Обучение производил в Google Colab, использовал TensorFlow/Keras.

**Ссылка на модель:**

https://drive.google.com/file/d/1lxROITBGdlUc77qwen75DUh4s6Wt\_-4n/view?usp=sharing

# Загрузка данных

С помощью библиотеки google\_drive\_downloader скачиваем архив данных, затем через tarfile разархивируем его и убеждаемся в правильности загрузки данных путем подсчета изображений в каждой из папок.

# Подготовка данных к подаче в модель

Данные будем подавать в модель через ImageDataGenerator().flow\_from\_dataframe(), так как с данными представленными в виде pandas DataFrame удобно работать. Формируем необходимые DataFrame и затем разбиваем исходную совокупность на train/validation в пропорции: 90% в train и 10% в test (Andrew Ng советует делать train больше при больших датасэт). При это и train и valid множества сбалансированы с точки зрения классов.

# Загрузка модели

В качестве подхода будем использовать Transfer learning. Соответственно загружаем нейронную сеть, которая ранее решала схожую задачу, а именно задачу классификации изображений (лиц). Я выбрал сеть, архитектура которой представляет из себя ResNet, обученную на дата сете VGGFace2. Данная сеть имеет свою препроцессинговую функцию preprocess\_input(), и ожидает на вход картинку размерность 224х224.

# Первый этап обучения

Для формирования **базовой модели** отрезаем классифицирующий слой у исходной модели, благодаря чему базовая модель на выходе будет иметь 2048-мерный вектор признаков и делаем все слои не обучаемыми. В качестве выходного слоя добавляем полносвязный слой с активационной функцией sigmoid, т.к. она подходит для бинарной классификации. Таким образом имеем 2 049 параметров модели.

После первого этапа имеем точность на train 0,988 и на valid 0,985 – из чего делаем вывод, что модель для нашей задачи подходит, переобучения не произошло. Пробуем улучшить метрику.

# Второй этап обучения

Попробуем fine tuning. Для этого у базовой модели делаем все слои начиная с последнего сверточного обучаемыми, предполагая, что именно последний сверточный слой содержит признаки, специфичные для датасета на котором обучалась базовая модель. Также уменьшаем learning rate, для того чтобы не сильно менять уже подобранные веса, базовой модели. Таким образом сложность модели возрастет до 1 054 721 параметров.

После данного этапа имеем точность на train 0,996 и на valid 0,987 – удалось улучшить метрику, но, кажется, что модель переобучилась.

# Третий этап обучения

Для борьбы с переобучением добавим аугментацию, а именно поработаем с освещением и с масштабом картинок, так как именно этими параметрами отличаются изображения.

Грузим лучшую модель с прошлого этапа обучения. Данная модель на данных с аугментацией дает точность на train 0,991 и на valid 0,990. (В ноутбуке видно, что прошло только 17 из 20 заявленных эпох, но это потому, что у Google Colab срабатывает ограничение по ресурсам).

# Строим ROC кривую

Для использования модели необходимо понять оптимальный threshold, для этого строим ROC кривую.

# Инструкция для запуска тренировки

Обучение проводится с использованием фреймворка Tensorflow/Keras c версией tensorflow\_version 1.x. Тренировку желательно проводить в Google Colab, в таком случае сработают команды по выбору версии tensorflow.

# Инструкция для запуска нейронной сети

Для запуска нейронной сети необходимо вызвать функцию **use\_model()**. И передать ей в качестве **обязательных** параметров:

* path\_to\_data: Путь к папке с данными;
* model: Модель, с возможностью применить метод .predict();

И в качестве необязательных параметров:

* threshold: Трешхолд, значение по умолчанию установлено равным найденному в Пункте 7;
* path\_to\_json: Путь с именем файла, в котором сохраняться результаты. По умолчанию файл сохраняется в папку, из которой был запущен скрипт с именем файла «process\_results.json».

P.S.

PyTorch начинаю учить сразу после отправки тестового задания на проверку😊.