kamera yang digunakan, mengirimkan kode pada arduino berupa string untuk memerintah servo untuk bergerak sesuai dengan posisi miniatur kendaraan.

Dalam pembuatan program python terdapat beberapa library yang akan digunakan, yaitu math, cv2, cvzone, pickle, numpy, ultralytics, serial, dan time. Program python diawali dengan pembuatan bagian yang akan mengakses keempat kamera yang digunakan. Bagian ini memanfaatkan library cv2 untuk menghubungkan keempat kamera menuju komputer.

```
001   cam_width, cam_height = 1280, 720  # CAM RES
002
003   cap1 = cv2.VideoCapture(p1)  # WEB CAM ASSIGN
004   cap2 = cv2.VideoCapture(p2)
005   cap3 = cv2.VideoCapture(p3)
006   cap4 = cv2.VideoCapture(roadCam)
007
008   cap2.set(3, cam_width)
009   cap2.set(4, cam_height)
010   cap3.set(3, cam_width)
011   cap3.set(4, cam_height)  # SET ALL FRAME SIZE
012   cap1.set(3, cam_width)
013   cap1.set(4, cam_height)
014   cap4.set(3, cam_width)
015   cap4.set(4, cam_height)
```

Potongan kode di atas merupakan bagian yang menginisialisasi kamera. Kamera yang akan diinisialisasi dilambangkan dengan angka dari 0 hingga 3 yang ditampung dalam variabel p1,p2,p3, dan roadcam. Selanjutnya pada baris 008 hingga 015 merupakan bagian inisialisasi resolusi dari citra kamera yang nanti akan ditampilkan.

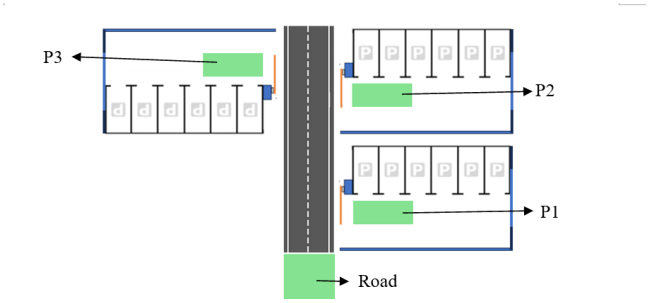
Bagian selanjutnya adalah bagian yang membaca citra kamera yang sudah diinisialisasi dan melakukan proses citra menggunakan model YOLO. Bagian ini dapat dilihat pada potongan kode di bawah ini.

```
001  success1, img1 = cap1.read()
002  success2, img2 = cap2.read()  # READING CAMERA ASSIGNED
003  success3, img3 = cap3.read()
004  success4, img4 = cap4.read()
005
006
007  object_list1 = get_object_list_yolo(model, img1,
     class_names, confidence, draw=True)
008  object_list2 = get_object_list_yolo(model, img2,
     class_names, confidence, draw=True)
009  object_list3 = get_object_list_yolo(model, img3,
     class_names, confidence, draw=True)
010  object_list4 = get_object_list_yolo(model, img4,
     class_names, confidence, draw=True)
```

Pada potongan kode di atas, pada baris 001 hingga 004 merupakan bagian yang membaca citra dari kamera. Baris 007 hingga 010 merupakan bagian yang memberikan citra kamera pada model untuk dideteksi oleh model YOLO menggunakan fungsi get_object_list_yolo. Objek yang terdeteksi akan ditampilkan dalam citra kamera dengan *bounding box* pada objek tersebut dan disimpan dalam format list ke dalam variabel penyimpan object_list1, object_list2, object_list3, dan object_list4. Masing – masing variabel penyimpan bertanggung jawab untuk menyimpan objek yang terdeteksi dari citra pada img1, img2, img3, dan img4. Objek yang telah terdeteksi dapat dilihat pada gambar 4.15.

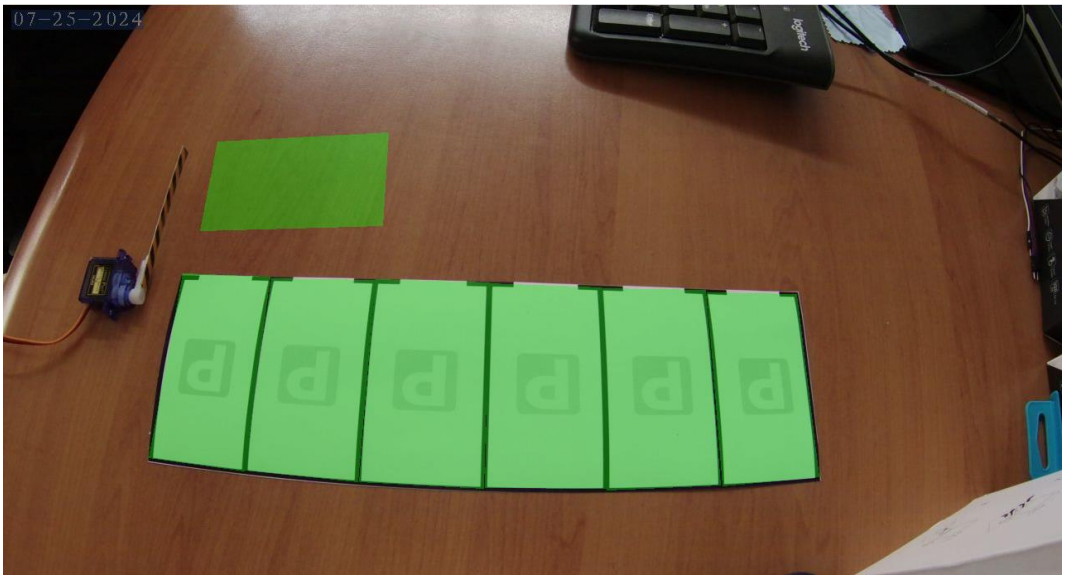
Gambar 4.15 Objek terdeteksi

Bagian selanjutnya adalah bagian yang mengatur arduino untuk mengirim perintah menuju ketiga servo. Dalam bagian ini posisi kendaraan akan memicu arduino jika sudah berada pada area yang ditentukan. Dapat dilihat pada gambar 4.16 bahwa terdapat empat area yang akan digunakan. Area pertama yaitu “Road” akan memicu arduino untuk menggerakan servo tertentu sesuai dengan jenis klasifikasi miniatur kendaraan yang ada di dalam area tersebut. Area selanjutnya adalah “P1”. Area tersebut akan memicu arduino untuk menggerakan servo yang ada pada lahan parkir P1. Sama halnya dengan area “P2” dan “P3” keduanya hanya akan menggerakan servo pada lahan parkirnya masing masing. Servo yang dibuka dengan memicu area “Road”, akan menutup setelah miniatur telah masuk ke lahan parkir, sedangkan servo yang dibuka dengan memicu area lain akan langsung menutup 500ms setelah miniatur kendaraan tidak lagi di dalam area tersebut.



Gambar 4.16 Area polygons

Area akan dibuat membaca empat titik koordinat pada citra kamera. Setelah keempat koordinat telah tersimpan maka berdasarkan koordinat – koordinat tersebut akan dijadikan acuan sudut dari kotak yang akan digambarkan pada citra, sehingga pada tampilanya area tersebut dapat terlihat seperti pada gambar 4.17. Setelah area telah digambar maka selanjutnya akan disimpan dengan menggunakan pickle. Format *file* yang disimpan akan berubah menjadi format pickle yaitu “.p”



Gambar 4.17 Area yang telah digambar

```
001   def mousePoints(event, x, y, flags, params):
002       global counter, current_polygon
003
004       # If left mouse button is clicked
005       if event == cv2.EVENT_LBUTTONDOWN:
006
007           current_polygon.append((x, y))
008
009           # If we have collected four points for one polygon
010           if len(current_polygon) == 4:
011               polygons.append(current_polygon)
012               current_polygon = []
013               counter += 1  # Increment the counter
014               print(polygons)  # Print the collected polygons
```

Potongan kode di atas merupakan fungsi yang digunakan untuk mengambil koordinat pada citra kamera. Baris 005 bertugas untuk mengambil koordinat jika *mouse* diklik. Koordinat yang telah diambil akan disimpan pada sebuah *array*. Fungsi tersebut akan diulang hingga *array* yang menampung koordinat tersebut sudah berisi empat.

```
001   if counter == totalSpaces:
002       fileObj = open('Polygons.p', 'wb')
003       pickle.dump(polygons, fileObj)
004       fileObj.close()
```

```
005       print("Saved all polygon points.")
006           Break
007   cv2.imshow("Original Image", img)
008   cv2.setMouseCallback("Original Image", mousePoints)
```

Potongan kode di atas pada baris 008 adalah baris kode yang memanggil fungsi mousePoints sebanyak variabel totalSpaces. Total spaces dapat diatur sesuai dengan kebutuhan. Ketika variabel totalSpaces sudah terpenuhi maka *array* yang menampung semua koordinat akan disimpan menggunakan library pickle dengan nama Road.p ke *folder* yang sama dengan lokasi *folder* kode tersebut. *File* tersebut dapat disebut dengan *file* polygons. Selanjutnya *file* Road.p akan digunakan untuk menentukan area yang akan memicu arduino untuk menggerakan servo.

```
001   polygonKeluar1 = 'Polygons.p'
002   file_obj_keluar1 = open(polygonKeluar1, 'rb')
003   roisKeluar1 = pickle.load(file_obj_keluar1)
004   file_obj_keluar1.close()
```

Potongan kode di atas merupakan kode yang akan menginisialisasi *file* polygons yang telah dibuat sebelumnya. Baris 002 berfungsi untuk membuka *file* dengan nama yang telah ditampung pada variabel polygonParkir1 yaitu Polygons.p. Selanjutnya isi dari *file* polygons akan ditampung pada variabel "roisParkir1", lalu *file* polygons akan ditutup kembali pada baris 004.

Tahap selanjutnya adalah menggunakan polygons yang telah diambil untuk menjadi polygons yang dapat memicu arduino untuk mengirimkan perintah. Untuk memenuhi tugas itu program membandingkan titik tengah polygons dengan titik tengah dari *bounding box* pada objek yang terdeteksi. Kode di bawah ini merupakan kode yang berfungsi untuk membandingkan titik tengah dari polygons dan *bounding box* dari objek.

```
001   empty_status1 = overlay_polygons(img1, object_list1,
 roisKeluar1, _draw_occupied=True)
```

Pada potongan kode di atas terdapat beberapa parameter yang digunakan. Parameter “img1” merupakan citra kamera yang digunakan. Parameter “object_list1” merupakan parameter yang menampung klasifikasi dari objek, dan titik tengah *bounding box* dari objek yang terdeteksi. Parameter roisKeluar1 merupakan polygons yang akan digunakan untuk pembanding. Luaran dari potongan kode di atas adalah boolean yang menyatakan tidak ada objek di dalam polygons dengan boolean *true* dan *false* jika ada objek di dalam polygons. Dari luaran ini program akan mengambil klasifikasi objek yang ada di dalam polygons menggunakan potongan kode di bawah ini.

```
001   if ((len(object_list1) != 0) and (empty_status1 == False)):
002       # Check if any car is present in this polygon
003       for obj in object_list1:
004           car_center = obj[“center”]
005           result = cv2.pointPolygonTest(keluar_space1,
   car_center, False)
006           if result > 0:
007               center1 = obj
```

Jenis objek yang ada di dalam polygons akan disimpan di dalam variabel “center1”. Dari informasi mengenai ada atau tidaknya objek di dalam polygons dan klasifikasi objek di dalam polygons maka program dapat memberikan kondisi yang cukup untuk menggerakan servo.

```
001   if ((center1["class"] == "SUV") and (statusPalang1 ==
   False)):
002       write("2")
003       statusPalang1 = True
004   elif((center1["class"] != "SUV") and (statusPalang1 ==
   True)):
005       write("3")
```

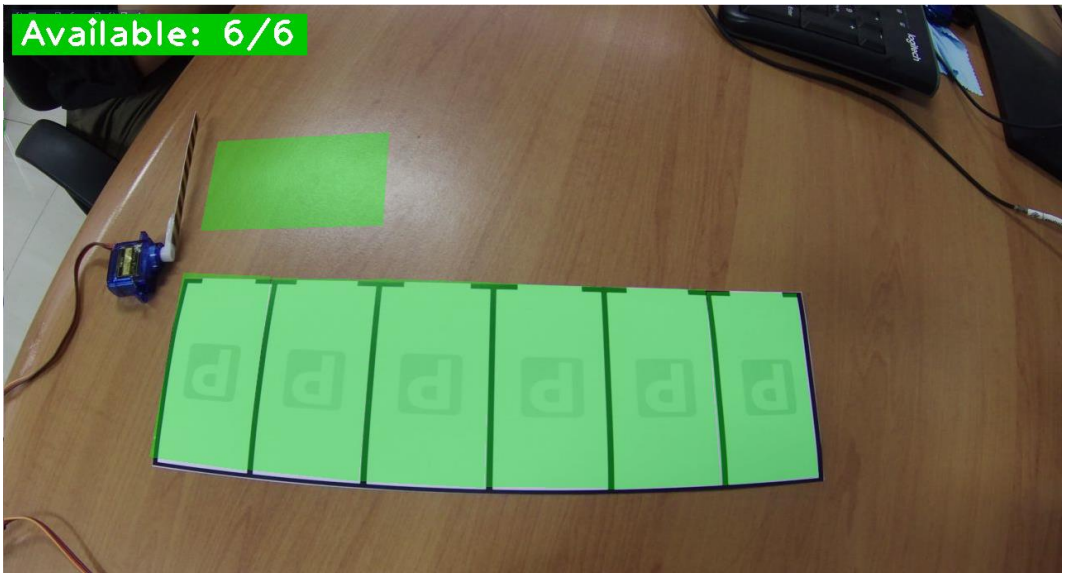
Potongan kode di atas berfungsi untuk mengirimkan string menuju arduino berdasarkan kondisi yang telah ditentukan. Kondisi yang digunakan adalah jika objek yang tedeteksi dengan klasifikasi SUV dan palang dalam kondisi tertutup (False), maka program akan mengirimkan string 2 ke arduino dan jika tidak klasifikasi objek yang ada di dalam polygons bukan SUV makan program akan

mengirimkan string 3. Terdapat beberapa string yang digunakan pada program. Masing – masing string melambangkan servo dan fungsi yang berbeda. string yang digunakan adalah 2,3,4,5,6,7. Program mengirimkan string menggunakan koneksi serial melalui *port* USB. Fungsi “write” pada program merupakan cara program untuk mengirim string menuju arduino.

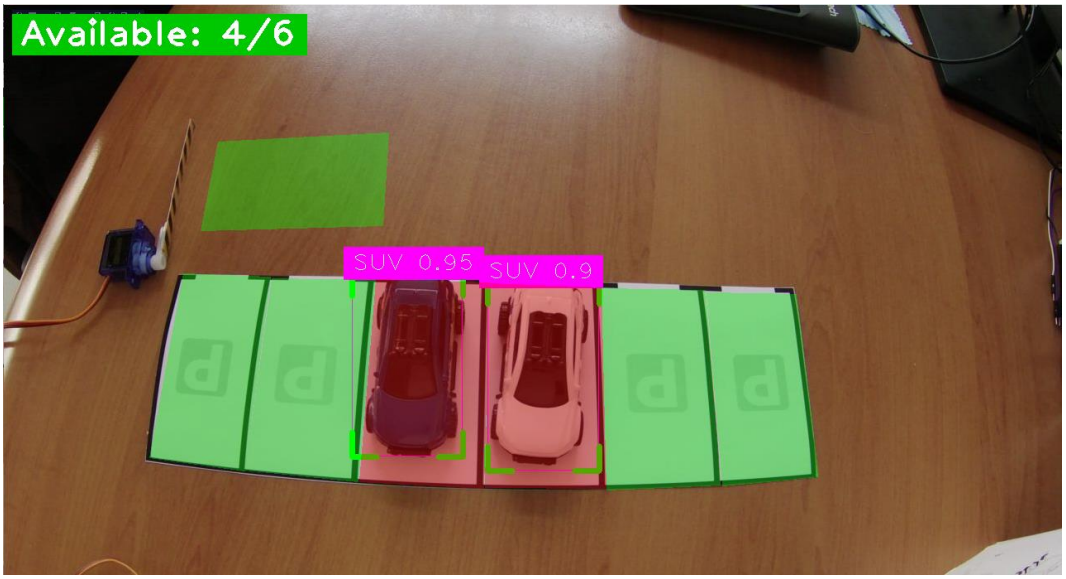
```
001   arduino = serial.Serial(port='COM7', baudrate=9600,
   timeout=.1)
002   def write(x):
003       arduino.write(bytes(x, 'utf-8'))
004       time.sleep(0.05)
```

Potongan kode diatas merupakan fungsi “write” pada program. Potongan kode di atas pada baris 001 merupakan bagian yang memulai koneksi dengan arduino dengan koneksi menggunakan *port* COM7, dan *baudrate* 9600. Di dalam fungsi “write” pada baris 003 merupakan bagian yang mengirimkan string dengan variabel x menuju ke arduino.

Selain polygons yang memicu arduino untuk menggerakan servo terdapat polygons yang digunakan untuk menghitung jumlah lahan parkir yang tersedia. Polygons pada lahan parkir tidak membutuhkan jenis objek yang diklasifikasi karena hanya kode hanya pertujuan untuk menghitung miniatur kendaraan yang terparkir. Gambar 4.18 menunjukan jika tidak ada objek di dalam polygons. Gambar 4.19 menunjukan jika terdeteksi ada objek di dalam polygons.



Gambar 4.18 Polygons tanpa objek



Gambar 4.19 Polygons dengan objek

```
001    ava1 = count_Spaces(4, object_list1, roisParkir1)
002    if ava1 == 0:
003        color = (0, 0, 255)  # Red for no available spaces
004    else:
005        color = (0, 200, 0)  # Green for available spaces
006    cvzone.putTextRect(img1, f"Available: {ava1}/4", (20, 50),
    colorR=color)
```

Potongan kode di atas merupakan kode yang menghitung jumlah objek di dalam polygons. Kode tersebut menghitung jumlah objek menggunakan fungsi count_Spaces. Pada baris 003 merupakan warna yang digunakan jika ada objek pada polygons, dan baris 005 adalah warna yang digunakan jika tidak ada objek pada polygons.

4.4 Pembuatan Program Arduino

Program arduino memiliki tugas utama yaitu untuk memulai koneksi dengan ketiga servo dan memerintah servo untuk bergerak sesuai dengan string yang diterima dari program python. Masing – masing perintah pada servo memiliki stringnya sendiri.

```
001   #include <Servo.h>
002
003   Servo servo1;  // create servo object to control a servo
004   Servo servo2;
005   Servo servo3;
006
007   void setup() {
008     Serial.begin(9600);
009     Serial.setTimeout(1);
010     servo1.attach(9);
011     servo2.attach(6);
012     servo3.attach(5);
013   }
```

Potongan kode di atas merupakan kode yang diggunakan untuk *setup* koneksi dan memulai koneksi dengan ketiga servo. *Setup* dimulai dengan baris 001 dengan memanggil library servo. Selanjutnya pada baris 003 hingga 005 variabel servo akan diinisialisasi dengan servo1, servo2, dan servo3. *Setup* selanjutnya adalah menetapkan *baudrate* dari koneksi serial yaitu 9600 pada baris 008. Untuk mengakses servo maka servo akan di hubungkan dengan digital pin 5, 6, dan 9, pin yang terhubung dengan servo akan diinisialisasi pada baris 010 hingga 012.

```
001 void loop() {
002   if (Serial.available() > 0){
003         code = Serial.readString().toInt();
004     }
005   }
```

Potongan kode diatas adalah kode yang bertugas untuk membaca string yang dikirim oleh program python dengan koneksi serial dan mengirim perintah kepada servo untuk bergerak. Pada baris 002 adalah bagian yang menunggu koneksi serial untuk menerima data. Pada baris 003 adalah bagian yang membaca data yang telah dikirim melalui koneksi serial serta mengubah data yang dikirim dalam format string ke format *integer*. Data tersebut akan ditampung dalam variabel "code" untuk nantinya menjadi kondisi untuk menggerakan servo.

Selanjutnya merupakan bagian kode yang akan bertugas untuk menggerakan servo, namun terdapat perbedaan pada salah satu servo dari tiga servo yang digunakan. Pada umumnya mini servo dengan jenis SG90 merupakan *continuous* servo, yang berarti bahwa servo tidak memiliki sensor yang mendeteksi sudut putaran servo, sehingga pada servo jenis ini jika kode pada arduino mengirimkan integer 0 hingga 89 servo akan bergerak berlawanan arah dengan jarum jam dan integer 91 hingga 180 servo akan bergerak searah jarum jam. Semakin jauh integer yang dikirimkan dengan nilai 90 maka semakin cepat servo berputar. Integer 90 akan memerintahkan servo untuk berhenti berputar. Pada satu servo yang berbeda ini, servo memiliki sensor yang mendeteksi sudut putaran servo sehingga integer yang dikirimkan ke arduino merupakan sudut putar yang akan dituju oleh servo.

```
001       if (code == 2){
002         servo1.write(180)
003       }
004       else if (code == 3){      //Pin 9
005         delay(500);
006         servo1.write(0);
007
008       }
```

Potongan kode di atas merupakan kode yang bertugas untuk mengatur servo dengan sensor sudut putar yang terhubung dengan pin 9. Pada baris 001 dan 002 adalah bagian yang akan membuka palang dengan memutar servo dengan sudut 180 jika isi dari variabel "code" adalah "2". Pada baris 004 hingga 006 adalah bagian yang akan menutup palang dengan memutar servo dengan sudut 0 atau kembali ke sudut awal setelah 500ms terlewati. *Delay* 500ms digunakan untuk kebutuhan demo.

```
001       if (code == 4){
002         servo2.write(81);
003         delay(500);
004         servo2.write(90);           //Pin 6
005       }
006       else if (code == 5){
007         delay(500);
008         servo2.write(104);
009         delay(491);
010         servo2.write(90);
011       }
012       if (code == 6){
013         servo3.write(81);
014         delay(500);
015         servo3.write(90);           //Pin 5
016       }
017       else if (code == 7){
018         delay(500);
019         servo3.write(104);
020         delay(491);
021         servo3.write(90);
022       }
```

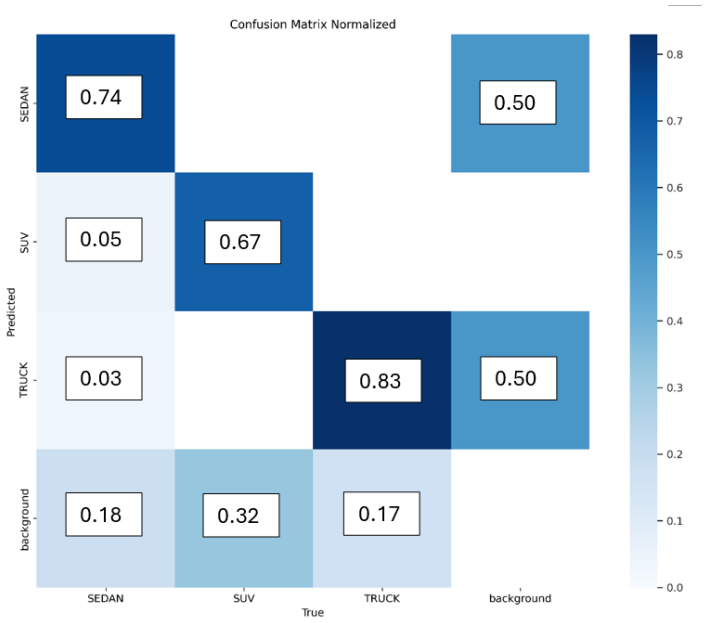
Kode diatas merupakan kode yang bertugas untuk menggerakkan servo lainya yang merupakan servo *continuous* yang terhubung pada pin 5 dan 6. Baris 001 hingga 004 bertugas untuk membuka palang pada servo yang terhubung pada pin 6 jika isi variabel "code" adalah 4 dengan mengirimkan integer 81 ke servo, yang berarti servo akan bergerak berlawanan arah dengan jarum jam selama 500ms. Selanjutnya kode akan mengirimkan integer 90 untuk menghentikan servo untuk berputar. Selanjutnya pada baris 006 hingga 010 memiliki tugas untuk menutup palang pada servo yang terhubung dengan pin 6. Kode akan menutup palang dengan mengirimkan integer 104 setelah *delay* 500ms di awal untuk keperluan demo, yang berarti setelah *delay* 500ms servo akan berputar searah

jarum jam. Selanjutnya kode akan menghentikan kode dengan mengirimkan integer 90 setelah *delay* 491ms. *Delay* dan kecepatan pada servo *continuous* terlihat bukan merupakan angka bulat untuk meningkatkan konsistensi sudut putar servo.

4.5 Pengujian Sistem

Bagian ini merupakan bagian terakhir dalam pembuatan program yaitu pengujian pada model YOLO dan pengujian arduino. Pengujian dilakukan untuk mengetahui performa dari model YOLO yang telah melalui proses training dengan *data set* dan klasifikasi khusus. Pengujian yang dilakukan adalah *precision*, *recall*, dan F1 – *score* serta terdapat *confusion matrix* yang akan merangkum hasil deteksi dari model.

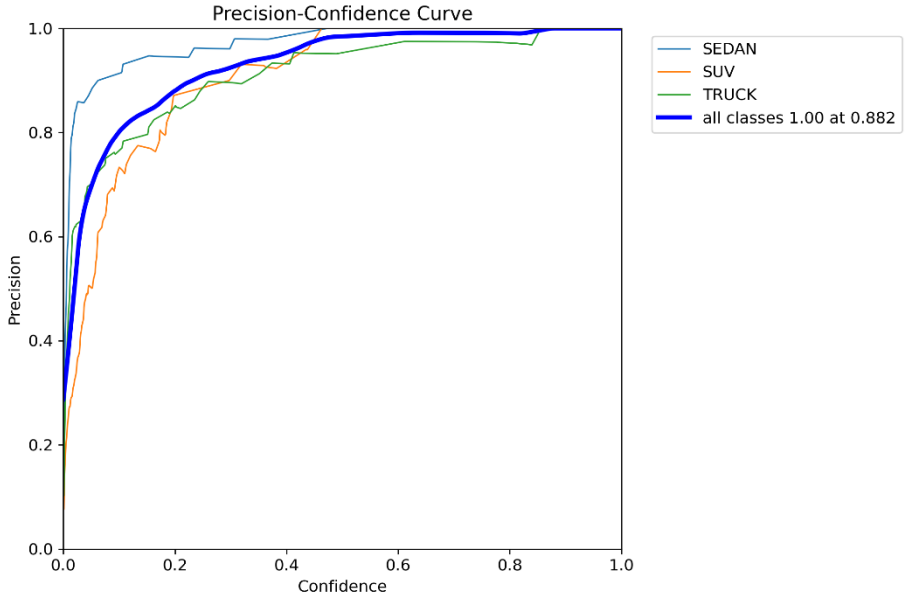
Pengujian yang dilakukan telah secara otomatis dilakukan saat *training* berlangsung. Setiap *epoch* yang selesai maka model akan melalui pengujian menggunakan *test set* yang telah disiapkan. Untuk melihat kemampuan deteksi secara *general* pada model, *confusion matrix* dapat digunakan.



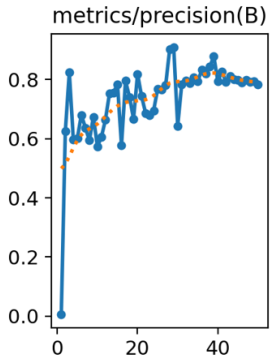
Gambar 4.20 *Confusion matrix*

Pada gambar 4.20 di atas deteksi yang dilakukan oleh model tersebar ke seluruh bagian, namun sebagian besar objek yang terdeteksi terdapat pada area yang menunjukan TP (*True Positive*). *Matrix* pada gambar 4.20 menunjukan persentase dari kemampuan model untuk mendeteksi objek. Bagian horizontal merupakan keadaan sesungguhnya dari objek sedangkan bagian vertical adalah luaran deteksi dari model. Dari *confusion matrix* di atas objek yang terdeteksi dengan benar oleh model ada pada angka 0.67 hingga 0.83 hal ini menunjukan lebih dari 50% citra dapat terdeteksi dengan akurat. Nilai lainya tersebar ke bagian lain yaitu *false negative* (FN) dan *false positive* (FP)

Selanjutnya adalah pengujian *precision* dan *recall*. Pengujian *precision* dilakukan untuk mengetahui seberapa sering model mendeteksi klasifikasi yang benar dari objek, *precision* memberi gambaran dari kemampuan model untuk menghindari deteksi *false positive* dengan nilai 0 jika model tidak pernah mengklasifikasikan objek dengan benar dan nilai 1 jika model selalu benar dalam mengklasifikasikan objek.



Gambar 4.21 *Precision – confidence curve*

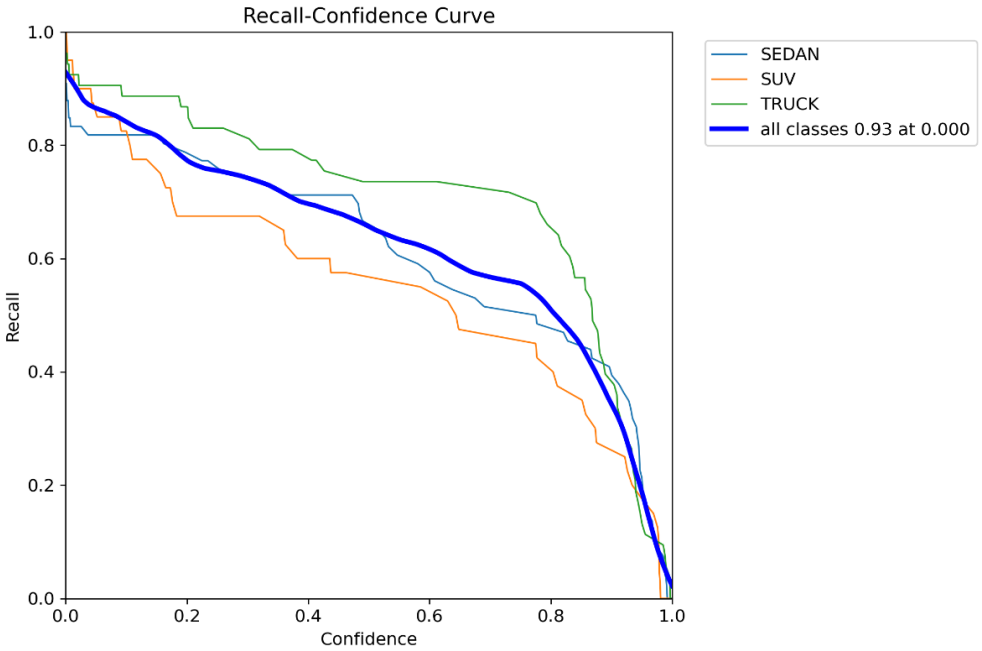


Gambar 4.22 *Precision* per *epoch*

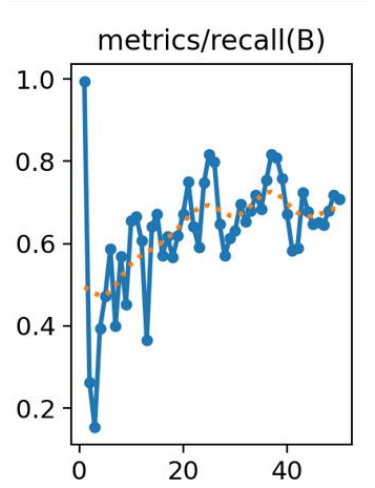
Pada gambar 4.21 merupakan grafik yang menunjukan hubungan *precision* jika dibandingkan dengan *confidence*. Grafik tersebut memberikan nilai *precision* pada nilai *confidence* tertentu. Dari grafik pada gambar 4.21 menunjukan bahwa *precision* pada model meningkat seiring dengan semakin tingginya *confidence* atau model akan semakin presisi dalam menentukan klasifikasi seiring dengan meningkatnya *confidence* saat mendeteksi klasifikasi objek, hingga pada *confidence* 0.9 dapat dipastikan bahwa semua objek yang terdeteksi maka akan dengan benar diklasifikasikan.

Gambar 4.22 merupakan grafik yang menunjukan pengujian *precision* pada model tiap *epoch* yang selesai saat proses *training* dilakukan. Pada grafik tersebut nilai *precision* meningkat pada beberapa *epoch* pertama lalu nilai *precision* bertahan pada 0.6 hingga 0.8.

Selanjutnya merupakan pengujian *recall*. Pengujian ini dilakukan untuk mengetahui kemampuan dari model untuk dapat mendeteksi semua objek yang ada pada klasifikasi objek. Pengujian ini memberikan gambaran tentang model untuk mengabaikan objek selain objek yang ada pada klasifikasi.



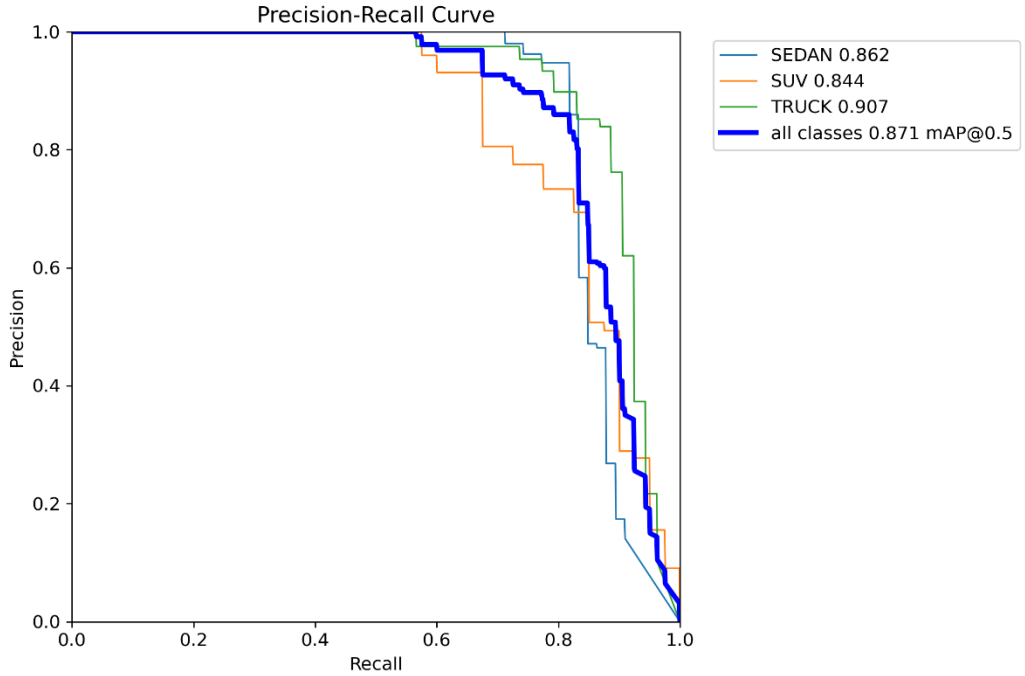
Gambar 4.23 *Recall – confidence curve*



Gambar 4.24 *Recall* per *epoch*

Gambar 4.23 merupakan grafik yang menunjukan hubungan antara nilai *recall* dan *confidence*. Nilai *recall* akan sangat tinggi pada nilai *confidence* 0 karena dengan nilai *confidence* 0 makan semua objek akan terdeteksi sebagai objek di dalam klasifikasi. Hal sebaliknya terjadi saat nilai *confidence* 1 maka nilai dari *recall* akan sangat rendah karena model hanya akan mendeteksi objek jika model memiliki *confidence* dengan nilai 1 atau pasti benar.

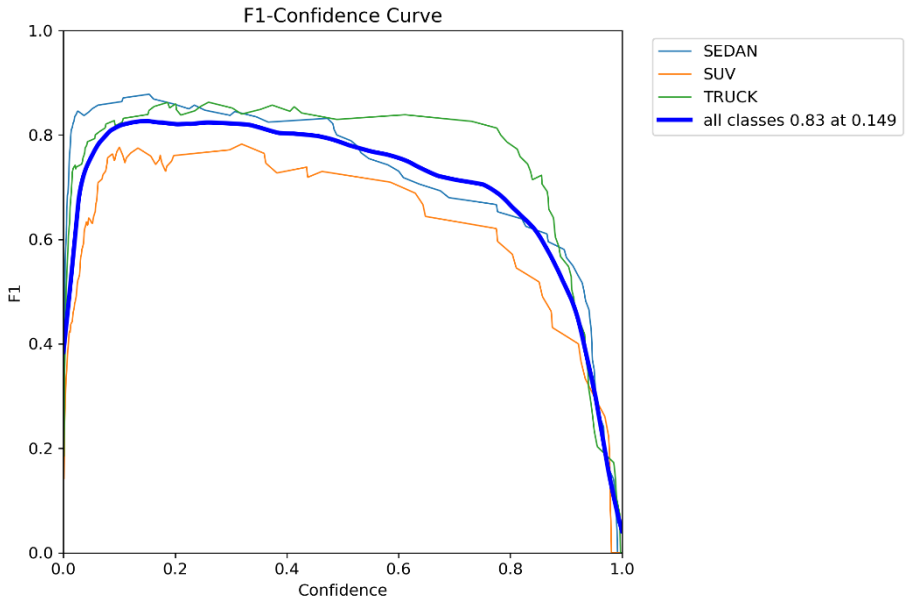
Gambar 4.24 adalah grafik yang menunjukan pengujian *recall* tiap *epoch* yang selesai dalam proses *training* model. Nilai recall pada *epoch* pertama yang didapat sangat tinggi dikarenakan model belum memiliki akurasi yang dibutuhkan untuk mendeteksi objek dengan benar sehingga pada tahap *epoch* pertama model mendeteksi semua objek termasuk ke dalam objek dalam klasifikasi. Selanjutnya nilai dari *recall* bertahan pada nilai 0.6 hingga 0.8.



Gambar 4.25 *Precision – recall curve*

Gambar 4.25 adalah grafik yang menunjukan hubungan antara *precision* dan *recall* untuk lebih menggambarkan kemampuan dari model. Pada grafik di atas nilai *precision* stabil pada nilai 1 pada *recall* dengan nilai 0 hingga 0.4, yang berarti model dapat mengklasifikasi dengan sangat akurat namun objek tidak selalu terdeteksi hanya kurang dari setengah dari seluruh *data set* yang terdeteksi. Sebaliknya saat *recall* bernilai 1 maka *precision* ada pada kisaran nilai 0.1 hingga 0.3 yang berarti hampir semua objek terdeteksi sebagai objek di dalam klasifikasi namun objek yang terdeteksi belum tentu objek yang seharusnya. Titik paling

stabil adalah pada nilai *recall* dan *precision* 0.8 dimana 80% objek akan terdeteksi dan terklasifikasi dengan cukup akurat.



Gambar 4.26 F1 – *score curve*

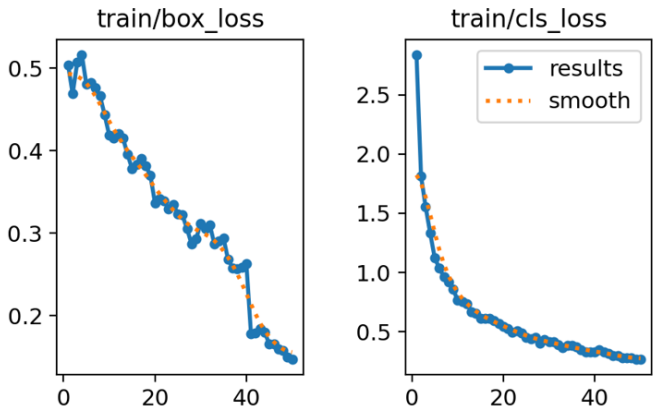
Pengujian f1 – *score* merupakan cara untuk mengetahui performa dari model. Pengujian ini adalah perhitungan *harmonic* mean dari *precision* dan *recall* yang memberikan nilai performa dari model. Semakin dengan nilai f1 ke 1 makan semakin tinggi keseimbangan antara *recall* dan *precision* tercapai yang berarti semakin tinggi performa dari model. Performa yang dimaksud adalah kemampuan model untuk mendeteksi objek dan mengklasifikasikannya dengan benar

Gambar 4.26 merupakan grafik yang menunjukan hubungan nilai f1 dan *confidence*. Pada grafik tersebut menunjukan model kesusahan untuk melakukan pendeteksian jika *confidence* yang diminta adalah 1. Pada pengujian ini hal yang sama terjadi saat *confidence* pada nilai 0 model tidak memberikan deteksi yang akurat dikarenakan saat nilai *confidence* 0 semua objek dapat dideteksi sebagai objek di dalam klasifikasi. Titik paling seimbang adalah dengan melimit *confidence* pada 0.5 hingga 1 sehingga performa yang didapat dari model ada

pada titik tertinggi namun nilai *confidence* tetap tinggi. Pengujian juga dilakukan dengan *data set* yang baru dengan jenis kendaraan yang tidak ada pada *data set* untuk *training*, *validation*, dan *test*.

4.5.1 Pencatatan *loss* dan mAP

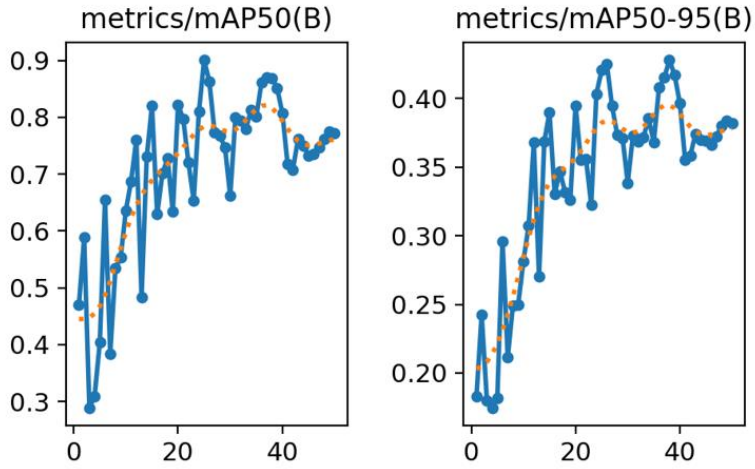
Loss merupakan suatu nilai yang yang mengukur seberapa dekat nilai prediksi dari model dengan nilai dari target yang didapat dari *data set*, dengan kata lain nilai *loss* dapat dijadikan acuan untuk mengetahui nilai *error* dari model dalam melakukan prediksi selama proses *training*. Semakin dekat nilai prediksi model dengan 0.0 maka semakin kecil *error* dari model atau semakin akurat model melakukan prediksi. mAP (*mean Average Precision*) adalah nilai rata-rata dari *precision* pada tiap tahapan nilai *recall* di dalam IoU (*Intersection over Union*) yang dihasilkan oleh model dan *bounding box* yang telah di tetapkan. Nilai ini dapat dijadikan acuan untuk akurasi dari model dalam melakukan deteksi dalam IoU tertentu.



Gambar 4.27 Grafik *loss*

Gambar 4.27 merupakan gambar yang menunjukan nilai *loss* dari prediksi model untuk *bounding box* di sebelah kiri dan klasifikasi di sebelah kanan pada saat proses training yang diambil tiap *epoch*-nya. Dari

kedua grafik terdapat tren menurun yang berarti dalam proses training, tiap *epoch*-nya model semakin akurat dalam melakukan prediksi untuk *bounding box* dan klasifikasi, namun pada grafik klasifikasi model lebih cepat untuk belajar.



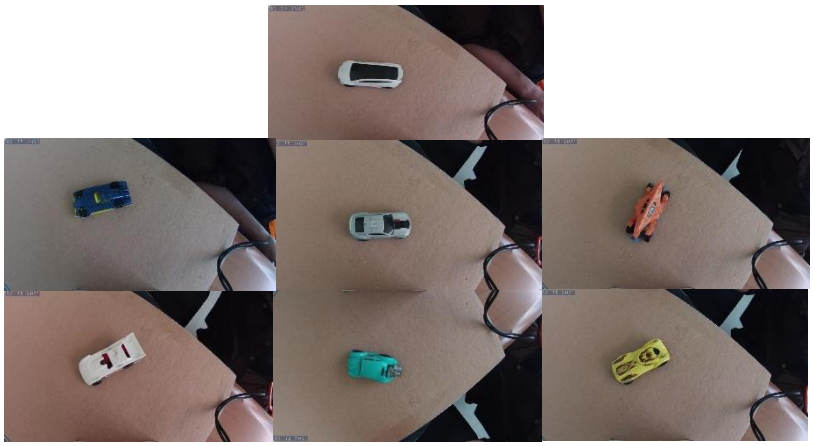
Gambar 4.28 Grafik mAP50 dan mAP50-95

Gambar 4.28 menunjukan nilai dari mAP pada model yang diambil tiap *epoch*-nya yaitu sebanyak 50 *epoch*. mAP50 merupakan perhintungan nilai mAP dengan IoU sebesar 50%, sedangkan mAP50-95 adalah perhitungan nilai mAP dengan IoU sebesar 50% hingga 95% dengan loncatan sebanyak 5%. Kedua grafik mengalami peningkatan pada tiap *epoch*-nya selama proses *training.* Dari kedua grafik ini dapat disimpulkan bahwa akurasi dari model pada mAP50 berada pada nilai 0.7 hingga 0.8 pada epoch terakhir, sedangkan mAP50-95 berada pada nilai 0.35 hingga 0.40 pada epoch terakhir.

4.5.2 Pengujian *data set* baru

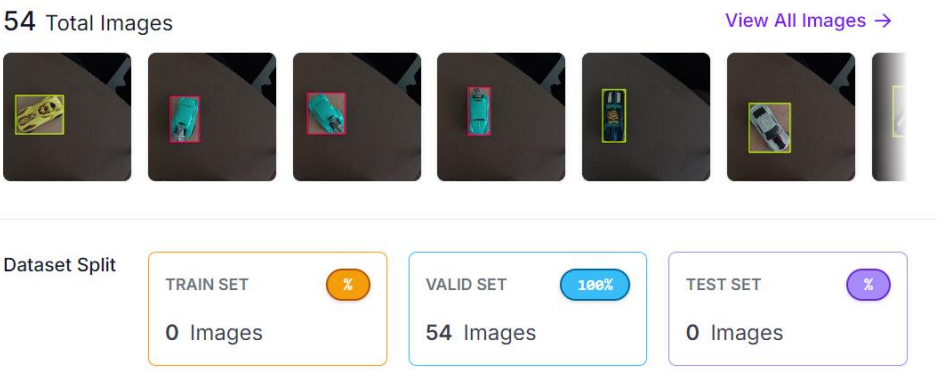
Pengujian ini dilakukan dengan menggunakan *data set* baru. *Data set* baru yang digunakan adalah *data set* yang tidak pernah digunakan dalam proses training. *Data set* baru yang digunakan, melalui tahap yang

sama dengan data sebelumnya yang digunakan untuk *training* model namun tidak melalui tahapan *training*.



Gambar 4.29 Citra baru

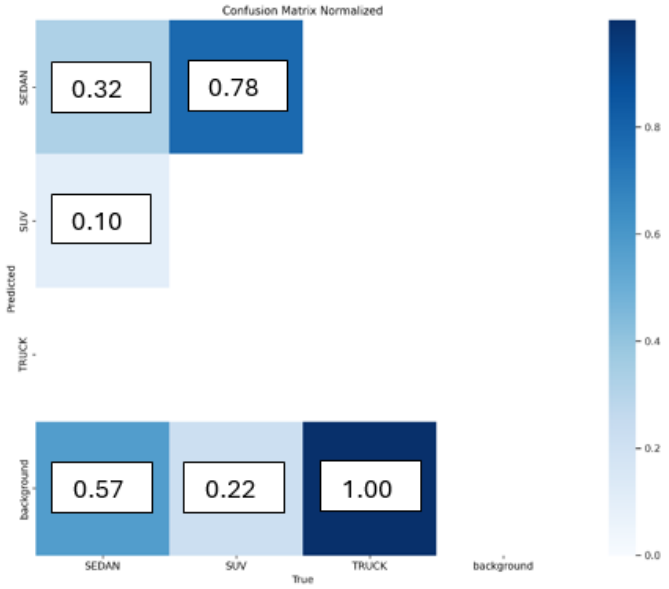
Gambar 4.29 merupakan beberapa citra yang ditangkap untuk dijadikan *data set* baru. Selanjutnya citra yang ditangkap dengan objek baru ini, akan melalui tahap *preprocessing* dengan *tools* yang sama, yaitu menggunakan roboflow. Jumlah citra yang ditangkap sebanyak 60 namun setelah melalui tahap *data filtering* jumlah citra yang dapat digunakan adalah sebesar 54. Citra akan diberikan *tag* berupa *bounding box*, koordinat, dan klasifikasi dari objek. Perbedaan dari pembuatan *data set* ini adalah bagian terakhir yaitu *data splitting*, dimana data hanya dialokasikan untuk *validation set* seperti pada gambar 4.30.



Gambar 4.30 *Data splitting* dari *data set* baru

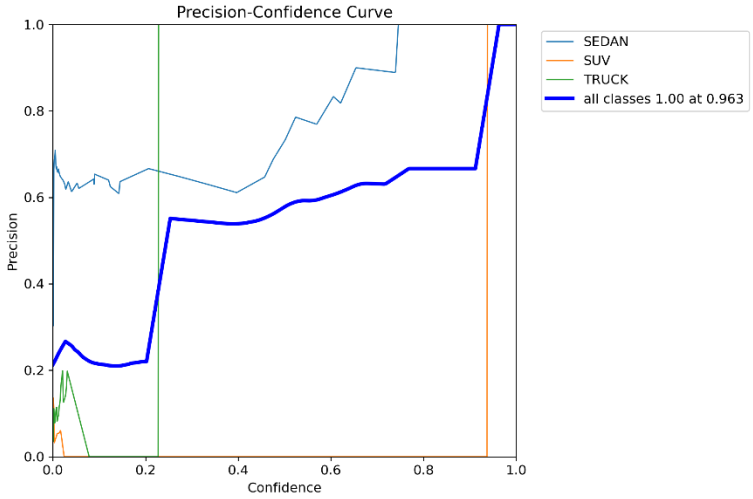
Selanjutnya *data set* akan melewati tahap validasi dengan menggunakan google colab dan model yang telah di-*training* sebelumnya, untuk mendapatkan hasil pengujian dari model tersebut dengan *data set* yang berbeda. Potongan kode di bawah merupakan potongan kode yang digunakan untuk melakukan proses *validation*.

```
001   !yolo val model=/content/drive/MyDrive/best.pt
   data=/content/Unknown-data-1/data.yaml
```



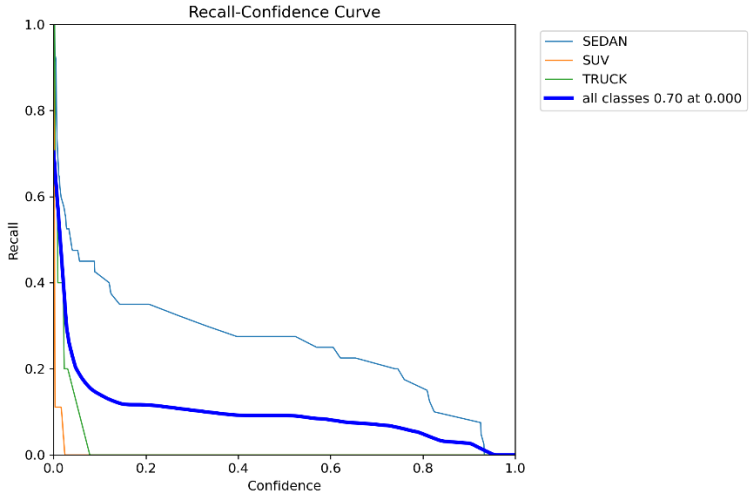
Gambar 4.31 *Confusion matrix data set* baru

Pada gambar 4.31 merupakan *confusion matrix* yang dihasilkan dari *data set* dengan objek baru. Dari *confusion matrix* tersebut dapat disimpulkan bahwa sebagian besar dari objek tidak dapat terdeteksi oleh model. Terdapat 32% kemungkinan untuk objek dengan klasifikasi SEDAN dapat terdeteksi dengan benar, namun untuk objek lainya dengan klasifikasi SUV dan TRUCK tidak dapat terdeteksi.



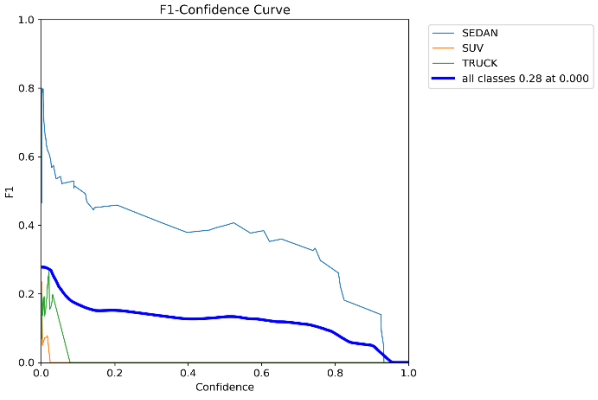
Gambar 4.32 *Precision-Confidence Curve* data baru

Gambar 4.32 merupakan grafik *precision* yang dihasilkan dengan *data set* baru. Dari grafik tersebut objek dengan klasifikasi SEDAN dapat terdeteksi dengan mudah dengan nilai *precision* yang selalu di atas nilai 0.6 di seluruh nilai *confidence*. Untuk klasifikasi TRUCK dan SUV terdapat lonjakan yang sangat signifikan pada nilai *confidence*, dimana nilai *precision* dari klasifikasi TRUCK melonjak dari 0 hingga 1.0 pada nilai *confidence* di atas 0.2. Sama halnya dengan klasifikasi SUV terdapat lonjakan nilai *confidence* dari 0 hingga 1.0 pada nilai *confidence* di atas 0.9. Dapat disimpulkan bahwa model dapat mengklasifikasikan objek dengan sangat akurat saat nilai *confidence* di atas 0.8.



Gambar 4.33 *Recall-Confidence Curve* data baru

Gambar 4.33 merupakan grafik *recall* yang dihasilkan dengan *data set* baru. Dari grafik tersebut objek dengan klasifikasi SEDAN dapat memiliki nilai *recall* yang relatif rendah yaitu di bawah 0.4 pada nilai *confidence* 0.5 hingga 1.0. Untuk objek dengan klasifikasi SUV dan TRUCK nilai *recall* turun ke nilai 0 dengan pada nilai *confidence* di bawah 0.2. Dapat disimpulkan bahwa pada nilai *confidence* 0.5 hingga 1.0 objek dengan klasifikasi sedan dapat dideteksi dengan kemungkinan di bawah 40%. Sedangkan untuk objek dengan klasifikasi TRUCK dan SUV tidak dapat terdeteksi dengan nilai *confidence* 0.5.



Gambar 4.34 *F1-Curve* data baru

Gambar 4.34 adalah grafik F1 yang dihubungkan dengan nilai *confidence.* Dari grafik tersebut dapat terlihat bahwa performa dari model relatif rendah dengan nilai F1 di bawah 0.5 pada nilai *confidence* 0.5 hingga 1.0 untuk objek dengan klasifikasi SEDAN. Untuk nilai F1 objek dengan klasifikasi SUV dan TRUCK di bawah 0.4 pada nilai *confidence* 0.0 hingga 0.2. Dari grafik tersebut dapat disimpulkan bahwa model tidak dapat menghasilkan performa yang baik jika dihadapkan pada *data set* baru yang tidak ada pada *data set* saat proses *training*, hanya objek dengan klasifikasi SEDAN yang dapat terdeteksi. Sebaliknya objek dengan klasifikasi SUV dan TRUCK tidak dapat terdeteksi .

Bab V
Kesimpulan dan Saran

5.1 Simpulan

Penelitian menghasilkan model yang mampu untuk mendeteksi dan mengklasifikasikan objek dengan nilai F1 – *score* tertinggi pada 0.83 pada *confidence* 0.149. Penggunaan model untuk melakukan klasifikasi objek dengan *reliable* maka tingkat *confidence* dapat dibatasi pada nilai minimum 0.7 sehingga dengan nilai *confidence* yang tinggi tetap mendapatkan nilai F1 – *score* yang cukup tinggi. Waktu yang dibutuhkan untuk menjalankan *training* dengan menggunakan CPU adalah 5 jam sedangkan *training* menggunakan GPU adalah 9.6 menit.

Program dapat berjalan dengan mengintegrasikan empat kamera dengan proses citranya masing – masing, seluruh objek yang ingin dideteksi dapat dengan baik dideteksi oleh keempat kamera.

5.2 Saran

Untuk penelitian selanjutnya, peningkatan jumlah *data set* merupakan tahap yang dapat diambil untuk meningkatkan akurasi dan jenis klasifikasi dari objek. Jika peneliti ingin lebih jauh meningkatkan akurasi dari model maka jenis model YOLO yang lebih besar dapat digunakan untuk proses *training*, namun harus dipastikan perangkat keras yang digunakan untuk melakukan deteksi objek dapat menangani proses deteksi dengan performa yang tinggi.