Нижегородский государственный университет им. Н.И. Лобачевского
Институт информационных технологий, математики и механики
Кафедра Математического обеспечения и суперкомпьютерных технологий

Образовательный курс «Современные методы и технологии глубокого обучения в компьютерном зрении»

Практическая работа №3
Сопровождение объектов на видео

При поддержке компании Intel

Васильев Е.П.

Нижний Новгород
2020

Содержание

1 Введение

В данной практической работе рассматривается задача сопровождения (трекинга) объектов на видео через построение траекторий их движения и предлагается общая схема решения данной задачи с использованием Intel Distribution of OpenVINO Toolkit. В качестве натренированных моделей глубокого обучения используются модели из множества моделей Open Model Zoo [5].

2 Методические указания

2.1 Цели и задачи работы

Цель работы состоит в изучении и разработке программной реализации алгоритма сопровождения объектов, основанном на поиске соответствий, с применением инструмента Intel Distribution of OpenVINO Toolkit.

Для достижения поставленной цели необходимо решить следующие задачи:

- Изучить алгоритм сопровождения объектов через нахождение соответствий и реализовать его на языке Python.
- Разработать приложение на базе компонента Inference Engine в составе OpenVINO для сопровождения объектов. Результат сопровождения необходимо отобразить на исходном видео.
- Выполнить запуск и проверку корректности разработанного приложения.

2.2 Структура работы

Вначале в работе приводится описание алгоритма сопровождения объектов через нахождение соответствий. Далее поэтапно разрабатывается программный код, решающий задачу сопровождения объектов на видео. Выполняется запуск и проверка корректности работы приложения.

2.3 Рекомендации по проведению занятий

При выполнении данной практической работы рекомендуется следующая последовательность действий:

- Изучить алгоритм сопровождения объектов, основанный на нахождении соответствий.
- Настроить рабочее окружение для использования Intel Distribution of OpenVINO Toolkit.
- Разработать программное приложение для решения задачи сопрвождения объектов с применением компонента Inference Engine, входящего в состав инструмента Intel Distribution of OpenVINO Toolkit, проверить его работоспособность.

Отметим, что процедура настройки рабочего окружения подробно описана в первой работе, поэтому данный шаг в настоящем описании опущен.

3 Алгоритм сопровождения объектов через нахождение соответствий

3.1 Общая схема алгоритма сопровождения

Алгоритм принимает на вход последовательность кадров видеопотока. На выходе формируется последовательность положений объектов интересующих классов на входной последовательности кадров. Формальная постановка задачи сопровождения объектов приведена в соответствующей лекции курса.

Общая схема алгоритма состоит из нескольких этапов. При этом предполагается, что на некотором шаге уже построены отдельные части траекторий для набора объектов, обнаруженных к текущему шагу.

1. Получить новый кадр видео.

2. Обнаружить объекты на полученном кадре.
3. Вычислить матрицу сходства между обнаруженными объектами и объектами, для которых построены отдельные части траекторий.
4. По матрице сходства ответить на следующие вопросы:
   4.1. Каким траекториям какой обнаруженный объект соответствует?
   4.2. Какие объекты появились впервые на видео (не соответствуют ни одной из уже существующих траекторий)?
   4.3. Для каких траекторий на новом кадре не обнаружился объект (траектория завершилась в результате выхода объекта из области видимости камерой)?
5. В соответствии с полученными ответами обновить положения в траекториях, создать новые траектория для вновь обнаруженных объектов.
6. Повторить действия, начиная с шага 1, до момента завершения видео.

Первый шаг описанной схемы является техническим и выполняется стандартными средствами библиотек. Второй шаг представляет собой решение задачи детектирования объектов интересующих классов на изображении. Решение данной задачи с использованием методов глубокого обучения рассматривалось подробно в предыдущей практической работе. Здесь слушателю предлагается воспользоваться имеющимися знаниями и навыками. Рассмотрим детально шаги 3 – 5.

3.2 Вычисление матрицы сходства

Матрица сходства A для N траекторий и M объектов – это матрица размера $N \times M$, где каждый элемент a_{ij} представляет собой коэффициент сходства траектории T_i с объектом R_j. По данной матрице можно найти наилучшее соответствие между обнаруженными объектами и существующими траекториями.

Самый простой способ вычисления коэффициента сходства траектории T_i и нового объекта R_j выглядит следующим образом:

1. Получить последнее положение объекта в траектории T_i.
2. Сравнить объект, накрываемый окаймляющим прямоугольником в последнем положении траектории, и обнаруженный объект R_j по некоторому признаку.

Для сравнения можно использовать один или несколько следующих признаков: расположение, форму и внешний вид объекта (размер). Пусть D – расстояние между центрами окаймляющих прямоугольников (лежит на пересечении диагоналей), (w_1, h_1) – ширина и высота последнего прямоугольника в траектории, (w_2, h_2) – ширина и высота обнаруженного прямоугольника, C_1, C_2 – веса, определяющие вклад признака в результирующий коэффициент сходства. Тогда формула для вычисления коэффициента сходства по положению имеет вид:

$$affinity\_place = e^{-C_1\left(\frac{D^2}{w_1 h_1}\right)},$$

а формула для вычисления коэффициента сходства по размеру –

$$affinity\_shapes = e^{-C_2\left(\frac{w_1 - w_2}{w_1} + \frac{h_1 - h_2}{h_1}\right)}.$$

Исходя из этого, результирующий коэффициент сходства определяется следующим образом:

$$a_{ij} = affinity\_place * affinity\_shapes.$$

3.3 Поиск соответствий по матрице сходства

Задача поиска соответствий траекторий и обнаруженных положений по матрице сходства сводится к задаче о назначениях [8]. В канонической формулировке задача о назначениях имеет вид:

1. Имеется некоторое число работ $\{1, \ldots, N\}$ и то же самое число исполнителей $\{1, \ldots, N\}$.
2. Любой исполнитель i может быть назначен на выполнение любой (но только одной) работы $j = f(i)$, с затратами $a(i, j) \geq 0$.
3. Нужно распределить работы так, чтобы выполнить работы с минимальными суммарными затратами, т.е. необходимо минимизировать функционал $\sum_j a\big(i, f(i)\big)$.

Задача поиска наилучших соответствий траекторий и положений по матрице сходства – задача максимизации суммарного сходства. Чтобы свести нашу задачу к задаче о назначениях, необходимо выполнить следующие действия:

1. Сделать матрицу A квадратной. Для этого можно добавить некоторое количество пустых строк и столбцов, заполненных нулями.
2. Перейти от задачи максимизации к задаче минимизации. Поскольку $0 \leq a_{ij} \leq 1$, то достаточно заменить каждый элемент в матрице сходства по формуле $a'_{ij} = 1 - a_{ij}$.

Для решения полученной задачи можно воспользоваться функцией **`linear_sum_assignment`** пакета **`scipy.optimize`**, которая реализует Венгерский алгоритм [3, 4].

3.4 Обновление набора траекторий

После того как найдены соответствия траекторий и вновь обнаруженных объектов, остается обновить множество траекторий.

1. Если коэффициент сходства между объектом и траекторией превышает пороговое значение (обычно в диапазоне 0.02 – 0.2), то добавить объект в траекторию.
2. Если объект не добавлен ни в одну из имеющихся траекторий, то необходимо создать новую траекторию и добавить в нее обнаруженный объект.

4 Разработка приложения для сопровождения объектов на видео с использованием OpenVINO

4.1 Установка дополнительных зависимостей

В данной работе понадобится библиотека scientific-python, поэтому необходимо установить пакет SciPy. SciPy – библиотека для языка программирования Python с открытым исходным кодом, предназначенная для выполнения научных и инженерных расчетов.

```
pip install scipy
```

4.2 Разработка необходимых структур данных

Для выполнения практической работы необходимо создать два файла: файл **`matching_tracker.py`**, содержащий классы **`Tracklet`** и **`MatchingTracker`**, и файл **`tracking_sample.py`**, содержащий тестирующий код для запуска алгоритма сопровождения.

Базовой сущностью, которая содержит информацию о положении одного объекта на изображении, является именованный кортеж **`DetectedObject`**.

```
DetectedObject = namedtuple('DetectedObject',
   ['confidence', 'frame_idx', 'object_id', 'timestamp', 'class_id',
    'x_left', 'y_bottom', 'x_right', 'y_top',])
```

- **`confidence`** – достоверность наличия объекта в текущей области изображения (вещественное число).
- **`frame_idx`** – номер кадра видео (целое положительное число).
- **`object_id`** – идентификатор траектории (целое число, принимает значение -1 в том случае, если объект не назначен ни одной траектории).
- **`timestamp`** – временн***ая*** метка (целое число в миллисекундах), необходима для отслеживания момента времени детектирования объекта.

- **class_id** – идентификатор класса объектов (целое неотрицательное число, используется для вывода наименования класса объекта на экран).
- **x_left**, **y_bottom**, **x_right**, **y_top** – координаты левого верхнего и правого нижнего угла окаймляющего объект прямоугольника в формате, полученном от нейросети (т.е. нормированные координаты от 0 до 1).

Класс, который хранит информацию о траектории движения одного объекта в сцене, называется **Tracklet**. Данный класс хранит в себе список положений типа **DetectedObject** и предоставляет несколько методов для работы с ними.

- **__init__** – конструктор класса, создает внутри себя список для хранения **DetectedObject**.
- **__len__** – метод для получения длины списка объектов.
- **__getitem__** – метод для получения объекта по его номеру.
- **add_new_detection** – метод для добавления нового положения типа **DetectedObject** в список.

```
class Tracklet():
    def __init__(self, detection=[]):
        self._trackedObjects = []
        self._trackedObjects.append(detection)
    def __len__(self):
        return len(self._trackedObjects)
    def __getitem__(self, position):
        return self._trackedObjects[position]
    def add_new_detection(self, detection):
        self._trackedObjects.append(detection)
```

Данный подход с созданием нового типа данных из именованных кортежей и класса для удобного хранения и доступа к элементам является распространенной практикой в языке Python. Для того чтобы работа с траекториями происходила в парадигме языка Python, переопределены стандартные методы **__len__** и **__getitem__**.

Основной составляющей данного приложения является класс **MatchingTracker**. Данный класс создает и хранит траектории объектов, а также обеспечивает поиск соответствий между существующими траекториями и объектами, обнаруженными на последнем кадре. Класс **MatchingTracker** содержит следующие методы:

- **__init__** – конструктор класса, который создает пустой список траекторий.
- **add_new_track** – метод, который добавляет новую траекторию.
- **filter_detections** – метод, обеспечивающий фильтрацию выхода алгоритма детектирования и создание объектов типа **DetectedObject**.
- **_shape_affiinity** – метод, который вычисляет коэффициент сходства двух объектов на основе их размеров.
- **_place_affinity** – метод, которая вычисляет коэффициент сходства двух объектов на основе расстояния между ними.
- **_affinity** – метод, который смешивает два коэффициента сходства объектов.
- **_compute_dissimilarity_matrix** – метод, осуществляющий построение двумерной матрицы, которая состоит из мер сходства траекторий и обнаруженных объектов.
- **_solve_assignment_problem** – метод для поиска наилучших соответствий траекторий и обнаруженных объектов с помощью решения задачи о назначениях.
- **process_new_frame** – внешний метод, который обрабатывает новый кадр.
- **_draw_active_track** – метод для отрисовки на изображении существующих траекторий.

```
class MatchingTracker:
    """
    Class that stores and processes tracks based on images
    """

    def __init__(self, dist_weight=0.02, shape_affinity_weight=0.5,
            place_affinity_weight=0.7):
        pass
    def add_new_track(self, detection):
        pass
    def filter_detections(self, detect_mat, threshold=0.5):
        pass
    def process_new_frame(self, frame, detections, timestamp):
        pass
    def _solve_assignment_problem(self, tracks, detections):
        pass
    def _compute_dissimilarity_matrix(self, tracks, detections):
        pass
    def _affinity(self, obj1, obj2):
        pass
    def _shape_affinity(self, obj1, obj2, weight):
        pass
    def _place_affinity(self, o1, o2, weight):
        pass
    def draw_active_tracks(self, image):
        pass
```

Далее рассмотрим реализацию основных из приведенных методов.

4.3 Разработка метода вычисления коэффициента сходства траектории и объекта на основании положений

Метод вычисления коэффициента сходства траектории и обнаруженного объекта на основании положений на вход принимает последнее положение объекта в траектории и обнаруженное положение, а также вес признака «положение». Данный метод вычисляет коэффициент сходства согласно формуле, представленной в описании алгоритма. Результатом работы метода является число от 0 до 1.

```
def _shape_affinity(self, obj1, obj2, weight):
    obj1_width = obj1.x_right-obj1.x_left
    obj2_width = obj2.x_right-obj2.x_left
    obj1_height = obj1.y_top - obj1.y_bottom
    obj2_height = obj2.y_top - obj2.y_bottom
    w_dist = abs(obj1_width - obj2_width) / (obj1_width + obj2_width)
    h_dist = abs(obj1_height - obj2_height) / (obj1_height + obj2_height)
    return math.exp(-weight * (w_dist + h_dist))
```

4.4 Разработка метода вычисления коэффициента сходства траектории и объекта на основании размера

Метод вычисления коэффициента сходства траектории и обнаруженного объекта на основании размера на вход принимает последнее положение объекта в траектории и обнаруженное положение, а также вес признака «размер объекта». Данный метод вычисляет коэффициент сходства согласно формуле, представленной в описании алгоритма. Результатом работы метода является число от 0 до 1.

```
def _place_affinity(self, obj1, obj2, weight):
    obj1_x = (obj1.x_left + obj1.x_right) * 0.5
```

```
    obj1_y = (obj1.y_top + obj1.y_bottom) * 0.5
    obj2_x = (obj2.x_left + obj2.x_right) * 0.5
    obj2_y = (obj2.y_top + obj2.y_bottom) * 0.5
    obj2_width = obj2.x_right- obj2.x_left
    obj2_height = obj2.y_top - obj2.y_bottom
    x_dist = ((obj1_x - obj2_x) ** 2) / (obj2_width ** 2)
    y_dist = ((obj1_y - obj2_y) ** 2) / (obj2_height ** 2)
    return math.exp(-weight * (x_dist + y_dist))
```

4.5 Разработка метода вычисления общего коэффициента сходства траектории и объекта

Метод вычисления общего коэффициента сходства траектории и объекта **_affinity** смешивает коэффициенты сходства, полученные с помощью предыдущих двух методов. Результатом работы данной функции является число от 0 до 1.

```
def _affinity(self, obj1, obj2):
    shp_aff = self._shape_affinity(obj1, obj2,
        self._shape_affinity_weight)
    mot_aff = self._place_affinity(obj1, obj2,
        self._place_affinity_weight)
    return shp_aff * mot_aff
```

4.6 Разработка метода построения матрицы соответствия траекторий и объектов

Метод **_compute_dissimilarity_matrix** строит матрицу соответствий, в которой ячейка **[i,j]** содержит число, отвечающее коэффициенту сходства траектории с номером **i** и обнаруженного объекта **j**. Для решения данной задачи матрица должна быть квадратной – число траекторий должно соответствовать числу обнаруженных объектов. Если данное условие не выполняется, в матрицу добавляются пустые строки и столбцы с нулевыми элементами.

```
def _compute_dissimilarity_matrix(self, tracks, detections):
    size = max(len(tracks), len(detections))
    diss_mat = np.zeros(shape=(size, size), dtype=float)
    for i in range(len(tracks)):
        for j in range(len(detections)):
            diss_mat[i, j] = 1.0 - \
                self._affinity(tracks[i][-1], detections[j])
    return diss_mat
```

4.7 Разработка метода поиска наилучших соответствий между траекториями и обнаруженными объектами

Метод **_solve_assignment_problem** находит наилучшие соответствия между имеющимися траекториями и обнаруженными объектами через решение задачи о назначениях [8]. Таким образом на вход метод принимает наборы траекторий и обнаруженных объектов, на выходе формирует множество пар, отвечающих номеру траектории (строки матрицы сходства) и номеру соответствующего объекта (столбцу матрицы сходства). Для решения данной задачи, как отмечалось ранее, используется функция из пакета SciPy [9].

```
from scipy.optimize import linear_sum_assignment

def _solve_assignment_problem(self, tracks, detections):
    dissimilarity_mat = self._compute_dissimilarity_matrix(
        tracks, detections)
    row_ind, col_ind = linear_sum_assignment(dissimilarity_mat)
    return row_ind, col_ind
```

4.8 Разработка метода фильтрации обнаруженных объектов

Метод **filter_detections** получает на вход результат детектирования объектов на изображении при помощи глубокой модели (например, модели SSD300, использованной в предыдущей практике), и строит список из объектов типа **DetectedObject**, которые будут обрабатываться алгоритмом. Объекты, для которых достоверность меньше некоторого порогового значения (по умолчанию 0.5), отбрасываются и не включаются в список обнаруженных объектов.

```
def filter_detections(self, detect_mat, threshold=0.5):
    detect_mat = detect_mat[0, 0, :, :]
    detections = []
    for i in range(detect_mat.shape[0]):
        # Parse one string in ie_detection_output
        conf = detect_mat[i, 2]
        if conf > threshold:
            detection = DetectedObject(
                detect_mat[i, 2],
                -1,
                -1,
                -1,
                detect_mat[i, 1],
                detect_mat[i, 3],
                detect_mat[i, 4],
                detect_mat[i, 5],
                detect_mat[i, 6])
            detections.append(detection)
    return detections
```

4.9 Разработка метода добавления новой траектории

Данный метод получает на вход новый объект, создает для него новую траекторию и добавляет ее в список траекторий.

```
def add_new_track(self, detection):
    track = Tracklet(detection)
    self._tracks.append(track)
```

4.10 Разработка общего метода обработки очередного кадра видео

Метод получает на вход набор объектов, обнаруженных на новом кадре, строит матрицу сходства траекторий и обнаруженных объектов и решает задачу поиска наилучших соответствий траекторий и объектов. Затем для соответствий, коэффициент сходства для которых выше порогового значения, добавляет новые положения в существующие траектории. Если соответствие между существующей траекторией и объектом имеет низкий коэффициент сходства, то для таких объектов создаются новые траектории.

```
def process_new_frame(self, frame, detections, timestamp):
    if self._tracks and detections:
        row_indexes, col_indexes = self._solve_assignment_problem(
            self._tracks, detections)
        # For each assignment
        for i in range(len(row_indexes)):
            row_id = row_indexes[i]
            col_id = col_indexes[i]
            # If we find existing track and existing detection
            if col_id < len(self._tracks) and col_id < len(detections):
                # Add detection to track if objects are close
                dist = 1.0 - self._affinity(
                    self._tracks[row_id][-1], detections[col_id])
                if dist > self._dist_weight:
```

```
                    self._tracks[row_id].add_new_detection(
                        detections[col_id])
                elif col_id < len(detections):
                    self.add_new_track(detections[col_id])
    else:
        # Add new tracks
        for id, detection in enumerate(detections):
            self.add_new_track(detection)
    return
```

4.11 Разработка метода отображения траекторий движения объектов

Метод **draw_active_tracks** отрисовывает траектории на изображении. Каждая траектория рисуется следующим образом. Определяются центры окаймляющих прямоугольников на кадрах **i** и **i+1** и между ними отрисовывается прямая линия.

```
def draw_active_tracks(self, image):
    w, h = image.shape[:2]
    for i, track in enumerate(self._tracks):
        # Draw one track from segments
        for i in range(len(track) - 1):
            cv2.line(img=image,
                pt1=(int((track[i].x_left + track[i].x_right) * h)//2,
                    int((track[i].y_bottom + track[i].y_top) * w)//2),
                pt2=(int((track[i+1].x_left + track[i+1].x_right) * h)//2,
                    int((track[i+1].y_bottom + track[i+1].y_top) * w)//2),
                color=(0, 255, 0), thickness=3)
    return image
```

4.12 Создание объекта класса, обеспечивающего сопровождение

Конструктор класса **MatchingTracker** создает пустой список траекторий и устанавливает параметры для вычисления коэффициентов сходства объектов. Данные параметры подбираются для конкретного видео отдельно.

```
def __init__(self, dist_weight=0.02, shape_affinity_weight=0.5,
             place_affinity_weight=0.7):
    self._tracks = []
    self._dist_weight = dist_weight
    self._shape_affinity_weight = shape_affinity_weight
    self._place_affinity_weight = place_affinity_weight
    return
```

4.13 Создание тестирующего приложения

4.13.1 Разбор параметров командной строки

В данной работе потребуются следующие аргументы командной строки:

- Путь до входного видео (обязательный аргумент).
- Путь до весов нейронной сети (обязательный аргумент).
- Путь до конфигурации нейронной сети (обязательный аргумент).
- Путь до dll-библиотеки с нестандартными слоями (нужно для запуска SSD моделей на CPU) (необязательный аргумент).
- Путь до файла с именами классов (необязательный аргумент).

Ниже приведена реализация функции разбора аргументов командной строки средствами пакета **argparse**.

```
def build_argparser():
    parser = argparse.ArgumentParser()
    parser.add_argument('-m', '--model', help = 'Path to an .xml \
```

```
        file with a trained model.', required = True, type = str)
    parser.add_argument('-w', '--weights', help = 'Path to an .bin file \
        with a trained weights.', required = True, type = str)
    parser.add_argument('-i', '--input', help = 'Path to \
        input video', required = True, type = str)
    parser.add_argument('-l', '--cpu_extension', help='MKLDNN \
        (CPU)-targeted custom layers. Absolute path to a shared library \
        with the kernels implementation', type=str, default=None)
    parser.add_argument('-c', '--classes', help = 'File containing \
        classnames', type = str, default = None)
    return parser
```

4.13.2 Создание основной функции

Создайте функцию **main**, которая выполняет следующие действия:

1. Разбор аргументов командной строки.
2. Создание объекта класса **InferenceEngineDetector** с необходимыми параметрами для детектирования объектов на каждом кадре видео.
3. Создание объекта **tracker** класса **MatchingTracker** с эмпирически подобранными под видео параметрами.
4. Открытие видео с веб-камеры.
5. Выполнение последовательных действий для каждого кадра видео:
   5.1. Детектирование объектов на изображении.
   5.2. Фильтрация обнаруженных объектов с использованием метода **tracker.filter_detections**.
   5.3. Сопровождение объектов с использованием метода **tracker.process_new_frame**.
   5.4. Отрисовка траекторий на кадре и вывод полученного изображения на экран.

```
def main():
    args = build_argparser().parse_args()

    ie_detector = InferenceEngineDetector(configPath=args.model,
        weightsPath=args.weights, device=args.device,
        extension=args.cpu_extension, classesPath=args.classes)

    tracker = MatchingTracker()

    cap = cv2.VideoCapture(args.input)

    timestamp = 0
    while(cap.isOpened()):
        timestamp += 1
        _, frame = cap.read()

        detections_mat = ie_detector.detect(frame)
        detections = tracker.filter_detections(detections_mat, 0.5)

        tracker.process_new_frame(frame, detections, timestamp)

        tracks_image = tracker.draw_active_tracks(frame)

        result_image = ie_detector.draw_detection(detections_mat,
            tracks_image)

        cv2.imshow('Detections', result_image)
        if cv2.waitKey(1) & 0xFF == ord('q'):
```

```
            break

    cap.release()
    cv2.destroyAllWindows()
    return
```

5 Запуск приложения

Запуск разработанного приложения выполняется из командной строки. Для этого необходимо открыть командную строку. Строка запуска имеет следующий вид:

```
python tracking_sample.py -i video.mp4 -m ssd300.xml -w ssd300.bin \
    -l "C:\Program Files
(x86)\IntelSWTools\openvino\deployment_tools\inference_engine\bin\intel64\
Release\cpu_extension_avx2.dll"
```

где аргумент **-i** задает путь до видео, аргумент **-m** задает путь до конфигурации модели, аргумент **-w** задает путь до весов модели, аргумент **-l** задает путь до dll-библиотеки с нестандартными слоями.

Результат запуска приложения выглядит следующим образом. Выводится сообщение о старте приложения, затем открывается графическое окно, в котором отрисовываются кадры видео с обнаруженными объектами и траекториями (рис. 1).

```
[ INFO ] Start matching tracking sample
```

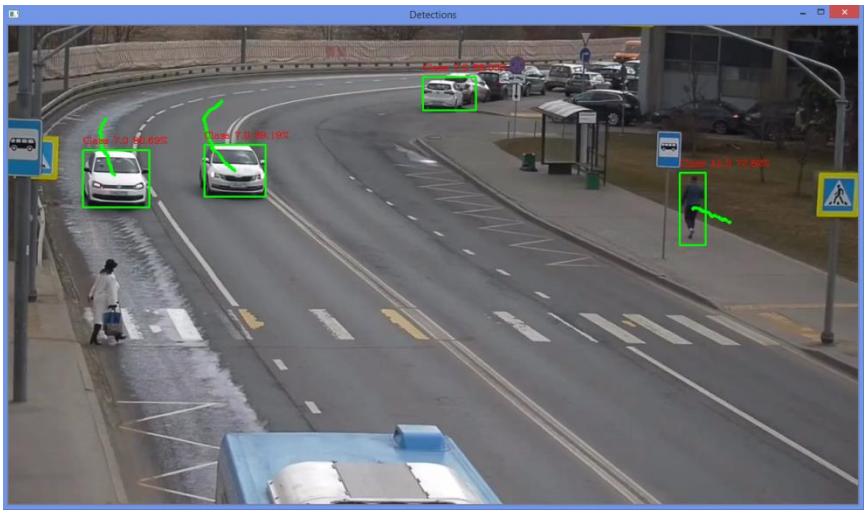

Рис. 1. Вывод приложения сопровождения объектов

6 Дополнительные задания

Созданный пример сопровождения объектов содержит минимально необходимый функционал. В качестве дополнительных заданий предлагается обеспечить поддержку следующих возможностей:

1. Вывод текущего количества обнаруженных объектов на каждом кадре видео и распределение количества объектов по классам. Предлагается выводить общее количество объектов и информацию по трем классам объектов, на которых получено максимальное количество объектов.
2. Ограничение количества интересующих классов объектов в зависимости от контекста видео. Например, если для детектирования объектов используется модель, обученная на наборе данных PASCAL VOC и обеспечивающая обнаружение 20 классов объектов, а на вход подается видео дорожного движения, то необходимо обнаруживать и сопровождать только объекты дорожного движения (автомобили, автобусы, пешеходы).
3. Поддержка разделения траекторий на «активные» и «неактивные». Под «неактивными» понимаются траектории, для которых не найден соответствующий объект в течение

некоторого промежутка времени (например, в течение нескольких секунд). Предлагается в течение этого временн***о***го промежутка рисовать траектории другим цветом, а по истечении этого времени не отображать эти траектории.

Приведенные задания предлагается выполнить самостоятельно, опираясь на документацию и примеры, входящие состав пакета OpenVINO.

7 Литература

7.1 Основная литература

1. Шолле Ф. Глубокое обучение на Python. – СПб.: Питер. – 2018. – 400с.

7.2 Дополнительная литература

2. Рамальо Л. Python. К вершинам мастерства / Пер. с англ. Слинкин А.А. – М.: ДМК Пресс. – 2016. – 768 с.
3. Kuhn H.W. The Hungarian Method for the assignment problem // Naval Research Logistics Quarterly. – 1955.
4. Kuhn H.W. Variants of the Hungarian method for assignment problems // Naval Research Logistics Quarterly. – 1956.

7.3 Ресурсы сети Интернет

5. Страница репозитория Open Model Zoo [https://github.com/opencv/open_model_zoo].
6. Документация Intel Distribution of OpenVINO Toolkit [https://docs.openvinotoolkit.org/2019_R3.1/index.html].
7. OpenVINO Multi Camera Multi Person Python Demo [https://docs.openvinotoolkit.org/2019_R3.1/_demos_python_demos_multi_camera_multi_person_tracking_README.html].
8. Задача о назначениях [https://wikimatik.ru/article/37].
9. Решение задачи о назначениях средствами библиотеки scipy [https://docs.scipy.org/doc/scipy-0.18.1/reference/generated/scipy.optimize.linear_sum_assignment.html].