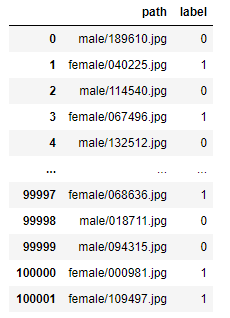
Процесс подготовки данных

Датасет весит всего 2ГБ, однако состоит из 100 000 файлов. Из-за этого при загрузке в RAM возникли следующие проблемы:

1. Первые несколько тысяч фотографий загружались за минуту, а дальше скорость загрузки заметно падала. В итоге, для того чтобы выгрузить весь датасет в оперативную память, требовалось большое количество времени. В Google Colab дело обстояло еще хуже.
2. Изначально планировалось обучать на Google Colab, однако там есть серьезные проблемы при манипулировании большим количеством файлов. Из-за ошибок Google Drive часто крашился весь процесс.

Чтобы решить эти проблемы, пришлось загружать данные в RAM по батчам во время обучения. Для этого с помощью функции create\_csv создается csv таблица данных с двумя столбцами в папке с датасетом:

* Path – путь к фото относительно папки датасета
* Label – класс: 0 – мужчина, 1 – женщина

Затем делим наши табличные данные:

* Train – 70 000 фото
* Validation – 15 000 фото
* Test – 15 000 фото

Дальше создается pytorch класс FaceDataset, в котором прописаны функции загрузки и преобразования данных. В класс затем и подаются данные из таблицы. Таким образом с помощью этого класса можно обращаться к данным как к итерируемому объекту. Например:

*train\_data = FaceDataset(X\_train, y\_train, data\_dir)*

*train\_data[0]* – первое изображение из тренировочного сета, ресайзнутое и нормализированное, уже приведенное к torch.Tensor.

Затем данные загружаются в DataLoader, генератор батчей. С помощью конструкции *for X, y in data\_loader* можно прогнать все данные, привязанные к data\_loader. Таким образом батчи сэмплируются из директории датасета прямо при обучении. Это увеличивает время обучения, но позволяет не держать в памяти весь датасет и не тратить время на предобработку до обучения.

Используемая нейросеть

Были попробованы несколько популярных сверточных архитектур. Очень глубокие сети, такие как ResNext101, занимают лидирующие позиции в ImageNet соревновании, но для нашей задачи они слишком мощные и требуют огромное количество времени и ресурсов на обучение. Поэтому было принято решение использовать “среднюю” модель ResNet18.

Я использовал pre-trained на ImageNet модель, а затем с помощью дообучил 2, 3, 4 слои и классификатор под нашу задачу. При использовании pre-trained сети, модель уже умеет выделять признаки (features) и нам не требуется учить ее с нуля этому. Нам лишь требуется дообучить классификатор и последние слои, чтобы они из этих признаков выделяли именно то, что нам нужно.

С математической точки зрения таким способом мы решаем задачу инициализации весов. Мы уже с самого начала обучения находимся в более удобной точке функции loss, откуда мы быстрее и вернее найдем минимум, чем если выбирать эту точку случайно (как при случайной инициализации весов).

Параметры обучения

* Оптимизатор: AdamW с AMSGrad
* Learning rate = 0.001
* Функция лосса = CrossEntropyLoss
* 6 эпох обучения

В качестве оптимизатора был выбран адаптивный оптимизатор AdamW c включенным AMSGrad (<https://arxiv.org/abs/1904.09237>). На задачах классификации с скросс энтропией данный оптимизатор обходит обычный Adam <https://arxiv.org/abs/1711.05101> (в нашей задаче он тоже сходится быстрее и лучше, чем Adam). Функция лосса – кросс энтропия как стандарт при обучении классификаторов. Learning rate стандартный 0.001.

Полученные результаты

Модель обучилась на 6 эпохах, лучшие показатели были на 4 эпохе (epoch\_3.pth). На тестовом датасете accuracy составила 0.991

Инструкция по запуску

Для запуска обучения нужно запустить файл “Скрипт обучения.ipynb”. Для этого нужен python 3 версии, а так же последние версии модулей torch, albumentations, PIL, pandas, matplotlib, numpy. В первой ячейке нужно указать полный путь к папке internship\_data. Если папка находится в одной директории со скриптом, то можно оставить пустую строку. Затем поочердно выполнить ячейки. Модель во время обучения сохраняет веса каждой из эпох в .pth файлы.

Для запуска скрипта для использования нейросети нужно перейти в папку с проектом и выполнить в командной строке команду: *python script.py “path to folder”* где path to folder – путь к папке с изображениями, которые нужно обработать (путь передавать в кавычках). В ней скрипт создаст файл process\_results.json. Скрипт автоматически загружает в себя веса модели 4 эпохи (epoch\_3.pth) из директории model\_weights, которая должна лежать в одной папке со скриптом.