Поиск ключевых точек на лице

Азамат Мифтахов, Влад Шахуро, Владимир Гузов, Даниил Киреев

Обзор задания

В данном задании предлагается реализовать регрессию ключевых точек лица с использованием сверточной нейронной сети. Перед тем как приступать к обучению, **внимательно** прочтите весь текст задания целиком.

Базовая часть

Обучение и тестирование

Для обучения алгоритма выдается публичная выборка лиц с размеченными ключевыми точками. Разрешается использовать только эти данные для обучения. Для реализации необходимо использовать библиотеку `PyTorch`.

Нельзя использовать какие-либо другие данные. **Нельзя** использовать библиотеки с готовыми реализациями архитектур моделей. **Нельзя** использовать более высокоуровневые библиотеки для обучения нейросетей (такие как `Lightning`).

То есть, от вас требуется самостоятельно реализовать код для обучения нейросети используя только функционал `PyTorch` и самостоятельно провести эксперименты по дизайну нейросетевых архитектур (выбору слоев, подбору их гиперпараметров и т.п.).

Метрикой качества является среднеквадратичная ошибка:

$$Q(X, Y) = \frac{1}{n} \sum_{i=1}^{n} (f(x_i) - y_i)^2,$$

где $f(x)$ – реализованный вами метод для поиска ключевых точек лица на изображении.

Для самостоятельной оценки качества модели, рекомендуется выделить из публичной выборки валидационный набор данных, который не будет участвовать в обучении. Качество на валидационной выборке стоит замерять периодически, через фиксированное количество итераций или эпох. Если качество модели на тренировочной выборке значительно лучше, чем на валидационной, это может говорить о переобучении.

Обратите внимание, что изображения как в тренировочной, так и в тестовой выборке могут быть черно-белыми, а не только цветными. Эту ситуацию нужно корректно обрабатывать (например, делать из одноканального изображения трехканальное путем дублирования канала).

Архитектура нейронной сети

На вход нейронной сети подается трехканальное изображение фиксированного размера. Размер входного слоя определяет компромисс между скоростью обучения и работы нейронной сети и качеством распознавания. Таким образом, все изображения обучающей выборки нужно привести к фиксированному размеру и перевести координаты соответствующих им ключевых точек.

Изображения перед подачей на вход нейросети рекомендуется нормализовать. Для этого можно для каждого канала посчитать попиксельное среднее и дисперсию по всей выборке и нормализовать каждое изображение с помощью посчитанных значений.

Слои нейронной сети

- Слой свертки – набор обучаемых фильтров.

  Фильтр представляет из себя окно, которое скользит по всей области предыдущего слоя и находит определенные признаки объектов. Размер фильтра обычно берут в пределах от 3×3 до 7×7.

  Предпочтительнее выбирать несколько слоев с маленьким размером фильтра (чередуя свертки с нелинейными функциями активации), чем один слой с большим. Это позволяет уменьшить количество обучаемых параметров модели.

- Слой активации – некоторая нелинейная функция, которая применяется к выходу сверточного слоя.

  Наиболее популярные функции активации: Sigmoid – $f(x) = \frac{1}{(1+e^{-x})}$ и ReLU – $f(x) = \max(0, x)$.

- Слой пулинга – уплотнение карты признаков, при котором непересекающиеся группы пикселей, сжимаются в один пиксель.

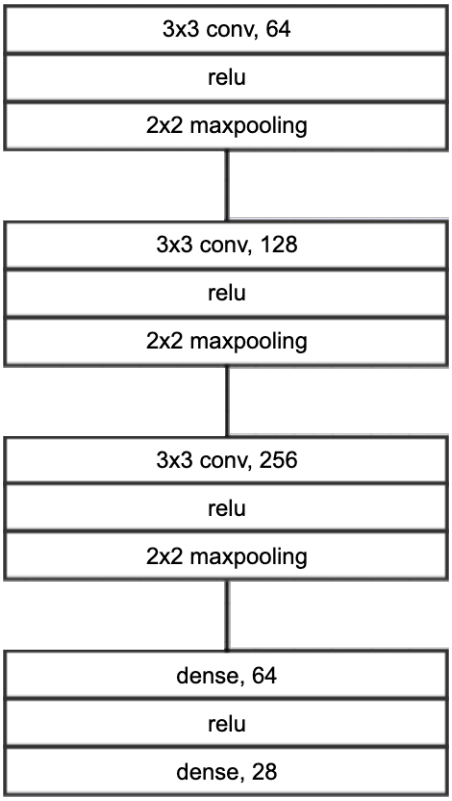
  Наиболее популярная функция пулинга – максимум из области 2×2. Операция пулинга позволяет существенно уменьшить пространственный объём изображения.

- Полносвязный слой – это слой обычного многослойного персептрона.

  После нескольких прохождений свертки изображения и уплотнения с помощью пулинга система перестраивается к более абстрактным картам признаков. В итоге остаётся набор каналов, хранящих самые абстрактные понятия, выявленные из исходного изображения.

  Эти данные объединяются и передаются на обычную полносвязную нейронную сеть, которая тоже может состоять из нескольких слоёв. При этом полносвязные слои уже утрачивают пространственную структуру пикселей.

В качестве базовой можно использовать следующую архитектуру:



Параметры градиентных оптимизаторов

Для настройки весов нейронной сети используются градиентные методы оптимизации (например, стохастический градиентный спуск). Скорость и качество обучения зависят от выбора гиперпараметров данных методов:

- Темп обучения (learning rate).

  Параметр градиентных алгоритмов обучения нейронных сетей, позволяющий управлять величиной коррекции весов на каждой итерации. При большом learning rate параметры модели будут изменяться быстрее, однако может снизиться точность настройки модели, либо алгоритм может перестать сходиться.

  Малые значения learning rate соответствуют меньшему шагу. В этом случае число шагов обучения, требуемое для поиска экстремума, как правило увеличивается, но возрастает и точность настройки алгоритма, что потенциально уменьшает ошибку на обучающей выборке.

  На практике коэффициент скорости обучения обычно подбирают экспериментально. Для темпа обучения также обычно задают расписание, например, умножать `lr` на 0.1 раз в несколько эпох. Постепенное уменьшение темпа обучения позволяет лучше приблизиться к локальному минимуму.

- Инерция (momentum).

  При использовании инерции, градиент определяется как взвешенная сумма градиентов на предыдущей и текущей итерации. Такое определение градиента позволяет избегать некоторых локальных минимумов.

Значение инерции выбирается в интервале от 0 до 1, на практике берется значение близкое к 0.9. При слишком маленьком значении инерции будут происходить колебания около локального минимума, при слишком большом значении обучение может сильно замедлится или даже перестать сходиться.

Существует также метод Нестерова, который является модификацией обычной инерции.

- Размер пакета (batch size).

  Для лучшей сходимости алгоритма градиентный шаг делается по нескольким примерам, множество этих примеров называется пакетом или батчем. Градиентный шаг вычисляется как усредненное значение градиента по всем примерам батча.

  Градиентные методы оптимизации могут плохо сходиться при слишком маленьких размерах батча. По мере увеличения размера батча, стохастическая природа оптимизационного процесса сглаживается, что обычно приводит к улучшению результатов обучения.

  На практике, увеличение размера батча подвержено закону убывающей отдачи. Соответственно, оптимальный размер батча стоит выбирать исходя из соотношения качества, скорости обучения и количества доступной памяти в графическом ускорителе.

- Выбор метода оптимизации.

  Кроме классического стохастического градиентного спуска существуют его различные модификации:

  – RMSprop. Данный метод уменьшает колебания и автоматически подбирает learning rate отдельно для каждого параметра.

  – Adam. Данный метод совмещает в себе преимущества momentum и RMSprop.

- Время обучения в эпохах.

  Эпоха – количество итераций, которое необходимо, чтобы просмотреть всю обучающую выборку с помощью батчей. Останавливать обучение имеет смысл, когда ошибка или качество на валидационной выборке перестает уменьшаться.

Интерфейс программы

Вам необходимо реализовать две функции:

`detect`

Данная функция принимает путь до файла с параметрами уже обученной вами модели и путь до папки с изображениями. Эта функция должна загрузить параметры модели из указанного файла и провести детектирование ключевых точек на всех изображениях в папке. В результате работы, функция должна вернуть словарь, ключи которого – имена файлов, а значения – массивы чисел длины 28 с координатами точек лица $[x_1, y_1, \ldots, x_{14}, y_{14}]$.

Не забывайте, что в тестовой системе есть ограничения по времени и пиковому потреблению памяти. Соответственно, архитектуру сети необходимо выбирать так, чтобы уложиться в данные ограничения ресурсов.

`train_detector`

Данная функция принимает разметку данных доступных вам для обучения (и валидации), путь до папки в которой находятся изображения и флаг `fast_train`. Эта функция должна обучать модель

используя переданные данные. В результате работы, функция должна вернуть обученную модель.

В результате вызова функции `train_detector` с флагом `fast_train=False`, должна получаться модель воспроизводящая качество вашей финальной модели. Ваш код в режиме `fast_train=False` может быть запущен проверяющими на своих машинах. Процесс обучения в данном режиме не должен использовать подключение к интернету, должен работать автономно (без интерактивного взаимодействия), должен занимать "адекватное" количество времени и использовать "адекватное" количество памяти. Если результаты обучения в данном режиме *значительно* отличаются от результатов вашей финальной модели, это приравнивается к плагиату.

В тестовой системе, функция `train_detector` будет вызываться **только** с флагом `fast_train=True`. В данном режиме, необходимо просто продемонстрировать, что ваш код для обучения минимально работоспособен. Для этого необходимо:

- указать маленькое количество эпох/шагов обучения
  (чтобы обучение шло не больше 5 минут, но не меньше двух батчей и не отключая валидацию)
- выключить использование графических ускорителей
  (в тестовой системе доступен только CPU)
- выключить все механизмы логирования/сохранения чекпоинтов на диск
  (в тестовой системе данные подмонтированы в режиме read-only)
- выключить параллелизм в механизмах загрузки данных
  (в тестовой системе вам будет доступно только одно ядро)

Обученная в этом режиме модель для тестирования не используется, этот этап необходим только для проверки работоспособности функции обучения. За исключением изменения выше указанных настроек, код для режимов `fast_train=True` и `fast_train=False` должен быть общим.

Результаты работы функции `train_detector` с флагом `fast_train=False` вам необходимо сохранить в файл `facepoints_model.pt` и загрузить его в тестовую систему. Итоговая оценка за задание будет определяется качеством работы загруженной модели на скрытой тестовой выборке.

Данные и скрипт для тестирования

Скрипт для тестирования оценивает качество детектирования путем подсчета среднеквадратичной ошибки err на изображении, приведенном к размеру 100×100 пикселей. Ошибка err на скрытой выборке конвертируется в итоговый балл исходя из следующих соответствий:

$err \leqslant 20$ – 2 балла,

$err \leqslant 18$ – 3 балла,

$err \leqslant 16$ – 4 балла,

$err \leqslant 14$ – 5 баллов,

$err \leqslant 12$ – 6 баллов,

$err \leqslant 10$ – 7 баллов,

$err \leqslant 8$ – 8 баллов,

$err \leqslant 6$ – 9 баллов,

$err \leqslant 5$ – 10 баллов.

Результаты скрытого теста и итоговый балл скрыты до даты дедлайна задания. Итоговый балл считается по последней посылке с ненулевой точностью.

Вы можете протестировать логику вашего решения на публичных данных, запустив скрипт для тестирования локально:

```
$ run.py path/to/public_tests
```

Обратите внимание! Тестирование на публичных данных использует те же данные, что были выданы вам для обучения. Таким образом, данное тестирование позволяет однозначно оценить только корректность вашего кода. Для самостоятельной оценки качества на публичных данных вам необходимо использовать валидационную выборку (см. раздел Обучение и тестирование).

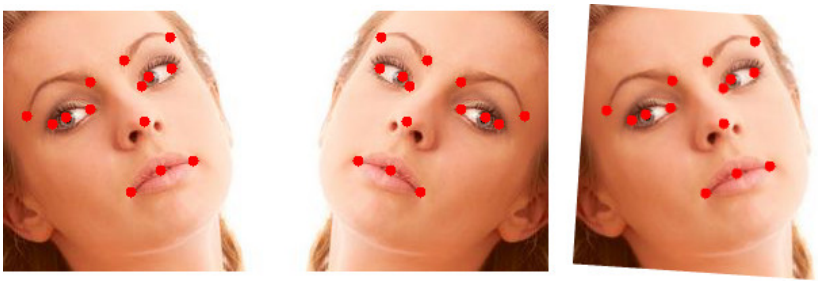
Продвинутая часть

Базовой архитектуры нейросети достаточно, чтобы получить среднеквадратичную ошибку $err \leqslant 20$ и минимальный положительный балл за решение. Для повышения качества предсказания нейросети можно использовать следующие приемы:

Изменение архитектуры

Поэкспериментируйте с количеством слоев и количеством нейронов на каждом слое.

Размножение данных



Для предотвращения переобучения и увеличения точности распознавания нейронных сетей тренировочная выборка обычно размножается (data augmentation). Количество обучающих данных при этом возрастает на порядки.

Изображения человеческих лиц можно размножить следующими случайными преобразованиями:

- зеркальное отражение относительно горизонтальной оси,
- поворот на небольшой угол,
- кадрирование с сохранением всех точек лица на изображении.

Не забывайте при этом применять преобразования и к координатам точек лица. Также, учитывайте, что некоторые операции меняют тип точек (например, после горизонтального отражения изображения точки “левый уголок рта” станет точкой “правый уголок рта”).

Batch normalization

Batch normalization – метод, который позволяет повысить производительность и стабилизировать работу искусственных нейронных сетей. Суть данного метода заключается в том, что некоторым слоям нейронной сети на вход подаются данные, предварительно обработанные и имеющие нулевое математическое ожидание и единичную дисперсию.

- Достигается более быстрая сходимость моделей, несмотря на выполнение дополнительных вычислений.

- Становится возможным использование более высокого темпа обучения, так как пакетная нормализация гарантирует, что выходы узлов нейронной сети не будут иметь слишком больших или малых значений.
- Модели становятся менее чувствительны к начальной инициализации весов.

Dropout

Для предотвращения переобучения рекомендуется использовать дропаут. Идея дропаута состоит в том, что во время обучения мы случайным образом обнуляем связи некоторых нейронов в сети. Дропаут для слоя нейросети определяется вероятностью p, с которой зануляется каждый нейрон слоя. Dropout ипользуют в полносвязных и сверточных слоях после функции активации.

Weight decay

Для предотвращения переобучения в нейросетях можно использовать L_1 или L_2-регуляризацию. К функции потерь прибавляется слагаемое, штрафующее слишком большие значения весов:

$$L'(w, X, Y) = L(w, X, Y) + \frac{\lambda}{2}||w||^2.$$

Здесь λ – параметр (обычно называется затуханием весов), определяющий силу регуляризации.