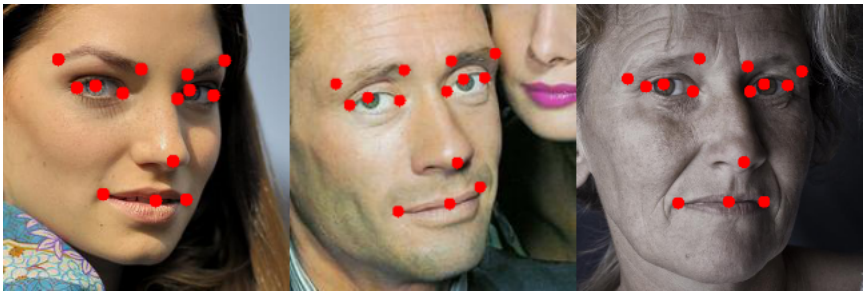
Поиск ключевых точек на лице

Влад Шахуро, Владимир Гузов, Даниил Киреев

Graphics & Media Lab
Vision Group

Обзор задания

В данном задании предлагается реализовать регрессию ключевых точек лица с использованием сверточной нейронной сети. Для реализации используйте библиотеку Keras поверх TensorFlow. Библиотека работает на CPU и GPU. Если у вас есть CUDA-совместимая видеокарта, мы советуем использовать её, тогда обучение нейросети будет идти на порядок быстрее.



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.

Архитектура сети (количество, типы и параметры слоев, функции активации) не закреплена. Рекомендуем начать с простых сетей (персептрон, затем сверточная сеть LeNet, адаптированная для регрессии и изображений выбранного размера) и затем экспериментировать (добавлять слои, изменять количество нейронов).

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

Стохастический градиентный спуск

Для настройки весов нейронной сети используется стохастический градиентный спуск. Скорость и качество обучения зависит от параметров градиентного спуска:

- Темп обучения (learning rate, lr). Чем он больше, тем сильнее обновляются веса на каждой итерации. Для темпа обучения обычно задают расписание, например, делить lr на 10 когда ошибка на тренировочной выборке перестала уменьшаться. Постепенное уменьшение темпа обучения позволяет лучше приблизиться к локальному минимому.

- Момент (momentum). При использовании момента градиент для спуска определяется как взвешенная сумма градиентов на предыдущей и текущей итерации. Такое определение градиента позволяет избегать некоторых локальных минимумов. Существует также момент Нестерова, который является модификацией обычного момента.
- Размер минибатча. Чем больше размер минибатча, тем более надежна стохастическая оценка градиента.
- Метод оптимизации. Кроме классического стохастического градиентного спуска существуют его различные модификации: Adam, RMSprop и другие.
- Время обучения в эпохах. Эпоха — количество итераций, которое необходимо, чтобы просмотреть всю обучающую выборку с помощью минибатчей.

Мониторинг обучения и регуляризации

Метрикой качества является среднеквадратичная ошибка:

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

где $f(w, x)$ — функция, вычисляемая нейросетью. Для мониторинга обучения нейронной сети обычно используется валидационная выборка. Для ускорения обучения качество на валидационной выборке проверяется через фиксированное количество итераций или в конце эпохи. Опишем несколько приемов, используемых при обучении.

Batch normalization. Для ускорения обучения и увеличения способности сети к обобщению в некоторых местах (к примеру, после maxpooling-слоев) можно добавить batch normalization. Batch normalization выполняет нормализацию значений предыдущего слоя для каждого минибатча в отдельности.

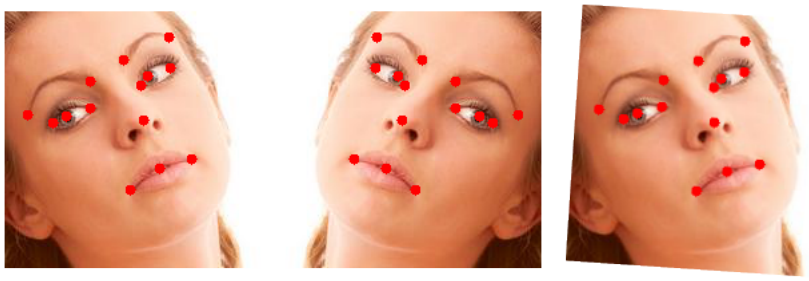
Dropout. Для предотвращения переобучения рекомендуется использовать дропаут. Идея дропаута состоит в том, что во время обучения мы случайным образом обнуляем связи некоторых нейронов в сети. Дропаут для слоя нейросети определяется вероятностью p, с которой зануляется каждый нейрон слоя.

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:

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

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

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. Данные можно размножить заранее и записать на диск или размножать в процессе обучения с использованием функций-предобработчиков. 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.

Интерфейс программы, данные и скрипт для тестирования

Необходимо реализовать две функции: `train_detector`, обучающую модель детектора, и `detect`, проводящую детектирование ключевых точек на изображениях с обученной моделью. Функция детектирования возвращает словарь размером N, ключи которого — имена файлов, а значения — массив чисел длины 28 с координатами точек лица $[x_1, y_1, \ldots, x_{14}, y_{14}]$. Здесь N — количество изображений.

Скрипт для тестирования обучает детектор и 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:

$err \leqslant 9$ — 10 баллов

$err \leqslant 11$ — 9 баллов

$err \leqslant 13$ — 8 баллов

$err \leqslant 15$ — 7 баллов

$err \leqslant 17$ — 6 баллов

$err \leqslant 20$ — 5 баллов

$err \leqslant 50$ — 2 балла

Для обучения алгоритма выдается публичная выборка лиц с размеченными ключевыми точками. Программа тестируется на двух тестах. В каждом из тестов нейросеть сначала обучается с флагом `fast_train=True` в функции `train_detector`. Функция обучения с этим флагом должна работать недолго, не больше 5 минут. Для этого поставьте 1 эпоху обучения и несколько батчей. Обученная модель для тестирования не используется, этот этап необходим только для проверки работоспособности функции обучения. Затем в первом тесте алгоритм тестируется на публичной выборке, во втором тесте — на скрытой выборке. Для тестирования загружается обученная модель `facepoints_model.hdf5`. Результаты второго теста и итоговый балл скрыты до окончания срока сдачи задания. Итоговый балл считается по последней посылке с ненулевой точностью. Для уменьшения количества потребляемой памяти в Keras можно при обучении и тестировании использовать генераторы батчей (функции `model.fit_generator` и `model.predict_generator`).

Полезные ресурсы

Материалы стэнфордского курса по нейронным сетям
Dropout: A Simple Way to Prevent Neural Networks from Overfitting — статья про дропаут
Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift — статья про batch normalization