**Описание модели нейронной сети для определения пола человека по изображению лица**

Было предложено разработать модель, определяющую пол лица по изображению. Для этих целей компания NtechLab предоставила изображения мужских и женских лиц в количестве 50001 каждого вида (итого 100002 изображения). Изображения хранятся в двух папках: male и female. Изображения не были разбиты на тренировочную, валидирующую и тестовую выборку, предлагалось это сделать самостоятельно.

**На первом этапе** была подготовлена таблица, состоящая из двух столбцов: название файла изображения и разметка (в данном случае, пол male или female). Далее был подготовлен список с путями к файлам изображений (в зависимости от пола, папки отличались), который соответствовал разметке. Из столбца разметки был сформирован двумерный массив, заполненный булевыми переменными True, False (см. Рис.1).

```
[array([False,  True]),
 array([False,  True]),
 array([ True, False]),
 array([ True, False])]
```

Рис.1

Таким образом, наш массив классифицирует изображения как мужские или женские лица. Так, первые два изображения в массиве на рис.1 - мужские, третье и четвертое - женские.

**На втором этапе** необходимо было разбить изображения на тренировочную, валидирующую и тестовую выборку и заодно их перемешать (до этого этапа у нас в начале списка располагались женские фотографии, затем мужские). И так как для отработки модели предполагалось использовать небольшое количество фотографий, то я ввел переменную NUM_IMAGES, которая отвечает за количество фотографий, с которыми в дальнейшем будем работать. В качестве тестовой выборки можно выбрать диапазон изображений, не попавший в тренировочную и валидирующую выборку.

**На третьем этапе** была написана функция для конвертации путей файлов в тензоры (нейросети работают с изображениями, которые представлены в виде тензоров). При конвертации размер всех картинок установлен как 200x200. Далее была определена функция для упаковки наших изображений и их разметок в батчи в виде кортежа (см. Рис.2).

```
((TensorSpec(shape=(None, 200, 200, 3), dtype=tf.float32, name=None),
  TensorSpec(shape=(None, 2), dtype=tf.bool, name=None)),
 (TensorSpec(shape=(None, 200, 200, 3), dtype=tf.float32, name=None),
  TensorSpec(shape=(None, 2), dtype=tf.bool, name=None)))
```

Рис.2

**На четвертом этапе** была создана и обучена модель. В качестве исходной модели была принята модель InceptionV3 (Рис.3) с использованием фреймворка Tensorflow-Keras.

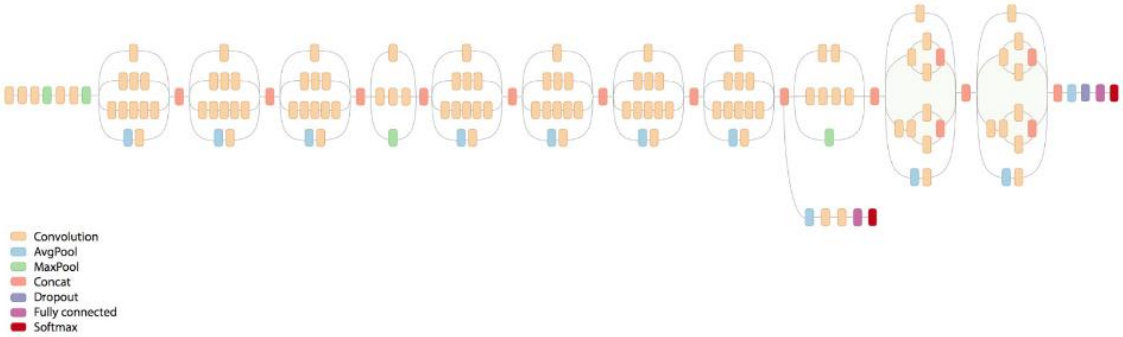


Рис.3

Были загружены веса модели, обученной на изображениях imagenet, верхние слои были убраны и добавлены свои для бинарной классификации. Были получены 3 модели, обученные на разном количестве изображений: 1 тыс., 10 тыс., и 100 тыс. Модели были сохранены в формате h5.

Модель, обученная на 10 тыс. изображениях, была оценена на тестовой выборке (2 тыс. изображений). Точность модели на тестовой выборке: 94.55%, при том что точность на валидирующей - 94,65 %.

**Параметры модели:**

- Исходная модель: InceptionV3
- Dropout – 0.5 (Dropout – это своего рода регуляризация, для того чтобы предотвратить переобучение, так как у нас большое количество картинок (если использовать все 100 тыс., возможно, имеет смысл уменьшить его до 0.2).
- Оптимизатор – стохастический градиентный спуск (SGD), скорость обучения -0,0001, момент – 0.9, функция штрафа – кроссэнтропия ('categorical_crossentropy'), метрика (была указана в ТЗ) – accuracy. Это значит, что мы будем вычислять в модели не только функцию штрафа, но и точность работы, то есть число правильно классифицированных примеров.
- Количество эпох -5(8)
- Размер пакета (Batch size) – 32 (Подсмотрено у Яна Лекуна https://twitter.com/ylecun/status/989610208497360896?s=20 )
- Callback функция – ModelCheckpoint с сохранением лучших весов

**To do**:

Добавить функцию, посредством которой можно отобразить те изображения в которых модель ошиблась, с вероятностью принадлежности изображения к двум классам.

Попытаться настроить гиперпараметры модели для улучшения точности классификации.

Добавить аугментацию изображений (сдвиг, поворот…) для увеличения количества изображений и повышения точности классификации (использовать методы model.fit_generator)

Добавить сравнение моделей друг с другом.

**Инструкция по тренировке модели**

Для тренировки модели необходимо в Jupyter Notebook открыть файлы Gender_detection.ipynb.

После открытия запустить первую исполняемую ячейку для импорта необходимых библиотек (см. Рис.1).

```
import pandas as pd
import os
import numpy as np
from IPython.display import Image,display

from matplotlib import pyplot as plt
import seaborn as sns
%matplotlib inline

from sklearn.model_selection import train_test_split

from keras.applications.inception_v3 import InceptionV3, preprocess_input
from keras import optimizers
from keras.models import Sequential, Model
from keras.layers import Dropout, Dense, Flatten, GlobalAveragePooling2D
from keras.callbacks import ModelCheckpoint
from keras.utils import np_utils
from keras.optimizers import SGD
import tensorflow as tf
```

Рис.1

Далее указать пути к изображениям мужских и женских лиц (Рис.2. Да, в моих путях оЧепятки юпитер через I, а gender_detetion ☺)

```
# Подготовим директории файлов
directory_female = r'C:\Users\User\Jupiter\Gender_detetion\internship_data\female'
directory_male = r'C:\Users\User\Jupiter\Gender_detetion\internship_data\male'
```

Рис.2

Далее исполняйте все ячейки по очереди до ячейки с переменной NUM_IMAGES. В ней укажите желаемое количество используемых изображений (Рис. 3).

**3. Разделение подготовленных данных на тренировочную, валидирующую, тестовую**

```
# Создадим X и y
X = paths
y = boolean_labels
```

Разделим выборку на тренировочную и тестовую Начнем экспериментировать с 1 000 изображений и будем увелчичивать количество изображений

```
# Зададим количество изображений для экспериментов
NUM_IMAGES = 1000
```

Рис. 3

После определения переменной NUM_IMAGES также исполняйте код в ячейках до ячейки с функцией process_image (Рис.4). Здесь можно указать размер картинок, которые

будут поступать на вход модели. Если есть желание поэкспериментировать, можно изменить переменные IMG_height и IMG_width на свои значения, которые отвечают за высоту и ширину изображений соответственно.

```
# Определим размер изображения
IMG_height = 200
IMG_width = 200

# Создадим функцию для препроцессинга изображений

def process_image(image_path, img_height=IMG_height, img_width=IMG_width):
    """
    На вход подается путь к изображению и функция конвертирует изображение в тезор.
    """
    # Считаем изображение
    image = tf.io.read_file(image_path)
    # Конвертируем изображение в формате jpeg в числовой тензор с 3 каналами (красный, зеленый и синий)
    image = tf.image.decode_jpeg(image, channels=3)
    # Конвертируем каналы цвета из диапазано 0-255 в диапазон 0-1(нормализуем)
    image = tf.image.convert_image_dtype(image, tf.float32)
    # Изменим размер наших рисунков до (300, 300)
    image = tf.image.resize(image, size=(IMG_height, IMG_width))

    return image
```

Рис. 4

Далее исполняем последовательно код до ячейки с функцией create_data_batches (Рис. 5). Здесь переменная BATCH_SIZE отвечает за размер пакета с изображением, который поступает на вход модели, при желании его также можно изменить.

```
# Определим размер батча, 32 это хороший размер
BATCH_SIZE = 32

# Создадим функцию для преобразования в батчи(пакеты)
def create_data_batches (X, y=None, batch_size=BATCH_SIZE, valid_data=False, test_data=False):
    """
    Создадает пакеты данных из изображений (X) и меток (y) в виде пары.
    Перемешивает данные если функция используется на тренировочной выборке, но не перемешивает - если
    на валидурующей.
    Также позволяет подготовить тестовую выборку (в которой нет меток)
    """
    # Если выборка представлет их себя тестовый датасет у нас скорее  всего не будет меток для ее
    if test_data:
        print("Создаются пакеты для тестовой выборки...")
        data = tf.data.Dataset.from_tensor_slices((tf.constant(X))) # Только пути к файлам (без меток)
        data_batch = data.map(process_image).batch(BATCH_SIZE)
        return data_batch
```

Рис. 5

Продолжите исполнять ячейки до ячейки, изменяющей верхние слои модели (Рис. 6). Если вы указали большое количество изображений для обучения (около 100 тыс. или более), вероятно, имеет смысл уменьшить Dropout (типичные значения от 0.2 до 0.5).

**Верхние слои (включая классификацию не включены). Эти слои будут замещены следующими слоями:**

```
x = inc_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(1024, activation='relu')(x)
x = Dropout(0.5)(x)
x = Dense(512, activation='relu')(x)
predictions = Dense(2, activation='softmax')(x)
```

Рис. 6

В следующей ячейке создается итоговая модель (Рис. 7). В этой ячейке можно поменять гиперпараметры модели, такие как оптимизатор (вместо SGD, например, указать adam), поменять скорость обучения, момент, функцию ошибок, а также метрику. Если настраивать гиперпараметры не нужно, просто оставьте те, которые есть.

```
# Создадим итоговую модель

model_ = Model(inputs=inc_model.input, outputs=predictions)

# Залочим слои, чтоб они не были обучены

for layer in model_.layers[:52]:
    layer.trainable = False

# Скомпилируем модель

model_.compile(optimizer=SGD(lr=0.0001, momentum=0.9), loss='categorical_crossentropy', metrics=['accur
```

Рис.7

Исполните последующие ячейки до ячейки, в которой проходит обучение модели (Рис. 8). Здесь можно указать количество эпох.

**ВНИМАНИЕ:** Обучение модели может занять длительное время, в зависимости от гиперпараметров, а главным образом от количества изображений, на котором будут обучать модель. Также время обучения зависит от мощности ПК или сервера.

```
# Обучим модель
num_epochs = 5
hist = model_.fit(x=train_data,
                  epochs=num_epochs,
                  validation_data=val_data,
                  validation_freq=1,
                  steps_per_epoch=len(X_train)/BATCH_SIZE,
                  callbacks=[checkpointer],
                  verbose=1)
```

Рис.8

После обучения необходимо сохранить модель, для этого написана функция (Рис. 9). В переменной directory, укажите путь к папке, в которой хотите сохранить модель (да, да, у меня в пути к папке все те же оЧепятки ☺ ☺ ). Исполните код функции. В последующей ячейке удалите знак комментария, укажите suffix для того, чтобы вы могли идентифицировать модель по названию, и исполните код ячейки.

```
# Создадим функцию для сохранения модели
def save_model(model, suffix=None):
    """
    Сохраняет данную модель в директорию models и добавляет суффикс(текст)
    """
    # Создадим путь к модели, а также текущее время
    directory = r'C:\Users\User\Jupiter\Gender_detetion\internship_data\models'
    modeldir = os.path.join(directory,
                            datetime.datetime.now().strftime("%d-%m-%Y_%H-%M-%S"))
    model_path = modeldir + "_" + suffix + ".h5" # расширение модели
    print(f"Сохранение модели в: {model_path}...")
    model.save(model_path)
    return model_path
```

```
# save_model(model_, suffix='100002_img')
```

Рис. 9

**Инструкция по работе со скриптом process.py**

Данный скрипт на вход получает путь к папке с файлами картинок, которые нужно классифицировать. На выход скрипт выдает файл json с прогнозом пола человека.

Прежде чем исполнять скрипт, необходимо провести его небольшую настройку.

Откройте файл process.py и укажите в переменные model_directory и path_json путь к сохраненной модели и путь, в котором вы хотите сохранить файл json (Рис.1.)

```
# Присвоим пути модели и пути сохранения файла json
model_directory = r'C:\Users\User\Jupiter\Gender_detetion\internship_data\models\08-09-2020_14-52-49_10000_img.h5'
path_json = r'C:\Users\User\Jupiter\Gender_detetion\internship_data\test'
```

Рис.1

Сохраните скрипт.

Теперь в терминале IDE запустите скрипт.

**Внимание!** Нужно, чтобы в среде, в которой вы работаете, были установлены необходимые библиотеки: tensorflow и numpy.

Итак, для запуска скрипта необходимо:

1. Войти в директорию, в которой находится файл process.py
2. В терминале ввести: python process.py
3. После пробела ввести путь к папке с изображениями, которые необходимо классифицировать.

На Рис.2 можно увидеть вышеперечисленные этапы по запуску скрипта.

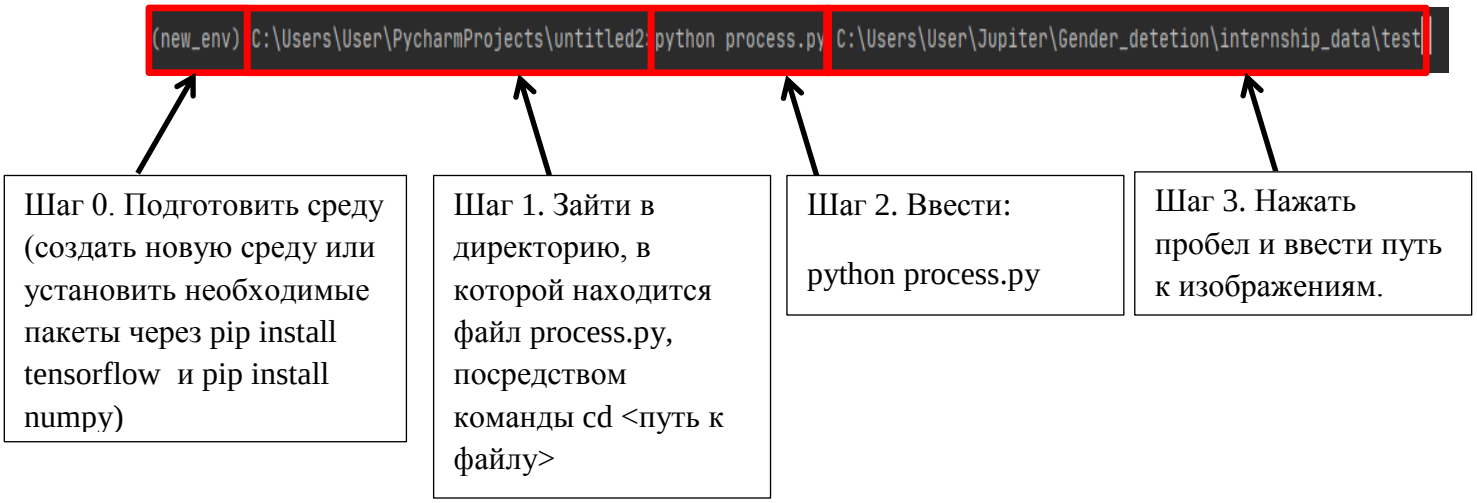


Рис.2

Скрипт запускается, после чего сохраняется файл «process_result.json» (рис.3)

```
(new_env) C:\Users\User\PycharmProjects\untitled2>python process.py C:\Users\User\Jupiter\Gender_detetion\internship_data\test
Создаются пакеты для тестовой выборки...
2020-09-10 12:45:33.379780: I tensorflow/core/platform/cpu_feature_guard.cc:142] This TensorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN)
to use the following CPU instructions in performance-critical operations:  AVX2
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
2020-09-10 12:45:33.899810: I tensorflow/compiler/xla/service/service.cc:168] XLA service 0xdafdba0 initialized for platform Host (this does not guarantee that XLA w
ill be used). Devices:
2020-09-10 12:45:33.906811: I tensorflow/compiler/xla/service/service.cc:176]   StreamExecutor device (0): Host, Default Version
1/1 [==============================] - 0s 0s/step
Файл process_result.json успешно сохранен
```

Рис. 3.

**Примечание:** На моем ПК в процессе работы скрипта появляются предупреждения о том, что библиотека tensorflow не оптимизирована под мой процессор (здесь была использована версия tensorflow cpu, так как видеокарта довольно-таки старая).

В результате работы скрипта в указанную папку записывается файл json (см. Рис. 4). В моем случае это папка, где лежат тестируемые изображения.

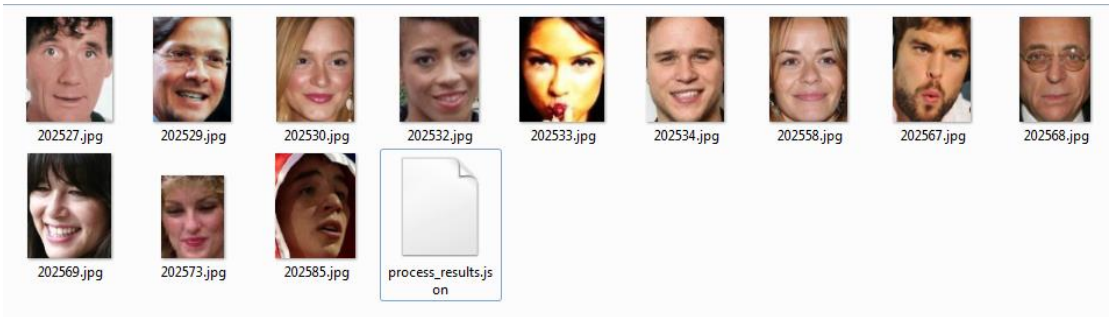


Рис. 4.

Можно открыть файл и посмотреть правильно ли записан файл (Рис. 5).

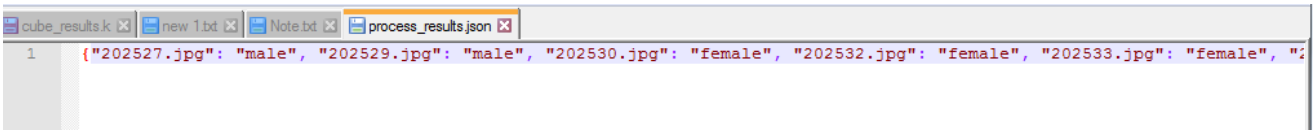


Рис. 5

Скрипт будет перезаписывать файл, существующий в этой директории, поэтому, чтобы не потерять результаты, рекомендуется перенести файл в другую папку.

**Что будет улучшено в последующих версиях скрипта:**

1. Отработка ошибок. Сейчас её практически нет.
    1.1 Проверка существования пути к файлам изображений
    1.2 Проверка наличия файлов .jpeg и возможность тестировать файлы других форматов (RAW, JPG, PNG, TIFF и т.д.)
    1.3 …
2. Добавлен вывод информации о процессе прогнозирования.
3. Добавлена возможность запуска скрипта из командной строки (например windows)
4. Проведена проверка работоспособности скрипта на других ОС, а также работу на разных версиях одной ОС (в данной версии проверено только на Windows 7)
5. Проведена проверка работоспособности при различных версиях используемых библиотек (скрипт разработан на Python - 3.7.7, tensorflow cpu - 2.3.0, numpy - 1.18.5)
6. Проведена проверка правильной записи файла json.