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 (fully convolutional) 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 Реализация примитивного детектора

1.1 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`, которая будет создавать модель классификатора. Архитектуру модели необходимо подобрать так, чтобы вход и **рецептивное поле** модели были размера 40×100. Для перехода к полносвязной части архитектуры используйте `Flatten` (не `Global Pooling`).

В сверточной части архитектуры, постарайтесь избегать граничные эффекты, особенно в последних слоях и слоях со страйдами > 1. На выходе у модели должны быть логиты для двух классов (нулевой логит – фон, первый логит – машина).

Также напишите функцию `fit_cls_model`, которая будет принимать на вход четырехмерный тензор с картинками и тензор меток классов, а также флаг `fast_train`. Данная функция должна создать, обучить и вернуть модель классификатора. При обучении не используйте аугментации совершающие геометрические преобразования входных изображений.

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 в режиме `fast_train=True` и проверяет точность на второй части. Точность этого быстро обученного классификатора на второй части должна быть не меньше 90%.

Самостоятельно обучите классификатор в режиме `fast_train=False` на вашей машине. Можете использовать для этого все данные из `train_data.npz`. Сохраните полученные параметры в файл `classifier_model.pt`, его нужно будет сдать в проверяющую систему.

1.2 Простой нейросетевой извлекатель тепловой карты

Теперь на основе обученного нейросетевого классификатора построим примитивный 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 `Flatten` слой и следующий за ним `Linear` слой.

   Пусть слой `Flatten` принимает на вход тензор размера $C_{in} \times H \times W$, вытягивает его в вектор-столбец, а затем передает `Linear` слою, который имеет C_{out} выходных значений. Два этих слоя можно заменить на один `Conv2d` слой с C_{out} выходными каналами и ядром размера $H \times W$.

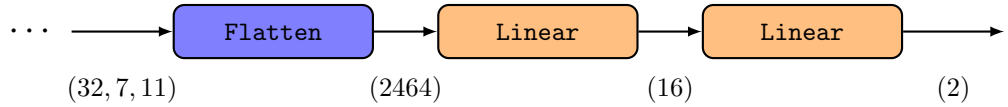
2. Все остальные `Linear` слои можно заменить на `Conv2d` слои с таким же размерами C_{in}/C_{out} и ядрами размера 1×1.

Обратите внимание, что при двух указанных преобразованиях не меняется ни количество параметров, ни выполняемые операции. Просто размерность всех вычислений меняется с C на $C \times 1 \times 1$. Таким образом, можно взять параметры для новых сверточных слоев из соответствующих полносвязных слоев и результаты вычислений (для изображений 40×100) при этом не изменятся.

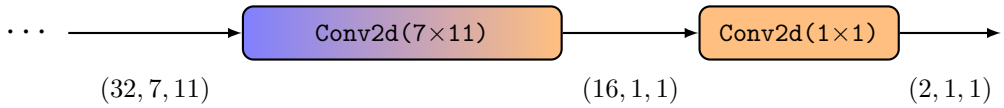
Пример

Предположим, что выбранная архитектура принимает изображения размера 40×100 и затем используя сверточные и пулинг слои понижает разрешение до размера 7×11 (с 32 каналами). После этого, применяется `Flatten` слой и два линейных слоя – с 16 и 2 выходными нейронами соответственно.

То есть последние слои в нейросети до конвертации выглядят так:



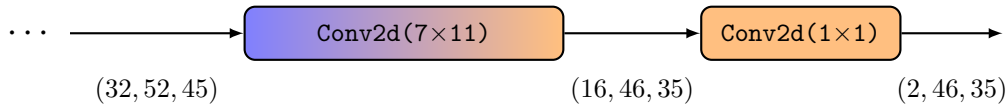
Применив выше описанный алгоритм, заменим все полносвязные слои на сверточные. В результате, получим полносверточную архитектуру, которая при обработке изображений размера 40×100 возвращает тензор размера $2 \times 1 \times 1$. При этом, значения в выходном тензоре будут совпадать с результатами работы исходной сети с полносвязными слоями.



Тепловая карта

В отличие от архитектур с `Flatten` слоем, полносверточные (fully convolutional) сети в теории способны обрабатывать изображения произвольных размеров. В таких сетях, увеличение размера входного изображения приводит к увеличению размера выходного изображения.

Например, в выше приведенном примере, сверточная часть сети-классификатора имела внутренний страйд 4×8. Тогда, увеличив размер входного изображения на 180×272 (до 220×372), мы также увеличим карты признаков в последних слоях нейросети на $\frac{180}{4} \times \frac{272}{8} = 45 \times 34$.



При условии отсутствия граничных эффектов в исходной сети-классификаторе, каждый пиксель в выходном изображении будет эквивалентен вычислению исходной сети-классификатора на фрагменте исходного изображения соответствующем рецептивному полю данного пикселя.

Как архитектура сети и размер входного изображения влияют на размер выходного изображения?

Как положение рецептивного поля зависит от положения пикселя в выходном изображении?

Как граничные эффекты в различных слоях в исходной сети-классификаторе могут влиять на результаты работы полносверточного детектора, полученного предложенным алгоритмом конвертации?

Таким образом, используя полносверточную нейросеть, мы получаем результаты применения классификатора как бегущего окна по бóльшему изображению. При этом, мы не тратим ресурсы на вырезание фрагментов изображения и не дублируем рассчет одних и тех же признаков на пересекающихся областях изображения.

Применив к полученному тензору размерности $2 \times 46 \times 35$ функцию `Softmax` (или `LogSoftmax`) и взяв последний элемент вдоль первой оси, мы получим карту вероятностей (или нормализованных логитов вероятностей) наличия автомобиля в различных позициях на исходном изображении.

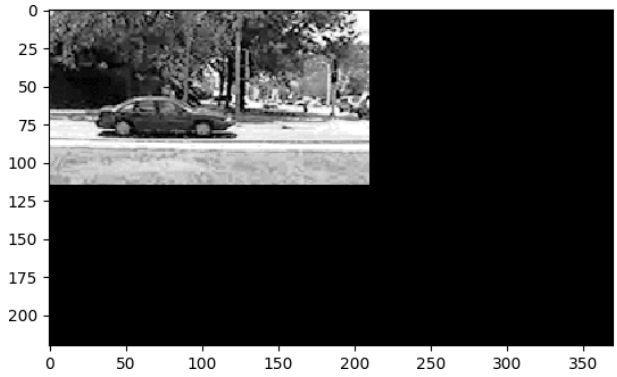
Реализация

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. Не забудьте скопировать (и сконвертировать) веса модели. Проверьте корректность вашей конвертации, сравнив результаты работы классификатора и детектора на изображениях 40×100.

Проверьте отсутствие граничных эффектов в исходной модели, сравнив результаты работы детектора на бóльших изображениях с результатами работы классификатора на соответствующих 40×100 фрагментах этих изображений. Большие изображения для тестирования детектора можно прочитать с помощью функции `read_for_detection(img_dir, gt_path)`, используя данные в папке `tests/03_unittest_detector_input`. Также, визуализируйте карты вероятностей детектора, на этих изображениях.

1.3 Простой нейросетевой детектор

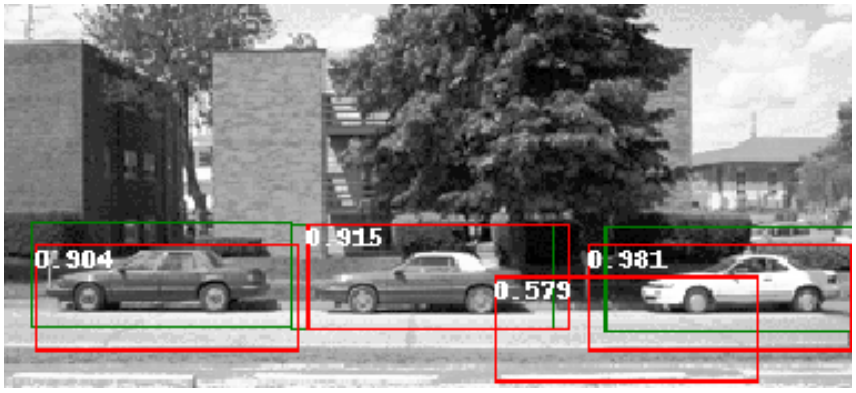
По тепловой карте полносверточного классификатора для изображений можно получить прямоугольники, ограничивающие найденные объекты. Заметим, что все машины в тестовой выборке одинакового размера, поэтому пирамиду разрешений строить не нужно. Однако для того, чтобы подать все тестовые изображения сразу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



Каждая точка полученной тепловой карты соответствует прямоугольной области на изображении. Чтобы получить достаточно точный детектор, нужно подобрать общий для всех изображений порог, по которому мы будем выбирать прямоугольники, для которых классификатор “выдвигает гипотезу” о наличии в них машины.

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.

После этого, 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` или `matplotlib`.



Пример визуализации разметки и обнаружений детектора

2 Реализация метрик

2.1 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
```

2.2 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.

3 Результаты работы детектора

3.1 Качество работы детектора (2 балла)

Проверьте, что AUC базового алгоритма детектора не меньше 20%:

```
$ ./run.py unittest detector
```

3.2 Подавление немаксимумов (2 балла)

Обычно один и тот же объект находится нашим примитивным 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 t – фиксированный порог, оптимальное значение которого вам необходимо подобрать.

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
```

3.3 Качество работы детектора с NMS (1 балл)

Теперь проверим, что AUC детектора с подавлением немаксимумов не меньше 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`.

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. Сделайте график понятным: подпишите оси, настройте сетку, в легенду добавьте AUC.