Классификация и локализация кошек и собак на изображениях

Задачу классификации (кошка или собака на изображении) и локализации (bounding box головы животного) решал двумя способами:

- создание собственной архитектуры CNN;
- использование техники transfer learning.

Решение реализовано в jupyter notebook на python 3 с использованием сервиса Google Colab (на графической карте Tesla T4). Каждый этап (подключение модулей, формирование датасета и т. п.) прокомментирован в ноутбуке, поэтому в данном описании остановимся на основных моментах решения задачи.

Способ 1 — собственная архитектура нейронной сети

Исходный тренировочный датасет (папка **train**) разбит на две части: тренировочную (**train** - 90% от исходного) и валидационную (**valid** - 10% от исходного), а изображения из исходной папки **valid** используются для тестирования.

Производил два варианта обучения собственной сети на двух датасетах:

- исходном;
- увеличенном в 4 раза за счет аугментаций изображений исходного датасета (использовал библиотеку imgaug).

Из результатов обучения модели на исходном датасете видно, что точность на тренировочных данных близка к 100%, следовательно происходит переобучение. Варьированием коэффициента регуляризации этот эффект устранить не удалось, поэтому для увеличения обобщающей способности сети датасет был увеличен за счет аугментированных изображений.

Смысл создания собственной небольшой сети в том, что возможно ее мощности будет достаточно для удовлетворительных результатов и

соответственно она будет, вычислительно, более быстрой. Структура сети и гиперпараметры выбраны на основе части сети resnet18, которую будем использовать в transfer learning.

Структура сети:

- сверточный слой с 32 нейронами (свертка = 7, padding=3);
- слой нормализации батчей;
- слой с нелинейной функцией Relu;
- слой с функцией MaxPool по 4 элементам;

- сверточный слой с 64 нейронами (свертка = 3, padding=1);
- такой же набор слоев, что и выше;

- сверточный слой с 128 нейронами (свертка = 3, padding=1);
- слой нормализации батчей;
- слой с нелинейной функцией Relu;
- слой с функцией MaxPool по 2 элементам.

- Выпрямляющий слой Flatten;
- Модифицированный линейный слой:

    * один линейный слой с входом от Flatten и двумя выходами (для классификации);

* один линейный слой с входом от Flatten и 500 выходами, нелинейный слой Relu, линейный слой с 500 входами и 4 выходами (для регрессии координат bounding box).

Гиперпараметры следующие:

- *функция потерь (для классификация)* — CrossEntropyLoss;
- *функция потерь (для регрессии) — MSELoss*;

- метод оптимизации Adam, с начальным коэффициентом обучения lr= 0,001 и коэффициентом регуляризации weight_decay=0,01;
- метод learning rate annealing — ReduceLROnPlateau;
- размер минибатча — 64 элемента;
- количество эпох = 30.

Результаты (на тестовых данных):

- <u>без</u> аугментаций:

mioU **32,8**%, classification accuracy **80,2**%, 0,26 ms, 2687 train, 298 valid, 400 test.

- <u>с</u> аугментациями:

mioU **34**%, classification accuracy **90**%, 0,26 ms, 10748 train, 298 valid, 400 test.

Варьирование архитектуры сети в тех же масштабах, а также варьирование гиперпараметров и методов оптимизации не принесли точность на тесте выше 90% и mioU выше 34%.

<u>Способ 2 — transfer learning</u>

Теперь попробуем воспользоваться техникой transfer learning. Для этого возьмем CNN — Resnet18, предобученную на датасете ImageNet. В связи с тем, что мы берем модель с предобученными весами, следовательно, она должна сойтись к хорошей точности довольно быстро.

1) <u>Попробуем сначала *заморозить все слои кроме последнего*, заменим его на новый и обучим (слой описан в способе 1)</u>.

Гиперпараметры следующие*:*

- *функция потерь (для классификация)* — CrossEntropyLoss;
- *функция потерь (для регрессии) — MSELoss*;
- метод оптимизации Adam, с начальным коэффициентом обучения lr= 0,01 ;
- метод learning rate annealing — ReduceLROnPlateau;

- размер минибатча — 64 элемента;
- количество эпох = 20.

Результаты (на тестовых данных):

- <u>без</u> аугментаций:

mioU **36,4**%, classification accuracy **97,7**%, 1,16 ms, 2687 train, 298 valid, 400 test.

- <u>с</u> аугментациями:

mioU **33,7**%, classification accuracy **98,7**%, 1,16 ms, 10748 train, 298 valid, 400 test.

Варьированием гиперпараметров не удалось пробить потолок *mioU* в 36,4%, поэтому попробуем следующий вариант обучения.

2) <u>Теперь попробуем обучить *всю сеть целиком*</u>.

Также заменим последний слой и обучим всю модель. При этом мы обучаем модель не с нуля, а после тренировки на ImageNet, поэтому она должна сойтись к хорошей точности довольно быстро.

Гиперпараметры следующие:

- *функция потерь (для классификация)* — CrossEntropyLoss;
- *функция потерь (для регрессии) — MSELoss;*
- метод оптимизации Adam с начальным коэффициентом обучения lr= 0,0001 и коэффициентом регуляризации weight_decay=0,01;
- метод learning rate annealing — ReduceLROnPlateau;
- размер минибатча — 64 элемента;
- количество эпох = 30.

Результаты (на тестовых данных):

- <u>без</u> аугментаций:

mioU **56**%, classification accuracy **98,2**%, 1,16 ms, 2687 train, 298 valid, 400 test.

- <u>с</u> аугментациями:

mioU **59,2**%, classification accuracy **99**%, 1,16 ms, 10748 train, 298 valid, 400 test.

Таблица с общими результатами

	Моя модель		**ResNet-18** (обучен последний слой)		**ResNet-18** (обучены все слои)	
	исходный датасет	*расширенный* датасет	*исходный* датасет	*расширенный* датасет	*исходный* датасет	*расширенный* датасет
Accuracy, % (точность)	*80,2*	*90*	97,7	*98,7*	*98,2*	***99***
mioU, %	*32,8*	*34*	36,4	*33,7*	*56*	***59,2***
Time, ms	*0,26*	*0,26*	1,16	*1,16*	*1,16*	*1,16*

Запуск эксперимента

Обучение сети производилось на GPU, поэтому рекомендуется использовать графический процессор.

Есть два файла jupyter notebook:

- Neural Network - обучение на исходном датасете;

- Neural Network(augmentation) - обучение на расширенном датасете.

В ноутбуке с расширенным датасетом есть раздел подготовки данных, его запускать не обязательно, т. к. данные уже подготовлены и упакованы в .zip, ссылки для скачивания этих файлов будут указаны в конце этого описания.

Тестирование модели рекомендуется производить в Google Colab (для этого в ноутбуках все настроено).

Необходимо создать папку ClassReg в корневом каталоге google drive и поместить в нее следующие файлы:

- два файла jupyter notebook:*Neural Network*, *Neural Network(augmentation)*;
- каталог *modules* с вложенными файлами;
- каталог *best_models_save* с вложенными файлами;
- три .zip файла: *cats_dogs_dataset*, *dataset*, *augmentation*.

Для повторения экспериментов в ноутбуке с кодом, следует поочередно запускать ячейки (запуск некоторых ячеек прокомментирован в следующем абзаце), помимо комментариев в коде, указаны заголовки ячеек, в которых описано, что происходит в данной ячейке. Все классы и функции находятся в модулях с названиями, соответствующими назначению классов и функций.

Вычисление среднего и стандартного отклонения уже произведено, так что эту ячейку можно запускать, если нужно проверить работоспособность кода. Также в папке проекта находятся файлы с моделями, поэтому можно не проводя повторного обучения проверить их на тестовых данных.

Ссылка для скачивания датасетов:

https://disk.yandex.ru/d/NWhwdn7HevFkAw