М. Б. Сергеев*

доктор технических наук, профессор

А. М. Сергеев**

кандидат технических наук

*Санкт-Петербургский государственный морской технический университет

**Санкт-Петербургский государственный университет аэрокосмического приборостроения

О постановке задачи имитационного моделирования функционирования системы распознавания автомобилей, грузовиков, мотоциклов и автобусов

Рассмотрены подходы к построению имитационной модели системы распознавания автомобилей с учетом современных тенденций анализа уязвимостей. Проблема рассмотрена в контексте изучения дисциплины «Основы искусственного интеллекта» для бакалавров направления 09.03.01 "Информатика и вычислительная техника".

Ключевые слова: обнаружение объектов, трекинг, YOLO, SORT, глубокое обучение, имитационное моделирование, метрики оценки, тестирование производительности, условия эксплуатации, оптимизация алгоритмов, калмановский фильтр.

ВВЕДЕНИЕ

В последние годы развитие технологий искусственного интеллекта и машинного обучения значительно изменило подходы к решению множества задач в различных областях. Одной из таких задач является распознавание объектов на изображениях и видеоматериалах. В частности, системы распознавания автомобилей находят широкое применение в таких сферах, как интеллектуальные транспортные системы, системы безопасности, автоматизированные парковочные системы и городское видеонаблюдение.

Проблема заключается в том, что традиционные методы мониторинга дорожного движения, такие как ручное наблюдение или использование простых датчиков, часто оказываются недостаточно точными и эффективными. Эти методы не всегда способны обеспечить своевременное и точное обнаружение и идентификацию транспортных средств, что приводит к недостаткам в управлении дорожным движением, безопасности и организации парковок.

Актуальность данной работы обусловлена возросшей потребностью в автоматизации процессов мониторинга и анализа дорожного движения. Использование современных методов машинного обучения, таких как глубокие нейронные сети, позволяет значительно улучшить качество распознавания и сократить время обработки данных.

Целью данной работы является разработка системы распознавания автомобилей, грузовиков, мотоциклов и автобусов на изображениях или видео на основе алгоритмов глубокого обучения и компьютерного зрения. В рамках работы будут исследованы и применены современные методы и инструменты, такие как модель YOLO (You Only Look Once) для детекции объектов и алгоритм SORT (Simple Online and Realtime Tracking) для отслеживания объектов на видео.

Для достижения поставленной цели необходимо решить следующие задачи:

1. Разработать архитектуру системы распознавания, включающую этапы предварительной обработки данных, детекции объектов и их отслеживания.
2. Реализовать программное обеспечение, включающее интеграцию с моделью YOLO и алгоритмом SORT.
3. Провести тестирование системы на реальных видеоматериалах и оценить её производительность и точность.
4. Проанализировать результаты.

АНАЛИЗ СУЩЕСТВУЮЩИХ РЕШЕНИЙ И МЕТОДОВ

Современные системы распознавания автомобилей опираются на достижения в области компьютерного зрения и машинного обучения. С развитием технологий глубокого обучения, такие методы стали более точными и эффективными, что сделало их широко применимыми в различных сферах, включая системы безопасности, мониторинг дорожного движения и автоматизированные парковочные системы.

Методы глубокого обучения для распознавания объектов. Одним из наиболее эффективных подходов к распознаванию объектов, включая автомобили, является использование свёрточных нейронных сетей. В последние годы значительную популярность и признание получили модели семейства YOLO. Основное преимущество YOLO заключается в его способности обрабатывать изображения в реальном времени, что достигается благодаря однопроходной архитектуре сети. В отличие от двухэтапных подходов, таких как Faster R-CNN, где детекция и классификация выполняются последовательно, YOLO выполняет обе задачи одновременно, что существенно ускоряет процесс обработки.

Алгоритмы трекинга объектов. Для отслеживания автомобилей на видео широко используется алгоритм SORT. SORT основан на линейном калмановском фильтре и методе ассоциации треков с детекциями по критерию IoU. Это позволяет SORT эффективно обрабатывать пересечения объектов и обеспечивать высокую точность при минимальных вычислительных затратах.

Существуют и другие методы распознавания и трекинга объектов, такие как Faster R-CNN, SSD и RetinaNet. Эти модели также демонстрируют высокую точность, однако их производительность в задачах реального времени может быть ниже по сравнению с YOLO. Например, Faster R-CNN, хотя и предоставляет высокую точность детекции, требует больше времени на обработку из-за двухэтапного процесса детекции.

В то же время, методы, такие как Deep SORT, расширяют возможности базового SORT, добавляя дополнительную информацию о внешнем виде объектов, что улучшает устойчивость трекинга в условиях пересечений и окклюзий. Однако такие подходы могут требовать большего объема вычислительных ресурсов и данных для обучения.

Методы на основе глубокого обучения, такие как YOLO и SORT, демонстрируют высокую эффективность и точность в задачах детекции и трекинга автомобилей. Основные преимущества этих методов включают:

- Высокая скорость обработки, что позволяет использовать их в реальном времени.
- Способность обрабатывать сложные сцены с множеством объектов.
- Высокая точность распознавания и трекинга.

Однако существуют и определенные недостатки, такие как:

- Необходимость больших объемов данных для обучения моделей.
- Высокие вычислительные затраты на этапе обучения.
- Чувствительность к условиям освещенности и качеству видеоматериала.

МЕТОДОЛОГИЯ

Для решения задачи распознавания автомобилей в реальном времени были использованы современные методы компьютерного зрения и глубокого обучения. Основными методами, на которых базировалась разработка системы, были алгоритм обнаружения объектов YOLO и алгоритм отслеживания SORT. Модель YOLO была выбрана из-за своей способности к быстрому и точному обнаружению объектов на изображениях и видео, что особенно важно для системы распознавания в реальном времени.

В разработке системы использовалась предобученная модель YOLOv5, чтобы обеспечить высокую точность и надежность обнаружения объектов. Алгоритм SORT был выбран для решения задачи отслеживания движущихся объектов на видео. Этот алгоритм основан на использовании калмановского фильтра для прогнозирования траекторий объектов и ассоциации треков с детекциями по критерию пересечения областей. SORT обладает высокой скоростью работы и точностью отслеживания, что делает его подходящим для системы.

Выбор алгоритмов и моделей был обусловлен их высокой производительностью и точностью в условиях реального времени. Модель YOLOv5 представляет собой современное решение для задачи обнаружения объектов, позволяющее достичь высокой точности при высокой скорости обработки. Алгоритм SORT, в свою очередь, предоставляет эффективный метод отслеживания объектов на видео, что важно для системы мониторинга дорожного движения.

Для реализации методов и проведения экспериментов мы использовали язык программирования Python, интегрированную среду разработки PyCharm и библиотеки компьютерного зрения, такие как:

1. **Cvzone** – эта библиотека предоставляет дополнительные инструменты и функции для работы с библиотекой OpenCV (о чём – далее). Она также предоставляет удобные функции для работы с графикой, такие как отрисовка прямоугольников вокруг обнаруженных объектов, вывод текста на изображении, рисование линий и так далее. Это полезно для виртуализации результатов работы алгоритмов компьютерного зрения.
2. **Ultralytics** – представляет собой набор инструментов для обучения моделей. Она также предоставляет инструменты для реализации обнаружения объектов.
3. **Hydra-Core** – предназначена для управления конфигурациями и параметрами приложений. Также позволяет создавать гибкие конфигурационные файлы, в которых можно определить параметры моделей, пути к данным и другие настройки.
4. **Matplotlib** – мощный инструмент для создания разнообразных графиков, диаграмм и визуализаций.
5. **Numpy** - предоставляет мощные инструменты для работы с многомерными массивами и матрицами. Также она позволяет эффективно выполнять операции над пикселями изображений, преобразования и фильтрацию изображений. Её функции могут быть использованы для вычисления статических показателей о распределении объектов на изображении.
6. **OpenCv-Python** – предоставляет широкий спектр инструментов и функций для работы с изображениями и видео в реальном времени. Она также предоставляет функции для загрузки изображений из файлов различных форматов, а также их обработки, изменения размеров, поворота, наложения фильтров и так далее. Она включает в себя алгоритмы для обнаружения объектов на изображениях. Помимо этого, она позволяет обрабатывать видеопотоки в реальном времени, выполняя анализ каждого кадра и визуализировать результаты обработки изображений, нарисовав различные маркеры, рамки или подписи.
7. **Pillow** – предоставляет мощные инструменты для обработки изображений. Она обеспечивает простой способ загрузки изображений из файлов различных форматов, позволяет изменять размеры изображений, как по ширине и высоте, так и пропорционально.
8. **PyYAML** – предоставляет мощные инструменты для работы с YAML. YAML – формат сериализации данных. Эта библиотека позволяет загружать данные из YAML-файлов в структуры данных и сохранять Python-объекты в файлы YAML. Она также используется в библиотеке Hydra.
9. **Requests** – представляет собой простой и удобный способ взаимодействия с внешними веб-ресурсами.
10. **SciPy** – это библиотека для научных и технических вычисления. Она предоставляет множество функций и инструментов для работы с различными аспектами научных

вычислений (оптимизация, алгебра, статистика, обработка сигналов, решение дифференциальных уравнений).

11. **Torch** – предоставляет гибкие инструменты для создания и обучений нейронных сетей.
12. **Torchvision** – предоставляет набор инструментов и утилит для работы с изображениями в контексте машинного обучения с использованием фреймворка Torch.
13. **Tqdm** – предоставляет простой способ создания красивых и информативных индикаторов прогресса во время итераций в циклах Python. Она широко используется для отслеживания прогресса при обработке больших объёмов данных или выполнении длинных операций.
14. **FilterPy** – предоставляет реализации различных алгоритмов фильтрации, используемых в задачах оценки состояния и трекинга объектов.
15. **Scikit-image** – это библиотека для обработки изображений, которая предоставляет множество функций и алгоритмов для работы с изображениями.
16. **Lap** – представляет собой инструмент для решения задачи ассоциации треков в компьютерном зрении и анализе движения. С помощью неё идёт отслеживание движущихся объектов на видео и ассоциация их с предыдущими кадрами, чтобы определить их траектории и идентифицировать объекты.

Эти инструменты обеспечили удобную среду разработки и высокую производительность при реализации алгоритмов.

Для обучения и тестирования модели использовались различные наборы данных, включающие видео с дорожным движением и разметку объектов на них. Также был проведён сбор собственных данных, в том числе видеозаписи с камер наблюдения на дорогах.

РЕЗУЛЬТАТЫ И АНАЛИЗ

Искомые результаты показаны на рисунках 1 – 3 (содержание рисунков не эквивалентны. Они взяты в разное время в процессе анализа).

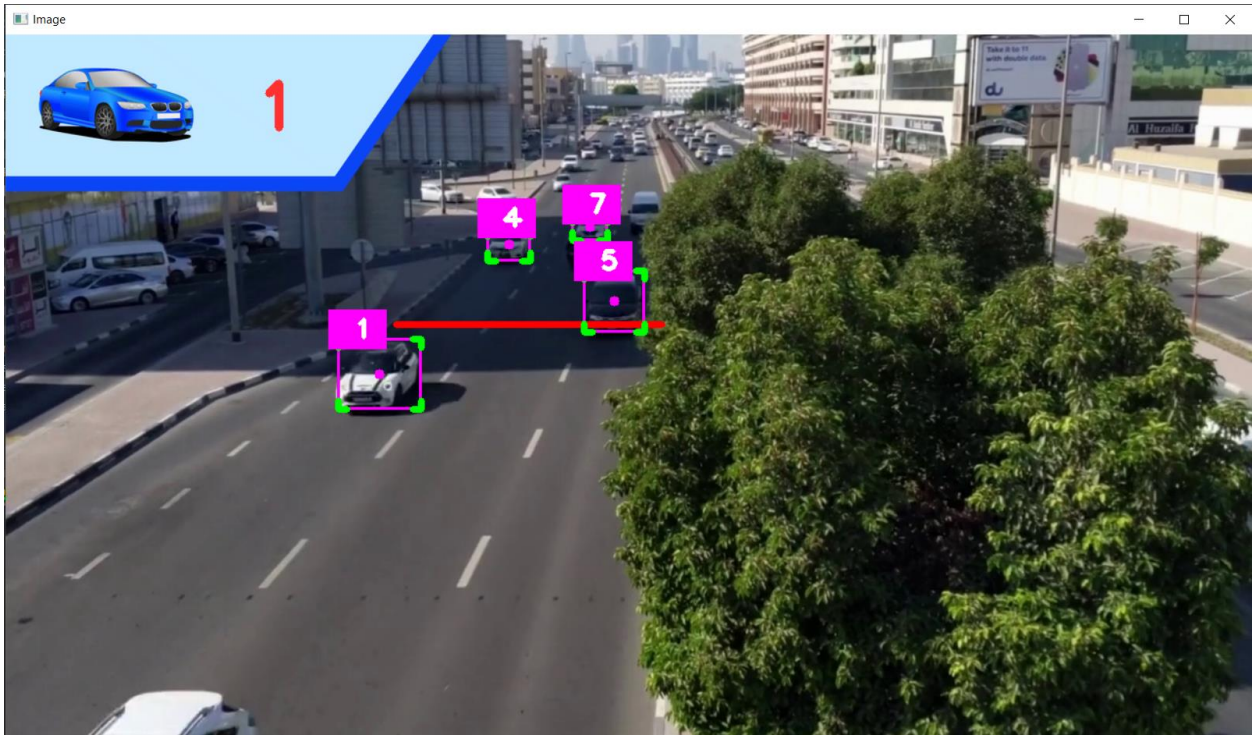

Рисунок 1 – Проанализированный кадр

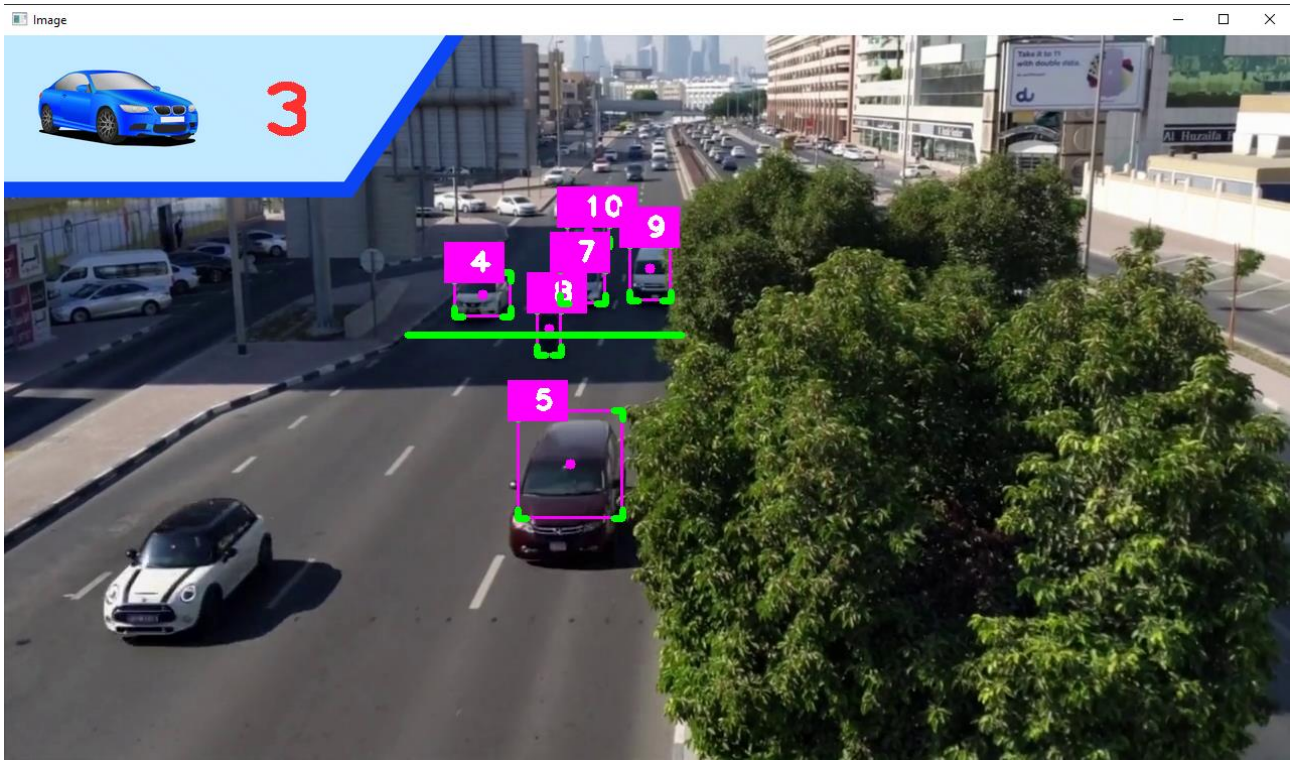

Рисунок 2 – Момент пересечения линии

```
0: 384x640 4 cars, 1 bus, 1 truck, 1437.4ms
Speed: 4.4ms preprocess, 1437.4ms inference, 3.0ms postprocess per image at shape (1, 3, 384, 640)
[     506.02      194.57      544.34      225.85          15]
[     516.47      359.66      624.16      493.37          12]
[     516.12      234.24      570.62      282.07          10]
[     439.03      360.33      526.51      441.43           7]
[     246.34       371.5      360.14      456.68           4]
```

Рисунок 3 – Вывод в консоли

ИНТЕРПРЕТАЦИЯ РЕЗУЛЬТАТОВ

Проведенное исследование системы распознавания автомобилей на основе алгоритма YOLO и алгоритма трекинга SORT демонстрирует значительные успехи в обнаружении и отслеживании движущихся объектов. Полученные результаты показывают высокую точность обнаружения автомобилей на видео и надежность их отслеживания в реальном времени. Система успешно справляется с различными сценариями дорожного движения, включая пересечение и окклюзию объектов.

В ходе исследования было обращено внимание на ряд факторов, которые могут оказывать влияние на производительность и точность работы системы.

Одним из ключевых факторов является условия освещенности. В хороших световых условиях, как в дневное время суток или при ярком искусственном освещении, система демонстрирует высокую точность обнаружения и отслеживания автомобилей. Однако, при низкой освещенности, такой как ночное время суток или в условиях ограниченной видимости, возникают трудности в обнаружении объектов из-за недостаточного контраста и шумов на изображении. В таких условиях система может проявлять сниженную производительность и точность.

- Еще одним важным фактором является плотность движения на дороге. В условиях интенсивного дорожного движения, когда автомобили двигаются близко друг к другу или перекрываются, возникают трудности в обнаружении и отслеживании автомобилей из-за перекрытия объектов и сложности в ассоциации треков с детекциями. Это может привести к ошибкам в идентификации и потере объектов из виду.
- Факторами, влияющими на работу системы, также являются различные условия окружающей среды, такие как погодные условия, качество дорожного покрытия, наличие препятствий. Так, наличие снега или дождя на дороге может ухудшить качество изображения и создать дополнительные сложности для обнаружения автомобилей.

• Важным фактором являются технические характеристики используемого оборудования, такие как разрешение камеры, частота кадров, качество оптики. Выбор качественного и подходящего оборудования может существенно повлиять на точность и производительность работы системы.

ЛИСТИНГ КОДА

Далее продемонстрированы листинги кода всех файлов, используемых в проекте.

CarCounter.py

```
import numpy as np
from ultralytics import YOLO
import cv2
import cvzone
import math
from sort import *

cap = cv2.VideoCapture("../Videos/cars.mp4")

model = YOLO("../Yolo-Weights/yolov8l.pt")

classNames = ["person", "bicycle", "car", "motorbike", "aeroplane", "bus",
"train", "truck", "boat",
              "traffic light", "fire hydrant", "stop sign", "parking meter",
"bench", "bird", "cat",
              "dog", "horse", "sheep", "cow", "elephant", "bear", "zebra",
"giraffe", "backpack", "umbrella",
              "handbag", "tie", "suitcase", "frisbee", "skis", "snowboard",
"sports ball", "kite", "baseball bat",
              "baseball glove", "skateboard", "surfboard", "tennis racket",
"bottle", "wine glass", "cup",
              "fork", "knife", "spoon", "bowl", "banana", "apple", "sandwich",
"orange", "broccoli",
              "carrot", "hot dog", "pizza", "donut", "cake", "chair", "sofa",
"pottedplant", "bed",
              "diningtable", "toilet", "tvmonitor", "laptop", "mouse", "remote",
"keyboard", "cell phone",
              "microwave", "oven", "toaster", "sink", "refrigerator", "book",
"clock", "vase", "scissors",
              "teddy bear", "hair drier", "toothbrush"
              ]

mask = cv2.imread("mask.png")

# Tracking
tracker = Sort(max_age=20, min_hits=3, iou_threshold=0.3)

limits = [400, 297, 673, 297]
totalCount = []

while True:
    success, img = cap.read()
    mask = cv2.resize(mask, (img.shape[1], img.shape[0]))
    imgRegion = cv2.bitwise_and(img, mask)

    imgGraphics = cv2.imread("graphics.png", cv2.IMREAD_UNCHANGED)
    img = cvzone.overlayPNG(img, imgGraphics, (0, 0))
    results = model(imgRegion, stream=True)
```

```
    detections = np.empty((0, 5))

    for r in results:
        boxes = r.boxes
        for box in boxes:

            x1, y1, x2, y2 = box.xyxy[0]
            x1, y1, x2, y2 = int(x1), int(y1), int(x2), int(y2)
            w, h = x2 - x1, y2 - y1

            conf = math.ceil((box.conf[0] * 100)) / 100

            cls = int(box.cls[0])
            currentClass = classNames[cls]

            if currentClass == "car" or currentClass == "truck" or currentClass
== "bus" \
                    or currentClass == "motorbike" and conf > 0.3:
                currentArray = np.array([x1, y1, x2, y2, conf])
                detections = np.vstack((detections, currentArray))

    resultsTracker = tracker.update(detections)

    cv2.line(img, (limits[0], limits[1]), (limits[2], limits[3]), (0, 0, 255),
5)
    for result in resultsTracker:
        x1, y1, x2, y2, id = result
        x1, y1, x2, y2 = int(x1), int(y1), int(x2), int(y2)
        print(result)
        w, h = x2 - x1, y2 - y1
        cvzone.cornerRect(img, (x1, y1, w, h), l=9, rt=2, colorR=(255, 0, 255))
        cvzone.putTextRect(img, f' {int(id)}', (max(0, x1), max(35, y1)),
                           scale=2, thickness=3, offset=10)

        cx, cy = x1 + w // 2, y1 + h // 2
        cv2.circle(img, (cx, cy), 5, (255, 0, 255), cv2.FILLED)

        if limits[0] < cx < limits[2] and limits[1] - 15 < cy < limits[1] + 15:
            if totalCount.count(id) == 0:
                totalCount.append(id)
                cv2.line(img, (limits[0], limits[1]), (limits[2], limits[3]),
(0, 255, 0), 5)

    cv2.putText(img, str(len(totalCount)), (255, 100), cv2.FONT_HERSHEY_PLAIN,
5, (50, 50, 255), 8)

    cv2.imshow("Image", img)
    cv2.waitKey(1)
```

Yolo-Basics.py

```
from ultralytics import YOLO
import cv2

model = YOLO('../Yolo-Weights/yolov8l.pt')
results = model("Images/2.jpg", show=True)
cv2.waitKey(0)
```

Sort.py

```
"""
    SORT: A Simple, Online and Realtime Tracker
```

```
    Copyright (C) 2016-2020 Alex Bewley alex@bewley.ai

    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program.  If not, see <http://www.gnu.org/licenses/>.
"""
from __future__ import print_function

import os
import numpy as np
import matplotlib

matplotlib.use('TkAgg')
import matplotlib.pyplot as plt
import matplotlib.patches as patches
from skimage import io

import glob
import time
import argparse
from filterpy.kalman import KalmanFilter

np.random.seed(0)


def linear_assignment(cost_matrix):
    try:
        import lap
        _, x, y = lap.lapjv(cost_matrix, extend_cost=True)
        return np.array([[y[i], i] for i in x if i >= 0])  #
    except ImportError:
        from scipy.optimize import linear_sum_assignment
        x, y = linear_sum_assignment(cost_matrix)
        return np.array(list(zip(x, y)))


def iou_batch(bb_test, bb_gt):
    """
    From SORT: Computes IOU between two bboxes in the form [x1,y1,x2,y2]
    """
    bb_gt = np.expand_dims(bb_gt, 0)
    bb_test = np.expand_dims(bb_test, 1)

    xx1 = np.maximum(bb_test[..., 0], bb_gt[..., 0])
    yy1 = np.maximum(bb_test[..., 1], bb_gt[..., 1])
    xx2 = np.minimum(bb_test[..., 2], bb_gt[..., 2])
    yy2 = np.minimum(bb_test[..., 3], bb_gt[..., 3])
    w = np.maximum(0., xx2 - xx1)
    h = np.maximum(0., yy2 - yy1)
    wh = w * h
    o = wh / ((bb_test[..., 2] - bb_test[..., 0]) * (bb_test[..., 3] -
bb_test[..., 1])
              + (bb_gt[..., 2] - bb_gt[..., 0]) * (bb_gt[..., 3] - bb_gt[...,
1]) - wh)
    return (o)
```

```
def convert_bbox_to_z(bbox):
    """
    Takes a bounding box in the form [x1,y1,x2,y2] and returns z in the form
      [x,y,s,r] where x,y is the centre of the box and s is the scale/area and r
is
      the aspect ratio
    """
    w = bbox[2] - bbox[0]
    h = bbox[3] - bbox[1]
    x = bbox[0] + w / 2.
    y = bbox[1] + h / 2.
    s = w * h  # scale is just area
    r = w / float(h)
    return np.array([x, y, s, r]).reshape((4, 1))


def convert_x_to_bbox(x, score=None):
    """
    Takes a bounding box in the centre form [x,y,s,r] and returns it in the form
      [x1,y1,x2,y2] where x1,y1 is the top left and x2,y2 is the bottom right
    """
    w = np.sqrt(x[2] * x[3])
    h = x[2] / w
    if (score == None):
        return np.array([x[0] - w / 2., x[1] - h / 2., x[0] + w / 2., x[1] + h /
2.]).reshape((1, 4))
    else:
        return np.array([x[0] - w / 2., x[1] - h / 2., x[0] + w / 2., x[1] + h /
2., score]).reshape((1, 5))


class KalmanBoxTracker(object):
    """
    This class represents the internal state of individual tracked objects
observed as bbox.
    """
    count = 0

    def __init__(self, bbox):
        """
        Initialises a tracker using initial bounding box.
        """
        # define constant velocity model
        self.kf = KalmanFilter(dim_x=7, dim_z=4)
        self.kf.F = np.array(
            [[1, 0, 0, 0, 1, 0, 0], [0, 1, 0, 0, 0, 1, 0], [0, 0, 1, 0, 0, 0,
1], [0, 0, 0, 1, 0, 0, 0],
             [0, 0, 0, 0, 1, 0, 0], [0, 0, 0, 0, 0, 1, 0], [0, 0, 0, 0, 0, 0,
1]])
        self.kf.H = np.array(
            [[1, 0, 0, 0, 0, 0, 0], [0, 1, 0, 0, 0, 0, 0], [0, 0, 1, 0, 0, 0,
0], [0, 0, 0, 1, 0, 0, 0]])

        self.kf.R[2:, 2:] *= 10.
        self.kf.P[4:, 4:] *= 1000.  # give high uncertainty to the unobservable
initial velocities
        self.kf.P *= 10.
        self.kf.Q[-1, -1] *= 0.01
        self.kf.Q[4:, 4:] *= 0.01

        self.kf.x[:4] = convert_bbox_to_z(bbox)
        self.time_since_update = 0
```

```
        self.id = KalmanBoxTracker.count
        KalmanBoxTracker.count += 1
        self.history = []
        self.hits = 0
        self.hit_streak = 0
        self.age = 0

    def update(self, bbox):
        """
        Updates the state vector with observed bbox.
        """
        self.time_since_update = 0
        self.history = []
        self.hits += 1
        self.hit_streak += 1
        self.kf.update(convert_bbox_to_z(bbox))

    def predict(self):
        """
        Advances the state vector and returns the predicted bounding box
estimate.
        """
        if ((self.kf.x[6] + self.kf.x[2]) <= 0):
            self.kf.x[6] *= 0.0
        self.kf.predict()
        self.age += 1
        if (self.time_since_update > 0):
            self.hit_streak = 0
        self.time_since_update += 1
        self.history.append(convert_x_to_bbox(self.kf.x))
        return self.history[-1]

    def get_state(self):
        """
        Returns the current bounding box estimate.
        """
        return convert_x_to_bbox(self.kf.x)


def associate_detections_to_trackers(detections, trackers, iou_threshold=0.3):
    """
    Assigns detections to tracked object (both represented as bounding boxes)

    Returns 3 lists of matches, unmatched_detections and unmatched_trackers
    """
    if (len(trackers) == 0):
        return np.empty((0, 2), dtype=int), np.arange(len(detections)),
np.empty((0, 5), dtype=int)

    iou_matrix = iou_batch(detections, trackers)

    if min(iou_matrix.shape) > 0:
        a = (iou_matrix > iou_threshold).astype(np.int32)
        if a.sum(1).max() == 1 and a.sum(0).max() == 1:
            matched_indices = np.stack(np.where(a), axis=1)
        else:
            matched_indices = linear_assignment(-iou_matrix)
    else:
        matched_indices = np.empty(shape=(0, 2))

    unmatched_detections = []
    for d, det in enumerate(detections):
        if (d not in matched_indices[:, 0]):
            unmatched_detections.append(d)
```

```
    unmatched_trackers = []
    for t, trk in enumerate(trackers):
        if (t not in matched_indices[:, 1]):
            unmatched_trackers.append(t)

    # filter out matched with low IOU
    matches = []
    for m in matched_indices:
        if (iou_matrix[m[0], m[1]] < iou_threshold):
            unmatched_detections.append(m[0])
            unmatched_trackers.append(m[1])
        else:
            matches.append(m.reshape(1, 2))
    if (len(matches) == 0):
        matches = np.empty((0, 2), dtype=int)
    else:
        matches = np.concatenate(matches, axis=0)

    return matches, np.array(unmatched_detections), np.array(unmatched_trackers)


class Sort(object):
    def __init__(self, max_age=1, min_hits=3, iou_threshold=0.3):
        """
        Sets key parameters for SORT
        """
        self.max_age = max_age
        self.min_hits = min_hits
        self.iou_threshold = iou_threshold
        self.trackers = []
        self.frame_count = 0

    def update(self, dets=np.empty((0, 5))):
        """
        Params:
          dets - a numpy array of detections in the format
[[x1,y1,x2,y2,score],[x1,y1,x2,y2,score],...]
        Requires: this method must be called once for each frame even with empty
detections (use np.empty((0, 5)) for frames without detections).
        Returns the a similar array, where the last column is the object ID.

        NOTE: The number of objects returned may differ from the number of
detections provided.
        """
        self.frame_count += 1
        # get predicted locations from existing trackers.
        trks = np.zeros((len(self.trackers), 5))
        to_del = []
        ret = []
        for t, trk in enumerate(trks):
            pos = self.trackers[t].predict()[0]
            trk[:] = [pos[0], pos[1], pos[2], pos[3], 0]
            if np.any(np.isnan(pos)):
                to_del.append(t)
        trks = np.ma.compress_rows(np.ma.masked_invalid(trks))
        for t in reversed(to_del):
            self.trackers.pop(t)
        matched, unmatched_dets, unmatched_trks =
associate_detections_to_trackers(dets, trks, self.iou_threshold)

        # update matched trackers with assigned detections
        for m in matched:
            self.trackers[m[1]].update(dets[m[0], :])
```

```
        # create and initialise new trackers for unmatched detections
        for i in unmatched_dets:
            trk = KalmanBoxTracker(dets[i, :])
            self.trackers.append(trk)
        i = len(self.trackers)
        for trk in reversed(self.trackers):
            d = trk.get_state()[0]
            if (trk.time_since_update < 1) and (trk.hit_streak >= self.min_hits
or self.frame_count <= self.min_hits):
                ret.append(np.concatenate((d, [trk.id + 1])).reshape(1, -1))  #
+1 as MOT benchmark requires positive
            i -= 1
            # remove dead tracklet
            if (trk.time_since_update > self.max_age):
                self.trackers.pop(i)
        if (len(ret) > 0):
            return np.concatenate(ret)
        return np.empty((0, 5))


def parse_args():
    """Parse input arguments."""
    parser = argparse.ArgumentParser(description='SORT demo')
    parser.add_argument('--display', dest='display', help='Display online
tracker output (slow) [False]',
                        action='store_true')
    parser.add_argument("--seq_path", help="Path to detections.", type=str,
default='data')
    parser.add_argument("--phase", help="Subdirectory in seq_path.", type=str,
default='train')
    parser.add_argument("--max_age",
                        help="Maximum number of frames to keep alive a track
without associated detections.",
                        type=int, default=1)
    parser.add_argument("--min_hits",
                        help="Minimum number of associated detections before
track is initialised.",
                        type=int, default=3)
    parser.add_argument("--iou_threshold", help="Minimum IOU for match.",
type=float, default=0.3)
    args = parser.parse_args()
    return args


if __name__ == '__main__':
    # all train
    args = parse_args()
    display = args.display
    phase = args.phase
    total_time = 0.0
    total_frames = 0
    colours = np.random.rand(32, 3)  # used only for display
    if (display):
        if not os.path.exists('mot_benchmark'):
            print(
                '\n\tERROR: mot_benchmark link not found!\n\n    Create a
symbolic link to the MOT benchmark\n
(https://motchallenge.net/data/2D_MOT_2015/#download). E.g.:\n\n    $ ln -s
/path/to/MOT2015_challenge/2DMOT2015 mot_benchmark\n\n')
            exit()
        plt.ion()
        fig = plt.figure()
        ax1 = fig.add_subplot(111, aspect='equal')
```

```
    if not os.path.exists('output'):
        os.makedirs('output')
    pattern = os.path.join(args.seq_path, phase, '*', 'det', 'det.txt')
    for seq_dets_fn in glob.glob(pattern):
        mot_tracker = Sort(max_age=args.max_age,
                           min_hits=args.min_hits,
                           iou_threshold=args.iou_threshold)  # create instance
of the SORT tracker
        seq_dets = np.loadtxt(seq_dets_fn, delimiter=',')
        seq = seq_dets_fn[pattern.find('*'):].split(os.path.sep)[0]

        with open(os.path.join('output', '%s.txt' % (seq)), 'w') as out_file:
            print("Processing %s." % (seq))
            for frame in range(int(seq_dets[:, 0].max())):
                frame += 1  # detection and frame numbers begin at 1
                dets = seq_dets[seq_dets[:, 0] == frame, 2:7]
                dets[:, 2:4] += dets[:, 0:2]  # convert to [x1,y1,w,h] to
[x1,y1,x2,y2]
                total_frames += 1

                if (display):
                    fn = os.path.join('mot_benchmark', phase, seq, 'img1',
'%06d.jpg' % (frame))
                    im = io.imread(fn)
                    ax1.imshow(im)
                    plt.title(seq + ' Tracked Targets')

                start_time = time.time()
                trackers = mot_tracker.update(dets)
                cycle_time = time.time() - start_time
                total_time += cycle_time

                for d in trackers:
                    print('%d,%d,%.2f,%.2f,%.2f,%.2f,1,-1,-1,-1' % (frame, d[4],
d[0], d[1], d[2] - d[0], d[3] - d[1]),
                          file=out_file)
                    if (display):
                        d = d.astype(np.int32)
                        ax1.add_patch(patches.Rectangle((d[0], d[1]), d[2] -
d[0], d[3] - d[1], fill=False, lw=3,
                                                        ec=colours[d[4] % 32,
:]))

                if (display):
                    fig.canvas.flush_events()
                    plt.draw()
                    ax1.cla()

    print("Total Tracking took: %.3f seconds for %d frames or %.1f FPS" % (
    total_time, total_frames, total_frames / total_time))

    if (display):
        print("Note: to get real runtime results run without the option: --
display")
```

ТЕОРЕТИЧЕСКИЕ ОСНОВЫ

В основе разработки лежат основные концепции и методы компьютерного зрения и глубокого обучения, применяемые для задачи распознавания автомобилей. Рассмотрим некоторые из них:

1. Компьютерное зрение является областью исследований, связанной с разработкой алгоритмов и методов для анализа и интерпретации изображений и видео с целью выделения и распознавания объектов на них. Основные методы компьютерного зрения включают в себя обнаружение объектов, сегментацию изображений, отслеживание объектов и классификацию.

2. Глубокое обучение представляет собой подраздел машинного обучения, основанный на использовании искусственных нейронных сетей с несколькими слоями. Оно позволяет автоматически извлекать высокоуровневые признаки из данных и применять их для решения различных задач, включая распознавание и классификацию объектов на изображениях и видео.

3. YOLO - это семейство алгоритмов обнаружения объектов, основанных на идее разбиения изображения на сетку ячеек и предсказании ограничивающих рамок и классов объектов в каждой ячейке. Этот подход позволяет достичь высокой скорости обработки видеопотока и высокой точности обнаружения объектов.

ВЫВОДЫ

Результаты исследования позволили разработать эффективную систему распознавания автомобилей и других транспортных средств на изображениях и видео. Модель YOLOv5 показала высокую точность обнаружения объектов, а алгоритм трекинга SORT обеспечил надежное отслеживание объектов в реальном времени. Разработанная система обладает несколькими ключевыми преимуществами, такими как высокая точность обнаружения объектов, высокая скорость работы и устойчивость к различным условиям окружающей среды. Она может быть успешно применена в широком спектре приложений, включая мониторинг дорожного движения и обеспечение безопасности на дорогах. Разработанная система имеет практическое применение в различных областях, таких как управление транспортом, безопасность на дорогах, городская инфраструктура и многое другое. Она может быть использована для автоматизации процессов мониторинга и контроля, что позволит снизить человеческий фактор и повысить эффективность систем управления.