Детектирование объектов на изображениях с помощью Python, OpenCV и YOLOv4

АННОТАЦИЯ

Детектирование объектов на изображениях с помощью нейронных сетей может быть сопряжено с определенными трудностями. Иногда недоступен мощный компьютер, оснащенный графическим ускорителем. В некоторых случаях, особенно в защищенных системах, нет возможности использовать специализированные фреймворки наподобие *torch* или *tensorflow*. В таких ситуациях для детектирования объектов можно применить связку из *Python*, библиотеки *opencv* и нейросети с архитектурой *YOLOv4-tiny*.

ОСНОВНОЙ ТЕКСТ

С появлением сверточных нейронных сетей и ростом вычислительной мощности компьютеров произошел настоящий прорыв в обработке изображений. За последние несколько лет появилось большое количество архитектур нейросетей для обработки изображений. Выбор архитектуры нейросети под конкретную задачу определяется сочетанием многих факторами, среди которых можно выделить требования к точности, располагаемой вычислительной мощностью “железа”, а также возможностью использования различных нейросетевых фреймворков.

Согласно исследованию, по состоянию на апрель 2020 года нейросети архитектуры YOLOv4 обеспечивали наилучший баланс между точностью и скоростью обработки в задачах детектирования объектов.

Используя эту информацию, а также принимая во внимание возможность использования нейросетевых алгоритмов обработки изображений с помощью модуля *opencv.dnn* известной библиотеки *opencv*, рассмотрим пример детектирования объектов с помощью нейронной сети *YOLOv4-tiny*, предобученной на наборе данных *MS COCO 2017* и позволяющей определять 80 видов объектов. Одна из особенностей этой нейронной сети - сравнительно небольшой (23 Мб) файл с параметрами сети.

Изображения будем получать с веб-камеры, а результаты детектирования - выводить на экран.

Скачиваем три файла: конфигурационный файл с архитектурой нейросети, файл с параметрами нейросети и файл с метками классов.

В первой части скрипта импортируем библиотеку *opencv*, задаем гиперпараметры модели, создаем модель на основе ранее скачанных файлов.

```
import cv2

# гиперпараметры модели
```

```
conf_thr = 0.4
nms_thr = 0.6

# цвет ограничивающих прямоугольников и шрифта
color = (0, 255, 255)

# загрузка модели
net = cv2.dnn.readNet('yolov4-tiny.weights', 'yolov4-tiny.cfg')
with open('coco.names', 'r') as f:
    labels = [line.strip() for line in f.readlines()]
model = cv2.dnn_DetectionModel(net)
model.setInputParams(scale=1/255, size=(416, 416))
```

Во второй части скрипта подключаемся к веб-камере, затем в цикле считываем очередной кадр с веб-камеры; детектируем объекты; добавляем на исходный кадр ограничивающие прямоугольники, метки классов и вероятности принадлежности объектов к классам; выводим полученное изображение на экран.

```
# подключение к веб-камере
cam = cv2.VideoCapture(0)

while True:

    # чтение кадра
    ret_val, img = cam.read()

    # детектирование
    classes, scores, boxes = model.detect(img, conf_thr, nms_thr)

    # рисование прямоугольников, меток и вероятностей
    for class_id, score, box in zip(classes, scores, boxes):
        cv2.rectangle(img, box, color, 2)
        label = labels[class_id[0]]
        text = '{} {:.3f}'.format(label, score[0])
        coord = (box[0] + 5, box[1] + 15)
        cv2.putText(img, text, coord,
                    cv2.FONT_HERSHEY_SIMPLEX,
                    0.5, color, 1)

    # вывод на экрана, проверка нажатия клавиши Esc
    cv2.imshow('object detection', img)
    if cv2.waitKey(1) == 27:
        break
```

Эксперименты на разных системах (см. таблицу ниже) показывают, что даже на компьютерах без графического ускорителя быстродействие достаточно для работы в режиме реального времени. Одноплатный микрокомпьютер также справляется с задачей детектирования, хотя и со значительно меньшей скоростью.

№ п/п	Компьютер, модель и тактовая	Программное	Производительность,

	частота процессора	обеспечение	кадров в секунду
1	Десктоп Intel i5-2500K 3,7 ГГц	Windows 7 Python 3.8.5 opencv 4.5.1	12,0
2	Ноутбук Intel i3-7020U 2,3 ГГц	Windows 10 Python 3.8.3 opencv 4.4.0	7,8
3	Одноплатный микрокомпьютер Raspberry Pi 3 Model B ARMv7 1,2 ГГц	Linux raspberry 5.10.52 Python 3.7 opencv 4.5.3	0,4

В модели детектирования используется два гиперпараметра: *confidence threshold* (переменная *conf_thr* в скрипте) и *non-maximum supression threshold* (переменная *nms_thr*). Первый гиперпараметр задает порог чувствительности при обнаружении объектов и, по сути, отвечает за баланс между ошибками 1-го рода (ложное срабатывание) и 2-го рода (несрабатывание). Второй гиперпараметр отвечает за поведение модели при наложении объектов. При очень больших значениях возможны множественные прогнозы для одного и того же объекта, при очень малых возможно объединение контуров для однотипных сливающихся объектов.

В целях демонстрации влияния гиперпараметров на поведение модели с помощью несколько модифицированной версии описанного выше скрипта была обработана запись с автомобильного регистратора. Вся запись опубликована в youtube, а один из кадров приведен ниже.

На этом кадре можно заметить два эффекта из вышеперечисленных: на правом верхнем изображении - несрабатывание детектора из-за высокого значения *confidence*

threshold, а на правом нижнем изображении - дублирование контуров из-за чрезмерно высокого значения *non-maximum supression threshold.*

Таким образом, мы реализовали детекцию объектов на изображениях на компьютере без графического ускорителя и специализированных фреймворков с производительностью несколько кадров в секунду. Разработанные скрипты, а также демонстрационные видеофрагменты можно найти в репозитории автора.

О том, как обучить нейросеть с архитектурой *YOLOv4-tiny* под конкретную задачу, поговорим в следующей статье.