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 №4
Решение задачи видеоаналитики, включающей детектирование, распознавание и сопровождение объектов на видео

При поддержке компании Intel

Васильев Е.П.

Нижний Новгород
2020

Содержание

1 Введение

Предлагается решение, основанное на детектировании транспортных средств с помощью глубокого обучения и их сопровождении на кадрах видеопотока. В качестве натренированных моделей глубокого обучения используются модели из множества обученных моделей Open Model Zoo [3].

2 Методические указания

2.1 Цели и задачи работы

Цель работы состоит в рассмотрении задачи подсчета транспортных средств разных классов на видео и разработке соответствующего приложения с применением инструмента Intel Distribution of OpenVINO Toolkit.

Для достижения поставленной цели необходимо решить следующие задачи:

- Изучить постановку задачи подсчета транспортных средств разных классов на видео и алгоритм решения поставленной задачи.
- Разработать приложение на базе компонента Inference Engine в составе OpenVINO для подсчета транспортных средств разных классов на видео. Результат необходимо отобразить на исходном видео.
- Выполнить запуск и проверку корректности разработанного приложения.

2.2 Структура работы

Вначале в работе рассматривается задача подсчета транспортных средств разных классов на видео. Описывается алгоритм ее решения. Далее поэтапно разрабатывается программный код, реализующий решение, которое основано на детектировании транспортных средств с помощью глубокого обучения и их сопровождения на кадрах видеопотока. Выполняется проверка корректности его работы.

2.3 Рекомендации по проведению занятий

При выполнении данной практической работы рекомендуется следующая последовательность действий:

- Изучить примеры решения задач видеоаналитики средствами Intel Distribution of OpenVINO Toolkit.
- Настроить рабочее окружение для использования Intel Distribution of OpenVINO Toolkit.
- Изучить постановку задачи подсчета транспортных средств разных классов на видео и алгоритм ее решения.
- Разработать программное приложение для решения поставленной задачи с использованием 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.

2.4 Примеры решения задач видеоаналитики, доступных в составе Intel Distribution of OpenVINO Toolkit

Intel Distribution of OpenVINO Toolkit содержит большой набор примеров решения задач видеоаналитики. Каждый пример сопровождается пояснениями, содержит описание принципа работы, входных и выходных данных, используемых глубоких моделей [5]. Ниже приведены некоторые примеры доступных приложений.

1. Action recognition Python demo [6]. Приложение для распознавания действий.

2. Crossroad camera demo [7]. Приложение реализует конвейер для обнаружения, распознавания и повторной идентификации людей. Используется модель детектирования лиц, к результатам работы которой применяются модели распознавания атрибутов лиц и идентификации лиц.
3. Security Barrier Camera C++ Demo [8]. Приложение имитирует работу системы слежения на парковке с ограниченным доступом. При этом демонстрируется работа модели обнаружения транспортных средств и номерных знаков, а также распознавания атрибутов транспортных средств и номерных знаков.
4. Smart Classroom C++ Demo [9]. Приложение имитирует работу системы слежения за действиями учеников в учебном классе. Демонстрируется совместное использование нескольких нейронных сетей для обнаружения действий ученика («сидит», «стоит», «поднимает руку вверх») и идентификации людей по лицам в аудитории.

Круг задач в составе Intel Distribution of OpenVINO Toolkit постоянно расширяется за счет новых демо-приложений.

3 Постановка задачи подсчета транспортных средств разных классов на видео

В данной практической работе рассматривается задача подсчета количества транспортных средств разных классов на видео, поступающем с дорожной камеры. Таким образом, на входе имеется последовательность кадров видео. Решение задачи подсчета транспортных средств разных классов предполагает, что на каждом текущем кадре видеопотока необходимо определить количество транспортных каждой категории («автомобиль», «автобус», «мотоцикл» и другие), которые уже наблюдались на видео к текущему кадру.

4 Алгоритм подсчета транспортных средств разных классов на видео

Общий алгоритм подсчета транспортных средств разных классов на видео на основе детектирования и сопровождения объектов, состоит из нескольких этапов, которые выполняются при обработке очередного полученного кадра видео.

1. Детектирование объектов на текущем кадре. Для детектирования можно использовать любую глубокую нейросетевую модель, способную обнаруживать транспортные средства разных классов («автомобиль», «автобус», «трамвай», «мотоцикл»). При этом необходимо отфильтровывать объекты, не принадлежащие множеству интересующих классов. Фильтрация может выполняться на этапе детектирования или на этапе выдачи статистики. При реализации рассматривается второй вариант. Также необходимо отметить, что алгоритм детектирования для одного и того же объекта на разных кадрах может выдавать разный класс. Здесь за класс принимается класс объекта, обнаруженного на текущем кадре (т.е. на последнем обработанном кадре). *Примечание:* в настоящей работе для детектирования объектов применяется модель SSD300, которая ранее использовалась при выполнении практической работы на детектирование объектов. Наряду с этим, используется разработанная программная реализация обертки детектора.
2. Сопровождение объектов, обнаруженных на предыдущем кадре видео, и обновление текущего состояния множества траекторий. *Примечание:* реализация данного этапа подробно рассмотрена в практической работе на сопровождение объектов, поэтому в настоящей работе ее описание опущено.
3. Сбор статистики о количестве транспортных средств разных классов, которые наблюдались к моменту обработки текущего кадра. Для этого необходимо завести счетчики для каждого интересующего класса и выполнить обход множества траекторий, увеличивая счетчики объектов соответствующего класса.

4. Отображение обнаруженных объектов и траекторий, вывод собранной информации на экран.

5 Разработка приложения подсчета транспортных средств разных классов на видео с использованием OpenVINO

5.1 Структура программных модулей

Для выполнения практической работы необходимо создать два файла: файл **videoanalytics.py** в директории **lib**, содержащий абстрактный класс **Videoanalytics** для отображения результатов решения задачи, и файл **videoanalytics_sample.py**, содержащий реализацию алгоритма решения задачи и тестирующий код. Предполагается, что модули **ie_detector.py** и **matching_tracker.py**, содержащие реализацию оберток алгоритмов детектирования и сопровождения объектов, уже разработаны при выполнении предыдущих практических работ и находятся в директории **lib**.

Рассмотрим более подробно класс **Videoanalytics**. Класс содержит следующие методы:

- **_init_** – конструктор класса. Загружает названия классов объектов, которые обнаруживаются моделью детектирования объектов, и номера интересующих классов транспортных средств в полученном перечне.
- **count_objects_per_classes** – метод, который производит подсчет транспортных средств каждого класса.
- **draw_videoanalytics** – метод для отображения результатов решения задачи подсчета транспортных средств на изображении.

Реализация конструктора достаточно простая, поэтому ее предлагается выполнить самостоятельно. Далее рассмотрим реализацию оставшихся методов.

5.2 Подсчет количества транспортных средств интересующих классов

Рассмотрим реализацию метода **count_objects_per_classes** для подсчета количества транспортных средств каждого класса. Данный метод проходит по всем траекториям, полученным в результате сопровождения, и подсчитывает количество объектов каждого интересующего класса.

```
def count_objects_per_classes(self, tracker):
    results = {}
    for tracklet in tracker._tracks:
        # Get object class from tracklet
        classid = tracklet[-1].class_id
        # Add +1 to counter for object
        if classid in results and classid in self.classIds:
            results[classid] += 1
        else:
            results[classid] = 1
    return results
```

5.3 Отображение результатов подсчета транспортных средств

Метод **draw_videoanalytics** обеспечивает отображение результатов подсчета транспортных средств и состоит из трех частей.

1. Отображение объектов интересующих классов. Предусматривает проход по всем траекториям.
   1.1. Поиск последнего окаймляющего прямоугольника в траектории.
   1.2. Если класс объекта удовлетворяет перечню интересующих классов и объект обнаружен на текущем кадре, то отображение окаймляющего прямоугольника.

2. Отображение траекторий объектов интересующих классов. Предусматривает проход по всем траекториям.
   2.1. Поиск последнего окаймляющего прямоугольника в траектории.
   2.2. Если класс объекта удовлетворяет перечню интересующих классов и объект обнаружен на текущем кадре, то проход по всем окаймляющим прямоугольникам, вычисление их центров, и отрисовка линии от центра к центру соседних объекта на соседних кадрах.
3. Отображение статистики.
   3.1. Получение количества объектов каждого интересующего класса с использованием метода **count_objects_per_classes**.
   3.2. Для каждого интересующего класса поиск его имени и формирование строки для отображения "Class <имя класса>: <количество объектов>".
   3.3. Вывод сформированных строк с информацией о количестве транспортных средств интересующих классов в левом верхнем углу изображения.

```
def draw_videoanalytics(self, frame, tracker):
    (h, w) = frame.shape[:2]

    # Draw detections of chosen classes
    for tracklet in tracker._tracks:
        # Get object class from tracklet
        classid = tracklet[-1].class_id
        if classid in self.classIds:
            # Draw bbox of the last detection
            x_left = int(tracklet[-1].x_left * w)
            y_bottom = int(tracklet[-1].y_bottom * h)
            x_right = int(tracklet[-1].x_right * w)
            y_top = int(tracklet[-1].y_top * h)
            cv2.rectangle(frame, (x_left, y_bottom), (x_right, y_top),
                (0, 255, 0), 2)
    # Draw tracklets of chosen classes
    for track in tracker._tracks:
        # Get object class from tracklet
        classid = track[-1].class_id
        if classid in self.classIds:
        # Draw one tracklet from segments
            for i in range(len(track)-1):
                x1 = int((track[i].x_left+track[i].x_right)*w)// 2
                y1 = int((track[i].y_bottom+track[i].y_top)*h)// 2
                x2 = int((track[i+1].x_left+track[i+1].x_right)*w)// 2
                y2 = int((track[i+1].y_bottom+track[i+1].y_top)*h)// 2
                cv2.line(img=frame, pt1=(x1,y1), pt2=(x2,y2),
                    color=(0, 255, 0), thickness=3)
    # Draw statistics
    counts = self.count_objects_per_classes(tracker)
    for i, elem in enumerate(counts):
        if self.classes:
            id = int(elem)
            text = 'Class {}: {} objects'.format(
                self.classes[id], counts[elem])
        else:
            text = 'Class {}: {} objects'.format(elem, counts[elem])
        text_pos = (0, i * 30 + 30)
        cv2.putText(frame, text, text_pos,
            cv2.FONT_HERSHEY_COMPLEX, 1.0, (0, 255, 255), 2)
```

```
    return frame
```

5.4 Разработка реализации алгоритма подсчета транспортных средств разных классов и тестирующего приложения

5.4.1 Разбор параметров командной строки

В данной работе потребуются следующие аргументы командной строки:

- Путь до входного видео (обязательный аргумент).
- Путь до весов нейронной сети детектирования (обязательный аргумент).
- Путь до конфигурации нейронной сети детектирования (обязательный аргумент).
- Путь до dll-библиотеки с нестандартными слоями (нужно для запуска SSD моделей на CPU) (необязательный аргумент).
- Путь до файла с именами детектируемых классов объектов (обязательный аргумент).
- Список номеров интересующих классов объектов (обязательный аргумент). Представляет собой несколько чисел, разделенных пробелом.

```
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('-d', '--device', help='Specify the target \
        device to infer on; CPU, GPU, FPGA or MYRIAD is acceptable. \
        Sample will look for a suitable plugin for device specified \
        (CPU by default)', default='CPU', type=str)
    parser.add_argument('-c', '--classes', help='File containing classes \
        names', type=str, default=None)
    parser.add_argument('-c_d', '--classes_detected', help='List of \
        classes to do videoanalysis', type=int, nargs='+',
        default='2 6 7 14 19')
    return parser
```

5.4.2 Создание основной функции

Функция **main** во многом похожа на функцию **main** из примера, разработанного для сопровождения объектов (предыдущая практическая работа), с небольшой разницей в том, что здесь используется метод **draw_videoanalytics** для отображения результатов видеоаналитики.

Рассмотрим подробно функцию **main**. Она предполагает выполнение следующих действий.

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** для сопровождения объектов с эмпирически подобранными под видео параметрами.
4. Создание объекта класса **Videoanalytics** с необходимыми параметрами для выполнения видеоаналитики.
5. Открытие видео или запуск веб-камеры.
6. Выполнение последовательных действий по кадрам видео:
   6.1. Детектирование объектов на изображении.

6.2. Фильтрация обнаруженных объектов с использованием метода **tracker.filter_detections**.
6.3. Сопровождение объектов с использованием метода **tracker.process_new_frame**.
6.4. Отображение объектов и траекторий на кадре и вывод полученного изображения на экран с помощью функции **draw_videoanalytics**.

```
def main():
    log.basicConfig(format="[ %(levelname)s ] %(message)s",
                    level=log.INFO, stream=sys.stdout)
    args = build_argparser().parse_args()

    log.info("Start videoanalytics sample")

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

    tracker = MatchingTracker()

    videoanalytics = Videoanalytics(args.classes_detected, args.classes)

    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)

        result_image = videoanalytics.draw_videoanalytics(frame, tracker)

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

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

6 Запуск приложения

Запуск разработанного приложения удобней всего произвести из командной строки. Для этого необходимо открыть командную строку. Строка запуска будет иметь следующий вид:

```
python videoanalytics_sample.py -i video.mp4 -m ssd300.xml -w ssd300.bin \
  -c pascal_voc.txt -c_d 2 6 7 19 \
  -l "C:\Program Files
(x86)\IntelSWTools\openvino\deployment_tools\inference_engine\bin\intel64\
Release\cpu_extension_avx2.dll"
```

где аргумент **-i** задает путь к видео, аргумент **-m_d** задает путь к конфигурации модели детектирования, аргумент **-w_d** задает путь к весам модели детектирования, аргумент **-l** задает путь до dll-библиотеки с нестандартными слоями, аргумент **-c** задает путь до файла с именами

детектируемых классов объектов, аргумент **-c_d** задает номера интересующих классов. Для моделей, обученных на данных PASCAL VOC, номера классов транспортных средств 2 (bicycle), 6 (bus), 7 (car), 19 (train).

Результат запуска приложения выглядит следующим образом. Выводится сообщение о старте приложения, затем открывается графическое окно, в котором отображаются кадры видео с обнаруженными объектами и траекториями (рис. 1), также на кадре отображается количество транспортных средств разных классов.

```
[ INFO ] Start videoanalytics sample
```

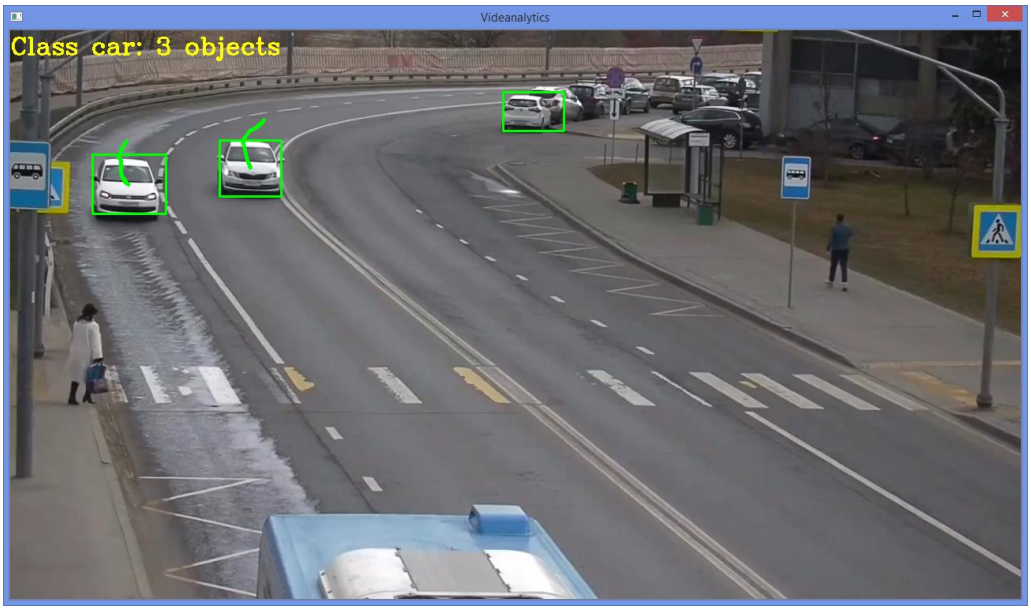

Рис. 1. Вывод результатов подсчета транспортных средств класса «автомобиль»

7 Дополнительные задания

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

1. Схема с сохранением информации обо всех обнаруженных объектах не является эффективной. Предлагается выполнить фильтрацию обнаруженных объектов сразу после запуска алгоритма детектирования на текущем кадре и осуществлять последующее сопровождение транспортных средств только интересующих классов.
2. Алгоритм детектирования на разных кадрах для одного и того же объекта может выдавать разные классы. Предлагается реализовать интеллектуальную схему определения класса объекта на основании полной информации о траектории. Наиболее простая схема – схема с голосованием, когда за класс объекта принимается класс, которому принадлежит большинство окаймляющих прямоугольников в траектории.

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

8 Литература

8.1 Основная литература

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

8.2 Дополнительная литература

2. Рамальо Л. Python. К вершинам мастерства / Пер. с англ. Слинкин А.А. – М.: ДМК Пресс. – 2016. – 768с.

8.3 Ресурсы сети Интернет

3. Страница репозитория Open Model Zoo [https://github.com/opencv/open_model_zoo].
4. Документация Intel Distribution of OpenVINO Toolkit [https://docs.openvinotoolkit.org/2019_R3.1/index.html].
5. Inference Engine demos [https://docs.openvinotoolkit.org/2019_R3.1/_demos_README.html].
6. Action recognition Python demo [https://docs.openvinotoolkit.org/2019_R3.1/_demos_python_demos_action_recognition_README.html].
7. Crossroad camera demo [https://docs.openvinotoolkit.org/2019_R3.1/_demos_crossroad_camera_demo_README.html].
8. Security barrier camera demo [https://docs.openvinotoolkit.org/2019_R3.1/_demos_security_barrier_camera_demo_README.html]
9. Smart Classroom C++ Demo [https://docs.openvinotoolkit.org/2019_R3.1/_demos_smart_classroom_demo_README.html]