Обучение и оценка качества детектора автомобилей

Андрей Бабичев, Борис Фаизов, Влад Шахуро,
Владимир Гузов, Иван Карпухин

В данном задании предлагается реализовать простой нейросетевой детектор машин на основе полносверточной нейросети, визуализировать результаты работы детектора, а также посчитать метрики качества и реализовать подавление множественных обнаружений. Для удобства выполнения задание разделено на пункты. Максимальная оценка за задание — 10 баллов.

1. Простой нейросетевой классификатор (2 балла)

Для начала обучим простой бинарный нейросетевой классификатор машин. В папке с первым тестом находится набор тренировочных изображений машин и фона.

Напишите функцию `get_cls_model(input_shape)`, которая будет возвращать нейросетевую модель. На вход функция принимает размер изображения (в данном случае `(40, 100, 1)`). Также напишите функцию `fit_cls_model(X, y)`, которая будет принимать на вход четырехмерный тензор с картинками и тензор меток классов, а также обучать и возвращать нейросетевую модель. Функция обучения не должна записывать модель в файл, т.к. в проверяющей системе папка с решением монтируется только для чтения. При обучении можно использовать простую аугментацию данных с помощью `torchvision.transforms`. Проверьте обучение классификатора с помощью теста:

```
$ ./run.py unittest classifier
```

При запуске тест загружает обучающую выборку, делит ее в отношении 3:1, обучает классификатор на первой части и проверяет точность на второй части. Точность классификатора на второй части должна быть не меньше 90%. Сохраните веса обученного классификатора в файл `classifier_model.pth`. Его нужно будет сдать в проверяющую систему.

2. Простой нейросетевой извлекатель тепловой карты

Теперь на основе обученного нейросетевого классификатора построим нейросетевой детектор, работающий методом скользящего окна. Для этого превратим классификатор в полносверточную нейросеть, которая выдает карту уверенности в том, что в фиксированной области изображения находится объект. Чтобы превратить классификатор в полносверточную нейросеть, заменим полносвязные слои свертками:

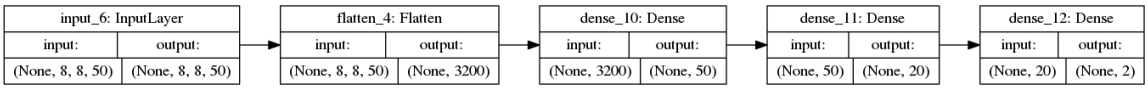
1. Рассмотрим последовательность `Flatten` и `Linear` слоев. Пусть слой `Flatten` принимает на вход тензор размера $H \times W \times C_{in}$, вытягивает его в вектор-столбец, а затем передает `Linear` слою, который имеет C_{out} выходных значений. Два этих слоя можно заменить на C_{out} сверток с ядром размера $H \times W$.

2. Рассмотрим одиночный `Linear` слой, который принимает на вход вектор из C_{in} чисел и выдает вектор из C_{out} чисел. Его можно заменить на C_{out} сверток с ядром 1×1.

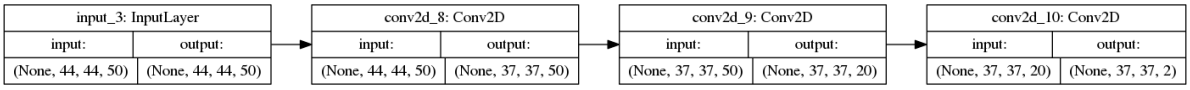
Обратите внимание, что при двух указанных преобразованиях не меняется ни количество параметров, ни выполняемые операции. Изменяется только размерность тензоров, количество элементов в тензорах остается прежним. При замене полносвязных слоев на сверточные достаточно скопировать веса слоев, изменив их размер с помощью функции `reshape`.

Приводим пример замены полносвязных слоев нейросети с $H = W = 8$, $C_{in} = 50$ перед `Flatten` слоем. На этом изображении отображается только преобразование последних полносвязных слоев. `InputLayer` должен быть заменен выходом сверточной части.

До замены:



После замены нейросеть можно применять к изображениям большего размера. Размер сверточных признаков увеличился до $H = W = 44$:



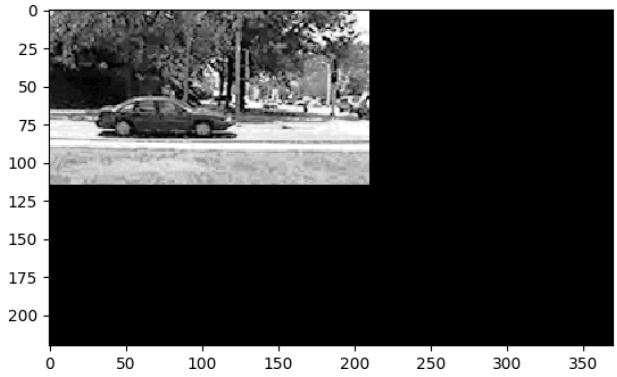
После замены всех полносвязных слоев на сверточные мы получим полносверточную сеть. Если размер входного изображения равен размеру входного изображения сети для классификации, то выходные данные будут иметь размер 1×1. Но если мы увеличим размер входного изображения, то размер выходного изображения автоматически увеличится. Например, если шаг свертки предыдущей сети составляет 4 пикселя, увеличение размера входного изображения на 100 пикселей вдоль оси приводит к увеличению размера выходного изображения на 25 пикселя вдоль оси. Применяя полносверточную нейросеть, мы получаем карту активации классификатора без вырезания фрагментов изображения и без выполняли рассчетов одних и тех же признаков на пересекающихся областях изображения.

Последний полносвязный слой с функцией активации softmax можно заменить на сверточный слой с линейной активацией. Чтобы разделить классы, нужно будет найти хороший порог для такой линейной функции.

Напишите функцию `get_detection_model(cls_model)`, которая принимает на вход обученную модель для классификации машин и создает эквивалентную ей модель для детектирования машин путем замены полносвязных слоев на сверточные. Веса модели можно скопировать используя метод `parameters`, предварительно приведя их к нужной форме. Тестовые изображения для детектора находятся во втором тесте. Прочитать их можно с помощью функции `read_for_detection(img_dir, gt_path)`.

3. Простой нейросетевой детектор

По тепловой карте полносверточного классификатора для изображений можно получить прямоугольники, ограничивающие найденные объекты. Заметим, что все машины в тестовой выборке одинакового размера, поэтому пирамиду разрешений строить не нужно. Однако для того, чтобы подать все тестовые изображения сразу в нейросеть, можно привести их к общему размеру `(220, 370)` (это максимальный размер изображения в тестовой выборке) путем дополнения нулями до правого нижнего угла изображения:



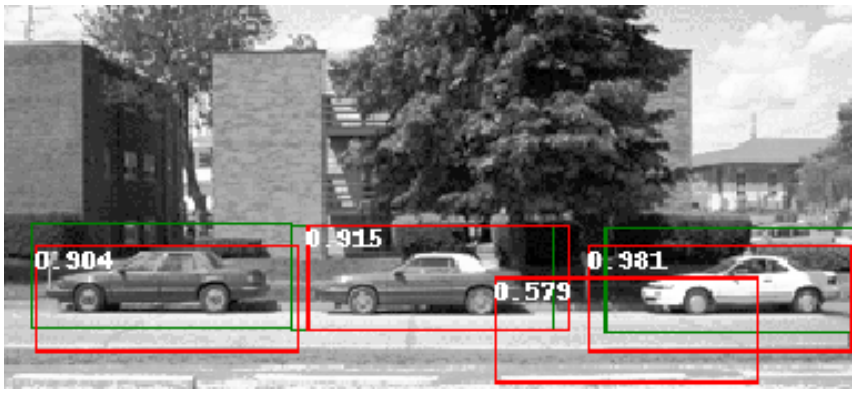
Каждая точка полученный тепловой карты соответствует прямоугольной области на изображении. Чтобы получить достаточно точный детектор, нужно подобрать общий для всех изображений порог,

по которому мы будем оставлять прямоугольники, в которых классификатор наиболее уверен. Также нужно отсеять прямоугольники, выходящие за границу изображения.

Напишите функцию `get_detections(detection_model, dictionary_of_images)`, которая на вход принимает модель обученного детектора и словарь изображений для тестирования. На выход она должна возвращать словарь из детекций. Качество детектора будет проверяться в одном из следующих пунктов.

4. Визуализация работы

Теперь визуализируйте результаты работы детектора и разметку изображения. При визуализации отсеивайте неуверенные ответы по фиксированному порогу, чтобы не засорять изображение. Для визуализации можно использовать библиотеку Pillow.



Пример визуализации разметки и обнаружений детектора

5. Мера IoU (1 балл)

Для подсчета метрики качества детектора нужно уметь сравнивать два прямоугольника по мере IoU (отношение площади пересечения к площади объединения). Напишите функцию `calc_iou(first_bbox, second_bbox)`, принимающую на вход два прямоугольника и возвращающую одно число — меру близости по IoU. Проверьте реализацию с помощью теста:

```
$ ./run.py unittest iou
```

6. Построение Precision-Recall кривой и подсчет AUC (2 балла)

На этапе тестирования нейросетевой детектор для каждого окна возвращает число — меру уверенности в том, что в данном окне находится объект. Обнаружениями будем считать окна, мера уверенности которых больше некоторого порога. Изменяя порог, мы можем получать детекторы с различными характеристиками — точностью и полнотой обнаружений. Множество детекторов, отличающихся только выбором порога, назовем семейством детекторов.

Для оценки качества семейства детекторов обычно строят кривую precision-recall. Опишем эффективную процедуру подсчета значений кривой:

1. Для каждого изображения нужно составить список из `tp` (true positive) и `fp` (false positive) обнаружений. Для этого нужно:

   (a) Отсортировать обнаружения в порядке убывания соответствующих мер уверенности классификатора. Подготовить список `gt` прямоугольников из разметки данного изображения.

(b) Для каждого обнаружения найти соответствующий ему прямоугольник из разметки, для которого мера IoU максимальна и $\geqslant$ `iou_thr` (в данном задании `iou_thr = 0.5`).

(c) Если такой прямоугольник найден, то добавить обнаружение в `tp`, иначе — в `fp`.

(d) Чтобы не сопоставлять один и тот же прямоугольник из разметки двум обнаружениями детектора, после добавления обнаружения в `tp` соответствующий ему прямоугольник нужно удалить из `gt`.

2. Объединим списки `tp` и `fp` всех изображений.

3. Теперь нам нужно два списка — все обнаружения (объединение `tp` и `fp`) и все `tp`. Сортируем эти списки по возрастанию мер уверенности классификатора.

4. Теперь пройдем по списку всех обнаружений. Пусть сейчас рассматривается обнаружение с мерой уверенности c. Найдем количество всех обнаружений с мерой уверенности $\geqslant c$ и количество `tp` обнаружений с уверенностью $\geqslant c$.

5. Имея полученные данные и общее количество прямоугольников в разметке, рассчитаем *recall* и *precision* для каждого порога уверенности c.

6. Сохраним полученные тройки (*recall*, *precision*, c) в общий список.

В итоге мы получим набор точек (*recall*, *precision*), которые задают кривую precision-recall. Кривые для разных детекторов можно сравнивать как визуально, так и с помощью интегральной метрики качества — площади под кривой (Area Under Curve, AUC). Для расчета AUC достаточно посчитать площади всех трапеций, образованных осью абсцисс, и прямыми, проходящими через точки PR-кривой.

Реализуйте функцию `calc_auc(pred_bboxes, gt_bboxes)` подсчета AUC, которая на вход принимает словарь из детекций, полученных в пункте 2 и словарь из ground truth детекций, а на выход дает значение метрики AUC. Пользоваться библиотечными функциями для подсчета AUC в данном пункте нельзя. Проверьте функцию с помощью теста:

```
$ ./run.py unittest auc
```

Кроме AUC часто используется другая интегральная метрика — Average Precision, AP. В этой метрике считается средняя точность в точках $recall = \{0, 0.1, \ldots, 0.9, 1\}$. Значения точности при этом интерполируются по соседним значениям.

7. Подавление немаксимумов (2 балла)

Обычно один и тот же объект находится детектором в нескольких близких окнах. Для того, чтобы оставить одно обнаружение, используется алгоритм подавления немаксимумов (non-maximum suppression, NMS). Простейший вариант алгоритма работает следующим образом:

1. Обнаружения сортируются по убыванию меры уверенности.

2. Для каждого обнаружения удаляются все следующие за ним обнаружения (те, у которых мера уверенности меньше), которые пересекаются с данным по мере IoU больше, чем на t. Здесь $0.3 \leqslant t \leqslant 0.7$ — фиксированный порог.

Реализуйте функцию `nms(detections_dictionary, iou_thr)`, которая на вход получает словарь с детекциями для каждого изображения и порог IoU, при котором детекции нужно считать совпадающими. На выходе функция должна возвращать словарь детекций, над которым был применен алгоритм подавления немаксимумов. Проверьте функцию с помощью теста:

```
$ ./run.py unittest nms
```

8. Проверка качества работы детектора (3 балла)

Проверьте, что AUC детектора до подавления немаксимумов не меньше 0.2:
`$ ./run.py unittest detector`
Теперь проверим, что AUC детектора с подавлением немаксимумов не меньше, чем 0.95:
`$ ./run.py unittest detector_nms`
Попробуйте улучшить качество, меняя порог в алгоритме подавления немаксимумов NMS. Подобранный порог сделайте значением по умолчанию второго аргумента функции `nms`.

9. Визуализация обнаружений детектора и PR-кривых

Визуализируйте три изображения с обнаружениями детектора до и после алгоритма подавления немаксимумов вместе с разметкой объектов. Нарисуйте график с PR-кривыми до и после подавления немаксимумов. График должен быть понятным: подпишите оси, настройте сетку, в легенду добавьте AUC.