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 №2
Детектирование объектов на изображениях

При поддержке компании Intel

Васильев Е.П.

Нижний Новгород
2020

Содержание

1 Введение

В данной практической работе рассматривается задача детектирования объектов на изображениях и предлагается общая схема ее решения средствами Intel Distribution of OpenVINO Toolkit [9]. В качестве натренированных моделей глубокого обучения используются модели из множества обученных моделей Open Model Zoo [7].

2 Методические указания

2.1 Цели и задачи работы

Цель работы состоит в изучении и применении глубоких моделей для решения задачи детектирования объектов на изображениях с применением инструмента Intel Distribution of OpenVINO Toolkit.

Для достижения поставленной цели необходимо решить следующие задачи:

- Изучить лекцию «Детектирование объектов разных классов на изображениях с использованием глубоких нейронных сетей».
- Изучить глубокие модели для детектирования объектов, входящие в состав Open Model Zoo, и выбрать модель для дальнейшего решения задачи. В данной работе демонстрируется пример использования модели SSD300 [4].
- Разработать приложение на базе компонента Inference Engine в составе OpenVINO для детектирования объектов с использованием выбранной модели. Результат детектирования необходимо отобразить на исходном изображении.
- Выполнить запуск и проверку корректности разработанного приложения.

2.2 Структура работы

Вначале в работе приводится краткое описание глубоких нейросетевых моделей для детектирования объектов, которые доступны в Open Model Zoo. Далее поэтапно разрабатывается программное приложение, решающее задачу детектирования объектов с использованием одной из рассмотренных моделей. Выполняется запуск и проверка корректности его работы.

2.3 Рекомендации по проведению занятий

При выполнении данной практической работы рекомендуется следующая последовательность действий:

- Изучить лекцию «Детектирование объектов разных классов на изображениях с использованием глубоких нейронных сетей».
- Настроить рабочее окружение для использования Intel Distribution of OpenVINO Toolkit.
- Изучить глубокие модели для детектирования объектов, входящие в состав Open Model Zoo, и выбрать модель для дальнейшего решения задачи.
- Разработать программный код для решения задачи детектирования 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 предыдущей работе, поэтому данный шаг в настоящем описании опущен.

3 Дополнительная техническая информация

3.1 Глубокие модели для детектирования объектов, входящие в состав Open Model Zoo

В настоящее время распространение получило несколько групп глубоких моделей, которые применяются для решения задачи детектирования объектов.

- Region-based Convolutional Networks (RCNN) [2].
- Region-Based Fully Convolutional Networks (RFCN) [3].
- Single Shot MultiBox Detector (SSD) [4].
- You Only Look Once (YOLO) [5].

Более полную информацию можно найти в лекции «Детектирование объектов разных классов на изображениях с использованием глубоких нейронных сетей», входящей в состав данного курса. Здесь же остановимся на некоторых моделях для детектирования объектов, входящих в Open Model Zoo.

В Open Model Zoo присутствует ***семейство моделей SSD***: ssd300, ssd512, mobilenet-ssd и другие. Указанная группа моделей имеет идентичный вход и выход. Вход SSD моделей, обученных в Caffe, представляет собой тензор размера $[B \times C \times H \times W]$, где B – количество обрабатываемых изображений в пачке, подаваемой на вход сети, C – количество каналов изображений, H, W – высота и ширина входного изображения соответственно. Выход SSD-моделей, сконвертированных в промежуточное представление OpenVINO, представляет собой тензор размера $[1 \times 1 \times N \times 7]$, в котором каждая строка (последняя размерность тензора) содержит следующие параметры: [image_number, classid, score, left, bottom, right, top], где image_number – номер изображения; classid – идентификатор класса; score – достоверность присутствия объекта в выделенной области; left, bottom, right, top – нормированные координаты окаймляющих прямоугольников в диапазоне от 0 до 1.

Наряду с этим, в состав Open Model Zoo входит семейство ***моделей, построенных на базе RCNN***. К таковым относятся faster_rcnn_resnet50_coco, faster_rcnn_inception_v2_coco, mask_rcnn_resnet50_atrous_coco, mask_rcnn_inception_v2_coco и некоторые другие. Вход RCNN-моделей, обученных с использованием библиотеки TensorFlow, отличается от SSD-моделей. Сконвертированые RCNN модели содержат два входных тензора: первый тензор размера $[B \times C \times H \times W]$, отвечающий набору обрабатываемых изображений, второй тензор размерности $[B \times C]$, где C – вектор, состоящий из трех значений в формате $[H, W, S]$, где S – коэффициент масштабирования исходного изображения. Выход сконвертированных моделей по аналогии с SSD-моделями представляет собой тензор размера $[1 \times 1 \times N \times 7]$ или $[N \times 7]$, в котором каждая строка (последняя размерность тензора) содержит следующие параметры: [image_id, label, conf, (x_min, y_min), (x_max, y_max)], где image_id – номер изображения; label – номер класса; conf – достоверность присутствия объекта в выделенной области; (x_min, y_min), (x_max, y_max) – нормированные координаты окаймляющих прямоугольников в диапазоне от 0 до 1.

В Open Model Zoo помимо публичных моделей, приведенных выше, также имеются собственные модели Intel, которые решают задачи детектирования специфичных объектов (транспортных средств, автомобильных номеров, людей, лиц, текста). К таким моделям, в частности, относятся face-detection-retail, person-detection-retail, vehicle-detection-adas, vehicle-license-plate-detection-barrier-0106 и другие. Данные модели имеют вход и выход, аналогичный SSD-моделям. Полное описание детектируемых классов объектов, входа и выхода для каждой модели доступно в документации или директории OpenVINO, указанной ниже.

```
<openvino_dir>/deployment_tools/open_model_zoo/models/intel
```

3.2 Расширения реализаций слоев для вывода глубоких моделей средствами Inference Engine

Глубокие модели состоят из большого количества слоев. Как правило, реализация слоев ложится на плечи разработчиков конкретного фреймворка. При исполнении глубоких моделей на CPU средствами OpenVINO используются реализации слоев из библиотеки Deep Neural Network Library (DNNL, предыдущее название MKL-DNN) [10]. С появлением новых моделей разрабатываются специфичные слои. Как следствие, в OpenVINO создан механизм подключения реализации слоев, отсутствующих в DNNL. Программный интерфейс Inference Engine содержит функции для подключения динамических библиотек с реализацией слоев. Таким образом, команда OpenVINO

поддерживает работоспособность моделей из Open Model Zoo, для которых реализация слоев отсутствует в DNNL. Примером такого слоя является слой подавления не-максимумов (Non-Maximum Suppression Layer) в конце топологии моделей, построенных на базе SSD [4]. Подключаемая динамическая библиотека со специфичными слоями может быть скомпилирована с использованием инструкций SSE или AVX для ускорения вывода. В Intel Distribution of OpenVINO Toolkit по умолчанию присутствует динамическая библиотека, скомпилированная с использованием инструкций AVX-2.

В примерах и демо-приложениях для подключения динамической библиотеки с реализованными слоями, как правило, нужно указать аргумент **-l** и путь до динамической библиотеки.

```
python any_openvino_sample.py -l "C:\Program Files
(x86)\IntelSWTools\openvino\deployment_tools\inference_engine\bin\intel64\
Release\cpu_extension_avx2.dll" <other_arguments...>
```

При разработке программного приложения после создания объекта среды **IECore** необходимо воспользоваться методом **add_extension**, который принимает два параметра – путь до динамической библиотеки и тип устройства (**'CPU'**), на котором будут выполняться вычисления.

```
ie = IECore()
ie.add_extension(cpu_extension_path, 'CPU')
```

3.3 Запуск примера OpenVINO для детектирования объектов на изображениях

Рассмотрим процедуру запуска примера для детектирования объектов, входящего в состав OpenVINO. В пакете OpenVINO содержится файл **detection_sample.py**, который позволяет детектировать объекты на изображениях при помощи глубоких нейронных сетей. На официальном сайте присутствует полное описание данного примера и инструкций по его запуску [9]. Здесь приведем краткую информацию, необходимую для быстрого старта.

Для запуска примера скачайте, сконвертируйте и запустите модель SSD300, последовательность команд приведена ниже. В приведенном перечне команд необходимо заменить пути в угловых скобках на реальные пути на вашем компьютере.

```
python "C:\Program Files
(x86)\IntelSWTools\openvino\deployment_tools\tools\model_downloader\downlo
ader.py" --name ssd300 --output_dir <destination_folder>
python "C:\Program Files
(x86)\IntelSWTools\openvino\deployment_tools\tools\model_downloader\conver
ter.py" --name ssd300 --download_dir <destination_folder>
python " C:\Program Files
(x86)\IntelSWTools\openvino\inference_engine\samples\python_samples\
object_detection_sample_ssd\object_detection_sample_ssd.py" -i
<path_to_image> -m <path_to_model>\ssd300.xml -l "C:\Program Files
(x86)\IntelSWTools\openvino\deployment_tools\inference_engine\bin\intel64\
Release\cpu_extension_avx2.dll"
```

После запуска данного приложения в консоли должен появиться выход работы данного примера, а в текущей директории появится изображение **out.bmp** с обведенными объектами (если директория будет открыта для записи файлов).

```
[ INFO ] Loading network files:
        ssd300.xml
        ssd300.bin
[ INFO ] Loading Inference Engine
[ INFO ] Device info:
        CPU
        MKLDNNPlugin version ......... 2.1
```

```
        Build ........... 32974
[ INFO ] CPU extension loaded: C:\Program Files
(x86)\IntelSWTools\openvino\inference_engine\bin\intel64\Release\cpu_exten
sion_avx2.dll
[ INFO ] File was added:
[ INFO ]         dog.jpg
[ WARNING ] Image dog.jpg is resized from (300, 300) to (300, 300)
[ INFO ] Preparing input blobs
[ INFO ] Batch size is 1
[ INFO ] Preparing output blobs
[ INFO ] Loading model to the device
[ INFO ] Creating infer request and starting inference
[ INFO ] Processing output blobs
[0,3] element, prob = 0.00579197    (149,576)-(179,600) batch id : 0
[1,3] element, prob = 0.00516128    (860,406)-(912,453) batch id : 0
[2,3] element, prob = 0.00462477    (617,493)-(653,537) batch id : 0

...

[195,17] element, prob = 0.0043105    (593,122)-(629,136) batch id : 0
[196,17] element, prob = 0.00430544    (617,498)-(657,520) batch id : 0
[197,17] element, prob = 0.00429488    (0,442)-(29,462) batch id : 0
[198,17] element, prob = 0.0042881    (638,136)-(682,155) batch id : 0
[199,17] element, prob = 0.00425323    (92,437)-(137,458) batch id : 0
[ INFO ] Image out.bmp created!
[ INFO ] Execution successful

[ INFO ] This sample is an API example, for any performance measurements
please use the dedicated benchmark_app tool
```

Код данного и остальных примеров можно использовать для изучения программного интерфейса OpenVINO. Далее приводится пошаговая инструкция по разработке приложения для детектирования объектов.

4 Разработка приложения для детектирования объектов на изображениях с использованием OpenVINO

4.1 Рабочие скрипты

Для выполнения практической работы необходимо создать два файла: файл **ie_detector.py**, содержащий абстрактный класс с реализацией детектора **InferenceEngineDetector** и файл **detection_sample.py**, содержащий тестирующий код для запуска детектора **InferenceEngineDetector**. Рассмотрим более подробно этот класс.

Класс **InferenceEngineDetector** содержит следующие методы:

- **_init_** – конструктор класса, инициализирует Inference Engine и загружает модель.
- **_prepare_image** – метод, который преобразует изображение в формат входа нейронной сети.
- **detect** – метод, обеспечивающий детектирование объектов при помощи нейронной сети.
- **draw_detection** – метод для отрисовки результатов детектирования на изображении.

```
class InferenceEngineDetector:
    def __init__(self, configPath = None, weightsPath = None,
            extension=None, classesPath = None):
        pass
```

```
    def _prepare_image(self, image, h, w):
        pass

    def detect(self, image):
        pass

    def draw_detection(self, detections, image, confidence=0.5,
                       draw_text=True):
        pass
```

Далее поэтапно реализуем приведенные методы.

4.2 Загрузка модели

Для того, чтобы выполнить загрузку модели, в файле **ie_detector.py** необходимо реализовать конструктор класса **InferenceEngineDetector**. Конструктор содержит следующие параметры:

- **configPath** – путь до xml-файла с описанием модели.
- **weightsPath** – путь до bin-файла с весами модели.
- **classesPath** – путь до файла с именами классов.
- **extension** – путь до библиотеки с реализацией нестандартных слоев глубоких моделей.

Конструктор выполняет следующие действия:

- Создание объекта класса **IECore**.
- Загрузка в объект класса **IECore** динамической библиотеки слоев при помощи функции **add_extension**.
- Создание объекта класса **IENetwork** с параметрами – путями до модели.
- Загрузка перечня классов изображений из файла.
- Загрузка нестандартных слоев.
- Загрузка созданного объекта класса **IENetwork** в **IECore**, что соответствует загрузке модели в плагин.

Объекты классов **IECore**, **IENetwork**, **ExecutableNetwork**, а также перечень классов объектов необходимо сохранить в качестве полей класса.

4.3 Загрузка и предобработка изображения

Для предобработки изображения реализуйте функцию **_prepare_image** в файле с тестирующим кодом. Обработка изображений глубокими моделями отличается от обработки изображений классическими алгоритмами тем, что сети принимают изображения поканально, а не попиксельно, изображения в подаваемой пачке необходимо преобразовать из формата RGBRGBRG… в формат RRRGGGBBB... Для этого можно воспользоваться функцией **transpose**.

```
image = image.transpose((2, 0, 1))
```

Также необходимо уменьшить или увеличить размер изображения до размера входа сети.

```
image = cv2.resize(image, (w, h))
```

В общем случае для модели SSD300 на вход должен подаваться четырехмерный тензор, например, [1,3,300,300], где первая координата – количество изображений в пачке; 3 – количество цветовых каналов изображения; 300 – ширина и высота изображения. Отметим, что если на вход сети подать трехмерный тензор [3,300,300], то OpenVINO автоматически добавит четвертую размерность.

4.4 Вывод модели

Следующий этап – реализация метода **detect** для детектирования объектов на изображении, который запускает исполнение глубокой модели на устройстве, указанном в конструкторе. Логика работы метода **detect** следующая.

1. Получить информацию о входе и выходе нейронной сети:

```
input_blob = next(iter(self.net.inputs))
out_blob = next(iter(self.net.outputs))
```

2. Из данных о входе нейронной сети получить требуемые нейросетью размеры входного изображения:

```
n, c, h, w = self.net.inputs[input_blob].shape
```

3. С помощью функции **_prepare_image** подготовить изображение.
4. Вызвать метод синхронного исполнения модели:

```
output = self.exec_net.infer(inputs = {input_blob: blob})
```

5. Из выхода модели получить тензор с результатом детектирования:

```
output = output[out_blob]
```

4.5 Обработка выхода модели

Выходом модели SSD300 является тензор размера [1x1x200x7], в котором каждая строка (последняя размерность тензора) содержит следующие параметры: [image_number, classid, score, left, bottom, right, top], где image_number – номер изображения (в разработанном приложении всегда 0, так как подается одно изображение); classid – номер класса; score – достоверность присутствия объекта в данном месте; left, bottom, right, top – нормированные координаты окаймляющих прямоугольников в диапазоне от 0 до 1.

Для обработки выхода необходимо реализовать метод **draw_detection**, который отрисует на изображении обнаруженные объекты. Последовательность работы указанного метода следующая.

1. Получить текущую строку матрицы.
2. Получить достоверность детектирования (третий параметр).
3. Если достоверность больше некоторого порогового значения (для определенности можно выбрать 0.5), то получить идентификатор класса и координаты окаймляющего прямоугольника. По идентификатору классу можно восстановить название класса. Чтобы получить координаты окаймляющего прямоугольника в системе координат, связанной с изображением, необходимо умножить полученные нормированные значения из выходного тензора на длину и ширину входного изображения.
4. Отрисовать прямоугольник на изображении средствами OpenCV. Для отрисовки прямоугольника воспользуйтесь функцией **cv2.rectangle**. Прототип и описание параметров данной функции приведены ниже.

```
cv2.rectangle(img, point1, point2, color, line_width)
```

- **img** – изображение, на котором необходимо выполнить отрисовку.
- **point1 = (x,y)** – кортеж из двух целых чисел, соответствующий координатам левого верхнего угла окаймляющего прямоугольника.
- **point2 = (x,y)** – кортеж из двух целых чисел, соответствующий координатам правого нижнего угла окаймляющего прямоугольника.
- **color = (B,G,R)** – кортеж из трех целых чисел от 0 до 255, определяющий цвет линий.
- **line_width = 1** – вещественное положительное число, определяющее толщину линий.

Для отображения текста, содержащего класс объекта, воспользуйтесь функцией **cv2.puttext**. Прототип и описание параметров данной функции приведены ниже.

```
cv2.putText(img, text, point, cv2.FONT_HERSHEY_COMPLEX, text_size, color,
1)
```

- **img** – изображение, на котором необходимо нарисовать.
- **text** – строка надписи.
- **point** – точка старта надписи на изображении.
- **color** – кортеж из трех целых чисел от 0 до 255, определяющий цвет текста.
- **text_size = 0.45** – вещественное положительное число, определяющее размер текста.

4.6 Создание тестирующего примера

4.6.1 Разбор параметров командной строки

В данном приложении потребуются следующие аргументы командной строки:

- Путь до входного изображения (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 \
        image file', 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
```

4.6.2 Создание основной функции

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

1. Разбор аргументов командной строки.
2. Создание объекта класса **InferenceEngineDetector** с необходимыми параметрами.
3. Чтение изображения.
4. Детектирование объектов на изображении.
5. Отрисовка детектированных объектов на изображении.
6. Отображение полученного изображения на экране.

```
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)

    img = cv2.imread(args.input)
```

```
    detections = ie_detector.detect(img)

    image_detected = ie_detector.draw_detection(detections, img)

    cv2.imshow('Image with detections', image_detected)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

    return
```

5 Запуск приложения

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

```
python ie_detection_sample.py -i image.jpg -m ssd300.xml -w ssd300.bin \
    -c voc_labels.txt \
    -l "C:\Program Files
(x86)\IntelSWTools\openvino\deployment_tools\inference_engine\bin\intel64\
Release\cpu_extension_avx2.dll"
```

где аргумент **-i** задает путь до изображения, аргумент **-m** задает путь до конфигурации модели, аргумент **-w** задает путь до весов модели, аргумент **-c** задает путь к файлу с именами классов для модели, аргумент **-l** задает путь до dll-библиотеки с нестандартными слоями.

Результат запуска приложения выглядит следующим образом: выводится сообщение о старте примера, затем открывается графическое окно, в котором отрисовано изображение с обнаруженными объектами (рис. 1).

```
 [ INFO ] Start IE detection sample
```

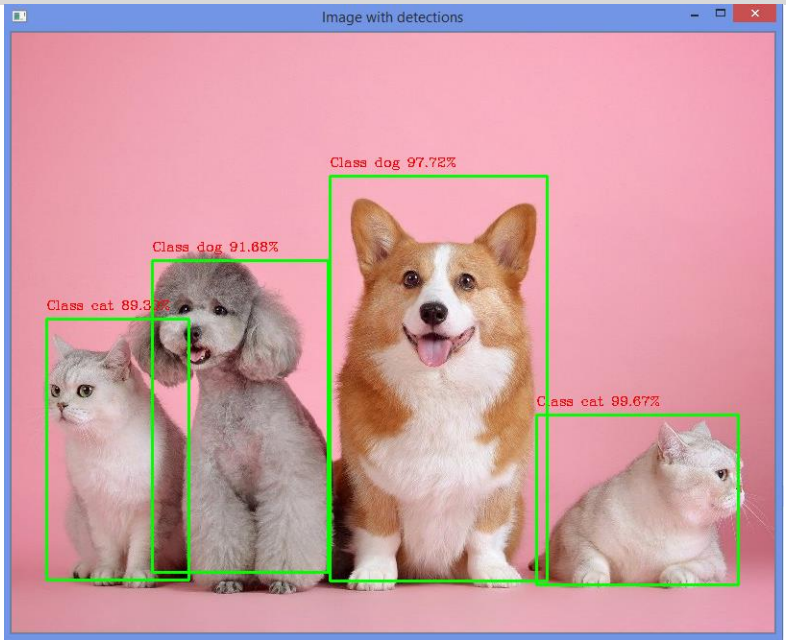

Рис. 1. Пример вывода программы детектирования

6 Дополнительные задания

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

1. Открытие видео и детектирование объектов на каждом кадре, отрисовка обнаруженных объектов на каждом кадре видеопотока.

2. Поддержка других глубоких моделей детектирования объектов на изображениях, содержащиеся в Open Model Zoo [7].
3. Вычисление времени обработки одного изображения.

Данные 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.
2. Girshick R., Donahue J., Darrell T., Malik J. Rich feature hierarchies for accurate object detection and semantic segmentation. – 2014.
3. Dai J., Li Y., He K., Sun J. R-FCN: Object detection via region-based fully convolutional networks. – 2016.
4. Liu W., Anguelov D., Erhan D., Szegedy C., Reed S., Fu C.Y., Berg A.C. SSD: Single Shot MultiBox Detector. – 2016.
5. Redmon J., Divvala S., Girshick R., Farhadi A. You only look once: Unified, real-time object detection. – 2015.

7.2 Дополнительная литература

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

7.3 Ресурсы сети Интернет

7. Страница репозитория Open Model Zoo [https://github.com/opencv/open_model_zoo].
8. Документация Intel Distribution of OpenVINO Toolkit [https://docs.openvinotoolkit.org/2019_R3.1/index.html].
9. OpenVINO detection sample [https://docs.openvinotoolkit.org/2019_R3.1/_inference_engine_ie_bridges_python_sample_object_detection_sample_ssd_README.html].
10. Репозиторий Deep Neural Network Library [https://github.com/intel/mkl-dnn].