# **LAMPIRAN**

**Lampiran 1**. Source Code Deteksi Objek dan Perhitungan Estimasi Kecepatan

|  |
| --- |
| # Import necessary packages  import cv2  import csv  import collections  import numpy as np  from tracker2 import \*  import time  import datetime  end = 0  ids\_lst = []  spd\_lst = []  frameIndex = 0  # Initialize Tracker  tracker = EuclideanDistTracker()  # Initialize the videocapture object  cap = cv2.VideoCapture('input/MVI\_2971.MP4')  input\_size = 416  fps = cap.get(cv2.CAP\_PROP\_FPS)  total\_frames = cap.get(cv2.CAP\_PROP\_FRAME\_COUNT)  frame\_count = 0  # Detection confidence threshold  confThreshold = 0.5  nmsThreshold = 0.2  font\_color = (0, 0, 255)  font\_size = 0.5  font\_thickness = 2  # Store Coco Names in a list  classesFile = "yolo-coco/skripsi.names"  classNames = open(classesFile).read().strip().split('\n')  # class index for our required detection classes  required\_class\_index = [0, 1, 2, 3, 4]  detected\_classNames = []  # Model Files  modelConfiguration = 'yolo-coco/yolov3-training.cfg'  modelWeigheights = 'yolo-coco/yolov3-training\_last.weights'  # configure the network model  net = cv2.dnn.readNetFromDarknet(modelConfiguration, modelWeigheights)  # Configure the network backend  net.setPreferableBackend(cv2.dnn.DNN\_BACKEND\_CUDA)  net.setPreferableTarget(cv2.dnn.DNN\_TARGET\_CUDA)  # Define random colour for each class  np.random.seed(42)  colors = np.random.randint(0, 255, size=(len(classNames), 3), dtype='uint8')  # Function for finding the center of a rectangle  def find\_center(x, y, w, h):  x1 = int(w / 2)  y1 = int(h / 2)  cx = x + x1  cy = y + y1  return cx, cy  # Function for speed vehicle  def speed\_vehicle(box\_id, img):  x, y, w, h, id, index = box\_id  # Find the center of the rectangle for detection  center = find\_center(x, y, w, h)  ix, iy = center  # # Draw circle in the middle of the rectangle  # cv2.circle(img, center, 2, (0, 0, 255), -1)  # # get speed  # if tracker.getsp(id) < tracker.limit():  # cv2.putText(img, str(tracker.getsp(id)) + " km/h ", (center), cv2.FONT\_HERSHEY\_COMPLEX\_SMALL,  # 0.5, (255, 255, 0), 1)  #  # if tracker.getsp(id) > tracker.limit():  # cv2.putText(img, str(tracker.getsp(id)) + " km/h ", (center), cv2.FONT\_HERSHEY\_COMPLEX\_SMALL, 0.5,  # (0, 0, 255), 1)  #  # s = tracker.getsp(id)  # if tracker.f[id] == 1 and s != 0:  # tracker.capture(img, x, y, h, w, s, id)  # Function for finding the detected objects from the network output  def postProcess(outputs, img):  global detected\_classNames  height, width = img.shape[:2]  boxes = []  classIds = []  confidence\_scores = []  detection = []  for output in outputs:  for det in output:  scores = det[5:]  classId = np.argmax(scores)  confidence = scores[classId]  if classId in required\_class\_index:  if confidence > confThreshold:  # print(classId)  w, h = int(det[2] \* width), int(det[3] \* height)  x, y = int((det[0] \* width) - w / 2), int((det[1] \* height) - h / 2)  boxes.append([x, y, w, h])  classIds.append(classId)  confidence\_scores.append(float(confidence))  # Apply Non-Max Suppression  indices = cv2.dnn.NMSBoxes(boxes, confidence\_scores, confThreshold, nmsThreshold)  # print(classIds)  if len(indices) > 0:  for i in indices.flatten():  x, y, w, h = boxes[i][0], boxes[i][1], boxes[i][2], boxes[i][3]  color = [int(c) for c in colors[classIds[i]]]  name = classNames[classIds[i]]  detected\_classNames.append(name)  # Draw classname and confidence score  cv2.putText(img, f'{name.lower()} {int(confidence\_scores[i] \* 100)}%', (x, y - 10),  cv2.FONT\_HERSHEY\_COMPLEX\_SMALL, 1, color, 1)  # Draw bounding rectangle  cv2.rectangle(img, (x, y), (x + w, y + h), color, 1)  detection.append([x, y, w, h, required\_class\_index.index(classIds[i])])  # Update the tracker for each object  boxes\_ids = tracker.update(detection)  for box\_id in boxes\_ids:  speed\_vehicle(box\_id, img)  while True:  (success, img) = cap.read()  if success == True:  frame\_count += 1  # percent = 0  # width = int(img.shape[1] \* percent / 100)  # height = int(img.shape[0] \* percent / 100)  # dim = (width, height)  cv2.resize(img, (1080, 720), fx=0.5, fy=0.5)  ih, iw, channels = img.shape  blob = cv2.dnn.blobFromImage(img, 1 / 255, (input\_size, input\_size), [0, 0, 0], 1, crop=False)  # Set the input of the network  net.setInput(blob)  layersNames = net.getLayerNames()  outputNames = [(layersNames[i - 1]) for i in net.getUnconnectedOutLayers()]  # Feed data to the network  outputs = net.forward(outputNames)  # Find the objects from the network output  postProcess(outputs, img)  # cv2.line(img, (0, 235), (2000, 235), (0, 255, 45), 2) # start 2  # cv2.line(img, (0, 255), (2000, 255), (0, 255, 255), 2) # start 1  #  # cv2.line(img, (0, 135), (2000, 135), (0, 255, 45), 2) # finish 2  # cv2.line(img, (0, 155), (2000, 155), (0, 255, 255), 2) # finish 1  # # Draw counting texts in the frame  # cv2.putText(img, "Jumlah Kendaraan Yang Melintas: " + str(tracker.count), (20, 40), cv2.FONT\_HERSHEY\_SIMPLEX,  # font\_size,  # font\_color,  # font\_thickness)  #  # cv2.putText(img, "Jumlah Kendaraan Yang Melanggar: " + str(tracker.exceeded), (20, 60),  # cv2.FONT\_HERSHEY\_SIMPLEX,  # font\_size,  # font\_color,  # font\_thickness)  # DISPLAY DATE, TIME, FPS & CURRENT FRAME  cv2.line(img, (0, 10), (2000, 10), (79, 79, 47), 30)  d = str(datetime.datetime.now().strftime("%d-%m-%y"))  t = str(datetime.datetime.now().strftime("%H-%M-%S"))  cv2.putText(img, f'DATE: {d} |', (25, 19), cv2.FONT\_HERSHEY\_PLAIN, 1.1, (255, 255, 255), 2)  cv2.putText(img, f'TIME: {t} |', (209, 19), cv2.FONT\_HERSHEY\_PLAIN, 1.1, (255, 255, 255), 2)  cv2.putText(img, f'FPS: {fps} |', (393, 19), cv2.FONT\_HERSHEY\_PLAIN, 1.1, (255, 255, 255), 2)  cv2.putText(img, f'FRAMES: {frame\_count} of {total\_frames} ', (510, 19), cv2.FONT\_HERSHEY\_PLAIN, 1.1,  (255, 255, 255), 2)  cv2.line(img, (0, 26), (2000, 26), (255, 255, 255), 2)  cv2.imwrite("output/output\_hujan/frame\_detection/frame-{}.png".format(frameIndex), img)  # DATA ALLOCATION  ids\_lst, spd\_lst = tracker.dataset()  # Show the frames  cv2.imshow('Output', img)  frameIndex += 1  if cv2.waitKey(1) == ord('q'):  tracker.end()  tracker.datavis(ids\_lst, spd\_lst)  end = 1  break  if end != 1:  tracker.end()  # Finally realese the capture object and destroy all active windows  cap.release()  cv2.destroyAllWindows() |

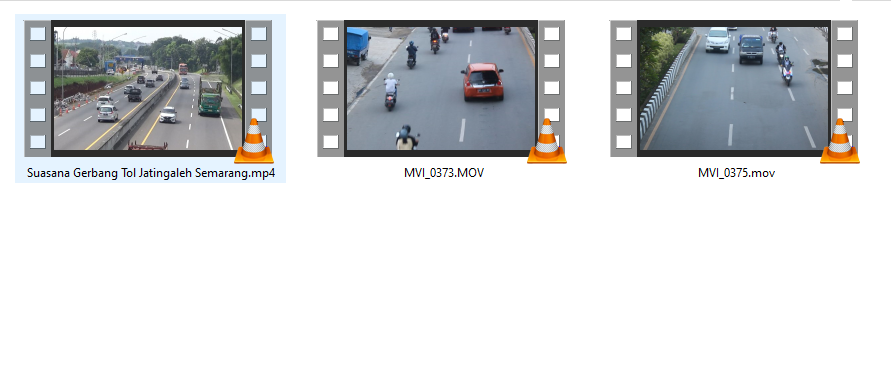
**Lampiran 2**. Source Code Tracking Objek

|  |
| --- |
| import math  import time  import numpy as np  import cv2  import matplotlib.pyplot as plt  from matplotlib import style  plt.rcParams.update({'font.size': 10})  limit = 40 # km/hr  distance = 16 # Field Of View Keseluruhan dalam sebuah footage  file = open("output/output\_hujan/SpeedRecord.txt", "w")  file.write("ID \t SPEED\n------\t-------\n")  file.close()  fps = 25  class EuclideanDistTracker:  def \_\_init\_\_(self):  # Store the center positions of the objects  self.center\_points = {}  self.id\_count = 0  # self.start = 0  # self.stop = 0  self.et = 0  self.s1 = np.zeros((1, 100000))  self.s2 = np.zeros((1, 100000))  self.s = np.zeros((1, 100000))  self.f = np.zeros(100000)  self.capf = np.zeros(100000)  self.count = 0  self.exceeded = 0  self.ids\_DATA = []  self.spd\_DATA = []  def update(self, objects\_rect):  objects\_bbs\_ids = []  # Get center point of new object  for rect in objects\_rect:  x, y, w, h, index = rect  cx = (x + x + w) // 2  cy = (y + y + h) // 2  # CHECK IF OBJECT IS DETECTED ALREADY  same\_object\_detected = False  for id, pt in self.center\_points.items():  dist = math.hypot(cx - pt[0], cy - pt[1]) # mengembalikan norma Euclidean  if dist < 40:  self.center\_points[id] = (cx, cy)  objects\_bbs\_ids.append([x, y, w, h, id, index])  same\_object\_detected = True  # START TIMER  if 235 <= y <= 255:  self.s1[0, id] = time.time()  # STOP TIMER and FIND DIFFERENCE  if 135 <= y <= 155:  self.s2[0, id] = time.time()  self.s[0, id] = self.s2[0, id] - self.s1[0, id]  # CAPTURE FLAG  if y <= 135:  self.f[id] = 1  break  # NEW OBJECT DETECTION  if same\_object\_detected is False:  self.center\_points[self.id\_count] = (cx, cy)  objects\_bbs\_ids.append([x, y, w, h, self.id\_count, index])  self.id\_count += 1  self.s[0, self.id\_count] = 0  self.s1[0, self.id\_count] = 0  self.s2[0, self.id\_count] = 0  new\_center\_points = {}  for obj\_bb\_id in objects\_bbs\_ids:  \_, \_, \_, \_, object\_id, index = obj\_bb\_id  center = self.center\_points[object\_id]  new\_center\_points[object\_id] = center  # Update dictionary with IDs not used removed  self.center\_points = new\_center\_points.copy()  return objects\_bbs\_ids  # SPEEED FUNCTION  def getsp(self, id):  if self.s[0, id] != 0:  s = distance / self.s[0, id] \* 3.6  else:  s = 0  return int(s)  # SAVE VEHICLE DATA  def capture(self, img, x, y, h, w, sp, id):  if (self.capf[id] == 0):  self.capf[id] = 1  self.f[id] = 0  crop\_img = img[y - 2:y + h + 2, x - 2:x + w + 2]  n = str(id) + "\_speed\_" + str(sp)  file = 'output/output\_hujan/detect/detect' + n + '.jpg'  cv2.imwrite(file, crop\_img)  self.count += 1  filet = open("SpeedRecord.txt", "a")  if sp > limit:  file2 = 'output/output\_hujan/overSpeed' + n + '.jpg'  cv2.imwrite(file2, crop\_img)  filet.write(str(id) + " \t " + str(sp) + " KM/H" + " <---overspeed\n")  self.exceeded += 1  if sp < limit:  filet.write(str(id) + " \t " + str(sp) + "\n")  filet.close()  self.ids\_DATA.append((id))  self.spd\_DATA.append((sp))  # STORE DATA  def dataset(self):  return self.ids\_DATA, self.spd\_DATA  # DATA VISUALIZATION  def datavis(self, id\_lst, spd\_lst):  x = id\_lst  y = spd\_lst  valx = []  for i in x:  valx.append(str(i))  plt.figure(figsize=(20, 5))  style.use('dark\_background')  plt.axhline(y=limit, color='r', linestyle='-', linewidth='5')  plt.bar(x, y, width=0.5, linewidth='3', edgecolor='yellow', color='blue', align='center')  plt.xlabel('ID')  plt.ylabel('SPEED')  plt.xticks(x, valx)  plt.legend(["speed limit"])  plt.title('SPEED OF VEHICLES CROSSING ROAD\n')  plt.savefig("output//datavis.png", bbox\_inches='tight', pad\_inches=1, edgecolor='w', orientation='landscape')  # SPEED\_LIMIT  def limit(self):  return limit  # TEXT FILE SUMMARY  def end(self):  file = open("output/output\_hujan/SpeedRecord.txt", "a")  file.write("\n-------------\n")  file.write("-------------\n")  file.write("SUMMARY\n")  file.write("-------------\n")  file.write("Total Vehicles :\t" + str(self.count) + "\n")  file.write("Vehicle Overspeed :\t" + str(self.exceeded))  file.close() |

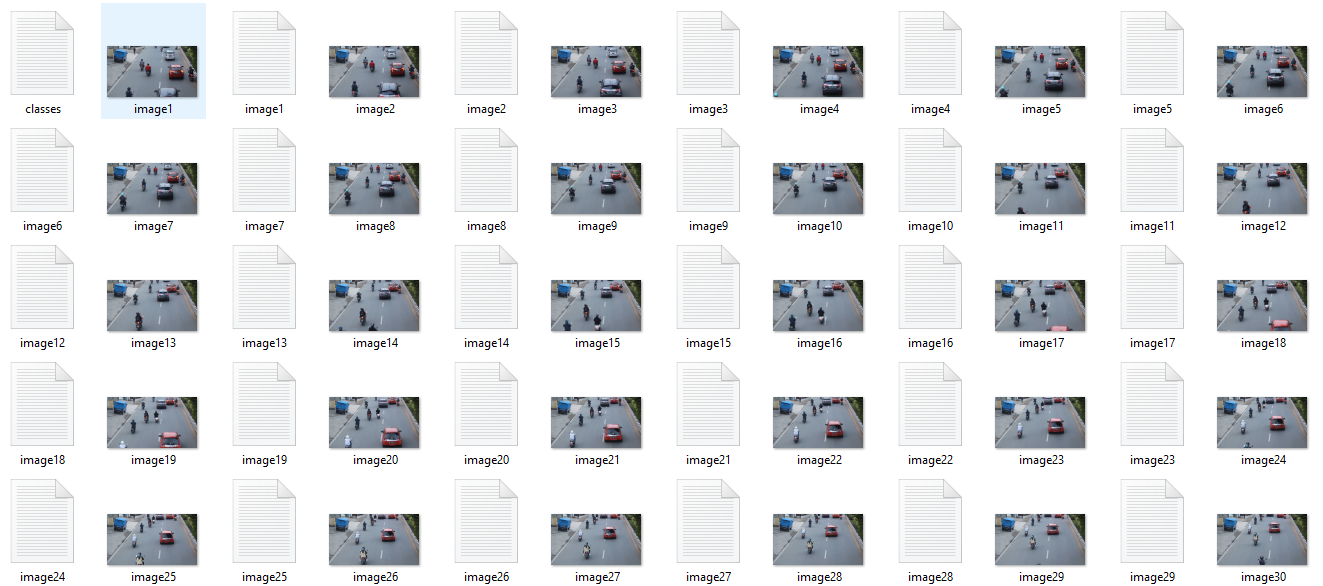
**Lampiran 3.** Source Code Pengenalan Plat Nomor Kendaraan

|  |
| --- |
| import cv2  import pytesseract  import numpy as np  import glob  pytesseract.pytesseract.tesseract\_cmd = 'C:\\Program Files\\Tesseract-OCR\\tesseract.exe'  class PlateReader:  def tesseract\_ocr(self, image, lang="eng1", psm=13, oem=3):  alphanumeric = "ABCDEFGHIJKLMNOPQRSTUVWXY0123456789"  options = "-l {} --psm {} --oem {} -c tessedit\_char\_whitelist={}".format(lang, psm, oem, alphanumeric)  return pytesseract.image\_to\_string(image, config=options)  class PlateDetector:  def load\_model(self, weight\_path: str, cfg\_path: str):  self.net = cv2.dnn.readNet(weight\_path, cfg\_path)  self.net.setPreferableBackend(cv2.dnn.DNN\_BACKEND\_CUDA)  self.net.setPreferableTarget(cv2.dnn.DNN\_TARGET\_CUDA)  with open("weights/detection/classes.names", "r") as f:  self.classes = [line.strip() for line in f.readlines()]  self.layers\_names = self.net.getLayerNames()  self.output\_layers = [self.layers\_names[i-1] for i in self.net.getUnconnectedOutLayers()]  def load\_image(self, img\_path):  img = cv2.imread(img\_path)  height, width, channels = img.shape  return img, height, width, channels  def detect\_plates(self, img):  blob = cv2.dnn.blobFromImage(img, scalefactor=0.00392, size=(320, 320), mean=(0, 0, 0), swapRB=True, crop=False)  self.net.setInput(blob)  outputs = self.net.forward(self.output\_layers)  return blob, outputs    def get\_boxes(self, outputs, width, height, threshold=0.3):  boxes = []  confidences = []  class\_ids = []  for output in outputs:  for detection in output:  scores = detection[5:]  class\_id = np.argmax(scores)  confidence = scores[class\_id]  if confidence > threshold:  center\_x = int(detection[0] \* width)  center\_y = int(detection[1] \* height)  w = int(detection[2] \* width)  h = int(detection[3] \* height)  x = int(center\_x - w / 2)  y = int(center\_y - h / 2)  boxes.append([x, y, w, h])  confidences.append(float(confidence))  class\_ids.append(class\_id)  return boxes, confidences, class\_ids  def draw\_labels(self, boxes, confidences, class\_ids, img):  indexes = cv2.dnn.NMSBoxes(boxes, confidences, 0.1, 0.1)  font = cv2.FONT\_HERSHEY\_SIMPLEX  plats = []  for i in range(len(boxes)):  if i in indexes:  x, y, w, h = boxes[i]  label = str(self.classes[class\_ids[i]])  color\_green = (0, 255, 0)  crop\_img = img[y:y+h, x:x+w]  gry = cv2.cvtColor(crop\_img, cv2.COLOR\_BGR2GRAY)  # thr = cv2.threshold(gry, 220, 255, cv2.THRESH\_BINARY\_INV + cv2.THRESH\_OTSU)[1]  try:  crop\_resized = cv2.resize(gry, dsize=(470, 110))  plats.append(crop\_resized)  cv2.rectangle(img, (x, y), (x + w, y + h), color\_green, 1)  confidence = round(confidences[i], 3) \* 100  cv2.putText(img, str(confidence) + "%", (x + 20, y - 20), font, 0.5, (0, 255, 0), 1)  except cv2.error as err:  print(err)  return img, plats  os.environ['QT\_DEVICE\_PIXEL\_RATIO'] = '0'  os.environ['QT\_AUTO\_SCREEN\_SCALE\_FACTOR'] = '1'  os.environ['QT\_SCREEN\_SCALE\_FACTORS'] = '1'  os.environ['QT\_SCALE\_FACTOR'] = '1'  pytesseract.pytesseract.tesseract\_cmd = 'C:\\Program Files\\Tesseract-OCR\\tesseract.exe'  OCR\_MODES = enum('TESSERACT')  file = open("hasil.txt", "w")  file.write("PLATE\n------\n")  file.close()  class MainWindow(QMainWindow):  def on\_resize(self):  self.dimension = self.size()  width, height = self.dimension.width(), self.dimension.height()  self.exceeded = 0  self.car\_image.setGeometry(QtCore.QRect(width \* 0.275, height \* 0.15, width \* 0.3, height \* 0.5))  self.load\_image.setGeometry(QtCore.QRect(width \* 0.03, height \* 0.2, width \* 0.125, height \* 0.05))  self.line.setGeometry(QtCore.QRect(width \* 0.179, height \* 0.1, 20, height))  self.line\_2.setGeometry(QtCore.QRect(0, height \* 0.06, width, 16))  self.car\_detection.setGeometry(QtCore.QRect(width \* 0.625, height \* 0.15, width \* 0.3, height \* 0.5))  self.cropped\_plat.setGeometry(QtCore.QRect(width \* 0.25, height \* 0.8, width \* 0.2, height \* 0.075))  self.start\_detection.setGeometry(QtCore.QRect(width \* 0.03, height \* 0.26, width \* 0.125, height \* 0.05))  self.exit.setGeometry(QtCore.QRect(width \* 0.03, height \* 0.74, width \* 0.125, height \* 0.05))  self.line\_3.setGeometry(QtCore.QRect(width \* 0.21, height \* 0.69, width \* 0.765, 20))  self.plate\_ocr.setGeometry(QtCore.QRect(width \* 0.49, height \* 0.8, width \* 0.2, height \* 0.075))  self.tesseract\_ocr.move(width \* 0.29, height \* 0.735)  label\_4\_width = self.label\_4.fontMetrics().boundingRect(self.label\_4.text()).width()  label\_4\_height = self.label\_4.fontMetrics().boundingRect(self.label\_4.text()).height()  self.label\_4.setGeometry(  QtCore.QRect((width - label\_4\_width + 1) \* 0.5, height \* 0.015, label\_4\_width + 1, label\_4\_height))  label\_2\_width = self.label\_4.fontMetrics().boundingRect(self.label\_2.text()).width()  label\_2\_height = self.label\_4.fontMetrics().boundingRect(self.label\_2.text()).height()  self.label\_2.setGeometry(QtCore.QRect(width \* 0.21, height \* 0.1, label\_2\_width, label\_2\_height))  label\_3\_width = self.label\_4.fontMetrics().boundingRect(self.label\_3.text()).width()  label\_3\_height = self.label\_4.fontMetrics().boundingRect(self.label\_3.text()).height()  self.label\_3.setGeometry(QtCore.QRect(width \* 0.21, height \* 0.73, label\_3\_width, label\_3\_height))  def resizeEvent(self, event):  QMainWindow.resizeEvent(self, event)  self.on\_resize()  def setup(self, width, height):  self.setObjectName("window")  self.resize(width, height)  self.setMinimumSize(512, 512)  self.central\_widget = QtWidgets.QWidget(self)  self.central\_widget.setObjectName("central\_widget")  self.car\_image = QtWidgets.QLabel(self.central\_widget)  self.car\_image.setText("")  self.car\_image.setScaledContents(True)  self.car\_image.setObjectName("car\_image")  self.load\_image = QtWidgets.QPushButton(self.central\_widget)  self.load\_image.setObjectName("load\_image")  self.line = QtWidgets.QFrame(self.central\_widget)  self.line.setFrameShape(QtWidgets.QFrame.VLine)  self.line.setFrameShadow(QtWidgets.QFrame.Sunken)  self.line.setObjectName("line")  self.line\_2 = QtWidgets.QFrame(self.central\_widget)  self.line\_2.setFrameShape(QtWidgets.QFrame.HLine)  self.line\_2.setFrameShadow(QtWidgets.QFrame.Sunken)  self.line\_2.setObjectName("line\_2")  self.car\_detection = QtWidgets.QLabel(self.central\_widget)  self.car\_detection.setText("")  self.car\_detection.setScaledContents(True)  self.car\_detection.setObjectName("car\_detection")  self.cropped\_plat = QtWidgets.QLabel(self.central\_widget)  self.cropped\_plat.setText("")  self.cropped\_plat.setObjectName("cropped\_plat")  self.label\_4 = QtWidgets.QLabel(self.central\_widget)  font = QtGui.QFont()  font.setPointSize(12)  font.setBold(True)  font.setWeight(75)  self.label\_4.setFont(font)  self.label\_4.setObjectName("label\_4")  self.start\_detection = QtWidgets.QPushButton(self.central\_widget)  self.start\_detection.setObjectName("start\_detection")  self.load\_video = QtWidgets.QPushButton(self.central\_widget)  self.exit = QtWidgets.QPushButton(self.central\_widget)  self.exit.setObjectName("exit")  self.line\_3 = QtWidgets.QFrame(self.central\_widget)  self.line\_3.setFrameShape(QtWidgets.QFrame.HLine)  self.line\_3.setFrameShadow(QtWidgets.QFrame.Sunken)  self.line\_3.setObjectName("line\_3")  self.label\_2 = QtWidgets.QLabel(self.central\_widget)  font = QtGui.QFont()  font.setBold(True)  self.label\_2.setFont(font)  self.label\_2.setObjectName("label\_2")  self.label\_3 = QtWidgets.QLabel(self.central\_widget)  self.label\_3.setFont(font)  self.label\_3.setObjectName("label\_3")  self.plate\_ocr = QtWidgets.QLabel(self.central\_widget)  self.plate\_ocr.setTextInteractionFlags(QtCore.Qt.TextSelectableByMouse)  self.plate\_ocr.setAlignment(QtCore.Qt.AlignCenter)  self.plate\_ocr.setObjectName("plate\_ocr")  font = self.plate\_ocr.font()  font.setPointSize(20)  self.plate\_ocr.setFont(font)  self.car\_image.setStyleSheet("border: 1px solid black;")  self.car\_detection.setStyleSheet("border: 1px solid black;")  self.cropped\_plat.setStyleSheet("border: 1px solid black;")  self.plate\_ocr.setStyleSheet("border: 1px solid black;")  self.load\_image.clicked.connect(self.on\_click\_load)  self.exit.clicked.connect(self.exit\_app)  self.start\_detection.clicked.connect(self.trained\_anpr)  self.tesseract\_ocr = QCheckBox(self.central\_widget)  self.tesseract\_ocr.setObjectName("tesseract\_ocr")  self.tesseract\_ocr.setText("General Plate (Tesseract-OCR)")  self.ocrButtonGroup = QButtonGroup()  self.ocrButtonGroup.addButton(self.tesseract\_ocr, 1)  self.ocr\_mode = OCR\_MODES.TESSERACT  self.tesseract\_ocr.setChecked(True)  self.setCentralWidget(self.central\_widget)  self.statusbar = QtWidgets.QStatusBar(self)  self.statusbar.setObjectName("statusbar")  self.setStatusBar(self.statusbar)  self.retranslate()  QtCore.QMetaObject.connectSlotsByName(self)  self.image\_path = ""  self.detector = PlateDetector()  self.detector.load\_model("./weights/detection/lapi.weights", "./weights/detection/Darknet-yolov3.cfg")  self.reader = PlateReader()  def ocr\_switch(self, btn):  if btn.text() == self.tesseract\_ocr.text():  self.ocr\_mode = OCR\_MODES.TESSERACT  self.clear\_ocr()  def retranslate(self):  \_translate = QtCore.QCoreApplication.translate  self.setWindowTitle(\_translate("window", "PENGENALAN PLAT NOMOR KENDARAAN"))  self.label\_4.setText(\_translate("window", "DETEKSI DAN PENGENALAN PLAT NOMOR KENDARAAN"))  self.label\_2.setText(\_translate("window", "Detection :"))  self.label\_3.setText(\_translate("window", "Recognition :"))  self.load\_image.setText(\_translate("window", "Load Image"))  self.start\_detection.setText(\_translate("window", "Start Detection"))  self.exit.setText(\_translate("window", "Exit"))  def popup\_close(self, input):  if input.text() == '&Yes':  sys.exit(1)  def exit\_app(self):  message = QMessageBox()  message.setWindowTitle('Warning')  message.setText('Are you sure you want to exit ?')  message.setIcon(QMessageBox.Warning)  message.setStandardButtons(QMessageBox.Yes | QMessageBox.No)  message.setDefaultButton(QMessageBox.No)  message.buttonClicked.connect(self.popup\_close)  x = message.exec\_()  def closeEvent(self, event):  self.exit\_app()  def on\_click\_load(self):  self.clear\_ocr()  self.image\_path = ""  image = QFileDialog.getOpenFileName(None, 'OpenFile', '', "Image file (\*.jpg \*.png)")  self.image\_path = image[0]  pixmap = QPixmap(self.image\_path)  self.car\_image.setScaledContents(True)  t = QtGui.QTransform()  rotated\_pixmap = pixmap.transformed(t)  self.car\_image.setPixmap(rotated\_pixmap)  def apply\_ocr(self):  if (self.ocr\_mode == OCR\_MODES.TESSERACT):  plate\_text = self.reader.tesseract\_ocr('./tmp/plate\_box.jpg')  if (len(plate\_text) > 0):  self.plate\_ocr.setText(plate\_text)  filet = open("hasil.txt", "a")  filet.write(str(plate\_text) + "\n")  self.exceeded += 1  def clear\_ocr(self):  self.plate\_ocr.clear()  def trained\_anpr(self):  if (self.image\_path == ""):  return  image, height, width, channels = self.detector.load\_image(self.image\_path)  blob, outputs = self.detector.detect\_plates(image)  boxes, confidences, class\_ids = self.detector.get\_boxes(outputs, width, height, threshold=0.3)  plate\_img, LpImg = self.detector.draw\_labels(boxes, confidences, class\_ids, image)  if len(LpImg):  cv2.imwrite('./tmp/car\_box.jpg', plate\_img)  cv2.imwrite('./tmp/plate\_box.jpg', LpImg[0])  self.car\_detection.setPixmap(QtGui.QPixmap('./tmp/car\_box.jpg'))  self.car\_detection.setScaledContents(True)  self.cropped\_plat.setPixmap(QtGui.QPixmap('./tmp/plate\_box.jpg'))  self.cropped\_plat.setScaledContents(True)  self.apply\_ocr() |

**Lampiran 4.** Data Training Dalam Bentuk Video



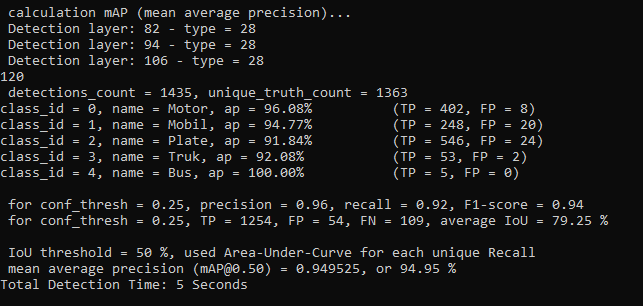
**Lampiran 5.** Data Training Setelah Dikonversi dan Dilakukan Proses Anotasi



**Lampiran 6.** Data Testing



**Lampiran 7.** Hasil Evaluasi Model YOLOv3 Dengan Darknet Framework



**Lampiran 8.** Hasil Perhitungan Accuracy, Precision, Recall, F1-Score dan mAP secara manual.

**Lampiran 9.** Hasil Perhitungan Relative Error dan Akurasi pada Hasil Perhitungan Estimasi Kecepatan

**Pengujian 1:**

**Pengujian 2:**

**Pengujian 3:**

Pengujian 4:

**Pengujian 5:**

**Pengujian 6:**

**Lampiran 10.** Hasil Perhitungan Akurasi pada Hasil Pengujian Pengenalan Plat Nomor Kendaraan.

**Pengujian 1:**

**Pengujian 2:**

**Pengujian 3:**

**Pengujian 4:**

**Pengujian 5:**

**Pengujian 6:**

**Pengujian 7:**

**Pengujian 8:**

**Pengujian 9:**

**Pengujian 10:**

**Lampiran 11.** Proses Pengambilan Data Video Training dan Testing



****

****

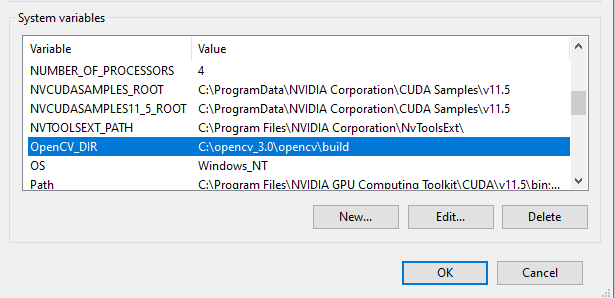
**Lampiran 12.**  Langkah-Langkah *Build* Darknet Sistem GPU Windows

Menginstal Darknet *framework* dengan GPU di Windows.

1. Untuk menggunakan Darknet *framework* di Windows diperlukan untuk menginstal OpenCV versi 3.3.0.
2. Unduh file OpenCV dari sumber resmi:

[https://sourceforge.net/projects/opencvlibrary/files/opencv-win/3.3.0/opencv-3.3.0-vc14.exe/download](https://sourceforge.net/projects/opencvlibrary/files/opencv-win/3.3.0/opencv-3.3.0-vc14.exe/download%20) dan ekstrak kedalam folder C:\opencv\_3.0.

1. Membuat variabel sistem dengan nama OpenCV\_DIR dan *path* ke *folder* *build*. Selengkapnya dapat dilihat pada gambar dibawah ini.



1. *Clone Repository* Darknet dan arahkan ke direktori yang diinginkan dengan menjalankan perintah berikut di Anaconda Prompt atau Command Prompt:

git clone https://github.com/AlexeyAB/darknet.git

Jika Anda belum menginstal git, jalankan perintah berikut di Anaconda Prompt atau Command Prompt:

conda install git

atau

pip install git

1. Temukan *file* **.dll** berikut dalam *folder* opencv\_3.0 yang telah diekstrak

C:\opencv\_3.0\opencv\build\x64\vc14\bin:

opencv\_ffmpeg320\_64.dll

opencv\_world320.dll

atau

opencv\_ffmpeg330\_64.dll

opencv\_world330.dll

atau

opencv\_ffmpeg340\_64.dll

opencv\_world340.dll

1. *Copy Paste* dua *file* tersebut kedalam direktori: darknet\build\darknet\x64.
2. Instalasi Visual Studio 2015/2017/2019. Untuk membangun Darknet *framework* pada Windows diperlukan untuk menginstal Visual Studio versi 2015 atau 2017 atau 2019.
3. Instalasi CUDA 11.5
4. Unduh NVIDIA CUDA Toolkit versi 11.5 dari arsip:

https://developer.NVIDIA.com/cuda-toolkit-archive

1. Instal CUDA Toolkit versi 11.5 dengan menjalankan instalasi CUDA dan mengikuti petunjuk di layar.
2. Instalasi cuDNN 8.4.2
3. Unduh arsip dari cuDNN versi 8.4.2 untuk versi CUDA 11.5 dari sumber resmi https://developer.NVIDIA.com/rdp/cudnn-archive.
4. Setelah cuDNN berhasil diunduh, lalu ekstrak cuDNN dan akan menghasilkan folder bin,lib, dan include. Setelah itu *Copy* dan *Paste* ketiga *folder* tersebut kedalam direktori *C:\Program Files\NVIDIA GPU* *Computing Toolkit*
5. Buat variabel sistem dengan nama CUDNN dan jalur ke tempat cuDNN diinstal. Untuk melakukannya:
   * + Buka *File Explorer*, klik kanan pada *This PC* dan buka Pengaturan, pilih *Advance setting*, lalu pilih *Environment Variabel.*
     + Klik *New* di *System variabel*.
     + Buat variabel baru dengan nama CUDNN dan Path tempat cuDNN diinstal, contoh: C:\Program Files\NVIDIA GPU Computing Toolkit
6. Temukan *file* dll berikut dalam direktori C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v11.5\bin:

cudnn64\_7.dll

*Copy* dan *Paste* *file* tersebut kedalam direktori darknet\build\darknet\x64.

1. Membangun kerangka Darknet dengan menggunakan Visual Studio yang telah terinstal, temukan dan buka file berikut di direktori darknet\build\darknet\:

darknet.sln

kemudian, di menu *drop-down* Visual Studio atur sebagai berikut:

x64

dan

Release

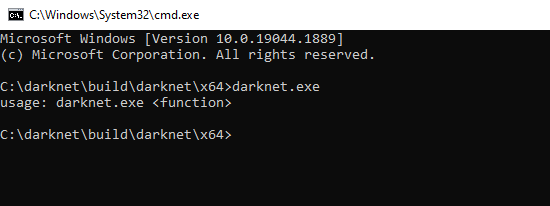
Terakhir, pilih *Build* dan *Build* Darknet atau dapat menggunakan hotkeys (Ctrl+B)

1. Temukan file darknet.exe atau darknet\_gpu.exe yang dibuat di direktori \build\darknet\x64.
2. Lakukan verifikasi instalasi yang telah berhasil dengan cara:
3. Buka Command Prompt atau Anaconda Prompt.
4. Arahkan direktori kedalam direktori darknet\build\darknet\x64.
5. Ketikkan salah satu perintah seperti dibawah ini:

darknet\_gpu.exe

darknet.exe

Apabila darknet sukses dibangun maka akan menghasilkan output seperti gambar dibawah ini.



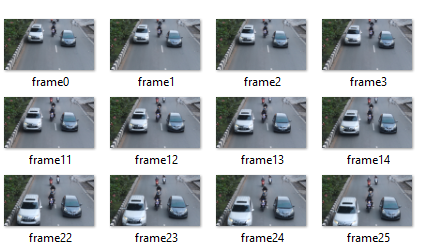
**Lampiran 13.** Langkah-Langkah Training Dengan Menggunakan Darknet Framework

Secara umum langkah-langkah untuk melakukan training dengan menggunakan Darknet *framework* adalah sebagai berikut:

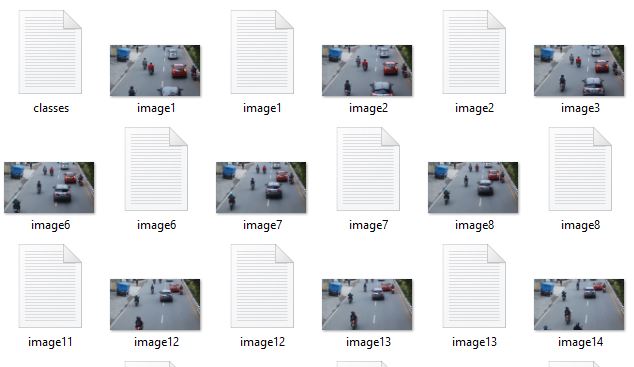
1. Konversi data video menjadi data gambar.
2. Anotasi dataset dengan menggunakan *software* LabelImg.
3. *Preparing dataset* (secara lengkap dapat dilihat pada sub bab 4.1).
4. *Training*.

Langkah-Langkah *Training:*

1. Lakukan konversi data video menjadi data gambar, seperti contoh gambar dibawah ini.



1. Lakukan anotasi *dataset* dengan menggunakan *software* LabelImg, sehingga menghasilkan file anotasi seperti gambar dibawah ini.



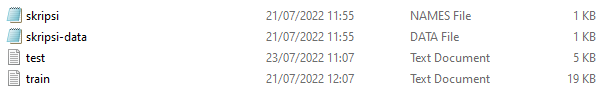
1. Membuat *file train, test,* data*,* dan *names* dengan menggunakan *Source Code* Python dibawah ini.

**Source Code** Membuat File Train dan Test

|  |
| --- |
| import os  full\_path\_to\_images = 'C:\\*Darknet*\\*dataset*-skripsi'  print(os.getcwd())  os.chdir(full\_path\_to\_images)  print(os.getcwd())  p = []  for current\_dir, dirs, files in os.walk('.'):  for f in files:  if f.endswith('.jpg'):  path\_to\_save\_into\_txt\_*file*s = full\_path\_to\_images + '/' + f  p.append(path\_to\_save\_into\_txt\_*file*s + '\n')  p\_*test* = p[:int(len(p) \* 0.15)]  p = p[int(len(p) \* 0.15):]  with open('*train*.txt', 'w') as *train*\_txt:  for e in p:  *train*\_txt.write(e)  with open('*test*.txt', 'w') as *test*\_txt:  for e in p\_*test*:  *test*\_txt.write(e) |

|  |
| --- |
| **Source Code** Membuat File Konfigurasi Data dan Classes |
| full\_path\_to\_images = 'C:\\*Darknet*\\*dataset*-skripsi'  c = 0  with open(full\_path\_to\_images + '/' + 'skripsi.names', 'w') as names, \  open(full\_path\_to\_images + '/' + 'classes.txt', 'r') as txt:  for line in txt:  names.write(line)  c += 1  with open(full\_path\_to\_images + '/' + 'skripsi-data.data', 'w') as data:  data.write('classes = ' + str(c) + '\n')  data.write('*train* = ' + full\_path\_to\_images + '/' + '*train*.txt' + '\n')    data.write('valid = ' + full\_path\_to\_images + '/' + '*test*.txt' + '\n')  data.write('names = ' + full\_path\_to\_images + '/' + 'skripsi.names' + '\n')  data.write('backup = backup') |

Dari *source code* tersebut akan menghasilkan *output* *file* seperti pada gambar dibawah ini.



1. Membuat *file* konfigurasi
2. Cari *file* yolov3.cfg dalam *folder* cfg pada direktori darknet.
3. Buka *file* cfg tersebut dan lakukan beberapa perubahan sebagai berikut:
4. Beri comment atau tanda (#) pada bagian *testing*.



1. *Uncomment* atau hilangkan tanda (#) pada bagian training.



1. *Update* beberapa parameter seperti contoh dibawah ini.

* Baris 6: set batch=64, ini berarti kita akan menggunakan 64 gambar untuk setiap langkah pelatihan
* Baris 7: set subdivisions=32, batch akan dibagi 32 untuk mengurangi kebutuhan VRAM GPU.
* Baris 20: set max\_batches = (classes × 2000), dalam penelitian ini max\_batches = 10000.
* Baris 22: set steps = 80%, 90% dari nilai max\_batches, dalam penelitian ini steps = 8000,9000.
* Baris 603: set filters = (classes + 5) ×3, dalam penelitian ini filters = 30.
* Baris 610: set classes = 5, jumlah kelas yang ingin dideteksi.
* Baris 689: set filters = (classes + 5) ×3, dalam penelitian ini filters = 30.
* Baris 696: set classes = 5, jumlah kelas yang ingin dideteksi.
* Baris 776: set filters = (classes + 5) ×3, dalam penelitian ini filters = 30.
* Baris 783: set classes = 5, jumlah kelas yang ingin dideteksi.

1. Setelah proses *preparing dataset* dan *file configuration* selesai dilakukan maka *copy* dan *paste file* data dan konfigurasi kedalam direktori darknet\build\darknet\x64\cfg.
2. Unduh *feature extractor* atau *backbone* YOLOv3 yaitu Darknet53-conv74 melalui link berikut <https://pjreddie.com/media/files/darknet53.conv.74>.
3. Buka Command Prompt dan arahkan direktori kedalam *folder* darknet\build\darknet\x64, lalu ketikkan perintah sebagai berikut:

***darknet.exe detector train cfg/nama\_file\_data.data cfg/nama\_komfigurasi.cfg weights/ darknet53.conv.74***