ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»
Московский институт электроники и математики

Казамбаев Семен Владимирович

Проект "Рассовая классификация людей по фотографиям".

"Прикладная математика"

Студент
С.В.Казамбаев

Руководитель ВКР
Профессор
В.Ю. Попов

Москва 2025г.

Содержание

Аннотация

В рамках данного проекта разрабатывалась модель нейронной сети для решения задачи классификации фотографий людей по их рассовой принадлежности , с использованием языка Python , а также библиотеки Keras, через которую происходило взаимодействие с библиотекой TensorFlow . Обе библиотеки предназначенны для глубокого машинного обучения. В качестве библиотеки данных для обучения и тестирования нейронной сети был выбран датасет "Human-Race-Identification"[**data**], взятый с сервиса kaggle , в последствии датасет был дополнен данными из других баз данных, а также фотографиями взятыми из открытых источников.

Затем были созданы и проанализированы несколько моделей с разными по сложности архитектурами.

Из которых наибольшую тестовую точность показали:

- CNN-модель "Model-2" 62%
- модель "Model-4" с внедренной архитектурой EfficientNetB7 50%
- модель "Model-3" с внедренной архитектурой ResNet50 45%

(все они приведены в google colab)

Такая точность определения расы связана с весьма сложными для классификации данными в датасете .

Повлияли следующие факторы :

1) Люди, на фотографиях в датасете, в абсолютном большинстве представители смешанных между собой рас , что сильно усложняет для нейронной сети задачу выделения каких-либо конкретных признаков.

2) Снимки в датасете сделаны с разным освещением , из-за этого становится практически невозможным выделения признаков цвета кожи.

3) Снимки в датасете сделаны не строго в одной проекции , большинство из них - под каким-либо углом, становится нейросети сложнее их распознавать .

4) На многих фотографиях в датасете присутствуют вотермарки / другие помехи.

5) Подведение фотографий под единый размер создало проблему геометрического искажения / потери частей снимков , что приводит к снижению количества выделенных нейросетью признаков .

В совокупности все вышеперечисленные факторы приводят к потери точности на этапе тестирования нейронной сети на незнакомой ей выборке , на обучающей выборке достигались результаты в 99.68% .

Для повышения точности результатов работы нейронной сети использовались следующие методы:

- Анализ и предобработка данных, включающая упрощение структуры, а также изменение размера каждого снимка.

- Использование в архитектурах моделей нейронной сети для классификации, многослойного персептрона, из библиотеки Keras Python модуль Sequential().[**keras**]
- Эксперементальный подбор оптимальной архитектуры нейронной сети.
- Использоваие предобученных моделей, таких как ResNet50[**ResNet50**] и EfficientNetB7.[**EfficientNetB7**]
- Использование L1 и L2 регуляции для смещений и активаций , а также Dropout для предотвращения переобучения.

Введение

В наши дни нейронные сети являются мощными и незаменимыми инструментами машинного обучения, которые могут быть использованы для решения широкого спектра задач, в том числе распознавание образов и последующую их классификацию. Эта технология нашла применение во многих областях , среди которых сфера безопасности, промышленность, экономика и медицина. Задача классификации одна из самых распространенных в мире, существует масса вариантов ее решения, в последнее время для этого активно применяются нейронные сети глубокого обучения. Нейронные сети хорошо подходят для решения таких задач, поскольку они могут научиться сложным взаимосвязям в данных.

В рамках данного проекта будет изучаться метод решения задачи классификации, использующий нейронную сеть для выявления закономерностей и признаков в данных и дальнейшего предсказания нужного класса. Проект выполнялся на языке программирования Python, в среде разработки Google Colaboratory, а также PyCharm.

Цели и задачи

Цель проекта :

Создание модели нейронной сети, которая способна определить расу человека по фотографии его лица.

Задачи проекта :

1. Изучить способы анализы изображений на языке Python.
2. Выбрать библиотеку для написания нейронной сети.
3. Подобрать подходящий датасет или составить его самостоятельно.
4. Провести обработку данных и разделить их на обучающюю и валидационную выборки.
5. Изучить, какие бывают архитектуры нейронных сетей.

6. Проанализировать и выбрать несколько архитектур нейросети.

7. Создать модели и обучить их на созданном датасете.

8. Оценить результаты и скорректировать подход для их улучшения.

Поиск и подготовка данных

Прототип датасета, который на данный момент используется для обучения расовой классификации моделей нейронной сети я получил из датасета, найденного на популярном в сфере анализа данных и машинного обучения сайте www.kaggle.com, который является хранилищем большого количества баз данных.
В Kagle есть удобная система поиска баз данных (раздел «Datasets»)(поисковая строка «Search dataset»). На Kagle я нашел только 2 совпадающих с моей темой датасета, из них я выбрал базу данных “Human-Race-Identification”[**data**], за ее упорядоченную, понятную структуру, второй датасет не был взят как основной из-за отсутствия классификации изображений для последующего обучения , также там были поврежденные файлы.

Перед предобработкой датасета было принято решение увеличить его в объеме. Для этого я вручную проклассифицировал часть файлов из поврежденного датасета, а также дополнил датасет снимками с открытой базы данных

IStock[**https://www.istockphoto.com/ru**].

Затем база данных была загружана на Google Disk , а Google Disk подключен к блокноту в Google Colab, в котором был реализован проект.

Создание Датасета

Архитектура датасета:

папка dataset , в которой находятся 2 папки: train и validation,

в каждой из которых - 4 папки - asian, european, indian, negroids - папки классов.

train и validation - обучающая и тестовая выборки соответственно, датасет разделен в соотношении 8 к 2

Общий обьем датасета - 1260 элементов .

После создания датасета было необходимо каждую фотографию привести к единому размеру для корректной передачи тензоров изображений - векторов, составленных из значений пикселей снимка, которые принимает на вход нейронная сеть.

Эта задача решается при загрузке датасета в

train (train_dataset*and*validation_dataset)

спомощью функции из библиотеки keras -

tf.keras.utils.image_dataset_from_directory(),

куда передаются данные ожидаемого нейронной сетью размера фотографии

(image_size=(img_height, img_width)),

в моем случае:

img_height, img_width = 600, 600 или **img_height, img_width = 224, 224**,

в зависимости от архитектуры.

После изменения своего размера снимки передаются в дальнейшую обработку

в формате **(img_width, img_height, 3)**,

где **img_width, img_height** - новые размеры фотографии,

а 3 - количество цветов (RGB).

Загрузка датасета, формирование тензоров изображений:

При подключении датасета к программе он делится на 2 части:

train_dataset и **validation_dataset**, в соотвтетствии с папками **train** и **validation**,

где

train_dataset и **validation_dataset** - обучающая и тестовая выборки соответственно, датасет разделен в соотношении 8 к 2. Этим занимается вышеупомянутая функция

image_dataset_from_directory(),

которая принимает в себя следующие аргументы :

train_data_dir - путь к папке с классами

shuffle=True - Логическое значение, указывающее, следует ли перемешивать порядок изображений в наборе данных.

image_size=(img_height, img_width) - Кортеж из двух целых чисел, указывающий желаемый размер для изменения размера изображений.

batch_size=batch_size - Количество изображений, которое будет возвращаться из набора данных за одну итерацию.

label_mode='categorical' - Преобразует метки в категориальные векторы, где каждый элемент представляет вероятность принадлежности к классу.

color_mode='rgb' - Загружает изображения в цветовом режиме RGB.

Для упрощения вычислений в процессе обучения и тренеровки нейронной сети значение каждого пикселя снимка делится на 255 (в пикселе закодировано значение от 0 до 255) для этого применяются следующие функции:

train_dataset = train_dataset.map(lambda x, y: (x / 255.0, y))

validation_dataset = validation_dataset.map(lambda x, y: (x / 255.0, y))

Результатом работы скрипта загрузки и обработки датасета является формирования тензоров, а также целевых векторов , попарно связных друг с другом (класс **_MapDataset**).

train_dataset: Found 929 files belonging to 4 classes.

validation_dataset: Found 287 files belonging to 4 classes.

train_dataset batch / epoch = 47

validation_dataset batch / epoch = 15 при размере **batch = 20**

(с ним у меня colab работает эффективнее всего (получено методом наблюдений)).

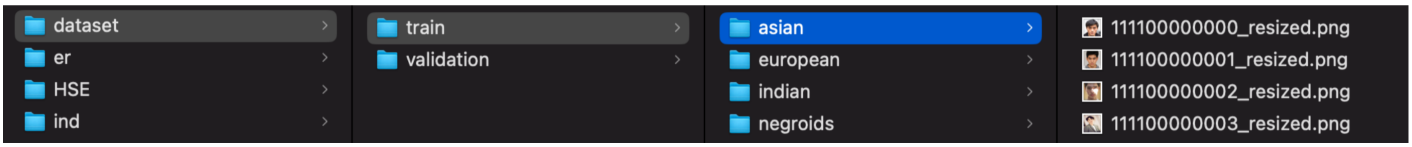

Рис. 1: Архитектура датасета.

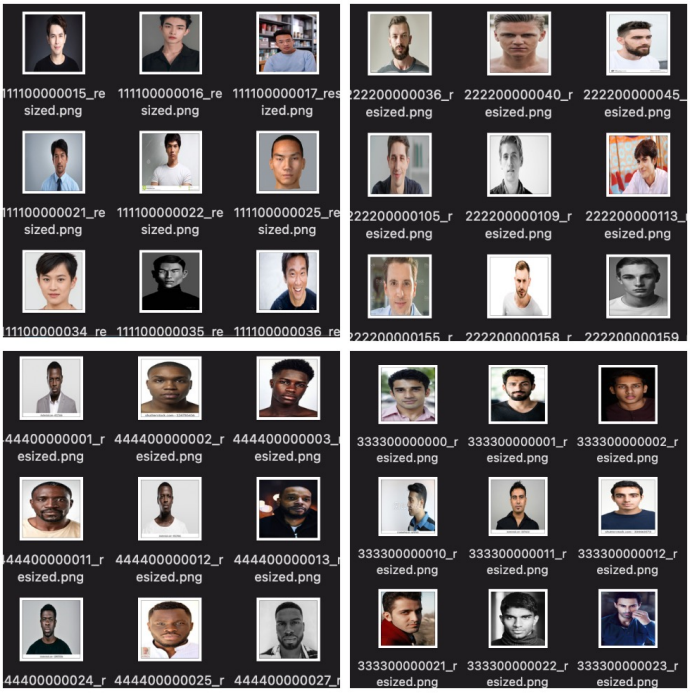

Рис. 2: Пример данных.

Построение архитектуры нейронной сети, обучение

Структура нейронной сети:

Классическая архитектура сети обычно состоит из следующих слоев:

Входной слой: Получает входные данные.

Скрытые слои: Выполняют вычисления и извлекают особенности(признаки) из входных данных.

Выходной слой: Производит окончательный вывод.

Каждый нейрон в скрытых слоях соединен с предыдущим и последующим слоями связями, каждая из которых имеет вес. Все веса регулируются во время процесса обучения, чтобы минимизировать ошибку между выходными данными сети и фактическими значениями.

Кроме весов в структуре нейронной сети существуют баисы(смещения). Они, также как и веса, настраиваются впроцессе обучения сети.

Для классификации людей по 4м расам я решил рассмотреть разные архитектуры нейронных сетей, а также вариации каждой из них.

Типы архитектур:

1. Model 1, based on convolutional neural network (CNN) layers
2. Model 2 (1 large), based on convolutional neural network (CNN) layers
3. Model 3, based on ResNet50
4. Model 4, based on EfficientNetB7

Принципы вариаций:

1. увеличение сколичества скрытых слоев
2. изменение количества нейронов в каждом слое
3. добавление штрафов к функции потерь за величину весов и активаций слоя
4. добавление слоя Dropout для предотвращения переобучения

Для написания каждой модели я использовал встроенный в keras модуль **Sequential()** с помощью которого создаётся модель многослойного персептрона.

Также использовался слой Flatten для выравнивания входа в модель,

активация - "relu".

relu

$$f(x) = \max(0, x)$$

sigmoid

$$f(x) = \frac{1}{1 + e^{-x}}$$

Скрытые слои восновном состояли из слоев класса Dense, с активацией - "relu"

и различным числом нейронов.

Выходной слой каждой модели представляет собой слой класса Dense , имеющий 4 выходных нейрона , так как целевой вектор состоит из 4х классов, а также имеет функцию активации 'sigmoid', чтобы получит значение от 0 до 1 в каждом нейроне.

Таргетированный вектор состоит из 4х значений, каждое из которых лежит на отрезке от 0 до 1.

1я Модель

Архитектура этой , начальной, модели состоит из

Блока их нескольких светрочных слоев

Conv2D, с активационной функцией - "relu"и **MaxPooling2D**

Свертка: 224x224 -> 112x112 -> 56x56 -> 28x28

Слои свертки выделяют на последнем этапе 256 признаков

Блока полносвязной нейронной сети с начальным слоем Flatten, и нескольких(от 3х до 13, в зависимости от конкретной вариации модели) слоев Dense, каждый из которых содержит от 16 до 256 нейронов, и финального слоя вывода - Dense, имеющий 4 выходных нейрона, с активацией 'sigmoid'.

Наилучший результат на тестировании на незнакомых данных - 45%

Данная концепция модели из-за своей простоты и, возможно, чрезмерной свертки показала не лучший результат на тестовой выборке.
В связи с чем была создана следующая концепция модели, в которой были уменьшено количество признаков и степень свертки, но увеличено количество скрытых слоев и нейронов в них.

2я Модель

Архитектура этой , продвинутой, модели состоит из

Блока их нескольких светрочных слоев

Conv2D, с активационной функцией - "relu"и **MaxPooling2D**

Свертка: 224x224 -> 112x112 -> 56x56

Слои свертки выделяют на последнем этапе 128 признаков

2)Блока полносвязной нейронной сети с начальным слоем Flatten, и нескольких(от 3х до 17, в зависимости от конкретной вариации модели) слоев Dense, каждый из которых содержит от 128 до 4096 нейронов, и финального слоя вывода - Dense, имеющий 4 выходных нейрона, с активацией 'sigmoid'. В этой модели к скрытым слоям были добавлены штрафы с раличными размерами от 0.001 до 0.1 .

Наилучший результат на тестировании на незнакомых данных - 62%

Данная концепция модели показала чуть лучший результат на тестовой выборке. Но мне хотелось увеличить точность модели, в связи с чем была создана 3тья концепция модели, использующая в своей архитектуре модель ResNet50, имеющую предобучение.

3я Модель

Архитектура этой модели состоит из

Предобученной на данных ImageNet модели ResNet50.

ResNet50 - это глубокая сверточная нейронная сеть, использует остаточные блоки для решения проблемы исчезающего градиента в глубоких нейронных сетях.

Архитектура ResNet50 состоит из 50 сверточных слоев, разделенных на четыре блока:

Блок 1: 64 сверточных фильтра размером 7x7, шаг 2

Блок 2: 64 остаточных блока с фильтрами размером 3x3

Блок 3: 128 остаточных блока с фильтрами размером 3x3

Блок 4: 256 остаточных блока с фильтрами размером 3x3

Модель завершается слоем глобального среднего пула и полностью связанным слоем с 1000 выходами для классификации по 1000 категориям ImageNet.

2)Блока полносвязной нейронной сети с начальным слоем Flatten, и нескольких(от 3х до 5, в зависимости от конкретной вариации модели) слоев Dense, каждый из которых содержит от 8 до 1000 нейронов, и финального слоя вывода - Dense, имеющий 4 выходных нейрона, с активацией 'sigmoid'.

Наилучший результат на тестировании на незнакомых данных - 45%

Данная концепция модели показала результат хуже на тестовой выборке. Наиболее вероятно это связано с плохим качеством изначального датасета, а также недостаточным из-за ограниченных мощностей обучением модели.

Я решил попробовать втроить более сложную с точки зрения архитектуры модель, по этому сделал 4тую концепцию, использующая в своей архитектуре модель EfficientNetB7, имеющую предобучение.

4я Модель

Архитектура этой модели состоит из

Предобученной на данных ImageNet модели EfficientNetB7.

EfficientNetB7 - это сверточная нейронная сеть, разработанная исследователями из Google AI, имеет архитектуру типа "глубина на ширину в которой глубина сети (количество слоев) и ширина (количество фильтров в каждом слое) масштабируются вместе. Это позволяет модели достичь высокого уровня точности при относительно небольшом количестве параметров и вычислительных затрат.

Архитектура EfficientNetB7 состоит из следующих блоков:

Сверточные слои с расширением MBConv: Эти слои используют свертки с глубиной и шириной, чтобы уменьшить количество параметров без ущерба для точности. Блоки сжимающего возбуждения (SE): Эти блоки динамически калибруют веса каналов, позволяя модели сосредоточиться на важных особенностях.
Инвертированное остаточное соединение: Это соединение позволяет градиентам течь через сеть более эффективно.

Модель , в случае предобучения завершается слоем глобального среднего пула и полностью связанным слоем с 1000 выходами для классификации по 1000 категориям ImageNet, с произвольными весами - любым количеством нейронов, я определил их как 4!, равное 24.

2)Блока полносвязной нейронной сети с начальным слоем Flatten, и нескольких(от 2х до 5, в зависимости от конкретной вариации модели) слоев Dense, каждый из которых содержит от 8 до 1000 нейронов, и финального слоя вывода - Dense, имеющий 4 выходных нейрона, с активацией 'sigmoid'.

Наилучший результат на тестировании на незнакомых данных - 40%

Данная концепция модели показала результат плохой на тестовой выборке.

Наиболее вероятно это связано как с плохим качеством изначального датасета, так и недостаточным из-за ограниченных мощностей обучением модели, 1 эпоха во время обучения которой занимала 1.5 часа. Заранее установить претренинные веса не удалось в связи с отсутствием их на гит , а также гугл ресурсах разарботчика модели.

Всего было создано и обучено 28 различных архитектур , некоторые из них вместе с отчетами по обучению и тестированию приведены в приложени к этому разделу.

Для оптимизации потерь в методе обратного распространения ошибки было выбрано 'categorical$_c rossentropy'$,

метрика - **'accuracy'**

Для обучения нейронной сети через библиотеку Keras был выбран метод fit, позволяющий задать обучающую выборку, валидационную выборку, количество эпох и шаг валидации. Валидацию по тестовой выборке я проводил после каждой эпохи.

```
# Размерность тензора на основе изображения для входных данных в нейронную сеть
input_shape = (img_width, img_height, 3)
model = Sequential()

# Сверточные слои 224x224 -> 112x112 -> 56x56 -> 28x28:
model.add(Conv2D(32, (3, 3), input_shape=input_shape))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2), strides=2))
model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2), strides=2))
model.add(Conv2D(128, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2), strides=2))
model.add(Conv2D(256, (3, 3)))
```

```
# Слои полносвязной нейронной сети:
model.add(Flatten())
model.add(Activation('relu'))
model.add(Dense(784))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(392))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(196))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(98))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(32))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(4))
```

Рис. 3: Итоговая конфигурация 1-ой модели.

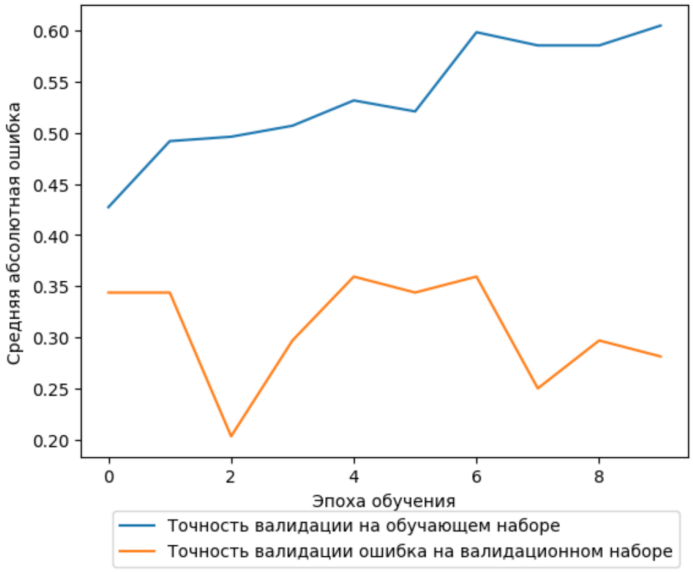

Рис. 4: График обучения 1-ой модели.

```
# Размерность тензора на основе изображения для входных данных в нейронную сеть
input_shape = (img_width, img_height, 3)                    # backend Tensorflow, channels_last
model = Sequential()

# Сверточные слои 224x224 -> 112x112 -> 56x56 :
model.add(Conv2D(32, (3, 3), input_shape=input_shape))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2), strides=2))
model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2), strides=2))
model.add(Conv2D(128, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2), strides=2))

# Слои полносвязной нейронной сети:
model.add(Flatten())
model.add(Dense(1024, kernel_regularizer=regularizers.l2(0.001), bias_regularizer=regularizers.l2(0.001), activity_regularizer=regularizers.l1(0.001)))
model.add(Activation('relu'))
model.add(Dense(2048, kernel_regularizer=regularizers.l2(0.001), bias_regularizer=regularizers.l2(0.001), activity_regularizer=regularizers.l1(0.001)))
model.add(Activat('relu'))
model.add(Dense(2048, kernel_regularizer=regularizers.l2(0.001), bias_regularizer=regularizers.l2(0.001), activity_regularizer=regularizers.l1(0.001)))
model.add(Activation('relu'))
model.add(Dense(2048, kernel_regularizer=regularizers.l2(0.001), bias_regularizer=regularizers.l2(0.001), activity_regularizer=regularizers.l1(0.001)))
model.add(Activation('relu'))
model.add(Dense(1024, kernel_regularizer=regularizers.l2(0.001), bias_regularizer=regularizers.l2(0.001), activity_regularizer=regularizers.l1(0.001)))
model.add(Activation('relu'))
model.add(Dense(512, kernel_regularizer=regularizers.l2(0.001), bias_regularizer=regularizers.l2(0.001), activity_regularizer=regularizers.l1(0.001)))
model.add(Activation('relu'))
model.add(Dense(256, kernel_regularizer=regularizers.l2(0.001), bias_regularizer=regularizers.l2(0.001), activity_regularizer=regularizers.l1(0.001)))
model.add(Activation('relu'))
model.add(Dense(128, kernel_regularizer=regularizers.l2(0.001), bias_regularizer=regularizers.l2(0.001), activity_regularizer=regularizers.l1(0.001)))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(4))
model.add(Activation('sigmoid'))
```

Рис. 5: Итоговая конфигурация 2-ой модели.

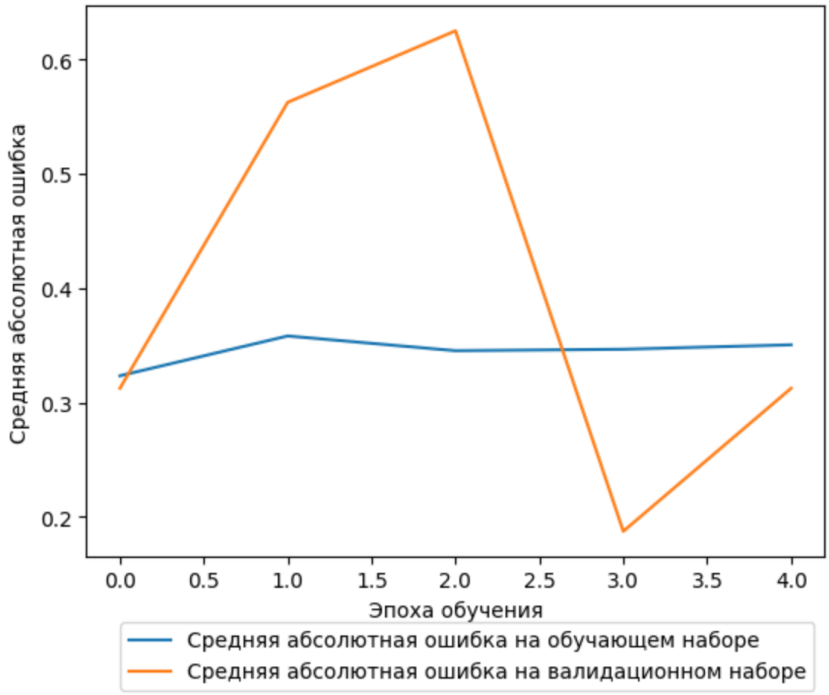

Рис. 6: График обучения 2-ой модели.

```
from tensorflow.keras.applications.resnet50 import preprocess_input, decode_

RS50 = tf.keras.applications.resnet50.ResNet50(
    include_top=True,
    weights='imagenet',
    input_tensor=None,
    input_shape=None,
    pooling=None,
    classes=1000
)
input_shape = RS50
model = Sequential()
model.add(input_shape)
model.add(Flatten())
model.add(Activation('relu'))
model.add(Dense(1000))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(784))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(784))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(512))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(256))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(128))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(64))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(32))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(4))
model.add(Activation('sigmoid'))
```

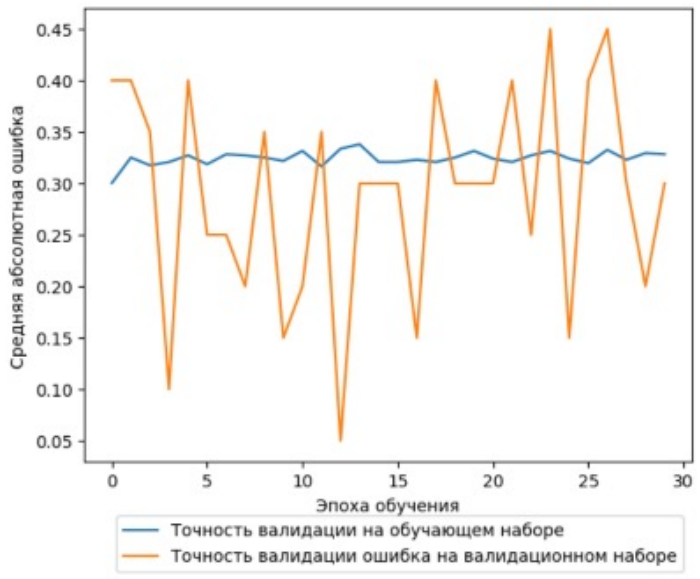

Рис. 7: Итоговая конфигурация 3-ей модели.

```
EfNetD7 = keras.applications.EfficientNetB7(
    include_top=True,
    weights="imagenet",
    input_tensor=None,
    input_shape=None,
    pooling=None,
    classes=1000,
    classifier_activation="softmax"
)

# Слои полносвязной нейронной сети:
input_shape = MyModuleWrapper(EfNetD7)

Flatten_layer = MyModuleWrapper(Flatten())
D_layer01 = MyModuleWrapper(Dense(1000, activation='relu'))
D_layer02 = MyModuleWrapper(Dense(256, activation='relu'))
D_layer03 = MyModuleWrapper(Dense(128, activation='relu'))
D_layer_mod01 = MyModuleWrapper(Dense(784, activation='relu', kernel_regularizer=regularizers.l2(0.01),
                                      bias_regularizer=regularizers.l2(0.01), activity_regularizer=regularizers.l1(0.01)))
D_layer_mod02 = MyModuleWrapper(Dense(256, activation='relu', kernel_regularizer=regularizers.l2(0.01),
                                      bias_regularizer=regularizers.l2(0.01), activity_regularizer=regularizers.l1(0.01)))
D_layer_mod03 = MyModuleWrapper(Dense(128, activation='relu', kernel_regularizer=regularizers.l2(0.01),
                                      bias_regularizer=regularizers.l2(0.01), activity_regularizer=regularizers.l1(0.01)))
Drop_layer = MyModuleWrapper(Dropout(0.5))
Output_layer = MyModuleWrapper(Dense(4, activation='sigmoid'))
```

```
    model = Sequential([
        input_shape,
        Flatten_layer,
        D_layer01,
        Drop_layer,
        D_layer_mod01,
        #Drop_layer,
        #D_layer02,
        #Drop_layer,
        #D_layer_mod02,
        #Drop_layer,
        D_layer03,
        #D_layer_mod03,
        Drop_layer,
        Output_layer])
```

Рис. 8: Итоговая конфигурация 4-ой модели.

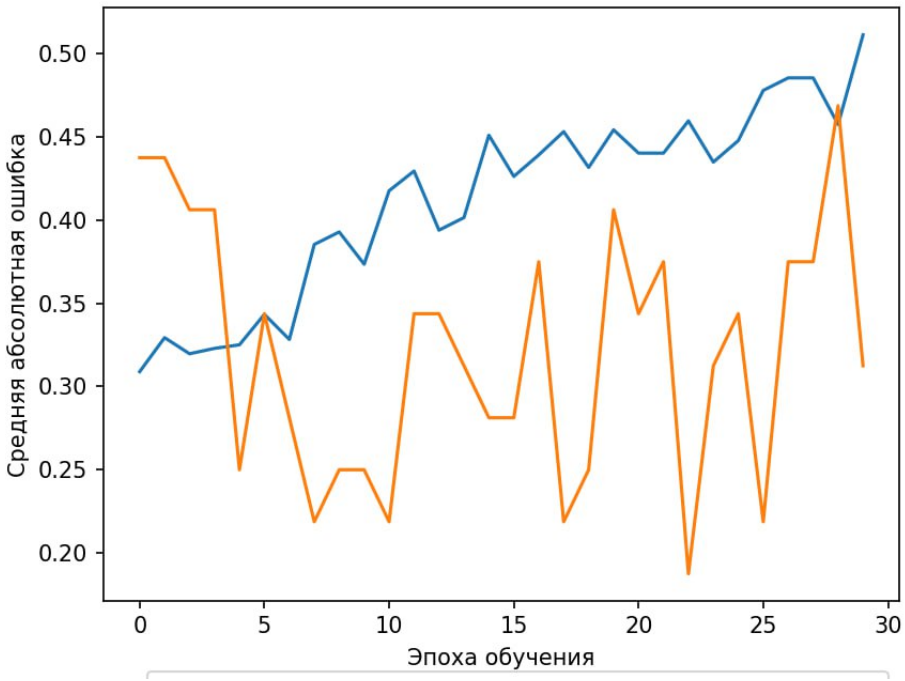

Рис. 9: График обучения 4-ой модели.

Результаты и выводы

Относительно низкая точность определения расы связана с весьма сложными для классификации данными в датасете .

Люди, на фотографиях в датасете, в абсолютном большинстве представители смешанных между собой рас , что сильно усложняет для нейронной сети задачу выделения каких-либо конкретных признаков.

Снимки в датасете сделаны с разным освещением, шумами , из-за этого становится практически невозможным выделения признаков цвета кожи.

Снимки в датасете сделаны не строго в одной проекции , большинство из них - под каким-либо углом, становится нейросети сложнее их распознавать .

На многих фотографиях в датасете присутствуют вотермарки / другие помехи.

Подведение фотографий под единый размер создало проблему геометрического искажения / потери частей снимков , что приводит к снижению количества выделенных нейросетью признаков .

В совокупности все вышеперечисленное приводит к потери точности на этапе тестирования нейронной сети как на обучающей , так и на незнакомой ей выборке , хоть на обучающей выборке и достигались результаты свыше 99% .

Рис. 10: Примеры. Попарно схожие фотографии разных рас.

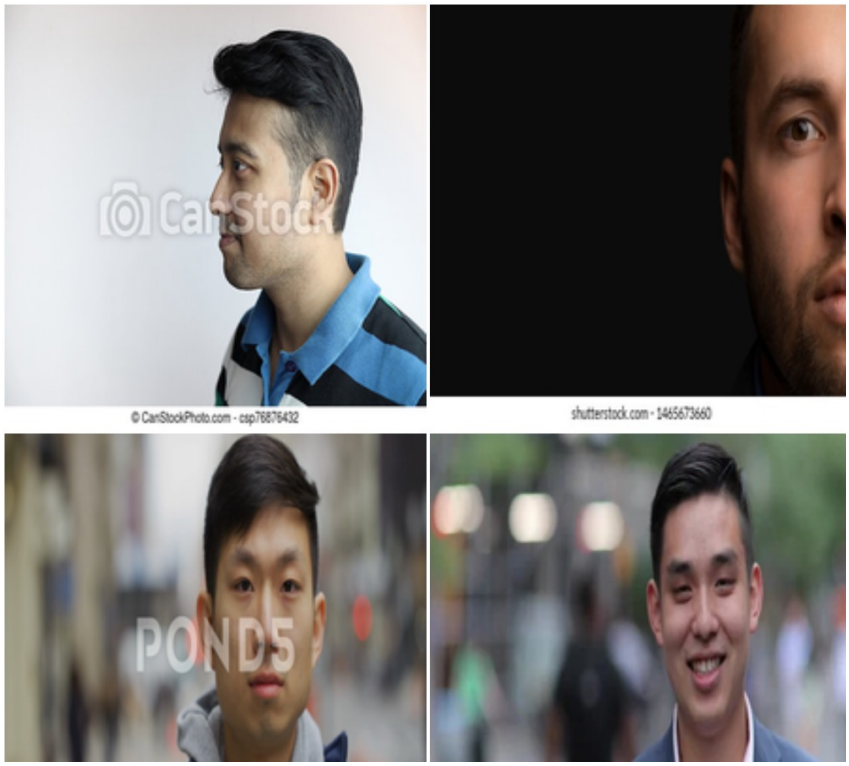

Рис. 11: Примеры. Искаженные, снятые с разного ракурса ,зашумленные фотографии.

Заключение

В конечном итоге поставленная в проекте цель была достигнута – разработаны модели, способные угадывать расу человека по фотографии. В процессе создания моделей я узнал о принципах работы и построения нейронных сетей для классификации.

В перспективе возможна доработка моделей на более продвинутых мощностях, искуственное формирование заведомо правильного(по описанным критериям выше) датасета, и последующее их обучение.

Работа над проектом дала мне знания о работе и устройстве нейронных сетей , а также повысила мои навыки работы в Google Colab и среде разработки Python

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

0. EfficientNetB7

1. NasNetLarge

2. ResNet

3. фрхитектуры синтаксис

4. Tensorflow

5. yolo lite

6. U^2*net*

7. keras