Привязка аэроснимков к местности

Презентация решения

Основные идеи

Был избран нестандартный путь. Вместо сети, обученной на огромном числе снимков, нарезанных из подложки, использовалась простейшая сверточная сеть PyTorch *без обучения*. В качестве весов ядер использовались сами снимки (с некоторой предобработкой). Использовалось свойство ядер сверточных сетей: если паттерн входного изображения совпадает с ядром, свертка с ним дает максимальный результат. Сеть содержит всего один сверточный слой состоящий из 400 ядер (по числу снимков) На вход подается подложка, на выходе получается 400 двумерных массивов. В массиве определяется максимум, он и соответствует положению центра снимка на подложке. С учетом того, что ориентация снимка может быть произвольной, пришлось перебрать все 360 возможных ориентаций подложки.

Детали реализации

С целью уменьшения объема данных все изображения были переведены в черно-белые. Кроме того, из-за ограничения по памяти, изображения были уменьшены в 4 раза.

Была вычислена автокорреляционная функция подложки и оказалось, что изображение имеет эффективное разрешение 3-4 пикселя, поэтому ресайз изображений в 4 раза не должен был сильно сказаться на результатах.

Далее, изображения были нормированы на среднее 0 и стандартное отклонение 1.

Преодоление трудностей

Выяснилось, что на ядрах размерности 256x256 результат не всегда получался удовлетворительным. Проблему создавали облака и аэродром в углу подложки - последний как бы притягивал к себе все снимки с облаками. Для решения этой проблемы ядро было поделено на сетку из квадратов 8x8 (так что таких квадратов было 32x32 = 1024)

Изображение в каждом квадрате было нормировано на среднее 0 и стандартное отклонение 1. Кроме того, был предложен простой алгоритм выявления облака в квадрате. Квадраты с облаками зануялись и не давали случайного вклада в свертку.

Данный подход позволил довести точность на снимках без облаков практически до 1, однако оставались некоторые сложности со снимками, обильно покрытыми облаками.

Воспроизведение вычислений

Все вычисления проводились на платформе kaggle. Один прогон (перебор всех 360 ориентаций подложки для 400 снимков) занимает около 4 часов (при условии использования GPU) Все результаты полностью воспроизводимы, поскольку нигде не используются случайные числа. Поскольку загрузка 2.7Gb данных занимает довольно много времени, я создал публичный датасет с названием “MPTI-champ” - его можно найти поиском и подключить к ноутбуку.

Для удобства запуска все ячейки ноутбука объединены в одну. При необходимости можно поменять параметры предобработки и задать режим Train | Test. В результате прогона в каталоге `submit` создаются JSON-файлы для каждого снимка. Эти файлы архивируются в zip.файл и могут быть загружены.

Воспроизведение вычислений

На кэгле был получен результат 0.968012, на снимках уменьшенных 1:4. При этом на снимках без облаков точность была 0.999+. Для получения более высокого результата вычисления были повторены на сервере с картой RTX-3090 24Gb. Брались снимки 1:2, из-за ограничений по памяти обработка велась батчами по 100 снимков. Прогон занял 38 часов, получена точность 0.980951

Исходный код

Исходный текст финального ноутбука может быть скачан из репозитория:

https://github.com/boangri/champ-mpti

Преимущества решения

Предложенный подход обладает тем преимуществом перед традиционным (когда на вход сети подается снимок, а не подложка), что при переходе к другой подложке не требуется переобучение сети. Замена/добавление ядра под новый снимок занимает секунды, но основную задержку вносит вычисление повернутых подложек. Предложенное решение эффективно для пакетной обработки (сотни) снимков для разных подложек. Однако для работы с одиночными снимками и фиксированной подложкой традиционное решение может оказаться удобнее.

Контактные данные

Борис Андреевич Грибовский

Email: xinu@yandex.ru

Github: https://github.com/boangri

Telegram: @boangri

Phone: +7 910 408 1189