нетология

Пояснительная записка

к итоговому проекту на тему:

"Детекция дорожных объектов с применением Single Shot MultiBox Detector"

Автор: Филипп Максим Игоревич
Группа: DLL-24

Оглавление

1. Постановка задачи

Описание исходной задачи работы

Исходной задачей работы является реализация искусственной нейронной сети с использованием компьютерного зрения для классификации и детекции объектов на видео в режиме реального времени (Real-Time Object Detection). В качестве объектов детекции выбраны дорожные светофоры. Архитектура модели – SSD300.

Актуальность задачи

С развитием технологий и возрастанием интереса к автономным транспортным средствам, особенно актуально становится использование искусственного интеллекта для анализа дорожной обстановки. В настоящее время ведется активная разработка программного обеспечения для систем автономного управления транспортом, включающего в себя компьютерное зрение и нейросети.

Одной из основных задач в процессе анализа видеопотока, получаемого от различных датчиков и камер и отражающего информацию об окружающей обстановке, является обнаружение различных дорожных объектов, в том числе светофоров. Надежная и быстрая детекция подобных объектов в беспилотных автомобилях обеспечивает выполнение безопасных маневров и позволяет снизить вероятность дорожно-транспортных происшествий.

Детекторы также находят применение в области систем управления трафиком, оптимизации потока транспорта, систем контроля скорости и предупреждения столкновений.

В целом точное и эффективное обнаружение объектов на дороге в режиме реально времени имеет значительную актуальность в контексте безопасности и оптимизации дорожного движения, а также в развитии автономных транспортных систем.

Метрики качества

Метрика качества – Mean Average Precision (mAP).

Mean Average Precision (средняя площадь под кривой Precision-Recall) – стандартная метрика для оценки качества детекторов объектов. Широко используется в задачах детекции, включая детекцию дорожных объектов. При оценке детектора объектов с помощью метрики mAP выполняются следующие шаги:

- Вычисление Precision и Recall: для каждого класса объектов вычисляются значения точности (Precision) и полноты (Recall) на разных порогах бинарной классификации (то есть, определение, считать ли объект обнаруженным или нет). Precision — это доля правильно предсказанных положительных объектов от всех предсказанных положительных объектов, а Recall — это доля правильно предсказанных положительных объектов от всех реальных положительных объектов.

- Построение кривой Precision-Recall: для каждого класса строится кривая, в которой по оси X откладывается Recall, а по оси Y - Precision. Эта кривая показывает, как меняется точность при разных уровнях полноты.
- Вычисление Average Precision (AP): Average Precision вычисляется путем вычисления площади под кривой. Это значение представляет собой среднюю точность для данного класса.
- Вычисление mAP: для получения mAP вычисляются средние значения AP по всем классам объектов. Для множества классов, mAP будет средним значением AP для каждого класса (Рис.1).

$$mAP = \frac{1}{n}\sum_{k=1}^{k=n} AP_k$$
$$AP_k = \text{the AP of class } k$$
$$n = \text{the number of classes}$$

Рисунок 1. Формула mAP

2. Анализ данных

Исходные данные:

Датасет – «S2TLD_720x1280 Traffic Light Detection XML Format»

https://www.kaggle.com/datasets/sovitrath/s2tld-720x1280-traffic-light-detection-xml-format

Количество изображений в обучающей выборке – 3785 шт.

Количество изображений в тестовой выборке – 779 шт.

Количество классов – 4.

Классы – «red», «yellow», «green», «off».

Формат аннотаций – XML.

Формат представления боксов объектов – «[x_min, y_min, x_max, y_max]»

Особенности: особенностью датасета является несбалансированное распределение классов. Дисбаланс наблюдается как по суммарному количеству объектов, так и по среднему количеству объектов на одно изображение. Далее будет предложен метод калибровки модели для устранения дисбаланса.

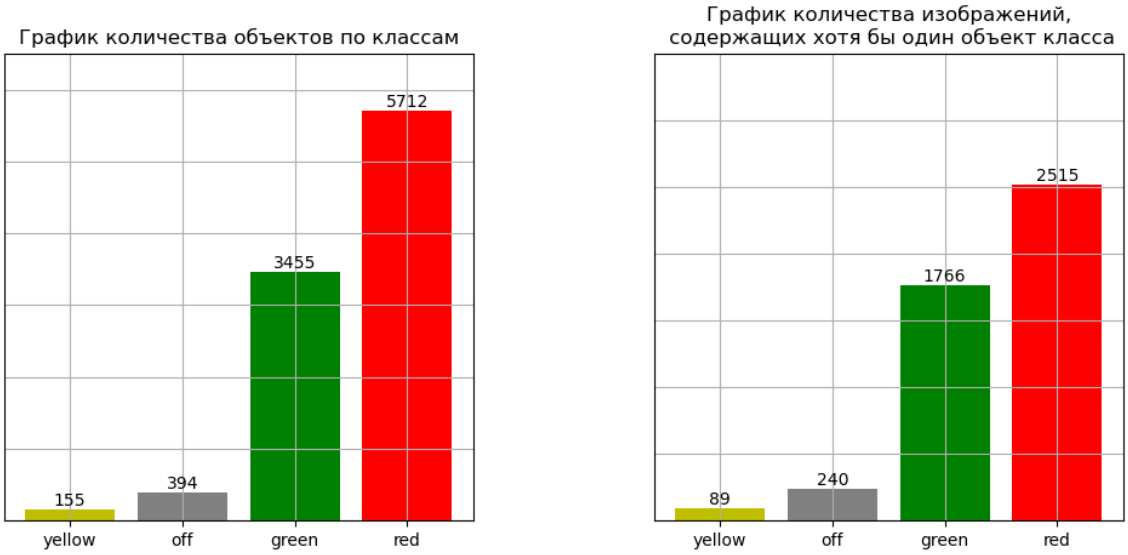

Рисунок 2. Графики распределения объектов по классам

Код для анализа данных: смотреть в ноутбуке analysis_and_inference.ipynb.

Анализ аналогичных решений

Детекция дорожных объектов с использованием сверточных нейронных сетей (CNN) — это активно развивающаяся область компьютерного зрения и автономной навигации. Существует множество архитектур и подходов, используемых для решения этой задачи:

- Faster R-CNN (Region Convolutional Neural Network): один из пионерских методов, сочетающий сверточные слои для извлечения признаков и региональную свёртку для детекции объектов. Модель генерирует предложения для областей, которые могут содержать объекты, а затем применяет свёрточные операции для определения объектов и их координат. Faster R-CNN с ResNet-50 backbone достигает примерно 30-35% mAP на COCO датасете [1].
- YOLO (You Only Look Once): делит изображение на сетку и предсказывает ограничивающие рамки и классы объектов для каждой ячейки сетки. Он известен своей высокой скоростью работы и способностью детектировать объекты в реальном времени. В зависимости от версии (например, YOLOv3), достигается примерно 28-33% mAP на COCO датасете [2].
- RetinaNet: этот метод объединяет преимущества Faster R-CNN и SSD, используя фокусировку на слабых сторонах предыдущих методов. Он использует Feature Pyramid Network (FPN) для более эффективного извлечения признаков разных масштабов и вводит новый тип якорей для борьбы с проблемой дисбаланса классов. Максимальный mAP достигает 44-45% на валидационном датасете [3].
- EfficientDet: этот метод стремится найти баланс между скоростью и точностью детекции. Он использует компаунд-масштабирование для увеличения эффективности использования вычислительных ресурсов при обучении на разных масштабах.
- Cascade R-CNN: этот метод предлагает каскадный подход к детекции, где последовательно применяются модели разной сложности для уточнения детекций. Это позволяет добиться высокой точности за счёт меньшего числа ложных срабатываний.
- CenterNet: вместо предсказания ограничивающих рамок, этот метод предсказывает центры объектов и расстояния до краёв объектов. Он также способен детектировать объекты различных форм, таких как ключевые точки или лица.
- Detectron2: фреймворк для разработки и обучения детекторов объектов, разработанный Facebook AI Research. Он предоставляет реализации многих современных методов детекции и обладает гибкой настройкой и расширяемостью [4].

По результатам анализа существующих решений в области детекции объектов, наиболее подходящим для данной задачи был выбран проект *a-PyTorch-Tutorial-to-Object-Detection*, являющийся одной из реализаций одностадийной нейронной сети SSD300 (Single Shot MultiBox Detector) [5]. Данная архитектура отлично подходит для детекции объектов в режиме реального времени за счет высокой производительности (более 25 кадров в секунду). Проект создан пользователем *sgrvinod* и является одной из лучших реализаций SSD300 с открытым исходным кодом. В данной работе вышеупомянутый проект взят в качестве основы и доработан для решения поставленных задач.

3. Методика реализации

Этап 1 (create_data_lists.py)

На первом этапе необходимо получить из исходного датасета список пар в формате «изображение – объекты на изображении». Для этого выполняется парсинг XML файла и производится запись полученных боксов и классов в JSON файлы. Параллельно записывается второй JSON файл с путями до изображений. Данная процедура применяется к обучающему и валидационному датасетам. Также формируется файл JSON, содержащий *label_map* — словарь меток-индексов, с помощью которого метки кодируются в двух предыдущих JSON. Этот словарь также доступен в *utils.py*.

Этап 2 (train.py – формирование train_dataset)

Далее создается объект класса *torch.utils.data.Dataset*. Для получения изображения и объектов в нем реализован метод __getitem__, в котором применяется ряд трансформаций, описанных далее.

Этап 2.1 (аугментации)

Базовые аугментации модуля *torchvision.transforms.functional* заменены аугментациями библиотеки *albumentations* [7]. Основная причина – удобство автоматического изменения координат bounding box-ов при геометрических трансформациях изображения.

Рисунок 3. Пример аугментации из библиотеки albumentations

Учитывая специфику датасета, аугментации выбирались по ходу экспериментов и опирались на следующие ключевые моменты:

1. Необходимость в минимальном количестве цветовых аугментаций для сохранения оригинальных цветов исходного изображения, так как цвет светофора является его основным признаком.
2. Возможность имитации съемки в движении (эффект размытия). Данный эффект был реализован с помощью аугментации *MotionBlur*. Важно установить значение аргумента *allow_shifted = False*, так как в ином случае возникают смещения bounding box-ов по отношению к детектируемым объектам на изображении.

Рисунок 4. Пример смещения рамок объектов аргументе allow_shifted = True

3. Имитация пыли, попадания грязи на объектив при съемке с использованием PixelDropout.

Рисунок 5. Применение PixelDropout

4. Имитация снега с использованием RandomSnow.

Рисунок 6. Применение RandomSnow

Более подробно со списком всех примененных аугментация можно ознакомиться в ноутбуке albumentations.ipynb.

Этап 2.2 (нормализация данных)

Поскольку используются веса предварительно обученной на датасете ImageNet сети VGG-16, для использования модели требуется предварительная обработка изображения. Список необходимых трансформация подробно описан в документации torchvision: значения пикселей должны находиться в диапазоне от нуля до единицы, а затем должна быть выполнена нормализация изображения по среднему и стандартному отклонению RGB каналов в изображениях ImageNet [6].

Этап 3 (model.py)

На данном шаге последовательно создаются составные части модели SSD300. Сначала формируется стандартная архитектура VGG-16 с пятью блоками свертки и пулинга. Полносвязные слои заменяются двумя сверточными слоями. Копируются параметры модели из *torchvision*, обученной на ImageNet. Получаем сеть, способную извлекать общие признаки из изображений.

Далее, поверх базовой сети создаем дополнительные свертки для карт признаков более высокого уровня. Получаем 4 дополнительные карты признаков. Здесь важно придать начальное значение весам. Крайне не рекомендуется инициализировать начальные веса нулями. Можно получить одинаковые фильтры и одинаковые градиенты по этим фильтрам. Получатся тождественные фильтры с одинаковыми преобразованиями. Поэтому инициализируем веса с помощью *torch.nn.init.xavier_uniform_* [8]. К этому моменту имеем 6 карт признаков: conv4_3, conv7, conv8_2, conv9_2, conv10_2 и conv11_2.

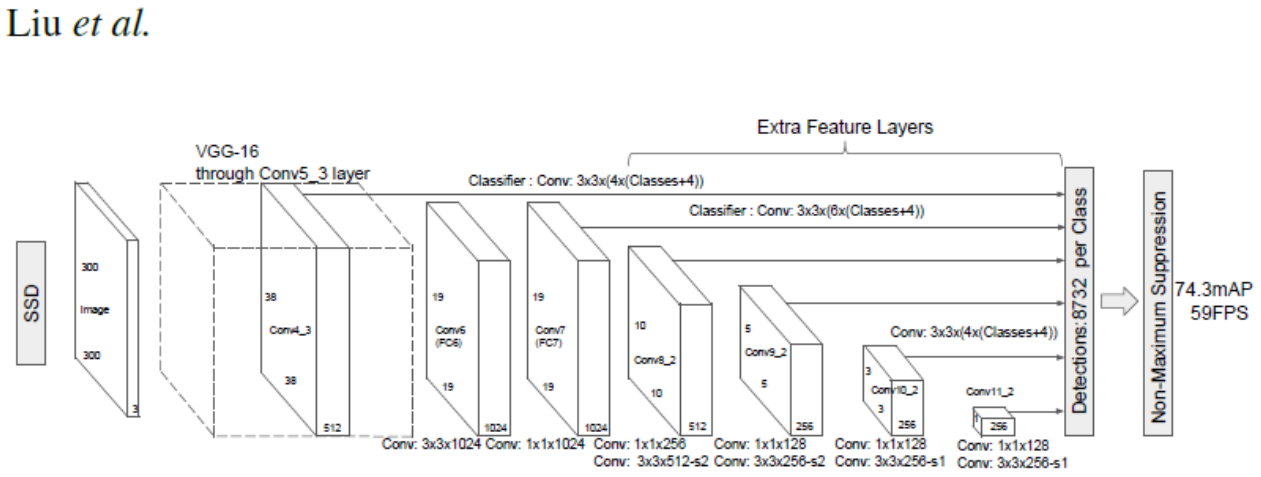

Рисунок 7. Общая схема архитектуры SSD

Для каждой ячейки каждой feature map предсказываем от 4 до 6 bounding box-ов с различным соотношением сторон. Суммарно получаем 8732 начальных рамок объектов (default boxes). Для каждого bounding box с помощью двух сверточных слоев размером 3x3, примененных к текущей feature map, предсказываем смещении координат и вероятность принадлежности к классу. Первый слой для предсказания четырех смещений координат, имеет $n_{bb} \times 4$ фильтра, второй слой, предсказывающий класс, имеет $n_{bb} \times n_{classes}$ фильтров, где n_{bb}– количество bounding box-ов в ячейке, $n_{classes}$ – количество классов, включая класс *background*.

Для каждого bounding box считаем функцию потерь MultiboxLoss. MultiboxLoss – это особая функция потерь, сочетающая в себе прогноз четырех координат смещения рамки и *n* вероятностей принадлежности к классу. Для подсчета функции потерь выполняется предварительная обработка рамок объектов:

- Вычисляется Intersection over Union (IoU) между 8732 default boxes и *N* действительными (ground truth) объектами. Это будет тензор размером 8732, N.

- Каждый из 8732 default boxes сопоставляется с объектом, с которым он имеет наибольшее IoU.
- Если полученное значение IoU более 0,5 – это положительное совпадение, если менее – рамке присваивается класс “background”.

Для устранения множества сильно перекрывающихся блоков, ссылающихся на один и тот же объект, в ходе последующей обработки выхода модели используется техника Non-Maximum Suppression (NMS). Все блоки, имеющие вероятность меньше максимальной, подавляются.

Этап 4 (detect.py)

Этап заключается в детекции объектов на единичном изображении и включает в себя следующие шаги:

- Преобразует входное изображение в тензор и нормализует его.
- Прогоняет изображение через предобученную модель SSD300 для предсказания местоположения и оценок объектов.
- Использует функцию *detect_objects* модели для обнаружения объектов на основе предсказанных местоположений и оценок.
- Преобразует координаты обнаруженных объектов в размеры оригинального изображения.
- Преобразует целочисленные метки классов в текстовые метки на основе обратного отображения *rev_label_map*.
- Создает аннотированное изображение, добавляя прямоугольники вокруг обнаруженных объектов и текст с метками классов.
- Возвращает аннотированное изображение с обозначенными объектами и классами.

Этап 5 (video_detection.py)

Данная часть кода использует библиотеки *OpenCV* и *Pillow* для обработки и визуализации видео, применяет обученную сеть для анализа и детекции объектов на каждом кадре, а также измеряет частоту кадров и включает ее в итоговый видеоряд с распознанными объектами. Основной алгоритм может быть кратко описан следующим образом:

- Открытие видеофайла с помощью *cv2.VideoCapture*, получение информации о ширине, высоте и частоте кадров видео.
- Инициализация объекта для записи обработанного видео.
- Запуск цикла обработки каждого кадра видео, включающего в себя детекцию объектов обученной моделью, измерение времени работы детекции и запись обработанного кадра в выходной видеофайл.

Также для удобства работы с видео файлами реализованы модули *youtube_downloader* и *mp4_to_gif*.

Итоговая модель

В итоговой модели используется оптимизатор SGD с начальной скоростью обучения 0,001 и momentum 0,9. Также добавлена L2 регуляризация с коэффициентом *weight_decay* = 5e-4. Кроме того, используется

планировщик скорости обучения, для снижения скорости обучения в 2 раза каждые 10 эпох. Размер батча рекомендуется размером 16, для размера батча 32 возникала проблема взрывающихся градиентов, и приходилось выполнять отбрасывание градиента, что в целом негативно сказалось на метрике модели. В кросс-энтропию, являющуюся частью функции потерь MultiboxLoss, добавлены веса для каждого класса. Эти веса рассчитываются в зависимости от имеющегося дисбаланса классов с помощью *compute_class_weight* библиотеки *sklearn*. Для оценки точности модели, в целях ускорения процесса обучения, в тестовый *DataLoader* был добавлен семплер, позволяющий оценивать модель на случайной выборке из валидационного датасета. Такой способ значительно ускоряет время обучения и при использовании статистически значимого объема выборки не слишком проигрывает в точности оценки.

4. Итоги обучения модели

По результатам обучения 50 эпох, лучшая модель достигла mAP = 0.35. Стоит отметить, что данная метрика считает среднее по классам, а метрики по отдельным классам получились в соответствии с распределением объектов по классам: red = 0.67, green = 0.52, yellow = 0.08, off = 0.13. Модель лучше обучилась распознавать те объекты, которые больше представлены в обучающей выборке.

По итогам применения модели к видеоряду, можно сделать вывод о том, что в целом, модель выучила основные паттерны, для выявления светофоров на изображении. Однако остается еще много возможностей для улучшения модели. Например, можно заметить ложные срабатывания, когда модель путает желтый свет от фонарей при ночной съемке и желтый сигнал светофора. Также наблюдаются ложные срабатывания на обратною сторону светофора – модель помечает такие объекты как выключенный светофор. Это может говорить о возможном недообучении модели. Возможные варианты решения данной проблемы предложены далее.

Ознакомиться с результатами работы модели можно в репозитории, в папке *outputs*.

5. Выводы

В ходе проведенной работы была получена модель детекции дорожных объектов. Данная модель основана на архитектуре Single Shot MultiBox Detector с backbone VGG-16, предобученной на датасете ImageNet. Целевая метрика Mean Average Precision на тестовой выборке составила 0,35.

Для дальнейшей настройки модели рекомендуется:

- Увеличить размер обучающей выборки. 3785 изображений все-таки довольно маленький объем данных для обучения нейронной сети, даже с учетом применения аугментаций;
- Проводить валидацию на полном объеме тестовых данных, без использования семплирования.
- Проводить обучение на изображениях с более высоким разрешением;
- Разнообразить изображения в обучающей выборке, добавить объекты с разными масштабами, относительно размеров изображения;
- Подобрать и протестировать альтернативные параметры регуляризации, скорости обучения, величины ограничения градиентов;
- Обучить модель на большем количестве эпох.

В целом, архитектура SSD хорошо подходит для детекции дорожных объектов, включая светофоры, в режиме реального времени, однако представляет собой довольно сложную и многогранную задачу, требующую глубоких знаний в области компьютерного зрения и машинного обучения, а также оптимизации и настройки модели для каждого конкретного сценария применения.

Список источников:

1. Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks." NeurIPS 2015.
2. Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. "You Only Look Once: Unified, Real-Time Object Detection." CVPR 2016.
3. Traffic Light Detection Using RetinaNet and PyTorch. https://debuggercafe.com/traffic-light-detection-using-retinanet/
4. FAIR (Facebook AI Research). https://github.com/facebookresearch/detectron2
5. https://github.com/sgrvinod/a-PyTorch-Tutorial-to-Object-Detection
6. https://pytorch.org/vision/stable/models/generated/torchvision.models.vgg16.html
7. https://albumentations.ai/
8. Understanding the difficulty of training deep feedforward neural networks - Glorot, X. & Bengio, Y. (2010)