Министерство образования и науки Российской Федерации
Федеральное государственное образовательное учреждение высшего образования
«Национальный исследовательский университет «МЭИ»
Институт ИВТ
Кафедра ПМИИ

КУРСОВАЯ РАБОТА
по дисциплине «Программная инженерия»
на тему:
«Разработка программы для решения задачи определения атрибутов личности по изображению лица с использованием свёрточных нейронных сетей»

Группа: А-05-19
Студент: Гусамова Камила Линаровна
Научный руководитель: Михайлов Илья Сергеевич

Москва 2021

Оглавление

Введение

Задача автоматического определения атрибутов личности по изображению лица сегодня приобретает все большее значение, так как человек является наиболее часто встречающимся объектом на фотографиях и видео. Основными атрибутами личности, определение которых имеет первостепенное значение, являются возраст человека, его этническая принадлежность и пол [4]. Существует множество задач, в которых именно эти характеристики являются наиболее значимыми. К ним относятся, например [5]:

- Автоматическая аннотация результатов поиска в Интернете;
- Анализ аудитории при проведении маркетинговых мероприятий;
- Формирование адаптивной рекламы;
- Создание адаптивного человеко-машинного интерфейса;
- Контроль доступа личности к информации или определенным действиям.

Широкая практическая применимость определяет *актуальность* поставленной задачи и исследования методов её решения.

Целью данной работы является разработка программного продукта, решающего задачу определения базовых атрибутов личности (возраста, этнической принадлежности, пола) по изображению лица с использованием свёрточных нейронных сетей в соответствии с моделью жизненного цикла программного обеспечения.

Для достижения указанной цели необходимо выполнить следующие *задачи*:

1. Изучить теоретические основы поставленной задачи, нейронных сетей и их применения в контексте обработки изображений;

2. Реализовать три нейронные сети – для определения возраста, этнической принадлежности и пола;

3. Провести тестирование и проанализировать полученные результаты.

1. Анализ задачи

1.1. Постановка задачи

Задача определения атрибутов по изображению является задачей классификации, сформулируем её [2].

Задано конечное множество классов и имеется множество объектов, для конечного подмножества которых известно, к какому классу они относятся. Это подмножество называется обучающей выборкой. Классовая принадлежность остальных объектов неизвестна. Требуется построить алгоритм, способный классифицировать произвольный объект из исходного множества.

Классифицировать объект значит указать номер (или наименование класса), к которому относится данный объект.

В задачи классификации изображений лиц объекты – это фотографии людей.

Теперь запишем формальную постановку задачи классификации изображений лиц.

Пусть $X = \{x_1, x_2, \ldots, x_n\}$ – множество фотографий людей. Каждая фотография $x \in X$ представляет собой матрицу пикселей с числовыми значениями от 0 до 255.

$Y = \{y_1, y_2, \ldots, y_n\}$ – множество меток классов.

$y^*: X \to Y$ – неизвестная целевая зависимость, значения которой известны только на объектах обучающей выборки $X^m = \{(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m),\}$.

Требуется построить алгоритм $a: X \to Y$, способный классифицировать произвольный объект $x \in X$.

1.2. Набор данных

Для решения поставленной задачи используем набор данных «Age, gender and ethnicity (face data) CSV», расположенный в свободном доступе на сайте Kaggle – онлайн-сообществе специалистов по обработке данных и машинному обучению.

Датасет представляет собой CSV (Comma-Separated Values – значения, разделенные запятыми) файл из 27305 строк и 5 столбцов. Каждая строка включает в себя черно-белое изображение лица человека в виде матрицы пикселей размера 48 на 48, название изображения, а также информацию о:

- Возрасте (значения от 1 до 120);
- Этнической принадлежности (значения от 0 до 4);
- Поле (0 – мужской, 1 – женский).

Соответственно, набор данных позволяет нам произвести классификацию по трём параметрам: возрасту, этнической принадлежности и полу.

Изображения подобраны таким образом, что лицо занимает практически все пространство на фотографии, поэтому нам не понадобится дополнительно обрабатывать их. Метки представлены в виде чисел.

Примеры изображений можно увидеть на Рис. 1.

Рис. 1. Примеры изображений, содержащихся в датасете, с выводом меток

1.3. Методы решения задачи классификации

Задача классификации решается с помощью аналитических моделей, называемых классификаторами. Разработано большое количество различных классификаторов, в целом их можно разделить на две категории: статистические и использующие методы машинного обучения. К статистическим классификаторам относят Байесовский классификатор, логистическую регрессию, дискриминантный анализ; к использующим методы машинного обучения – деревья решений, метод ближайших соседей, машины опорных векторов и др.

Согласно исследованию, проведённому в [3], нейросетевые алгоритмы способны составить конкуренцию классическим методам распознавания образов, указанным выше. Нейронные сети корректнее использовать для более сложных задач, для задачи классификации линейно неразделимых классов и при работе с большими объёмами данных – поставленная задача соответствует приведённым критериям.

2. Теория

2.1. Биологический прототип

Развитие искусственных нейронных сетей вдохновлено биологией [7]. Нейрон является структурной единицей нервной системы, предназначенной для приёма, обработки, хранения и вывода информации с помощью электрохимических сигналов. Он состоит из тела клетки, дендритов и одного аксона.

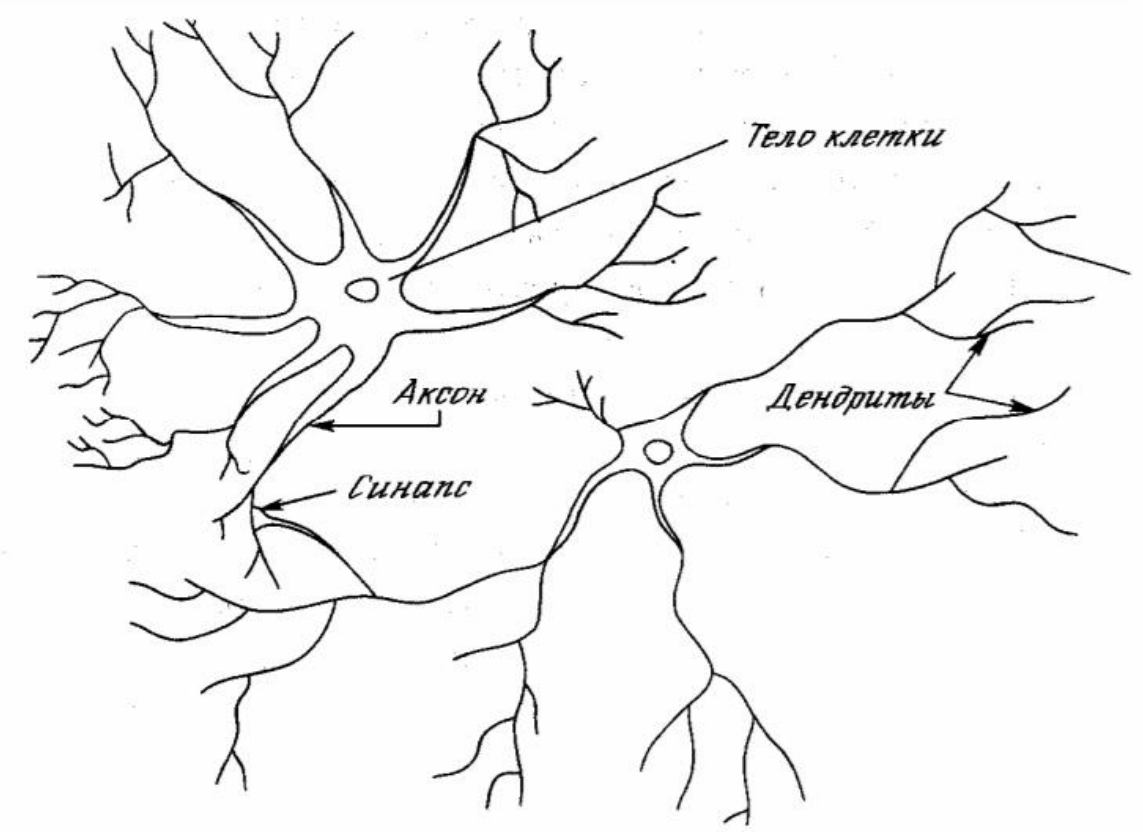


Рис. 2. Биологический нейрон

На Рис. 2 показана структура пары типичных биологических нейронов. Дендриты идут от тела нервной клетки к другим нейронам, где они принимают сигналы через контакты, называемые синапсами. Принятые синапсом входные сигналы подводятся к телу нейрона. Здесь они суммируются, причём одни входы пытаются возбудить нейрон, другие – воспрепятствовать его возбуждению. Когда суммарное возбуждение в теле нейрона превышает некоторый порог, нейрон возбуждается, посылая по аксону сигнал другим нейронам.

2.2. Искусственный нейрон

Искусственный нейрон имитирует свойства биологического нейрона. На вход искусственного нейрона поступают сигналы, которые является или исходными данными, или выходными сигналами других нейронов. Каждый вход умножается на соответствующий вес, аналогичный синаптической силе, и все произведения суммируются, определяя уровень активации нейрона. На Рис. 3 представлена модель, реализующая эту идею.

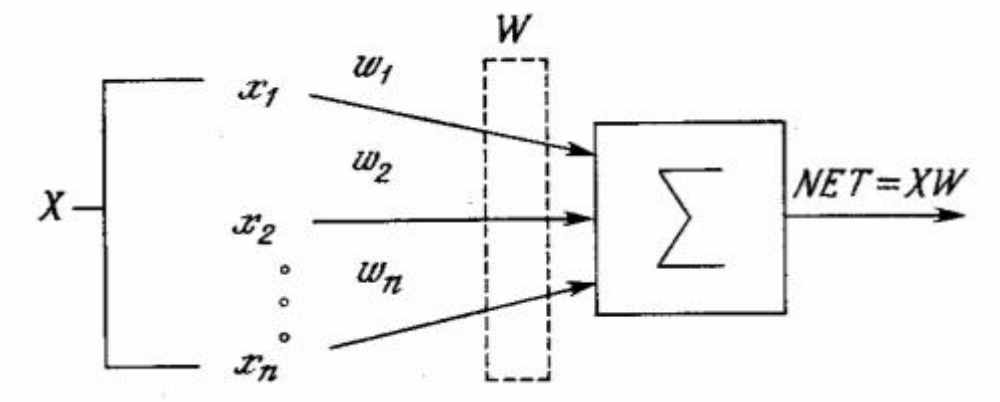


Рис. 3. Искусственный нейрон

Здесь множество входных сигналов, обозначенных $x_1, x_2, \ldots, x_n$ (вектор X) поступает на искусственный нейрон. Каждый сигнал умножается на соответствующих вес $w_1, w_2, \ldots, w_n$ (вектор W) и поступает на суммирующий блок, обозначенный Σ. Суммирующий блок складывает взвешенные входы алгебраически, создавая вход, который мы будем называть NET. В векторных обозначениях это может быть записано следующим образом:

$$NET = XW$$

2.3. Функция активации

Сигнал NET далее преобразуется функцией активации F и даёт выходной нейронный сигнал OUT. В качестве функции активации можно использовать:

- Логистическую функцию (сигмоиду);

$$\mathrm{F(x)} = \sigma(\mathrm{x}) = \frac{1}{1 + \mathrm{e}^{-\mathrm{x}}}$$

- ReLU (Rectified linear unit – полулинейный элемент);

$$\mathrm{F(x)} = \begin{cases} 0, \mathrm{x} < 0 \\ \mathrm{x}, \mathrm{x} \geq 0 \end{cases}$$

- Гиперболический тангенс;

$$\mathrm{F(x)} = \mathrm{t}h(\mathrm{x}) = \frac{\mathrm{e}^{\mathrm{x}} - \mathrm{e}^{-\mathrm{x}}}{\mathrm{e}^{\mathrm{x}} + \mathrm{e}^{-\mathrm{x}}}$$

- Softmax – обобщение логистической функции для многомерного случая;

$$\mathrm{F_i}(\vec{\mathrm{x}}) = \frac{\mathrm{e}^{\mathrm{x_i}}}{\sum_{\mathrm{j}=1}^{\mathrm{N}} \mathrm{e}^{\mathrm{x_j}}}, \mathrm{i} = 1, \ldots, \mathrm{N}$$

Таким образом, $\mathrm{OUT} = \mathrm{F(NET)}$.

2.4. Искусственные нейронные сети

Хотя и один нейрон способен выполнять простейшие процедуры, наибольший интерес представляют нейронные сети – системы соединённых и взаимодействующих между собой искусственных нейронов. На Рис. 4 изображена схема элементарного перцептрона – модели нейронной сети, представляющей собой основу для всех последующих видов искусственных нейронных сетей.

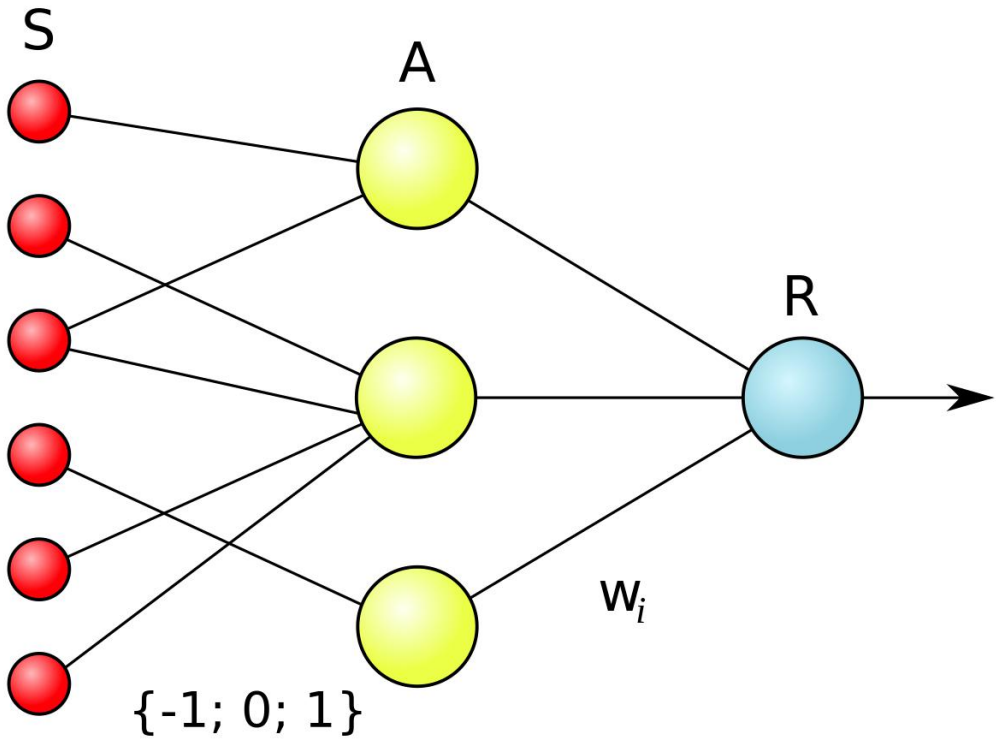


Рис. 4. Логическая схема элементарного перцептрона

Существует множество разнообразных видов нейронных сетей, классифицирующихся по [9]:

Количеству слоёв:

- Однослойные;
- Многослойные.

Характеру обучения:

- Обучение с учителем – выходное пространство решений известно;
- Обучение без учителя – нейронная сеть самостоятельно формирует выходное пространство решений на основе входных воздействий;
- Обучение с подкреплением – система назначения штрафов и поощрений от среды.

Характеру связей:

- Нейронные сети прямого распространения (однонаправленные) – связи направлены строго от входных нейронов к выходным;
- Рекуррентные нейронные сети (с обратными связями) – сигнал с выходных нейронов или нейронов скрытого слоя частично передаётся обратно на входы нейронов входного слоя.

И это далеко не все разновидности нейронных сетей. Однако совершенно точно известно, что наиболее подходящими для работы с изображениями являются свёрточные нейронные сети.

3. Свёрточные нейронные сети

С появлением больших объёмов данных и вычислительных возможностей нейронные сети стали применяться более активно. Особую популярность получили свёрточные нейронные сети, направленные на эффективную обработку изображений и распознавание образов. Своё название архитектура сети получила из-за наличия операции свёртки, суть которой состоит в том, что каждый фрагмент изображения умножается на матрицу (ядро) свёртки поэлементно, а результат суммируется и записывается в соответствующую позицию выходного изображения. Благодаря этому свёрточная нейронная сеть имеет следующие преимущества [6]:

- Большая временная эффективность по сравнению, например, с перцептроном за счёт меньшего количество настраиваемых параметров;
- Улучшенные способности выделения отдельных элементов на изображении (углы, кривые, прямые, яркие области и т.д.) за счёт использования нескольких карт признаков на одном слое;
- Способность формирования высокоуровневых признаков на основе низкоуровневых в пределах одного класса за счёт использования ядер свёртки сравнительно небольшого размера вместо соединения нейронов двух соседних слоёв, как у полносвязного перцептрона.

3.1. Архитектура свёрточной нейронной сети

Свёрточная нейронная сеть представляет собой чередование свёрточных слоёв (convolution layers), субдискретизирующих слоёв (subsampling layers или pooling layers, слоёв подвыборки) и полносвязных слоёв (fully-connected layers) на выходе. Все три вида слоёв могут чередоваться в произвольном порядке.

В свёрточном слое нейроны, которые используют одни и те же веса, объединяются в карты признаков (feature maps), а каждый нейрон карты признаков связан с частью нейронов предыдущего слоя. При вычислении сети получается, что каждый нейрон выполняет свёртку некоторой области предыдущего слоя.

Пример архитектуры свёрточной нейронной сети представлен на Рис. 5.

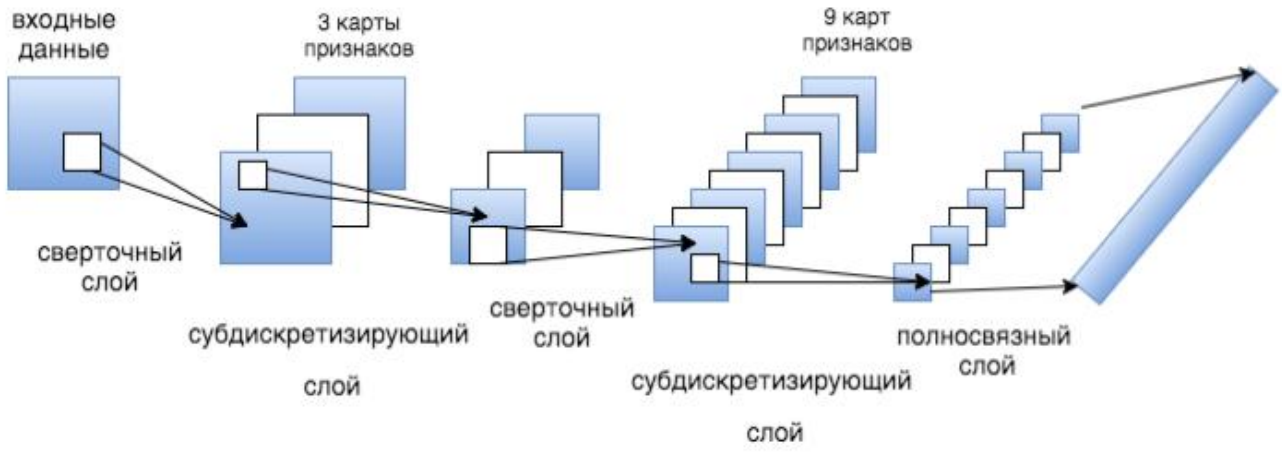


Рис. 5. Архитектура свёрточной нейронной сети

3.2. Полносвязный слой

Слой, в котором каждый нейрон соединён со всеми нейронами на предыдущем уровне, причём каждая связь имеет свой весовой коэффициент. На Рис. 6 показан пример полносвязного слоя.

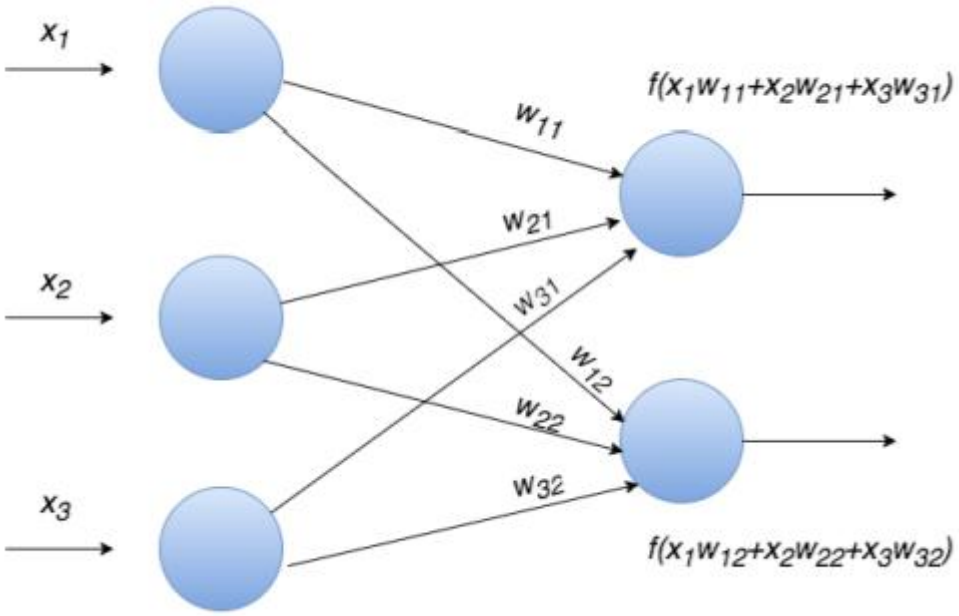


Рис. 6. Полносвязный слой

3.3. Свёрточный слой

В отличие от полносвязного слоя, в свёрточном слое нейрон соединён лишь с ограниченным количеством нейронов предыдущего уровня, т.е. свёрточный слой аналогичен применению операции свёртки, где используется лишь матрица весов небольшого размера (ядро свёртки), которую «двигают» по всему обрабатывающему слою.

Ещё одна особенность свёрточного слоя состоит в том, что он немного уменьшает изображение за счёт краевых эффектов.

На Рис. 7 показан пример свёрточного слоя с ядром свёртки размера 3 на 3.

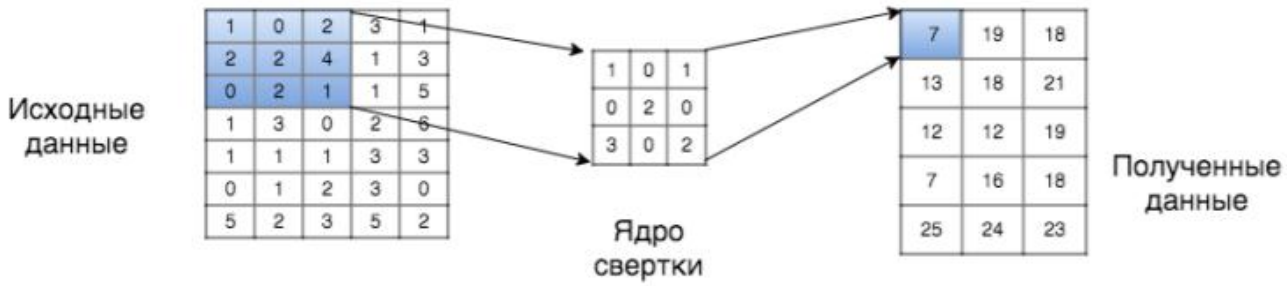


Рис. 7. Свёрточный слой

3.4. Субдискретизирующий слой

Слои этого типа выполняют уменьшение размерности. Это можно делать разными способами, но зачастую используется метод выбора максимального элемента (max-pooling) – вся карта признаков разделяется на ячейки, из которых выбираются максимальные по значению.

На Рис. 8 показан пример субдискретизирующего слоя с методом выбора максимального элемента.

Рис. 8. Субдискретизирующий слой

3.5. Dropout слой

Dropout слой применяется для борьбы с переобучением в нейронных сетях. Переобучение – явление, при котором построенная модель хорошо объясняет только примеры из обучающей выборки, адаптируясь к обучающим примерам, вместо того, чтобы учиться классифицировать примеры, не участвующие в обучении.

Dropout регуляция заключается в изменении структуры сети: каждый нейрон исключается с некоторой вероятностью p. «Исключение» нейрона означает, что при любых входных данных или параметрах он возвращает 0. Таким образом, происходит усреднение моделей внутри нейронной сети, в результате чего более обученные нейроны получают в сети больший вес.

Этот приём значительно увеличивает скорость обучения, повышает качество обучения на тренировочных данных и предсказаний модели на новых тестовых данных.

3.6. Обучение свёрточной нейронной сети

Для обучения свёрточной нейронной сети используется специфическая версия алгоритма обратного распространения ошибки, который относится к методам обучения с учителем [1].

Рассмотрим обучение полносвязных слоёв. Ошибка формируется на последнем слое нейронов свёрточной сети и определяется как разность между выходным сигналом сети (значениями нейронов последнего слоя) y^* и эталоном y:

$$\gamma_j = y_j^* - y_j, j = 1,2,\dots,n$$

Далее происходит изменение значений весов по формуле:

$$w_{ij}(t+1) = w_{ij}(t) - \alpha\gamma_j F'(s_j)y_j^*, j = 1,2,\dots,n,$$

где $t, t+1$ – моменты времени до и после изменения весов;

$F(s_j)$ – значение функции активации от взвешенной суммы s_j;

α – скорость обучения сети, принимающая значения в промежутке $(0, 1]$.

Ошибка для скрытого слоя с индексом i вычисляется через ошибки следующего за ним слоя с индексом j как:

$$\gamma_i = \sum_j \gamma_j F'(s_j)w_{ij}$$

Рассмотрим обучение свёрточного и субдискретизирующего слоёв.

Обратное распространение ошибки субдискретизирующего слоя зависит от функции пулинга. При использовании метода выбора максимального элемента ошибка присваивается тому нейрону блока, с которого было взято минимальное значение по блоку.

Основой обратного распространения ошибки по свёрточному слою служит операции свёртки. При передаче матрицы ошибок от субдискретизирующего слоя к свёрточному производится обратная свёртка, при выполнении которой получается матрицы большего размера, чем входная. Затем производится свёртка входа свёрточного слоя с матрицей ошибок данного слоя, повёрнутой на 180 градусов. Далее полученная матрица умножается на скорость обучения α и вычитается из ядра свёртки данного слоя.

3.7. Функция потерь

Введём обозначения: пусть X – множество описаний объектов, Y – множество допустимых ответов. Предполагается, что существует неизвестная целевая зависимость – отображение $y^*: X \to Y$, значения которой известны только на объектах конечной обучающей выборки $X^m = \{(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m),\}$.

Вводится функция потерь $E(y, y')$, характеризующая величину отклонения ответа y от правильного ответа $y' = y^*(x)$ на произвольном объекте $x \in X$. Она позволяет оценить качество работы нейронной сети и используется в процессе обучения, описанного выше.

4. Проектирование

Для реализации решения поставленной задачи используем язык программирования Python и библиотеки TensorFlow, Keras, Scikit-Learn – открытые программные библиотеки для машинного обучения и работы с искусственными нейронными сетями.

В качестве среды разработки используем Kaggle Kernels – облачную вычислительную среду разработки. Она содержит все необходимые нам библиотеки и не требует дополнительной настройки.

Решение будет выполнено в три этапа:

1. Загрузка, анализ и подготовка набора данных;

2. Построение трёх свёрточных нейронных сетей по каждой категории классификации, обучение;

3. Применение созданных нейронных сетей для классификации примеров из тестовой выборки. При получении точности 70% и выше будем считать задачу выполненной.

Итоговый продукт представляет собой программу, объединяющую код и вывод в виде единого документа. Её запуск возможен так же в Kaggle или в средах Jupyter Notebook или Google Colab.

5. Реализация

Подключим необходимые библиотеки

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Input, Conv2D, MaxPooling2D, Flatten, Dropout,
Dense
import time
```

5.1. Этап 1

Выполним загрузку набора данных из CSV-файла, выведем пять первых строк

```
data = pd.read_csv("../input/age-gender-and-ethnicity-face-data-
csv/age_gender.csv")
print(data.shape)
data.head()
```

```
(23705, 5)
```

	age	ethnicity	gender	img_name	pixels
0	1	2		020161219203650636.jpg.chip.jpg	129 128 128 126 127 130 133 135 139 142 145 14...
1	1	2		020161219222752047.jpg.chip.jpg	164 74 111 168 169 171 175 182 184 188 193 199...
2	1	2		020161219222832191.jpg.chip.jpg	67 70 71 70 69 67 70 79 90 103 116 132 145 155...
3	1	2		020161220144911423.jpg.chip.jpg	193 197 198 200 199 200 202 203 204 205 208 21...
4	1	2		020161220144914327.jpg.chip.jpg	202 205 209 210 209 209 210 211 212 214 218 21...

Столбец, содержащий названия изображений не имеет отношения к решаемой задаче, поэтому уберём его

```
data = data.drop('img_name', axis=1)
print(data.shape)
data.head()
```

```
(23705, 4)
```

	age	ethnicity	gender	pixels
0	1	2		0129 128 128 126 127 130 133 135 139 142 145 14...
1	1	2		0164 74 111 168 169 171 175 182 184 188 193 199...
2	1	2		067 70 71 70 69 67 70 79 90 103 116 132 145 155...
3	1	2		0193 197 198 200 199 200 202 203 204 205 208 21...
4	1	2		0202 205 209 210 209 209 210 211 212 214 218 21...

Обратим внимание на то, что объективно задача классификации по возрасту является сложной – даже человек не всегда может определить возраст другого человека с приемлемой точностью. Для того, чтобы немного облегчить задачу, разделим изображения на 10 категорий по возрасту (метка 0 – от 1 до 9 лет, метка 1 – от 10 до 19 лет, метка 2 – от 20 до 29 лет и т.д. до метки 9 – от 90 до 120 лет)

```
data.loc[(data['age'] > 0) & (data['age'] <= 10), 'age'] = 0
data.loc[(data['age'] > 10) & (data['age'] <= 20), 'age'] = 1
data.loc[(data['age'] > 20) & (data['age'] <= 30), 'age'] = 2
data.loc[(data['age'] > 30) & (data['age'] <= 40), 'age'] = 3
data.loc[(data['age'] > 40) & (data['age'] <= 50), 'age'] = 4
data.loc[(data['age'] > 50) & (data['age'] <= 60), 'age'] = 5
data.loc[(data['age'] > 60) & (data['age'] <= 70), 'age'] = 6
data.loc[(data['age'] > 70) & (data['age'] <= 80), 'age'] = 7
data.loc[(data['age'] > 80) & (data['age'] <= 90), 'age'] = 8
data.loc[(data['age'] > 90) & (data['age'] <= 120), 'age'] = 9
data.head()
```

	age	ethnicity	gender	pixels
0	0	2	0	129 128 128 126 127 130 133 135 139 142 145 14...
1	0	2	0	164 74 111 168 169 171 175 182 184 188 193 199...
2	0	2	0	67 70 71 70 69 67 70 79 90 103 116 132 145 155...
3	0	2	0	193 197 198 200 199 200 202 203 204 205 208 21...
4	0	2	0	202 205 209 210 209 209 210 211 212 214 218 21...

Проанализируем набор данных, построив графики

```
plt.figure(figsize=(40,10))
sns.countplot(x=data['age'])
plt.title('Age countplot', fontsize=20)
```

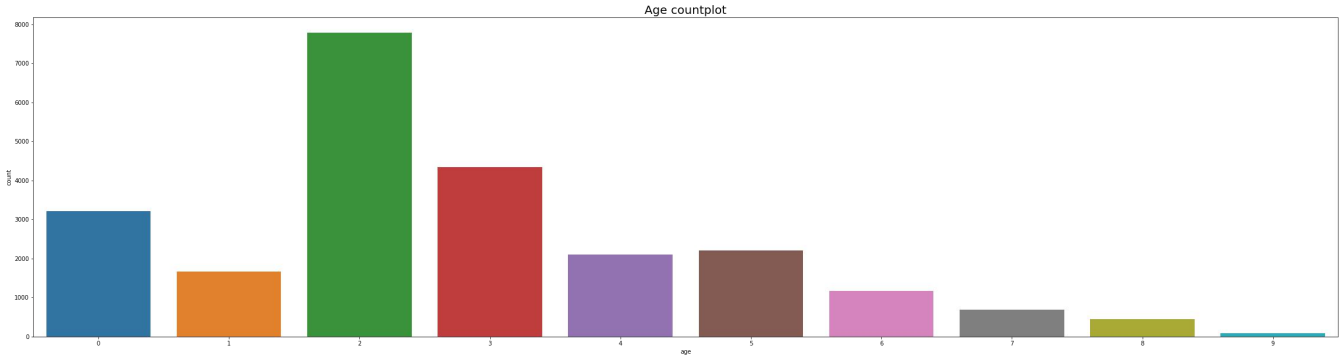


Рис. 9. График, показывающий количество изображений по возрастам

```
fig, axes = plt.subplots(1, 2, figsize=(20, 10), sharey=True)
sns.countplot(ax=axes[0], x=data['ethnicity'])
axes[0].set_title('Ethnicity countplot')
sns.countplot(ax=axes[1], x=data['gender'])
axes[1].set_title('Gender countplot')
```

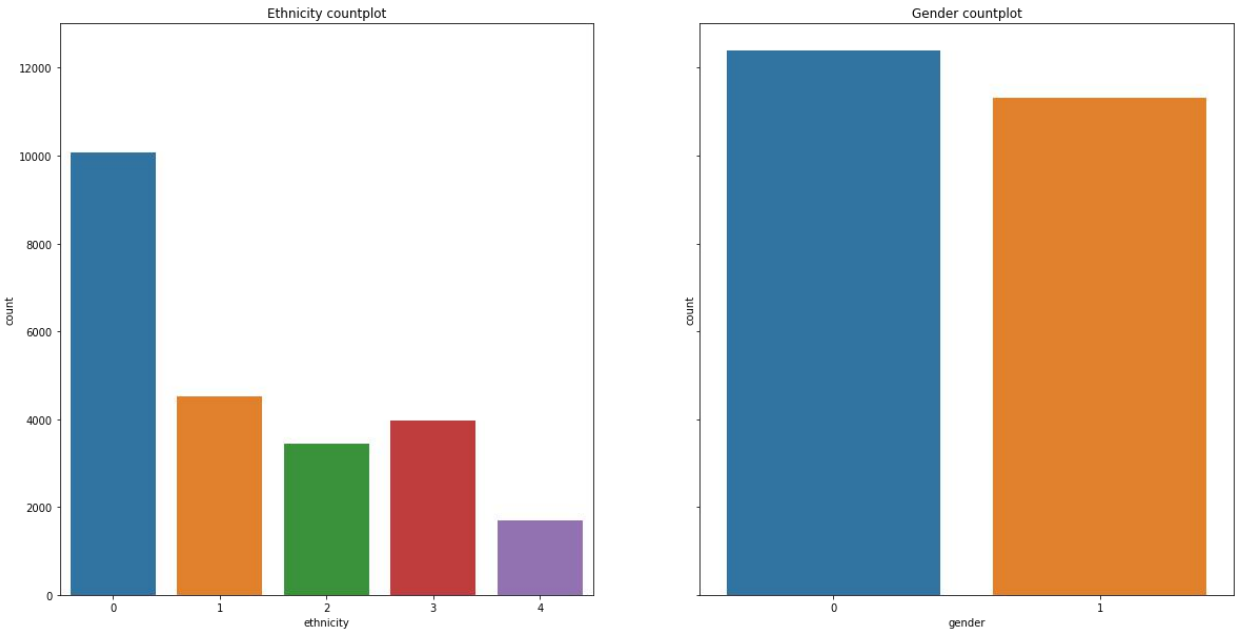


Рис. 10. Графики, показывающие количество изображений по этнической принадлежности и полу

Преобразуем столбец, содержащий значения пикселей, в матрицу размера 48 на 48

```
data['pixels'] = data['pixels'].map(lambda x: np.array(x.split(' '),
dtype=np.float32).reshape(48, 48))
```

Выведем пять случайных изображений с указанием возраста, этнической принадлежности и пола

```
fig, axes = plt.subplots(1, 5, figsize=(20, 10))
 for i in range(5):
    random_face = np.random.choice(len(data))
    age = data['age'][random_face]
    ethnicity = data['ethnicity'][random_face]
    gender = data['gender'][random_face]
    axes[i].set_title('Age group: {0}, Ethnicity: {1}, Gender:
{2}'.format(age, ethnicity, gender))
    axes[i].imshow(data['pixels'][random_face], cmap='gray')
    axes[i].axis('off')
```

Рис. 11. Пять случайных изображений с указанием возрастной группы, этнической принадлежности и пола

Задаём матрицу пикселей x, нормализуем её значения; input_shape для дальнейшего использования

```
x = np.array(data['pixels'].to_list())
x = x.reshape(x.shape[0], 48, 48, 1) / 255
input_shape = x.shape[1:]
```

Задаём массивы меток y1, y2, y3. При этом преобразуем y1 и y2 в кодировку one-hot (к примеру, метка «3» для этнической принадлежности превращается в [0. 0. 0. 1. 0. 0.])

```
y1 = to_categorical(np.array(data['age']), 10)
y2 = to_categorical(np.array(data['ethnicity']), 5)
y3 = np.array(data['gender'])
```

Разбиваем датасет на три обучающих (70%) и тестовых набора (30%)

```
x_train1, x_test1, y_train1, y_test1 = train_test_split(x, y1, test_size =
0.3, random_state = 42)
x_train2, x_test2, y_train2, y_test2 = train_test_split(x, y2, test_size =
0.3, random_state = 42)
x_train3, x_test3, y_train3, y_test3 = train_test_split(x, y3, test_size =
0.3, random_state = 42)
```

5.2. Этап 2

Нейронная сеть для классификации по возрасту. Модель 1

Построим НС с двумя свёрточными, двумя субдискретизирующими, двумя полносвязными слоями и одним слоем Dropout. Не забываем, что выходной полносвязный слой должен включать в себя столько нейронов, сколько исходных классов мы имеем – в данном случае 10.

В качестве функции активации выходного слоя используем Softmax. Функция потерь – Categorical Crossentropy как функция наиболее подходящая для мультиклассификационной задачи.

```
model_age = Sequential()

model_age.add(Conv2D(8, kernel_size=(3, 3), input_shape=input_shape,
padding='same', activation='relu'))
model_age.add(MaxPooling2D(pool_size=(2, 2), strides=2))
model_age.add(Conv2D(16, kernel_size=(3, 3), padding='same',
activation='relu'))
model_age.add(MaxPooling2D(pool_size=(2, 2), strides=2))

model_age.add(Flatten())

model_age.add(Dense(16, activation='relu'))
model_age.add(Dropout(0.3))

model_age.add(Dense(10, activation='softmax'))

model_age.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])

model_age.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 48, 48, 8)         80
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 24, 24, 8)         0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 24, 24, 16)        1168
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 12, 12, 16)        0
_________________________________________________________________
flatten (Flatten)            (None, 2304)              0
_________________________________________________________________
dense (Dense)                (None, 16)                36880
```

```
_________________________________________________________________
dropout (Dropout)            (None, 16)                0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                170
=================================================================
Total params: 38,298
Trainable params: 38,298
Non-trainable params: 0
_________________________________________________________________
```

Обучим НС на 25 эпохах, в качестве валидационной выборки возьмём тестовую

```
history_age = model_age.fit(x_train1, y_train1, batch_size=128, epochs=25,
validation_data=(x_test1, y_test1))
```

Нейронная сеть для классификации по этнической принадлежности. Модель 2

Построим НС с двумя свёрточными, двумя субдискретизирующими, двумя полносвязными слоями и одним слоем Dropout. Выходной полносвязный слой включается в себя 5 нейронов.

Задача аналогична предыдущей, поэтому так же в качестве функции активации выходного слоя используем Softmax; функция потерь – Categorical Crossentropy.

```
model_ethnicity = Sequential()

model_ethnicity.add(Conv2D(2, kernel_size=(3, 3), input_shape=input_shape,
padding='same', activation='relu'))
model_ethnicity.add(MaxPooling2D(pool_size=(2, 2), strides=2))
model_ethnicity.add(Conv2D(4, kernel_size=(3, 3), padding='same',
activation='relu'))
model_ethnicity.add(MaxPooling2D(pool_size=(2, 2), strides=2))

model_ethnicity.add(Flatten())

model_age.add(Dense(4, activation='relu'))
model_age.add(Dropout(0.3))

model_ethnicity.add(Dense(5, activation='softmax'))

model_ethnicity.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
```

```
model_ethnicity.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_4 (Conv2D)            (None, 48, 48, 2)         20
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 24, 24, 2)         0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 24, 24, 4)         76
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 12, 12, 4)         0
_________________________________________________________________
flatten_2 (Flatten)          (None, 576)               0
_________________________________________________________________
dense_5 (Dense)              (None, 5)                 2885
=================================================================
Total params: 2,981
Trainable params: 2,981
Non-trainable params: 0
_________________________________________________________________
```

Обучим НС

```
history_ethnicity = model_ethnicity.fit(x_train2, y_train2, epochs=25,
validation_data=(x_test2, y_test2))
```

Нейронная сеть для классификации по полу. Модель 3

Построим НС с двумя свёрточными, двумя субдискретизирующими, двумя полносвязными слоями и одним слоем Dropout. Выходной полносвязный слой включает в себя только 1 нейрон, так как в данном случае это бинарная классификация.

В качестве функции активации выходного слоя используем сигмоиду. Функция потерь – Binary Categorical Crossentropy.

```
model_gender = Sequential()

model_gender.add(Conv2D(2, kernel_size=(3, 3), input_shape=input_shape,
padding='same', activation='relu'))
model_gender.add(MaxPooling2D(pool_size=(2, 2), strides=2))
model_gender.add(Conv2D(4, kernel_size=(3, 3), padding='same',
```

```
activation='relu'))
model_gender.add(MaxPooling2D(pool_size=(2, 2), strides=2))

model_gender.add(Flatten())

model_age.add(Dense(4, activation='relu'))
model_age.add(Dropout(0.3))

model_gender.add(Dense(1, activation='sigmoid'))

model_gender.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])

model_gender.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_6 (Conv2D)            (None, 48, 48, 2)         20
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 24, 24, 2)         0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 24, 24, 4)         76
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 12, 12, 4)         0
_________________________________________________________________
flatten_3 (Flatten)          (None, 576)               0
_________________________________________________________________
dense_7 (Dense)              (None, 1)                 577
=================================================================
Total params: 673
Trainable params: 673
Non-trainable params: 0
_________________________________________________________________
```

Обучим НС

```
history_gender = model_gender.fit(x_train3, y_train3, epochs=25,
validation_data=(x_test3, y_test3))
```

6. Тестирование

6.1. Этап 3

Для того, чтобы протестировать получившиеся модели, выведем информацию о двух метриках, которые вычислялись при обучении: потери и точность для обучающего и тестового набора. Кроме того, построим их графики в зависимости от эпохи.

Также необходимо выполнить предсказания обученными моделями. После этого мы выведем изображения с указанием предсказанных и настоящих данных и неправильно классифицированные изображения для того, чтобы проанализировать работу нейронных сетей.

Модель 1

```
val_loss, val_acc = model_age.evaluate(x_test1, y_test1)
print('Accuracy:', history_age.history['accuracy'][-1])
print('Val (test) accuracy:', history_age.history['val_accuracy'][-1])
```

```
223/223 [==============================] - 1s 3ms/step - loss: 1.2319 - acc
uracy: 0.5179
Accuracy: 0.5000301599502563
Val (test) accuracy: 0.5178571343421936
```

Точность для обучающего набора составляет 50%, для тестового – 51.79%.

Графики потерь и точности:

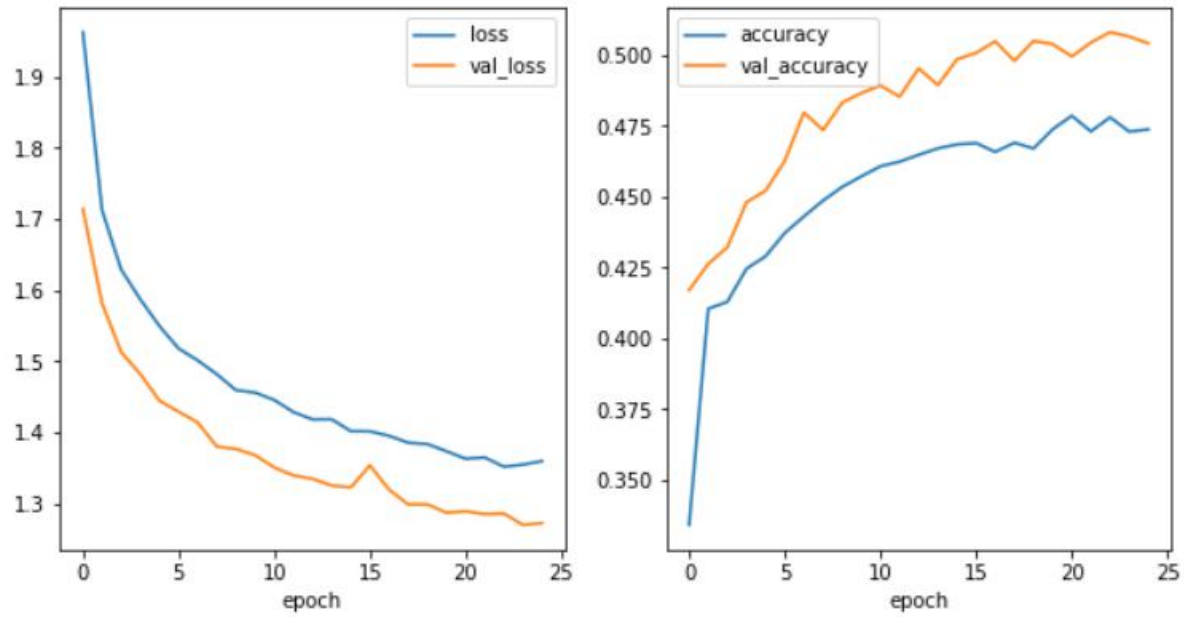


Рис. 12. Графики потерь и точности модели 1

Точность в 51.79%, строго говоря, является низкой, кроме того, она не соответствует нашей исходной цели (получение точности 70% и выше). Основной проблемой в данной задаче является то, что люди одинакового возраста могут выглядеть совершенно по-разному из-за индивидуальных особенностей и разного проявления процесса старения.

Попробуем изменить разбиение на категории по возрасту:

- Дети: 0 – возраст от 1 до 10;
- Подростки: 1 – возраст от 11 до 20;
- Молодые люди: 2 – возраст от 21 до 39;
- Взрослые: 3 – от 40 до 59;
- Пожилые люди: 4 – от 60 до 89;
- Старые люди: 5 – от 90 до 120.

```
data.loc[(data['age'] > 0) & (data['age'] <= 10), 'age'] = 0
data.loc[(data['age'] > 10) & (data['age'] <= 20), 'age'] = 1
```

```
data.loc[(data['age'] > 20) & (data['age'] <= 30), 'age'] = 2
data.loc[(data['age'] > 30) & (data['age'] <= 40), 'age'] = 2
data.loc[(data['age'] > 40) & (data['age'] <= 50), 'age'] = 3
data.loc[(data['age'] > 50) & (data['age'] <= 60), 'age'] = 3
data.loc[(data['age'] > 60) & (data['age'] <= 70), 'age'] = 4
data.loc[(data['age'] > 70) & (data['age'] <= 80), 'age'] = 4
data.loc[(data['age'] > 80) & (data['age'] <= 90), 'age'] = 4
data.loc[(data['age'] > 90) & (data['age'] <= 100), 'age'] = 5

plt.figure(figsize=(40,10))
sns.countplot(x=data['age'])
plt.title('Age countplot', fontsize=20)

y1 = to_categorical(np.array(data['age']), 10)

x_train1, x_test1, y_train1, y_test1 = train_test_split(x, y1, test_size =
0.3, random_state = 42)
```

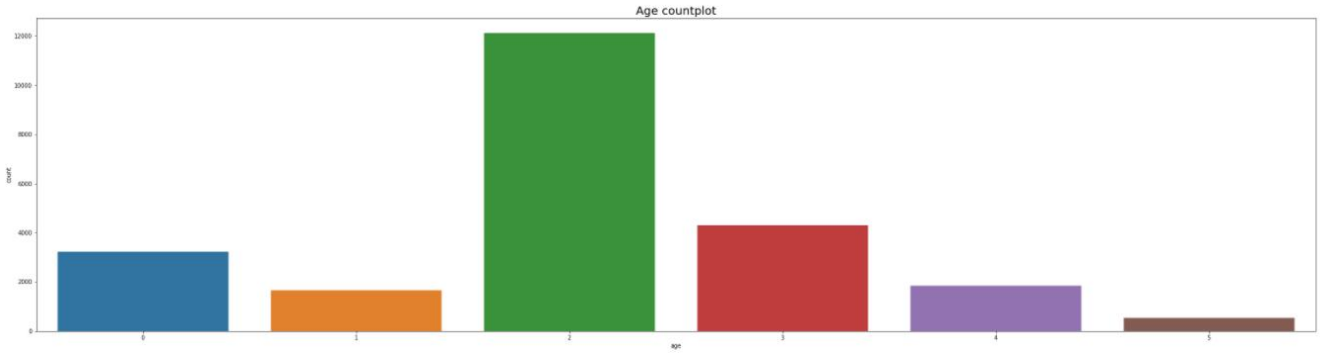


Рис. 13. Обновленный график, показывающий количество изображений по возрастам

Так как количество классов изменилось, необходимо поменять количество нейронов на последнем слое с 10 на 6 и обучить обновлённую нейронную сеть.

```
model_age.add(Dense(6, activation='softmax'))

history_age = model_age.fit(x_train1, y_train1, batch_size=128, epochs=25,
validation_data=(x_test1, y_test1))

val_loss, val_acc = model_age.evaluate(x_test1, y_test1)
print('Accuracy:', history_age.history['accuracy'][-1])
print('Val (test) accuracy:', history_age.history['val_accuracy'][-1])
```

```
223/223 [==============================] - 1s 3ms/step - loss: 0.7489 - acc
uracy: 0.7106
Accuracy: 0.6829385757446289
Val (test) accuracy: 0.710629940032959
```

Точность для обучающего набора составляет 68.29%, для тестового – 71.06%.

Графики потерь и точности для обновлённой модели:

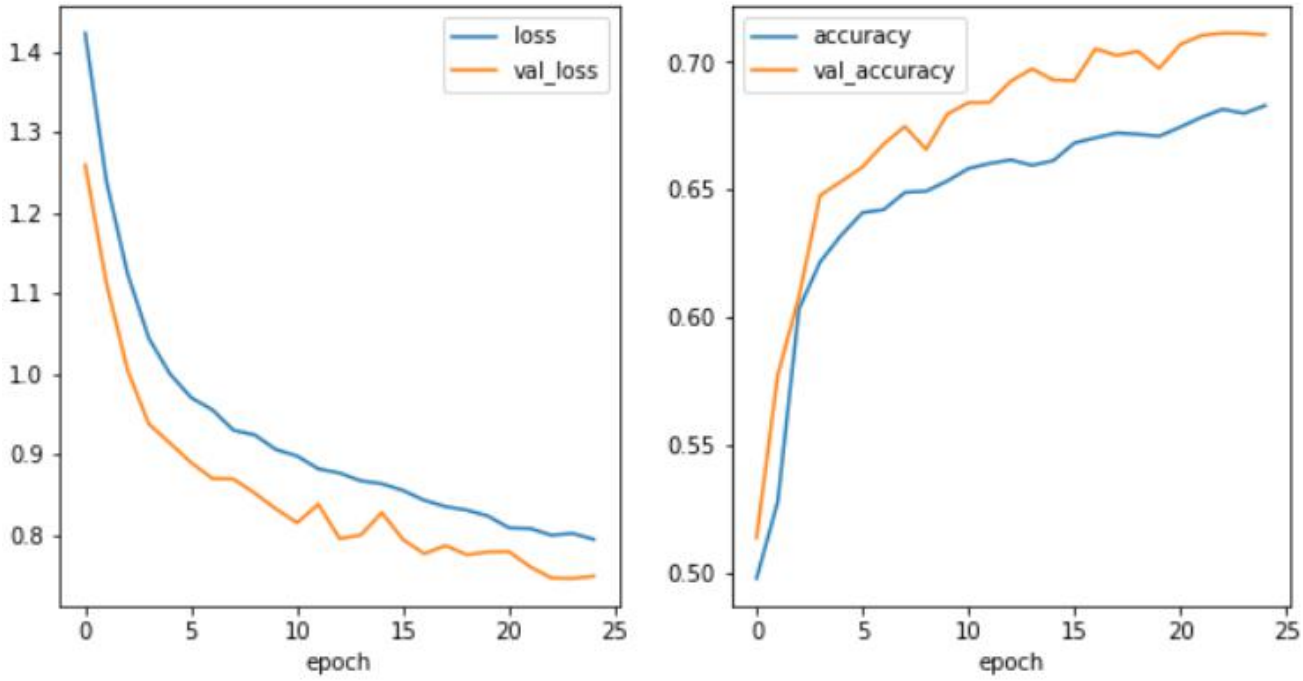


Рис. 14. Графики потерь и точности обновлённой модели 1

Теперь поставленная задача выполнена.

Строим предсказание

```
predictions1 = model_age.predict(x_test1)
predictions1 = np.argmax(predictions1, axis=1)
y_test1 = np.argmax(y_test1, axis=1)
```

Выведем пять случайных изображений с указанием предсказанной и настоящей возрастной группы

```
fig, axes = plt.subplots(1, 5, figsize=(20, 10))for i in range(5):
    random_face = np.random.choice(len(predictions1))
    axes[i].set_title('Pred age group: ' + str(predictions1[random_face]) +
' Act age group: ' + str(y_test1[random_face]))
    axes[i].imshow(x_test1[random_face], cmap='gray')
    axes[i].axis('off')
```

Рис. 15. Пять случайных изображений с указанием предсказанной и настоящей возрастной группы

Выведем пять неверно классифицированных по возрасту изображений

```
mask1 = predictions1 == y_test1
x_false1 = x_test1[~mask1]
false_predictions1 = predictions1[~mask1]
y_false1 = y_test1[~mask1]

fig, axes = plt.subplots(1, 5, figsize=(20, 10))for i in range(5):
    random_face = np.random.choice(len(false_predictions1))
    axes[i].set_title('Pred age group: ' +
str(false_predictions1[random_face]) + ' Act age group: ' +
str(y_false1[random_face]))
    axes[i].imshow(x_false1[random_face], cmap='gray')
    axes[i].axis('off')
```

Рис. 16. Пять неверно классифицированных по возрасту изображений

Модель 2

```
val_loss, val_acc = model_ethnicity.evaluate(x_test2, y_test2)
print('Accuracy:', history_ethnicity.history['accuracy'][-1])
print('Val (test) accuracy:', history_ethnicity.history['val_accuracy'][-
1])
```

```
223/223 [==============================] - 1s 3ms/step - loss: 0.7703 - acc
uracy: 0.7264
Accuracy: 0.7463388442993164
Val (test) accuracy: 0.7263779640197754
```

Точность для обучающего набора составляет 74.63%, для тестового – 72.64%. Задача выполнена.

Графики потерь и точности:

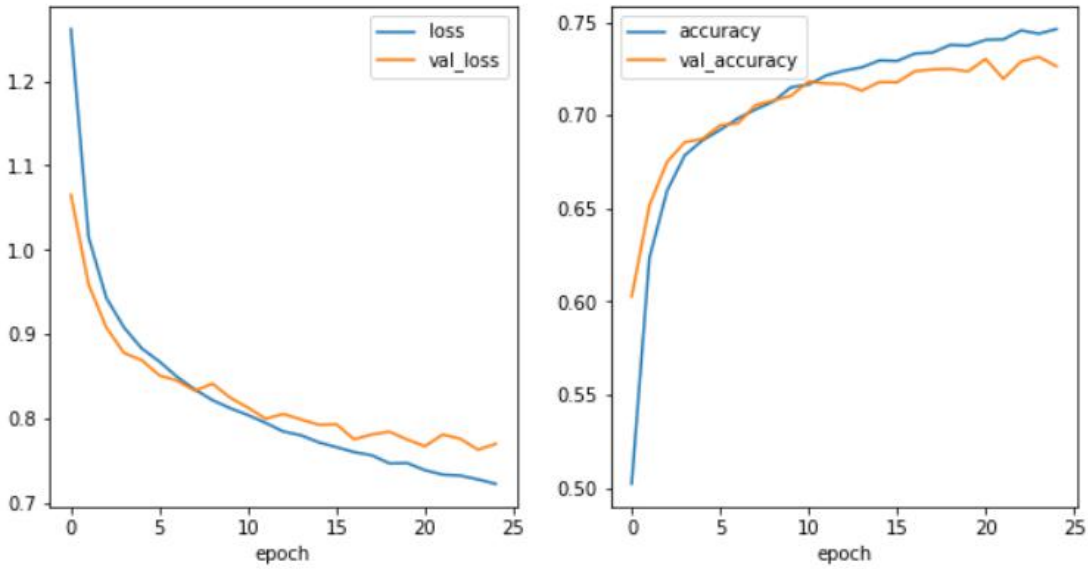


Рис. 17. Графики потерь и точности модели 2

Строим предсказание

```
predictions2 = model_ethnicity.predict(x_test2)
predictions2 = np.argmax(predictions2, axis=1)
y_test2 = np.argmax(y_test2, axis=1)
```

Выведем пять случайных изображений с указанием предсказанной и настоящей этнической принадлежности

```
fig, axes = plt.subplots(1, 5, figsize=(20, 10))for i in range(5):
    random_face = np.random.choice(len(predictions2))
    axes[i].set_title('Pred ethnicity: ' + str(predictions2[random_face]) +
' Act ethnicity: ' + str(y_test2[random_face]))
    axes[i].imshow(x_test2[random_face], cmap='gray')
    axes[i].axis('off')
```

Рис. 18. Пять случайных изображений с указанием предсказанной и настоящей этнической принадлежности

Выведем пять неверно классифицированных по этнической принадлежности изображений

```
mask2 = predictions2 == y_test2
 x_false2 = x_test2[~mask2]
false_predictions2 = predictions2[~mask2]
y_false2 = y_test2[~mask2]

fig, axes = plt.subplots(1, 5, figsize=(20, 10))for i in range(5):
    random_face = np.random.choice(len(false_predictions2))
    axes[i].set_title('Pred ethnicity: ' +
str(false_predictions2[random_face]) + ' Act ethnicity: ' +
str(y_false2[random_face]))
    axes[i].imshow(x_false2[random_face], cmap='gray')
    axes[i].axis('off')
```

Рис. 19. Пять неверно классифицированных по этнической принадлежности изображений

Модель 3

```
val_loss, val_acc = model_gender.evaluate(x_test3, y_test3)
print('Accuracy:', history_gender.history['accuracy'][-1])
print('Val (test) accuracy:', history_gender.history['val_accuracy'][-1])
```

```
223/223 [==============================] - 1s 3ms/step - loss: 0.3453 - acc
uracy: 0.8553
Accuracy: 0.8588561415672302
Val (test) accuracy: 0.8553149700164795
```

Точность для обучающего набора составляет 85.89%, для тестового – 85.53%. Задача выполнена.

Графики потерь и точности:

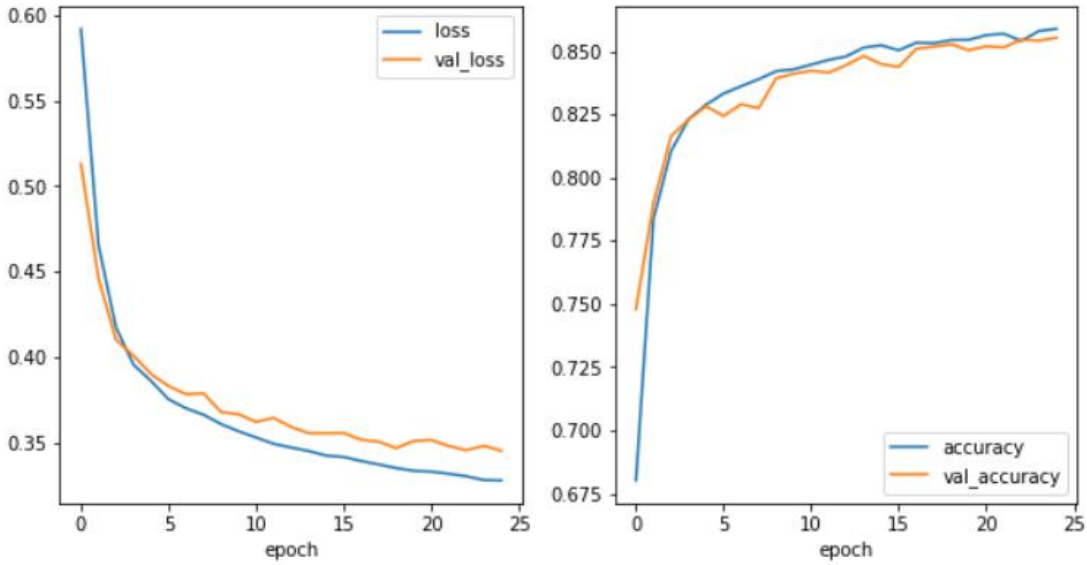


Рис. 20. Графики потерь и точности модели 3

Строим предсказание

```
predictions3 = model_gender.predict(x_test3)
```

Выведем пять случайных изображений с указанием предсказанного и настоящего пола

```
fig, axes = plt.subplots(1, 5, figsize=(20, 10))for i in range(5):
    random_face = np.random.choice(len(predictions3))
    axes[i].set_title("Pred gender: {0} Act gender:
{1}".format(np.round(predictions3[random_face]), y_test3[random_face]))
    axes[i].imshow(x_test3[random_face], cmap='gray')
    axes[i].axis('off')
```

Рис. 21. Пять случайных изображений с указанием предсказанного и настоящего пола

Выведем пять неверно классифицированных по полу изображений

```
false_predictions3 = []
x_false3 = []
y_false3 = []for i in range(len(y_test3)):
    pred_class = np.round(predictions3[i])
    true_class = y_test3[i]
    if pred_class != true_class:
        x_false3.append(x_test3[i])
        y_false3.append(true_class)
        false_predictions3.append(pred_class)

fig, axes = plt.subplots(1, 5, figsize=(20, 10))for i in
range(5):
    random_face = np.random.choice(len(false_predictions3))
    axes[i].set_title('Pred gender: ' +
str(false_predictions3[random_face]) + ' Act gender: ' +
str(y_false3[random_face]))
    axes[i].imshow(x_false3[random_face], cmap='gray')
    axes[i].axis('off')
```

Рис. 22. Пять неверно классифицированных по этнической принадлежности изображений

Заметим, что чаще всего неверно классифицируются дети и пожилые люди

Рис. 23. Двадцать пять неверно классифицированных по полу изображений

Заключение

Цель работы выполнена – был разработан программный продукт, решающий задачу определения базовых атрибутов личности (возраста, этнической принадлежности, пола) по изображению лица с использованием свёрточных нейронных сетей в соответствии с моделью жизненного цикла программного обеспечения.

Для достижения указанной цели были выполнены следующие задачи:

1. Изучены теоретические основы поставленной задачи, нейронных сетей и их применения в контексте обработки изображений;

2. Реализованы три нейронные сети – для определения возраста, этнической принадлежности и пола;

3. Проведено тестирование и проанализированы полученные результаты.

Модели показали точность более 70%.

В дальнейшем продукт можно доработать: например, усложнить архитектуру нейронных сетей и добавить возможность загружать пользовательские изображения.

Список литературы

1. Бредихин А.А. Алгоритмы обучения свёрточных нейронных сетей // Вестник Югорского государственного университета. – 2019. – №1 (25).

2. Воронцов К. В. Курс лекций по машинному обучению. – 2015.

3. Дуденков В. М. Разработка нейросетевых моделей человекомашинного общения: диссертация. Воронежский государственный университет. – 2016.

   URL:

   http://www.science.vsu.ru/dissertations/3715/Диссертация_Дуденков_В.М.pdf

4. Куликова А.А. Подход к классификации пользователей в социальных сетях // Восточно-Европейский журнал передовых технологий. – 2011. – №2 (51).

5. Рыбинцев А.В. Исследование, модификация и разработка методов компьютерного зрения для задач определения атрибутов личности по изображению лица: диссертация. Национальный исследовательский университет «МЭИ». – 2018.

   URL: https://mpei.ru/diss/Lists/FilesDissertations/369-Диссертация.pdf

6. Рысьмятова А.А. Использование свёрточных нейронных сетей для задачи классификации текстов: выпускная квалификационная работа. – Московский государственный университет. – 2016.

   URL:

   http://www.machinelearning.ru/wiki/images/c/c7/2016_417_RysmyatovaAA.pdf

7. Уоссермен Ф. Нейрокомпьютерная техника: теория и практика / перевод с английского Ю.А. Зуева, В. А. Точёнова. – 1992.

8. Классификация данных при помощи нейронных сетей. Электронный ресурс.

   URL: https://loginom.ru/blog/neural-classification

9. Нейронная сеть. Электронный ресурс.

   URL: https://ru.wikipedia.org/wiki/Нейронная_сеть

10. Age, gender and ethnicity (face data) CSV. Электронный ресурс.

    URL:

    https://www.kaggle.com/nipunarora8/age-gender-and-ethnicity-face-data-csv

Листинг программы

In [1]:

```
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns

import numpy as np

from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report

from tensorflow.keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Input, Conv2D, MaxPooling2D, Flatten, Dropout,
from keras.callbacks import Callback

import time
```

DATASET

In [2]:

```
data = pd.read_csv("../input/age-gender-and-ethnicity-face-data-csv/age_
print(data.shape)
data.head()
```

```
(23705, 5)
```

Out[2]:

	age	ethnicity	gender	img_name	pixels
0	1	2	0	20161219203650636.jpg.chip.jpg	129 128 128 126 127 130 133 135 139 142 145 14...
1	1	2	0	20161219222752047.jpg.chip.jpg	164 74 111 168 169 171 175 182 184 188 193 199...
2	1	2	0	20161219222832191.jpg.chip.jpg	67 70 71 70 69 67 70 79 90 103 116 132 145 155...
3	1	2	0	20161220144911423.jpg.chip.jpg	193 197 198 200 199 200 202 203 204 205 208 21...
4	1	2	0	20161220144914327.jpg.chip.jpg	202 205 209 210 209 209 210 211 212 214 218 21...

In [3]:

```
data = data.drop('img_name', axis=1)
print(data.shape)
data.head()
```

```
(23705, 4)
```

Out[3]:

	age	ethnicity	gender	pixels
0	1	2	0	129 128 128 126 127 130 133 135 139 142 145 14...
1	1	2	0	164 74 111 168 169 171 175 182 184 188 193 199...
2	1	2	0	67 70 71 70 69 67 70 79 90 103 116 132 145 155...
3	1	2	0	193 197 198 200 199 200 202 203 204 205 208 21...
4	1	2	0	202 205 209 210 209 209 210 211 212 214 218 21...

In [4]:
```
data.loc[(data['age'] > 0) & (data['age'] <= 10), 'age'] = 0
data.loc[(data['age'] > 10) & (data['age'] <= 20), 'age'] = 1
data.loc[(data['age'] > 20) & (data['age'] <= 30), 'age'] = 2
data.loc[(data['age'] > 30) & (data['age'] <= 40), 'age'] = 3
data.loc[(data['age'] > 40) & (data['age'] <= 50), 'age'] = 4
data.loc[(data['age'] > 50) & (data['age'] <= 60), 'age'] = 5
data.loc[(data['age'] > 60) & (data['age'] <= 70), 'age'] = 6
data.loc[(data['age'] > 70) & (data['age'] <= 80), 'age'] = 7
data.loc[(data['age'] > 80) & (data['age'] <= 90), 'age'] = 8
data.loc[(data['age'] > 90) & (data['age'] <= 120), 'age'] = 9
```

In [5]:
```
data
```

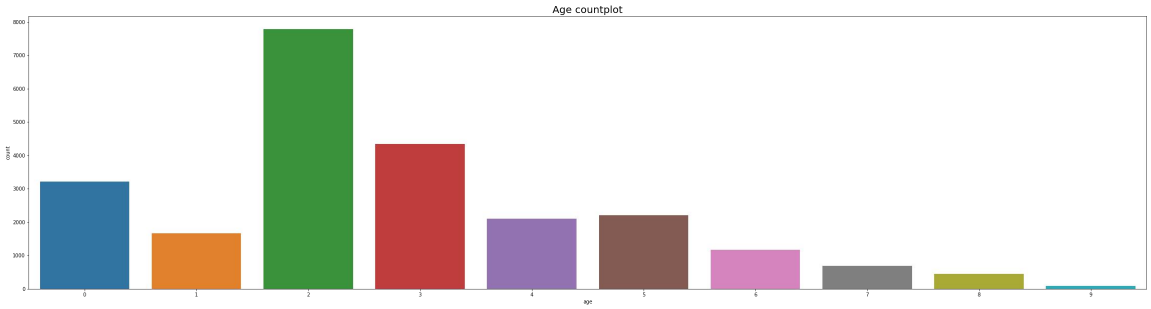
Out[5]:

	age	ethnicity	gender	pixels
0	0	2	0	129 128 128 126 127 130 133 135 139 142 145 14...
1	0	2	0	164 74 111 168 169 171 175 182 184 188 193 199...
2	0	2	0	67 70 71 70 69 67 70 79 90 103 116 132 145 155...
3	0	2	0	193 197 198 200 199 200 202 203 204 205 208 21...
4	0	2	0	202 205 209 210 209 209 210 211 212 214 218 21...
...	...	...	...	...
23700	9	0	1	127 100 94 81 77 77 74 99 102 98 128 145 160 1...
23701	9	1	1	23 28 32 35 42 47 68 85 98 103 113 117 130 129...
23702	9	2	1	59 50 37 40 34 19 30 101 156 170 177 184 187 1...
23703	9	2	1	45 108 120 156 206 197 140 180 191 199 204 207...
23704	9	0	1	156 161 160 165 170 173 166 177 183 191 187 18...

23705 rows × 4 columns

In [6]:
```
plt.figure(figsize=(40,10))
sns.countplot(x=data['age'])
plt.title('Age countplot', fontsize=20)
```

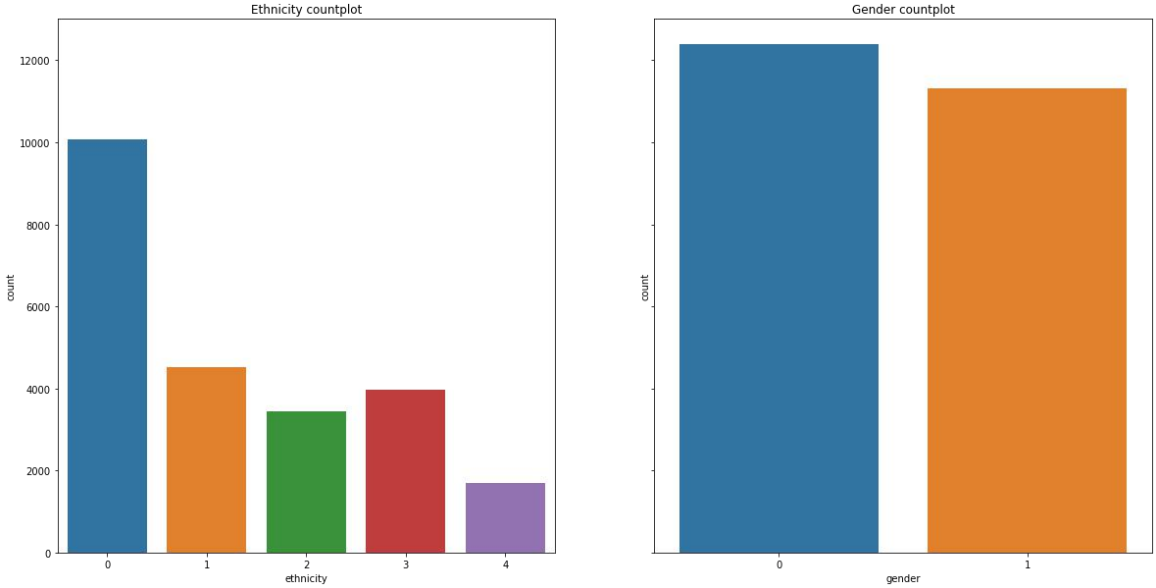
Out[6]:
```
Text(0.5, 1.0, 'Age countplot'
```



In [7]:
```
fig, axes = plt.subplots(1, 2, figsize=(20, 10), sharey=True)

sns.countplot(ax=axes[0], x=data['ethnicity'])
axes[0].set_title('Ethnicity countplot')
```

```
sns.countplot(ax=axes[1], x=data['gender'])
axes[1].set_title('Gender countplot')
```

Out[7]: Text(0.5, 1.0, 'Gender countplot')



In [8]:
```
data['pixels'] = data['pixels'].map(lambda x: np.array(x.split(' '), dty
```

In [9]:
```
fig, axes = plt.subplots(1, 5, figsize=(20, 10))
for i in range(5):
    random_face = np.random.choice(len(data))
    age = data['age'][random_face]
    ethnicity = data['ethnicity'][random_face]
    gender = data['gender'][random_face]

    axes[i].set_title('Age group: {0}, Ethnicity: {1}, Gender: {2}'.form
    axes[i].imshow(data['pixels'][random_face], cmap='gray')
    axes[i].axis('off')
```

In [10]:
```
x = np.array(data['pixels'].to_list())
x = x.reshape(x.shape[0], 48, 48, 1) / 255
input_shape = x.shape[1:]
```

In [11]:
```
y1 = to_categorical(np.array(data['age']), 10)
y2 = to_categorical(np.array(data['ethnicity']), 5)
y3 = np.array(data['gender'])
```

In [12]:
```
x_train1, x_test1, y_train1, y_test1 = train_test_split(x, y1, test_size
x_train2, x_test2, y_train2, y_test2 = train_test_split(x, y2, test_size
x_train3, x_test3, y_train3, y_test3 = train_test_split(x, y3, test_size
```

AGE

In [13]:
```
model_age = Sequential()

model_age.add(Conv2D(8, kernel_size=(3, 3), input_shape=input_shape, pad
model_age.add(MaxPooling2D(pool_size=(2, 2), strides=2))
model_age.add(Conv2D(16, kernel_size=(3, 3), padding='same', activation=
model_age.add(MaxPooling2D(pool_size=(2, 2), strides=2))

model_age.add(Flatten())

model_age.add(Dense(16, activation='relu'))
model_age.add(Dropout(0.3))

model_age.add(Dense(10, activation='softmax'))

model_age.compile(optimizer='adam', loss='categorical_crossentropy', met
model_age.summary()
```

```
Model: "sequential"

Layer (type)                 Output Shape              Param #

conv2d (Conv2D)              (None, 48, 48, 8)         80

max_pooling2d (MaxPooling2D) (None, 24, 24, 8)         0

conv2d_1 (Conv2D)            (None, 24, 24, 16)        1168

max_pooling2d_1 (MaxPooling2 (None, 12, 12, 16)        0

flatten (Flatten)            (None, 2304)              0

dense (Dense)                (None, 16)                36880

dropout (Dropout)            (None, 16)                0

dense_1 (Dense)              (None, 10)                170

Total params: 38,298
Trainable params: 38,298
Non-trainable params: 0
```

```
2021-12-07 19:27:56.529433: I tensorflow/stream_executor/cuda/cuda_gpu_ex
ecutor.cc:937] successful NUMA node read from SysFS had negative value (-
1), but there must be at least one NUMA node, so returning NUMA node zero
2021-12-07 19:27:56.613378: I tensorflow/stream_executor/cuda/cuda_gpu_ex
ecutor.cc:937] successful NUMA node read from SysFS had negative value (-
1), but there must be at least one NUMA node, so returning NUMA node zero
2021-12-07 19:27:56.614120: I tensorflow/stream_executor/cuda/cuda_gpu_ex
ecutor.cc:937] successful NUMA node read from SysFS had negative value (-
1), but there must be at least one NUMA node, so returning NUMA node zero
2021-12-07 19:27:56.615715: I tensorflow/core/platform/cpu_feature_guard.
cc:142] This TensorFlow binary is optimized with oneAPI Deep Neural Netwo
rk Library (oneDNN) to use the following CPU instructions in performance-
critical operations: AVX2 AVX512F FMA
To enable them in other operations, rebuild TensorFlow with the appropria
te compiler flags.
2021-12-07 19:27:56.616700: I tensorflow/stream_executor/cuda/cuda_gpu_ex
ecutor.cc:937] successful NUMA node read from SysFS had negative value (-
1), but there must be at least one NUMA node, so returning NUMA node zero
2021-12-07 19:27:56.617426: I tensorflow/stream_executor/cuda/cuda_gpu_ex
ecutor.cc:937] successful NUMA node read from SysFS had negative value (-
1), but there must be at least one NUMA node, so returning NUMA node zero
```

```
2021-12-07 19:27:56.618097: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node read from SysFS had negative value (-1), but there must be at least one NUMA node, so returning NUMA node zero
2021-12-07 19:27:58.431537: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node read from SysFS had negative value (-1), but there must be at least one NUMA node, so returning NUMA node zero
2021-12-07 19:27:58.432421: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node read from SysFS had negative value (-1), but there must be at least one NUMA node, so returning NUMA node zero
2021-12-07 19:27:58.433164: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node read from SysFS had negative value (-1), but there must be at least one NUMA node, so returning NUMA node zero
2021-12-07 19:27:58.433833: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1510] Created device /job:localhost/replica:0/task:0/device:GPU:0 with 15403 MB memory:  -> device: 0, name: Tesla P100-PCIE-16GB, pci bus id: 0000:00:04.0, compute capability: 6.0
```

In [14]:

```
start_time = time.time()

history_age = model_age.fit(x_train1, y_train1, batch_size=128, epochs=2

print("Time:", (time.time() - start_time) / 60)
```

```
2021-12-07 19:27:59.222238: I tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:185] None of the MLIR Optimization Passes are enabled (registered 2)
Epoch 1/25
2021-12-07 19:28:00.377032: I tensorflow/stream_executor/cuda/cuda_dnn.cc:369] Loaded cuDNN version 8005
130/130 [==============================] - 7s 8ms/step - loss: 1.9628 - accuracy: 0.3341 - val_loss: 1.7140 - val_accuracy: 0.4170 Epoch
2/25
130/130 [==============================] - 1s 5ms/step - loss: 1.7124 - accuracy: 0.4105 - val_loss: 1.5811 - val_accuracy: 0.4263 Epoch
3/25
130/130 [==============================] - 1s 5ms/step - loss: 1.6288 - accuracy: 0.4129 - val_loss: 1.5126 - val_accuracy: 0.4322 Epoch
4/25
130/130 [==============================] - 1s 5ms/step - loss: 1.5870 - accuracy: 0.4246 - val_loss: 1.4818 - val_accuracy: 0.4481 Epoch
5/25
130/130 [==============================] - 1s 5ms/step - loss: 1.5493 - accuracy: 0.4290 - val_loss: 1.4442 - val_accuracy: 0.4522 Epoch
6/25
130/130 [==============================] - 1s 5ms/step - loss: 1.5178 - accuracy: 0.4372 - val_loss: 1.4289 - val_accuracy: 0.4627 Epoch
7/25
130/130 [==============================] - 1s 5ms/step - loss: 1.5010 - accuracy: 0.4431 - val_loss: 1.4137 - val_accuracy: 0.4798 Epoch
8/25
130/130 [==============================] - 1s 5ms/step - loss: 1.4816 - accuracy: 0.4487 - val_loss: 1.3799 - val_accuracy: 0.4736 Epoch
9/25
130/130 [==============================] - 1s 5ms/step - loss: 1.4591 - accuracy: 0.4535 - val_loss: 1.3765 - val_accuracy: 0.4833 Epoch
10/25
130/130 [==============================] - 1s 5ms/step - loss: 1.4558 - accuracy: 0.4572 - val_loss: 1.3678 - val_accuracy: 0.4866 Epoch
11/25
130/130 [==============================] - 1s 5ms/step - loss: 1.4453 - accuracy: 0.4608 - val_loss: 1.3504 - val_accuracy: 0.4893 Epoch
12/25
130/130 [==============================] - 1s 6ms/step - loss: 1.4283 - accuracy: 0.4624 - val_loss: 1.3393 - val_accuracy: 0.4854 Epoch
13/25
130/130 [==============================] - 1s 5ms/step - loss: 1.4181 - a
```

```
ccuracy: 0.4648 - val_loss: 1.3345 - val_accuracy: 0.4954 Epoch
14/25
130/130 [==============================] - 1s 5ms/step - loss: 1.4183 - a
ccuracy: 0.4671 - val_loss: 1.3250 - val_accuracy: 0.4895 Epoch
15/25
130/130 [==============================] - 1s 5ms/step - loss: 1.4015 - a
ccuracy: 0.4685 - val_loss: 1.3223 - val_accuracy: 0.4986 Epoch
16/25
130/130 [==============================] - 1s 5ms/step - loss: 1.4013 - a
ccuracy: 0.4690 - val_loss: 1.3534 - val_accuracy: 0.5008 Epoch
17/25
130/130 [==============================] - 1s 5ms/step - loss: 1.3949 - a
ccuracy: 0.4659 - val_loss: 1.3193 - val_accuracy: 0.5049 Epoch
18/25
130/130 [==============================] - 1s 5ms/step - loss: 1.3853 - a
ccuracy: 0.4691 - val_loss: 1.2985 - val_accuracy: 0.4980 Epoch
19/25
130/130 [==============================] - 1s 5ms/step - loss: 1.3834 - a
ccuracy: 0.4671 - val_loss: 1.2986 - val_accuracy: 0.5051 Epoch
20/25
130/130 [==============================] - 1s 5ms/step - loss: 1.3733 - a
ccuracy: 0.4739 - val_loss: 1.2869 - val_accuracy: 0.5039 Epoch
21/25
130/130 [==============================] - 1s 5ms/step - loss: 1.3627 - a
ccuracy: 0.4786 - val_loss: 1.2888 - val_accuracy: 0.4996 Epoch
22/25
130/130 [==============================] - 1s 5ms/step - loss: 1.3646 - a
ccuracy: 0.4731 - val_loss: 1.2849 - val_accuracy: 0.5045 Epoch
23/25
130/130 [==============================] - 1s 5ms/step - loss: 1.3515 - a
ccuracy: 0.4781 - val_loss: 1.2856 - val_accuracy: 0.5082 Epoch
24/25
130/130 [==============================] - 1s 5ms/step - loss: 1.3545 - a
ccuracy: 0.4730 - val_loss: 1.2696 - val_accuracy: 0.5066 Epoch
25/25
130/130 [==============================] - 1s 5ms/step - loss: 1.3594 - a
ccuracy: 0.4738 - val_loss: 1.2722 - val_accuracy: 0.5042
Time: 0.6954338351885477
```

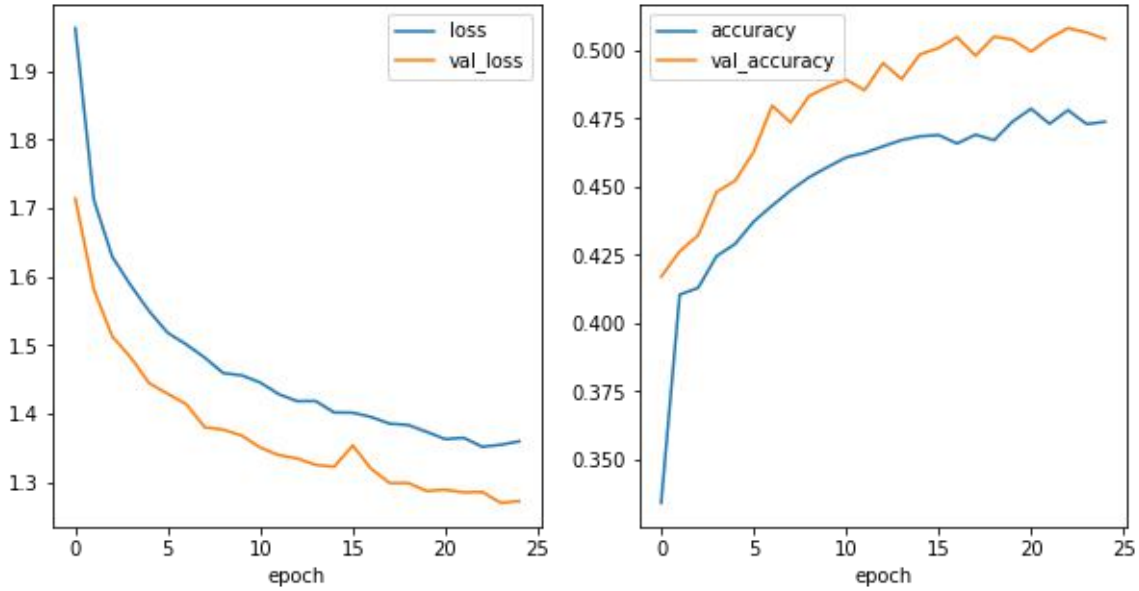
In [15]:

```
val_loss, val_acc = model_age.evaluate(x_test1, y_test1)

print('Accuracy:', history_age.history['accuracy'][-1])
print('Val (test) accuracy:', history_age.history['val_accuracy'][-1])
```

```
223/223 [==============================] - 1s 2ms/step - loss: 1.2722 - a
ccuracy: 0.5042
Accuracy: 0.4738142490386963
Val (test) accuracy: 0.5042182207107544
```

In [16]:

```
fig, axs = plt.subplots(1, 2, figsize=(10, 5))
axs[0].plot(history_age.history['loss'], label='loss')
axs[0].plot(history_age.history['val_loss'], label='val_loss')
axs[0].set_xlabel('epoch')
axs[0].legend()

axs[1].plot(history_age.history['accuracy'], label='accuracy')
axs[1].plot(history_age.history['val_accuracy'], label='val_accuracy')
axs[1].set_xlabel('epoch')
axs[1].legend()
plt.show()
```



In [17]:

```
predictions1 = model_age.predict(x_test1)
predictions1 = np.argmax(predictions1, axis=1)
y_test1 = np.argmax(y_test1, axis=1)
```

In [18]:

```
fig, axes = plt.subplots(1, 5, figsize=(20, 10))
for i in range(5):
    random_face = np.random.choice(len(predictions1))
    axes[i].set_title('Pred age group: ' + str(predictions1[random_face]
    axes[i].imshow(x_test1[random_face],   cmap='gray')
    axes[i].axis('off')
```

In [19]:

```
mask1 = predictions1 == y_test1

x_false1 = x_test1[~mask1]
false_predictions1 = predictions1[~mask1]
y_false1 = y_test1[~mask1]
```

In [20]:

```
fig, axes = plt.subplots(1, 5, figsize=(20, 10))
for i in range(5):
    random_face = np.random.choice(len(false_predictions1))
    axes[i].set_title('Pred age group: ' + str(false_predictions1[random
    axes[i].imshow(x_false1[random_face], cmap='gray')
    axes[i].axis('off')
```

ETHNICITY

In [21]:
```
model_ethnicity = Sequential()

model_ethnicity.add(Conv2D(2, kernel_size=(3, 3), input_shape=input_shap
model_ethnicity.add(MaxPooling2D(pool_size=(2, 2), strides=2))
model_ethnicity.add(Conv2D(4, kernel_size=(3, 3), padding='same', activa
model_ethnicity.add(MaxPooling2D(pool_size=(2, 2), strides=2))

model_ethnicity.add(Flatten())

model_age.add(Dense(4, activation='relu'))
model_age.add(Dropout(0.3))

model_ethnicity.add(Dense(5, activation='softmax'))

model_ethnicity.compile(optimizer='adam', loss='categorical_crossentropy
model_ethnicity.summary()
```

```
Model: "sequential_1"

Layer (type)                 Output Shape              Param #

conv2d_2 (Conv2D)            (None, 48, 48, 2)         20

max_pooling2d_2 (MaxPooling2 (None, 24, 24, 2)         0

conv2d_3 (Conv2D)            (None, 24, 24, 4)         76

max_pooling2d_3 (MaxPooling2 (None, 12, 12, 4)         0

flatten_1 (Flatten)          (None, 576)               0

dense_3 (Dense)              (None, 5)                 2885

Total params: 2,981
Trainable params: 2,981
Non-trainable params: 0
```

In [22]:
```
start_time = time.time()

history_ethnicity = model_ethnicity.fit(x_train2, y_train2, epochs=25,

print("Time:", (time.time() - start_time) / 60)
```

```
Epoch 1/25
519/519 [==============================] - 3s 5ms/step - loss:      1.2626 - a
ccuracy: 0.5014 - val_loss: 1.1078 - val_accuracy: 0.6146
Epoch 2/25
519/519 [==============================] - 2s 4ms/step - loss:      1.0278 - a
ccuracy: 0.6242 - val_loss: 0.9609 - val_accuracy: 0.6558
Epoch 3/25
519/519 [==============================] - 2s 4ms/step - loss:      0.9432 - a
ccuracy: 0.6608 - val_loss: 0.8985 - val_accuracy: 0.6808
Epoch 4/25
519/519 [==============================] - 2s 4ms/step - loss:      0.8990 - a
ccuracy: 0.6810 - val_loss: 0.8719 - val_accuracy: 0.6992
Epoch 5/25
519/519 [==============================] - 2s 4ms/step - loss:      0.8675 - a
ccuracy: 0.6919 - val_loss: 0.8638 - val_accuracy: 0.6877
Epoch 6/25
519/519 [==============================] - 2s 4ms/step - loss:      0.8396 - a
```

```
ccuracy: 0.7051 - val_loss: 0.8392 - val_accuracy: 0.7039 Epoch
7/25
519/519 [==============================] - 2s 4ms/step - loss: 0.8244 - a
ccuracy: 0.7108 - val_loss: 0.8610 - val_accuracy: 0.7013 Epoch
8/25
519/519 [==============================] - 2s 4ms/step - loss: 0.8103 - a
ccuracy: 0.7184 - val_loss: 0.8372 - val_accuracy: 0.6959 Epoch
9/25
519/519 [==============================] - 2s 4ms/step - loss: 0.7997 - a
ccuracy: 0.7186 - val_loss: 0.8390 - val_accuracy: 0.7066 Epoch
10/25
519/519 [==============================] - 2s 4ms/step - loss: 0.7881 - a
ccuracy: 0.7257 - val_loss: 0.8026 - val_accuracy: 0.7168 Epoch
11/25
519/519 [==============================] - 2s 3ms/step - loss: 0.7835 - a
ccuracy: 0.7251 - val_loss: 0.8088 - val_accuracy: 0.7167 Epoch
12/25
519/519 [==============================] - 2s 4ms/step - loss: 0.7767 - a
ccuracy: 0.7284 - val_loss: 0.8126 - val_accuracy: 0.7087 Epoch
13/25
519/519 [==============================] - 2s 4ms/step - loss: 0.7706 - a
ccuracy: 0.7305 - val_loss: 0.8024 - val_accuracy: 0.7184 Epoch
14/25
519/519 [==============================] - 2s 4ms/step - loss: 0.7622 - a
ccuracy: 0.7329 - val_loss: 0.8077 - val_accuracy: 0.7144 Epoch
15/25
519/519 [==============================] - 2s 4ms/step - loss: 0.7593 - a
ccuracy: 0.7315 - val_loss: 0.8038 - val_accuracy: 0.7127 Epoch
16/25
519/519 [==============================] - 2s 4ms/step - loss: 0.7546 - a
ccuracy: 0.7348 - val_loss: 0.8081 - val_accuracy: 0.7113 Epoch
17/25
519/519 [==============================] - 2s 4ms/step - loss: 0.7537 - a
ccuracy: 0.7351 - val_loss: 0.8110 - val_accuracy: 0.7125 Epoch
18/25
519/519 [==============================] - 2s 4ms/step - loss: 0.7493 - a
ccuracy: 0.7372 - val_loss: 0.8332 - val_accuracy: 0.7006 Epoch
19/25
519/519 [==============================] - 2s 4ms/step - loss: 0.7435 - a
ccuracy: 0.7401 - val_loss: 0.8115 - val_accuracy: 0.7118 Epoch
20/25
519/519 [==============================] - 2s 4ms/step - loss: 0.7406 - a
ccuracy: 0.7394 - val_loss: 0.7999 - val_accuracy: 0.7223 Epoch
21/25
519/519 [==============================] - 2s 4ms/step - loss: 0.7380 - a
ccuracy: 0.7421 - val_loss: 0.7998 - val_accuracy: 0.7217 Epoch
22/25
519/519 [==============================] - 2s 4ms/step - loss: 0.7371 - a
ccuracy: 0.7414 - val_loss: 0.7991 - val_accuracy: 0.7246 Epoch
23/25
519/519 [==============================] - 2s 4ms/step - loss: 0.7322 - a
ccuracy: 0.7434 - val_loss: 0.8123 - val_accuracy: 0.7130 Epoch
24/25
519/519 [==============================] - 2s 4ms/step - loss: 0.7317 - a
ccuracy: 0.7453 - val_loss: 0.8073 - val_accuracy: 0.7201 Epoch
25/25
519/519 [==============================] - 2s 4ms/step - loss: 0.7247 - a
ccuracy: 0.7451 - val_loss: 0.7899 - val_accuracy: 0.7230
Time: 1.3758671085039775
```

In [23]:

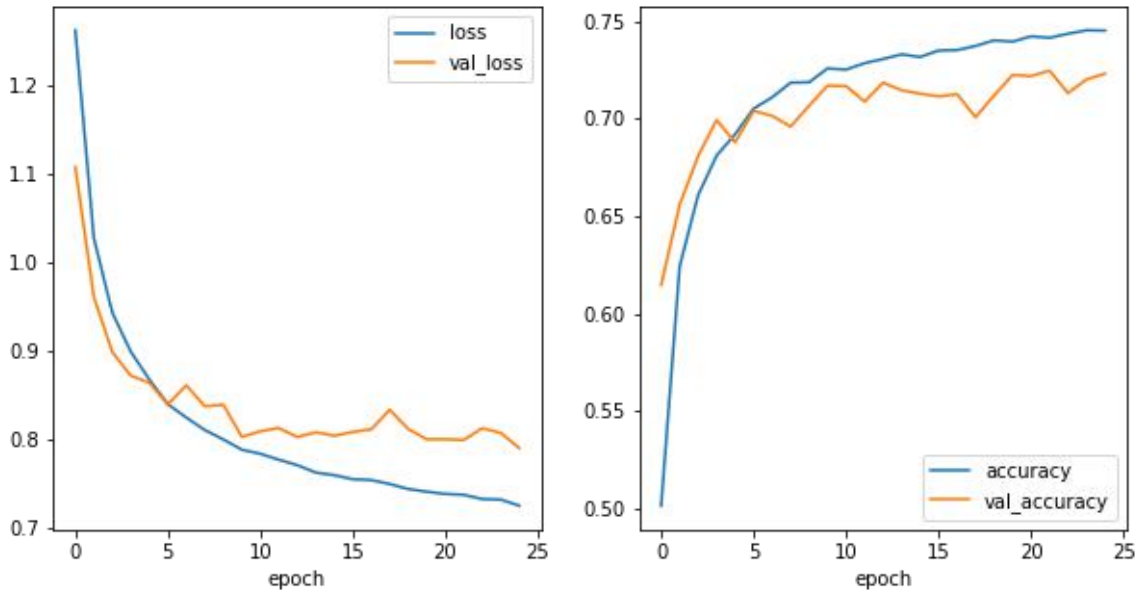
```
val_loss, val_acc = model_ethnicity.evaluate(x_test2, y_test2)

print('Accuracy:', history_ethnicity.history['accuracy'][-1])
print('Val (test) accuracy:', history_ethnicity.history['val_accuracy'][
```

```
223/223 [==============================] - 0s 2ms/step - loss: 0.7899 - a
ccuracy: 0.7230
```

```
Accuracy: 0.7450731992721558
Val (test) accuracy: 0.7230033874511719
```

In [24]:

```
fig, axs = plt.subplots(1, 2, figsize=(10, 5))
axs[0].plot(history_ethnicity.history['loss'], label='loss')
axs[0].plot(history_ethnicity.history['val_loss'], label='val_loss')
axs[0].set_xlabel('epoch')
axs[0].legend()

axs[1].plot(history_ethnicity.history['accuracy'], label='accuracy')
axs[1].plot(history_ethnicity.history['val_accuracy'], label='val_accura
axs[1].set_xlabel('epoch')
axs[1].legend()
plt.show()
```



In [25]:

```
predictions2 = model_ethnicity.predict(x_test2)
predictions2 = np.argmax(predictions2, axis=1)
y_test2 = np.argmax(y_test2, axis=1)
```

In [26]:

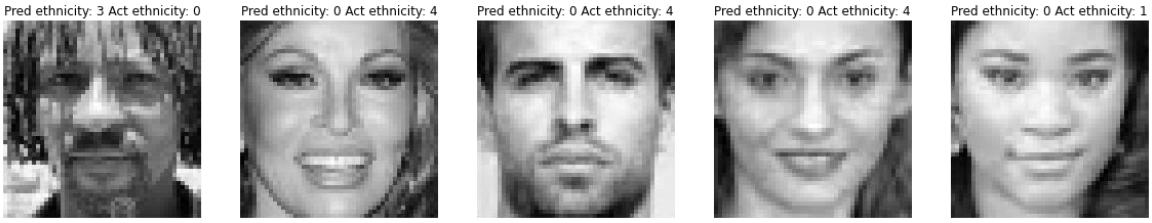
```
fig, axes = plt.subplots(1, 5, figsize=(20, 10))
for i in range(5):
    random_face = np.random.choice(len(predictions2))
    axes[i].set_title('Pred ethnicity: ' + str(predictions2[random_face]
    axes[i].imshow(x_test2[random_face],   cmap='gray')
    axes[i].axis('off')
```

In [27]:

```
mask2 = predictions2 == y_test2

x_false2 = x_test2[~mask2]
false_predictions2 = predictions2[~mask2]
y_false2 = y_test2[~mask2]
```

In [28]:

```
fig, axes = plt.subplots(1, 5, figsize=(20, 10))
for i in range(5):
    random_face = np.random.choice(len(false_predictions2))
    axes[i].set_title('Pred ethnicity: ' + str(false_predictions2[random
    axes[i].imshow(x_false2[random_face], cmap='gray')
    axes[i].axis('off')
```



GENDER

In [29]:

```
model_gender = Sequential()

model_gender.add(Conv2D(2, kernel_size=(3, 3), input_shape=input_shape,
model_gender.add(MaxPooling2D(pool_size=(2, 2), strides=2))
model_gender.add(Conv2D(4, kernel_size=(3, 3), padding='same', activatio
model_gender.add(MaxPooling2D(pool_size=(2, 2), strides=2))

model_gender.add(Flatten())

model_age.add(Dense(4, activation='relu'))
model_age.add(Dropout(0.3))

model_gender.add(Dense(1, activation='sigmoid'))

model_gender.compile(optimizer='adam', loss='binary_crossentropy', metri
model_gender.summary()
```

```
Model: "sequential_2"

Layer (type)                 Output Shape              Param #

conv2d_4 (Conv2D)            (None, 48, 48, 2)         20

max_pooling2d_4 (MaxPooling2 (None, 24, 24, 2)         0

conv2d_5 (Conv2D)            (None, 24, 24, 4)         76

max_pooling2d_5 (MaxPooling2 (None, 12, 12, 4)         0

flatten_2 (Flatten)          (None, 576)               0

dense_5 (Dense)              (None, 1)                 577

Total params: 673
Trainable params: 673
Non-trainable params: 0
```

In [30]:

```
start_time = time.time()

history_gender = model_gender.fit(x_train3, y_train3, epochs=25, validat

print("Time:", (time.time() - start_time) / 60)
```

```
Epoch 1/25
519/519 [==============================] - 3s 4ms/step - loss: 0.5702 - a
ccuracy: 0.6970 - val_loss: 0.4922 - val_accuracy: 0.7677 Epoch
2/25
519/519 [==============================] - 2s 4ms/step - loss: 0.4693 - a
ccuracy: 0.7842 - val_loss: 0.4556 - val_accuracy: 0.7885 Epoch
3/25
519/519 [==============================] - 2s 4ms/step - loss: 0.4466 - a
ccuracy: 0.7972 - val_loss: 0.4430 - val_accuracy: 0.8005 Epoch
4/25
519/519 [==============================] - 2s 4ms/step - loss: 0.4280 - a
ccuracy: 0.8102 - val_loss: 0.4290 - val_accuracy: 0.8081 Epoch
5/25
519/519 [==============================] - 2s 4ms/step - loss: 0.4162 - a
ccuracy: 0.8142 - val_loss: 0.4164 - val_accuracy: 0.8154 Epoch
6/25
519/519 [==============================] - 2s 4ms/step - loss: 0.4033 - a
ccuracy: 0.8217 - val_loss: 0.4059 - val_accuracy: 0.8158 Epoch
7/25
519/519 [==============================] - 2s 4ms/step - loss: 0.3952 - a
ccuracy: 0.8280 - val_loss: 0.3963 - val_accuracy: 0.8234 Epoch
8/25
519/519 [==============================] - 2s 4ms/step - loss: 0.3875 - a
ccuracy: 0.8309 - val_loss: 0.3969 - val_accuracy: 0.8221 Epoch
9/25
519/519 [==============================] - 2s 4ms/step - loss: 0.3794 - a
ccuracy: 0.8363 - val_loss: 0.3874 - val_accuracy: 0.8296 Epoch
10/25
519/519 [==============================] - 3s 5ms/step - loss: 0.3758 - a
ccuracy: 0.8362 - val_loss: 0.3789 - val_accuracy: 0.8328 Epoch
11/25
519/519 [==============================] - 2s 4ms/step - loss: 0.3697 - a
ccuracy: 0.8387 - val_loss: 0.3752 - val_accuracy: 0.8361 Epoch
12/25
519/519 [==============================] - 2s 4ms/step - loss: 0.3659 - a
ccuracy: 0.8428 - val_loss: 0.3729 - val_accuracy: 0.8376 Epoch
13/25
519/519 [==============================] - 2s 4ms/step - loss: 0.3606 - a
ccuracy: 0.8423 - val_loss: 0.3674 - val_accuracy: 0.8375 Epoch
14/25
519/519 [==============================] - 2s 4ms/step - loss: 0.3577 - a
ccuracy: 0.8475 - val_loss: 0.3718 - val_accuracy: 0.8314 Epoch
15/25
519/519 [==============================] - 2s 5ms/step - loss: 0.3541 - a
ccuracy: 0.8468 - val_loss: 0.3750 - val_accuracy: 0.8400 Epoch
16/25
519/519 [==============================] - 2s 4ms/step - loss: 0.3513 - a
ccuracy: 0.8496 - val_loss: 0.3616 - val_accuracy: 0.8394 Epoch
17/25
519/519 [==============================] - 2s 4ms/step - loss: 0.3483 - a
ccuracy: 0.8513 - val_loss: 0.3648 - val_accuracy: 0.8445 Epoch
18/25
519/519 [==============================] - 2s 4ms/step - loss: 0.3463 - a
ccuracy: 0.8513 - val_loss: 0.3594 - val_accuracy: 0.8442 Epoch
19/25
519/519 [==============================] - 2s 4ms/step - loss: 0.3432 - a
ccuracy: 0.8526 - val_loss: 0.3591 - val_accuracy: 0.8427 Epoch
20/25
519/519 [==============================] - 2s 4ms/step - loss: 0.3410 - a
ccuracy: 0.8542 - val_loss: 0.3572 - val_accuracy: 0.8487 Epoch
21/25
519/519 [==============================] - 2s 4ms/step - loss: 0.3409 - a
ccuracy: 0.8545 - val_loss: 0.3608 - val_accuracy: 0.8462 Epoch
22/25
519/519 [==============================] - 2s 4ms/step - loss: 0.3376 - a
ccuracy: 0.8554 - val_loss: 0.3526 - val_accuracy: 0.8456 Epoch
23/25
519/519 [==============================] - 2s 4ms/step - loss: 0.3374 - a
```

```
ccuracy: 0.8572 - val_loss: 0.3600 - val_accuracy: 0.8383 Epoch
24/25
519/519 [==============================] - 2s 4ms/step - loss: 0.3346 - a
ccuracy: 0.8555 - val_loss: 0.3608 - val_accuracy: 0.8459 Epoch
25/25
519/519 [==============================] - 2s 4ms/step - loss: 0.3325 - a
ccuracy: 0.8580 - val_loss: 0.3574 - val_accuracy: 0.8491
Time: 1.3759745756785076
```

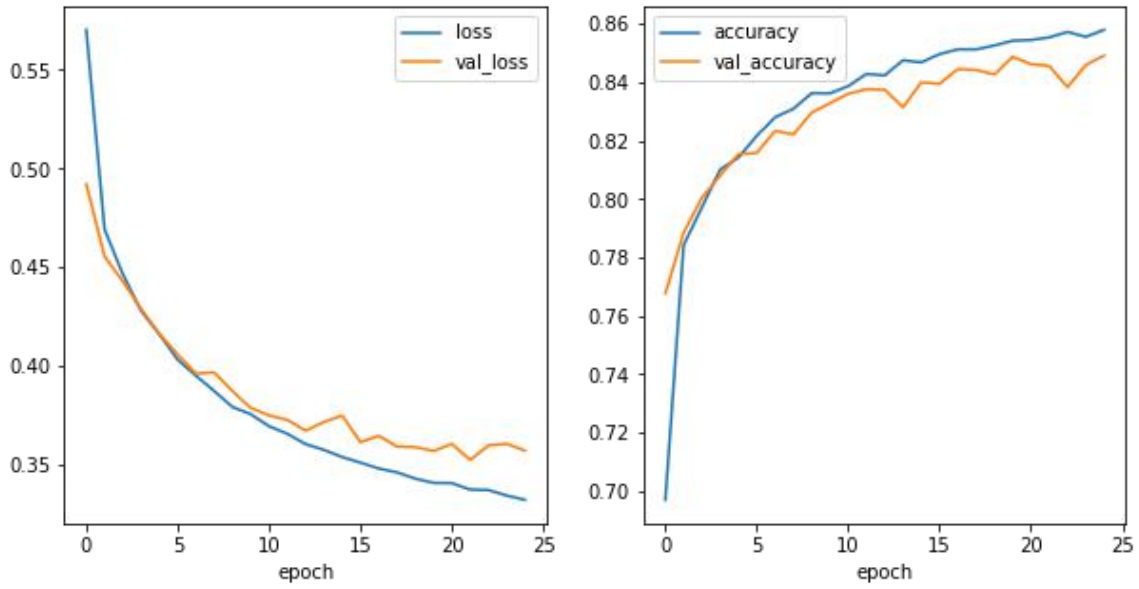
In [31]:

```
val_loss, val_acc = model_gender.evaluate(x_test3, y_test3)

print('Accuracy:', history_gender.history['accuracy'][-1])
print('Val (test) accuracy:', history_gender.history['val_accuracy'][-1]
```

```
223/223 [==============================] - 0s 2ms/step - loss: 0.3574 - a
ccuracy: 0.8491
Accuracy: 0.8579521775245667
Val (test) accuracy: 0.849128246307373
```

In [32]:

```
fig, axs = plt.subplots(1, 2, figsize=(10, 5))
axs[0].plot(history_gender.history['loss'], label='loss')
axs[0].plot(history_gender.history['val_loss'], label='val_loss')
axs[0].set_xlabel('epoch')
axs[0].legend()

axs[1].plot(history_gender.history['accuracy'], label='accuracy')
axs[1].plot(history_gender.history['val_accuracy'], label='val_accuracy'
axs[1].set_xlabel('epoch')
axs[1].legend()
plt.show()
```

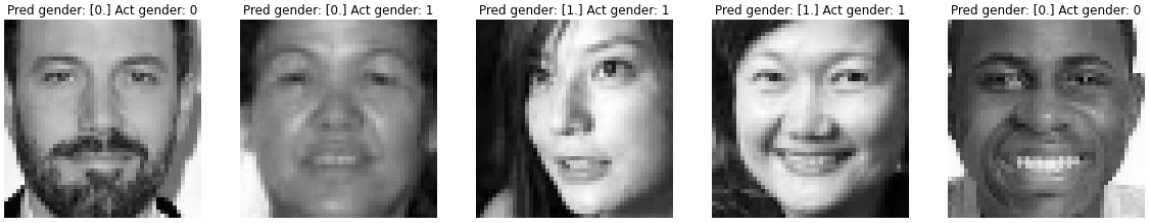


In [33]:

```
predictions3 = model_gender.predict(x_test3)
```

In [34]:

```
fig, axes = plt.subplots(1, 5, figsize=(20, 10))
for i in range(5):
    random_face = np.random.choice(len(predictions3))
    axes[i].set_title("Pred gender: {0} Act gender: {1}".format(np.round
    axes[i].imshow(x_test3[random_face],  cmap='gray')
    axes[i].axis('off')
```



In

```
false_predictions3 = []
x_false3 = []
y_false3 = []
for i in range(len(y_test3)):
    pred_class = np.round(predictions3[i])
    true_class = y_test3[i]
    if pred_class != true_class:
        x_false3.append(x_test3[i])
        y_false3.append(true_class)
        false_predictions3.append(pred_class)
```

In

```
fig, axes = plt.subplots(1, 5, figsize=(20, 10))
for i in range(5):
    random_face = np.random.choice(len(false_predictions3))
    axes[i].set_title('Pred gender: ' + str(false_predictions3[random_fa
    axes[i].imshow(x_false3[random_face], cmap='gray')
    axes[i].axis('off')
plt.savefig("faces1")
```