Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

«КАЗАНСКИЙ (ПРИВОЛЖСКИЙ) ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт математики и механики им. Н. И. Лобачевского

Кафедра теории функций и приближений

Направление подготовки: 02.03.01 – «Математика и компьютерные науки»

Профиль: Наука о данных

КУРСОВАЯ РАБОТА

Определение идентичности лиц на различных изображениях средствами искусственных нейронных сетей

Студент 3 курса

группы 05-804

«__» _____ 2021 г. _____ А. Н. Турашова

Научный руководитель:

к.п.н., доцент каф. ТФиП

«__» _____ 2021 г. _____ С. В. Маклецов

Казань – 2021

ОГЛАВЛЕНИЕ

Введение

С момента изобретения искусственных нейронных сетей они применяются в обширном спектре приложений, например: распознавание образов, прогнозирование, выявление зависимостей, сжатие данных, а также задачи распознавания образов.

В настоящее время распознавание образов находит все больше методов применения в повседневной жизни людей. Методы и алгоритмы теории распознавания широко применяются в следующих областях: медицина, геология, робототехника, астрономия, анализ изображений и идентификация личности. Одной из актуальных задач на данный момент является распознавание людей с помощью видеосъёмки. Например, для идентификации сотрудников компании на пропускных постах или для помощи правоохранительным органам.

Цель курсовой работы — изучить некоторые алгоритмы идентификации лиц на изображениях средствами искусственных нейронных сетей и написать программную реализацию одного из них.

Для достижения этой цели необходимо решить следующие задачи:

1) изучить методы идентификации лиц на изображении;

2) выбрать оптимальный алгоритм нейронной сети для нахождения лиц на фотографиях;

3) спроектировать, реализовать и протестировать нейронную сеть для идентификации лиц на изображении.

Курсовая работа объёмом 21 страниц состоит из введения, двух глав, заключения, списка литературы, содержащего 10 источников, и 4 приложения. В работе имеется 14 рисунков.

1. Методы распознавания лиц

1.1. Выбор метода распознавания лиц

Прежде чем приступить к определению идентичности двух лиц необходимо выяснить, действительно ли на изображении присутствует лицо, в каком месте оно находится. На данный момент существует множество алгоритмов распознавания лица, но все они имеют одинаковую структуру. В первую очередь происходит детектирование [4], то есть обнаружение наличия и местоположения лица на изображении. Затем производится выравнивание лица: подбирается приемлемая яркость и, используя геометрические вычисления, лицо располагают анфас. После получения анфас лица вычисляют особенные признаки и сравнивают эти признаки с заложенными в базу данных эталонами.

Одним из самых эффективных алгоритмов, позволяющих обнаруживать объекты на изображениях, является метод Виолы-Джонса. Он был представлен миру в 2001 году Паулом Виолой и Майклом Джонсом как алгоритм распознавания лиц, но его используют и для распознавания других объектов.

Метод Виола-Джонса работает по принципу скользящего окна. На изображении выделяется прямоугольное окно, размеры которого меньше изображения, в этом окне происходит поиск лица, используя каскад слабых классификаторов. Если лицо не обнаружено, окно смещается в следующую область изображения, со временем уменьшаясь в размерах. Таким образом алгоритм может найти несколько лиц разного размера на одном изображении.

Можно выделить два этапа метода: обучение и распознавание, на каждом из которых применяются соответствующие алгоритмы [1]. В практическом применении скорость работы алгоритма обучения не важна, так как обучается сеть перед началом использования. При этом скорость алгоритма распознавания является главным фактором эффективности работы программы.

Стоит заметить, что метод Виолы-Джонса обладает рядом преимуществ по сравнению с другими алгоритмами поиска лица. Во-первых, принцип скользящего окна даёт возможность обнаруживать более одного лица на изображении. Во-вторых, алгоритм распознавания через простые

классификаторы имеет хорошую скорость, что позволяет использовать его в видеопотоке.

Однако, для того чтобы обучить алгоритм распознавать людей, необходима обширная база данных человеческих лиц, включающая в себя как изображения с лицами, так и без них. На практике время обучения измеряется неделями.

1.2. Выделение объектов на изображении по методу Виолы-Джонса

Метод Виолы-Джонса выделяет объекты (чаще всего он используется для нахождения лиц) прямоугольной рамкой (рис. 1), если в скользящем по изображению окне появились признаки лица. Признаков лица может быть множество, например, похожий на глаз участок изображения или ромб с переходом яркости пикселей, как у переносицы (рис. 2). Но для того, чтобы не было ложных выделений, алгоритм проверяет, действительно ли в окне находится достаточное количество достоверных признаков лица.

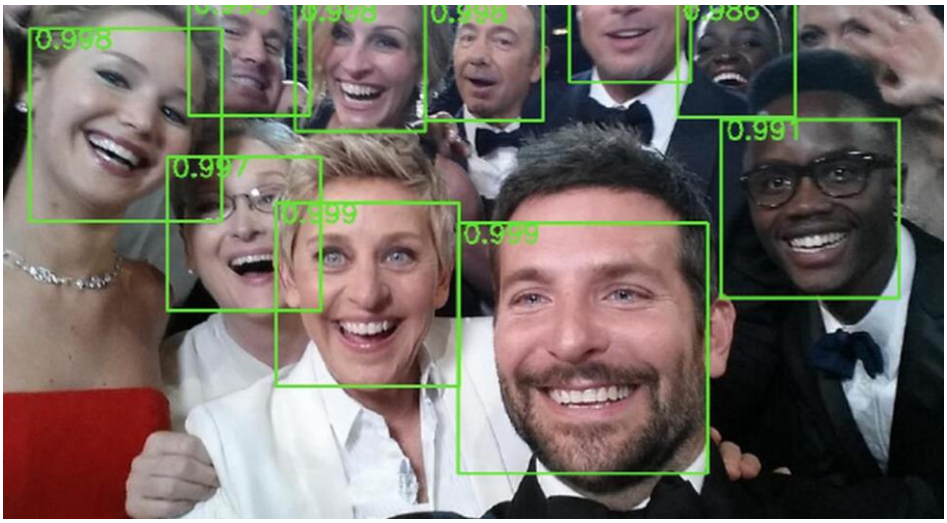


Рис. 1. Выделение лиц методом Виолы-Джонса

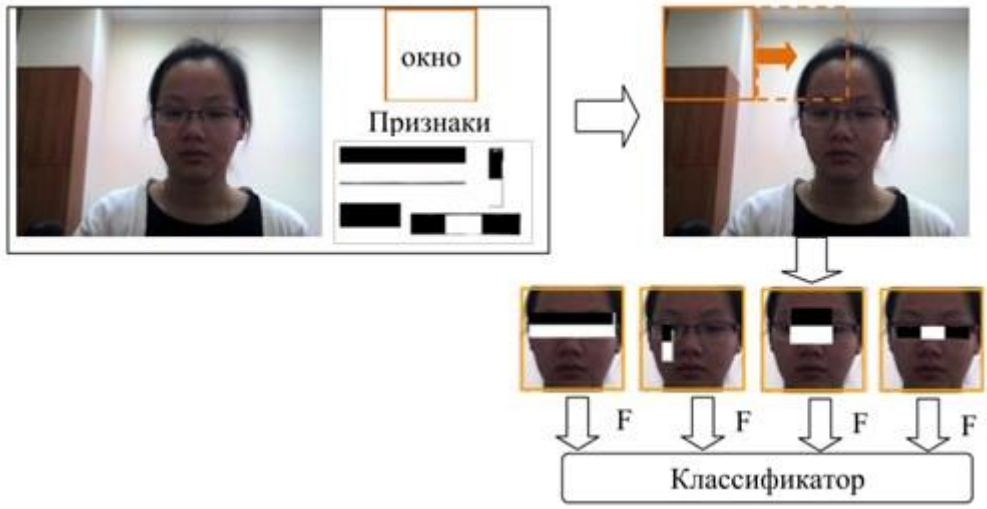


Рис. 2. Скользящее по изображению окно находит признаки лица

Такой подход к детектированию объектов на изображении комбинирует четыре ключевые концепции [6]:

- простые прямоугольные функции-признаки, называемые функциями Хаара;
- интегральное изображение для быстрого обнаружения функции;
- метод машинного обучения AdaBoost [3];
- каскадный классификатор для эффективного совмещения множественных функций.

В качестве признаков объекта были предложены признаки Хаара, основанные на вейвлетах Хаара.

Вейвлет — математическая функция, которая позволяет анализировать различные частотные компоненты данных. График функции вейвлета выглядит как волнообразные колебания с изменяющейся амплитудой [7].

Вейвлет Хаара представляет собой прямоугольные волны одинаковой длины. На плоскости прямоугольная волна выглядит как пара соседних прямоугольников — один светлый и один темный (рис. 3).

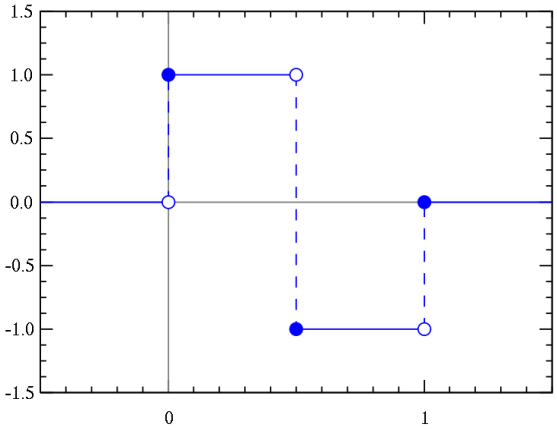


Рис. 3. Вейвлет Хаара

Функция Хаара — это кусочно-постоянная функция, определяемая на интервале [0,1). Две первые функции Хаара определены так:

$$\aleph_0{}^{(0)}(x) = 1,\ \aleph_0{}^{(1)} = \begin{cases} 1, x \in \left[0, \frac{1}{2}\right) \\ 0, x = \frac{1}{2} \\ -1, x \in \left(\frac{1}{2}, 1\right] \end{cases}.$$

Чтобы обнаружить, есть ли в окне функция Хаара, алгоритм вычитает среднее значение области светлых пикселей из среднего значения области тёмных пикселей. Полученное число сравнивается с пороговым. Пороговое число выявляется при обучении [9] алгоритма распознавать лица. Если полученное число равно или больше порогового, то алгоритм понимает, что функция Хаара в окне существует.

Пример выделения прямоугольных областей, являющихся функциями Хаара, изображён на рисунке 4.

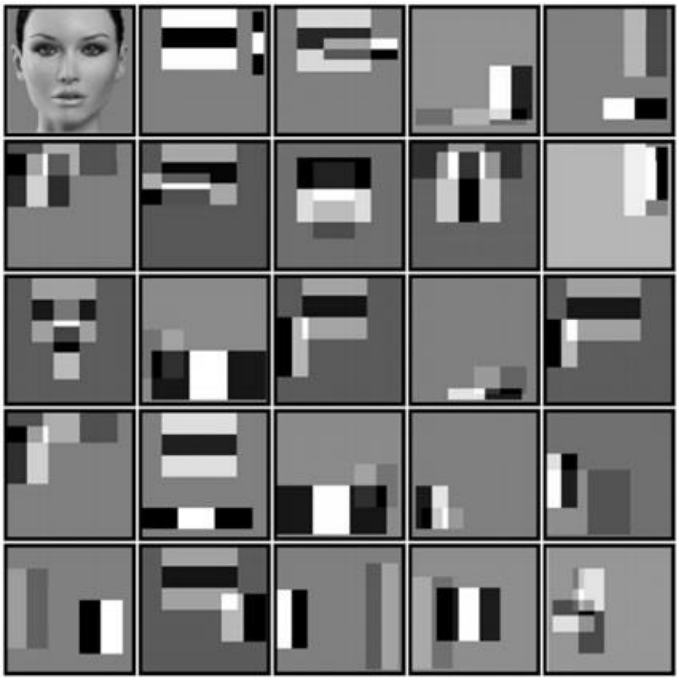


Рис. 4. Выделение прямоугольных областей (функций Хаара)

Чтобы эффективно определить наличие и отсутствие функций Хаара в разных по размеру и местоположению окнах на изображении, Виола и Джонс использовали технологию интегрального представления изображения.

Интегральное представление изображения — это матрица, размерность которой совпадает с размерностью исходного изображения [3].

Используя интегральное изображение, можно быстро вычислить суммирование по подобластям изображений. Интегральные изображения упрощают суммирование пикселей, так как каждый пиксель в нём — это сумма значений всех пикселей в области над ним и слева от него (включая сам пиксель) (рис.5).

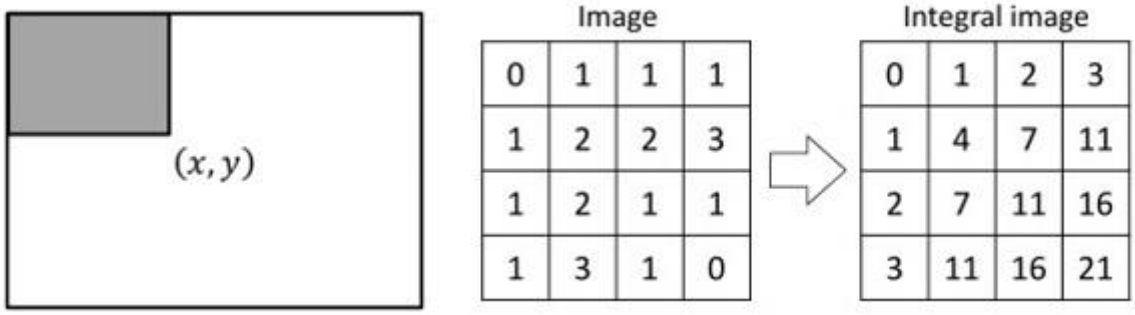


Рис. 5. Пример подсчёта интегральных значений

Чтобы получить значение порогового числа разности светлых и темных пикселей, необходимо провести качественное обучение алгоритма. Виола и Джонс использовали метод машинного обучение AdaBoost [2], так как он работает с классификаторами, выявляя кто из них слабее, а кто сильнее.

Метод машинного обучения получает множество выделенных объектов (областей изображения), часть из которых можно сразу разделить на классы (например, могут существовать классы «анфас положение брови» или «ряд зубов») — такие разделённые на классы объекты называются обучающей выборкой. К каким классом принадлежат остальные объекты не известно. Задачей метода машинного обучение является классификация объектов, которые невозможно со сто процентной вероятностью определить к одному или иному классу [8].

Классификатор — это функция, которая путём аппроксимации указывает, к какому классу и с каким процентом вероятности принадлежит объект.

AdaBoost в процессе обучения комбинирует множество «слабых» классификаторов в один «сильный». «Слабый» классификатор — это классификатор, который правильно классифицирует объект чаще, чем случайное угадывание. Имея множество «слабых» классификаторов, каждый из которых выдвигает ответ в немного верном направлении, можно получить комбинацию, дающую правильный ответ. Но разные классификаторы могут иметь разную частоту угадывания, поэтому AdaBoost присваивает каждому из них свой вес. Комбинация из «слабых» классификаторов с правильно подобранными весами является «сильным» классификатором [9].

В результате обучения и подбора весов формируется простой классификатор вида:

$$h_{(j)}(z) = \begin{cases} 1, \text{если } p_j f_j(z) < p_j \theta_j \\ 0, \text{если } p_j f_j(z) \geq p_j \theta_j \end{cases},$$

где p_j показывает направление знака неравенства; θ_j — значение порога; $f_j(z)$ — вычисленное значение признака; z — окно изображения.

Виола и Джонс объединили несколько «сильных» классификаторов в последовательность фильтров, каждый из которых является классификатором AdaBoost с небольшим числом «слабых» классификаторов. В этой последовательности порог прохождения отдельного фильтра устанавливается достаточно низким, чтобы пройти обучающую выборку (большую базу изображений лиц).

После обучения, во время работы алгоритма Виолы-Джонса, если какой-то из фильтров указывает, что выделенная область «не лицо», то следующие фильтры не начинают работать и окно с выделенной областью смещается дальше. Если область изображения прошла через все фильтры, то она классифицируются как «лицо» (рис. 6).

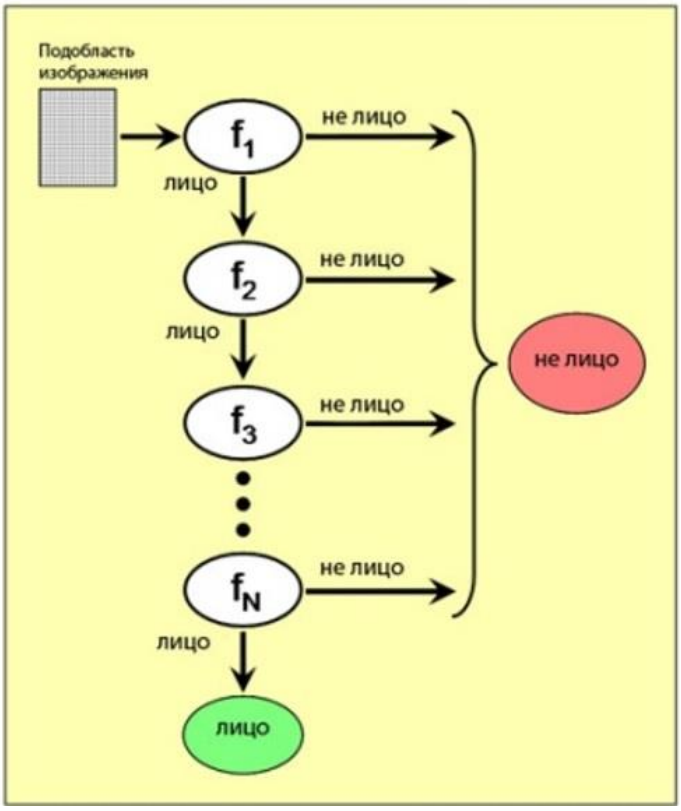


Рис. 6. Работа последовательности фильтров одной выделенной области

Порядок фильтров в последовательности основывается на весах классификаторов в них, которые присваивает AdaBoost. Фильтры с наибольшим значением весов идут в первую очередь для того, чтобы как можно быстрее выявить области без лица. Рисунок номер 7 показывает два первых фильтра в последовательности. Первый основан на том, что область щёк светлее, чем

область глав. А второй на том, что перегородка между глазами светлее области глаз.

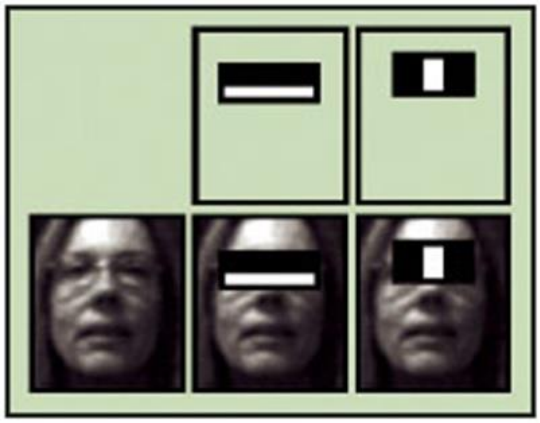


Рис. 7. Первые две функции каскада Виолы-Джонса

1.3. Сиамская нейронная сеть для распознавания лиц

После получения двух найденных и вырезанных лиц необходимо сравнить их. Если сравнивать фотографии попиксельно, то алгоритм никогда не найдёт одинаковые лица. Это связано в первую очередь с тем, что одно и то же лицо на разных фотографиях может быть повёрнутого относительно камеры под разным углом, из-за чего возникает проблема положения головы (рис. 8).

Рис. 8. Проблема положения головы

Помимо положения лица относительно камеры существуют другие проблемы. Например, при неравномерном или слабом освещении границы лица могут быть визуально сдвинуты или размыты, из-за чего возникает проблема освещённости (рис. 9).

Рис. 9. Проблема освещённости

Однако можно уменьшить количество ошибочных сравнений, подобрав и хорошо обучив нейронную сеть. Существует множество видов нейронных сетей, дающих хорошие результаты при работе с изображениями, но для их обучения необходимы большие базы данных.

Нейронная сеть может научиться различать несколько объектов (один объект является одним классом). Например, рассмотрим задачу обучения свёрточных нейронных сетей для распознавания ста различных классов (объектов). Если обучать с нуля такую нейронную сеть, то для получения хорошей точности нам понадобится большое количество фотографий, относящихся к каждому из классов, которые на практике иногда собрать сложно. Рассмотрим пример: поставлена задача распознавания сотрудников компании, но собрать большое количество фотографий каждого из сотрудников весьма проблематично.

Для решения этой проблемы можно воспользоваться алгоритмом сиамской нейронной сети. Он отличается тем, что состоит из двух подсетей с одинаковыми наборами весов. Каждая из подсетей принимает фотографию с лицом и выдаёт вектор неких признаков. Полученные два вектора нейронная сеть сравнивает. В зависимости от того, насколько схожи вектора, нейронная сеть определяет, похожи ли люди на фотографии.

Сиамская нейронная сеть хороша тем, что после её обучения возможно добавить ещё один класс, используя небольшое количество фотографий. Имея

одну фотографию нового сотрудника, при условии, что сеть хорошо обучена на других классах, мы сможем распознать его на кадрах из видео.

Обобщённую архитектуру сиамской нейронной сети можно описать, используя набор данных $\{(x_i, y_i), i = 1, \dots, n\}$ состоящий из n векторов признаков $x_i \in \mathbb{R}^m$ размера m с метриками $y_i \in \{1,2, \dots, C\}$, где C — число классов. Обучающий набор данных $S = \{(x_i, x_j, z_{ij}), i = 1, \dots, n; j = 1, \dots, 2\}$, состоит из пар (x_i, x_j) с бинарными метками z_{ij}. Если оба вектора признаков x_i и x_j принадлежат к одному и тому же классу, то $z_{ij} = 0$, иначе $z_{ij} = 1$. Обучающий набор данных S разделяется на два подмножества: одно — со схожими парами (или с $z_{ij} = 0$), другое — с различающимися парами (или с то $z_{ij} = 1$).

На рисунке 10 изображён общий случай сиамской сети, состоящей из двух подсетей, выходы которых подаются на вход другого модуля, который генерирует конечный выход. На вход подаются пары векторов x_i и x_j, W — общие веса/параметры, а $h_i \in R^D$ и $h_j \in R^D$ — выходы обеих подсетей. Сиамская сеть представляет собой отображение $h_i = f(x_i)$, для которого Евклидово расстояние $d(h_i, h_j)$ максимально мало́ при $y_i = y_j$ и максимально велико при $y_i \neq y_j$. Сеть возвращает оценку при o_W того, насколько различны x_i и x_j.

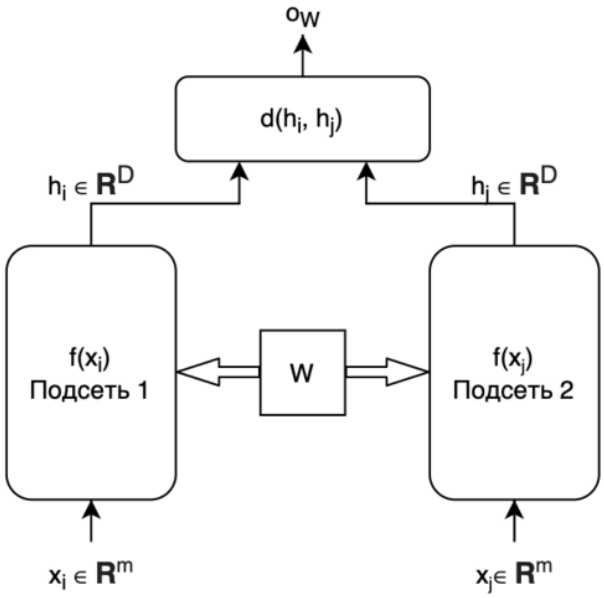


Рис. 10. Обобщённая архитектура сиамской нейронной сети.

2. Написание и тестирование сиамской нейронной сети

2.1. Сбор базы данных лиц из видеофайлов

Прежде чем написать программу, необходимо собрать и создать базу данных. Для этого необходимы сотни фотографий одних и тех же людей с разного ракурса. Наиболее простой способ сбора одинаковых фотографий — сбор с видеозаписи.

Для сбора фотографий взят видеофайл, который с помощью библиотеки OpenCV был разделён на кадры. Для более удобного нахождения лиц над кадрами были сделаны некоторые изменения цветовой гаммы, после чего кадры перевели в чёрно-белый цвет. Все распознанные с помощью метода Виолы-Джонса лица на кадре были обведены синим квадратом. Данный квадрат необходимо вырезать и поместить в папку с именем человека, которому принадлежит лицо.

Натренированные признаки и функции реализации алгоритма поиска лица человека на фотографии уже есть в библиотеке OpenCV [3]. Так как цель курсовой работы не в нахождении объектов, были взяты уже готовые, обученные классификаторы. Для поиска лица на изображении импортируется файл “haarcascade_frontalface_default.xml”, а для поиска глаз “haarcascade_eye.xml” (см. Приложение 1).

Стоит заметить, что библиотека OpenCV не идеально распознаёт лица. Например, она обнаружила лица на следующих фотографиях (Рис. 11.). Если внимательно приглядеться, на изображениях действительно видны первые функции Хаара.

Рис. 11. Ошибочное распознавание лиц библиотекой OpenCV.

Для того, чтобы снизить процент ошибочного распознавания лиц, все изображения будут проверяться на наличие хотя бы одного глаза. Для поиска глаз на изображении импортируется файл “haarcascade_eye.xml” (см. Приложение 1).

Итоговый сбор фотографий изображён на Рис. 12. По каждому человеку собрано около 2000 фотографий, все рассортированы по отдельным папкам.

Рис. 12. Пример собранных и отсортированных фотографий

2.2. Выбор архитектуры сиамской нейронной сети

После сбора изображений для базы данных можно приступать к выбору архитектуры нейронной сети. Как показывает практика, самый эффективный вид сети для классификации изображений — свёрточная нейронная сеть. Одним из её представителей является AlexNet [4]. Она состоит из пяти свёрточных слоёв, между которыми располагаются pooling-слои и слои нормализации, а завершают нейросеть три полносвязных слоя (Рис. 13.).

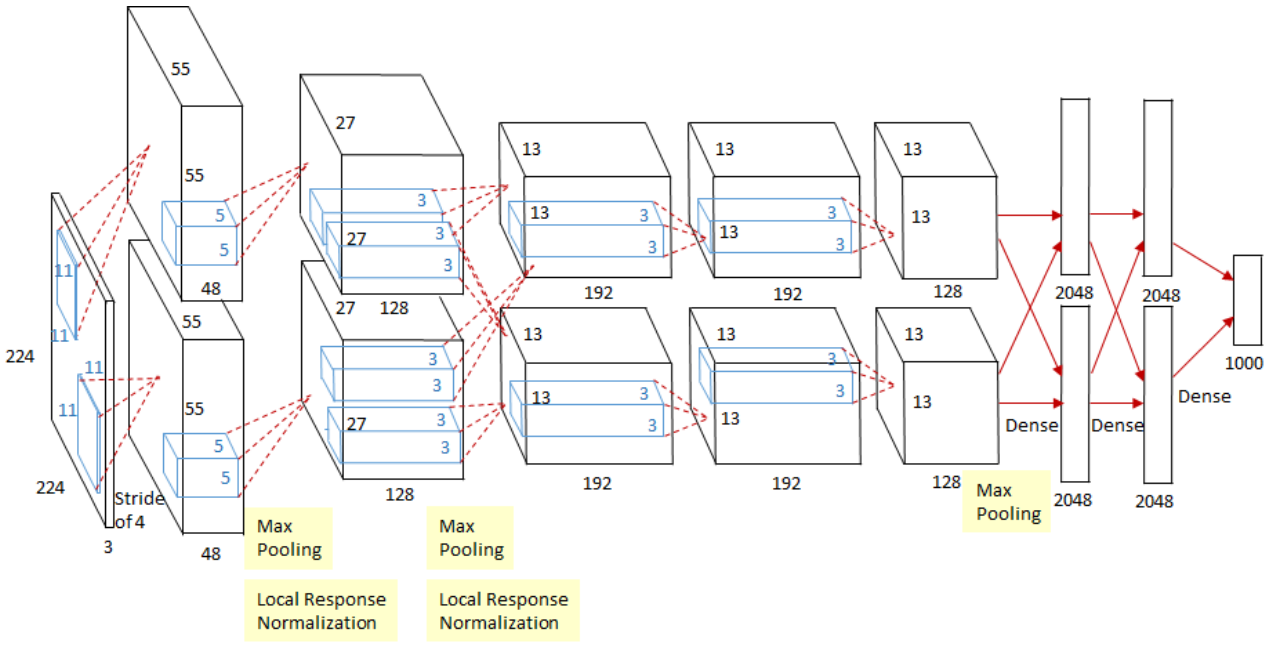


Рис. 13. Схема архитектуры AlexNet.

По схеме видно, что все выходные данные AlexNet разделяются на два потока — это связано с тем, что в 2012 году для её обучения использовалось по две видеокарты, которые параллельно выполняли вычисления над двумя участками изображения.

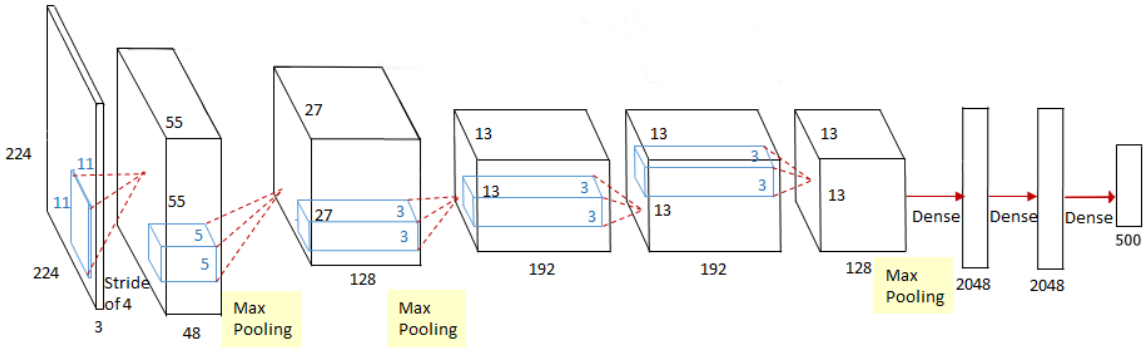


Рис. 14. Схема модифицированной архитектуры AlexNet

Поскольку на имеющемся оборудовании присутствует только одна видеокарта, для программы будет модифицирована архитектура сети — будет взят только один из её участков. Также, в целях повышения производительности обучения, из исходной архитектуры AlexNet будут исключены два последних

слоя. Таким образом структура сети примет следующий вид, изображённый на рисунке 14.

2.3. Алгоритм обучения и тестирования сиамской нейронной сети

После создания модели нейронной сети необходимо приступить к разбивке фотографий из базы данных на пары (Приложение 2.). Каждая пара будет проходить через одну нейронную сеть, получая на выходе два вектора. Сходство фотографий будет оценено путём сравнения этих двух векторов. Для того, чтобы правильно сравнить векторы, они будут переданы в функцию энергии, которая выдаст результат сходства.

После того, как сеть собрана, необходимо оценить её эффективность. Для этого была создана функция метрики. Такая функция помогает оценить производительность модели. После обучения нейронной сети полученные веса были сохранены в файл “checkpoint” (Приложение 3). Обучение происходило на 7000 фотографиях, наилучшая точность определения человеческого лица при обучении составила 82%. Тестовый набор фотографий содержал в себе 3000 фотографии, точность определения составила 72%.

Для проверки нейронной сети на практической задаче был вновь взят исходный видеофайл, из которого были извлечены фотографии “изученных” нами людей. Для распознавания лиц на кадрах видеопотока, была написана программа, используя библиотеку OpenCV (Приложение 3). Функция создания модифицированной модели нейросети AlexNet совпадает с такой же функцией в приложении 2. Как и в предыдущем разделе, в программе создаются пары изображений, которые проходят через одну нейронную сеть. В качестве первого изображения берётся найденное на кадре видеопотока лицо человека. В качестве второго изображения импортируется фотография человека, которого мы хотим сравнить с человеком из видео.

В программе импортированы фотографии двух людей, имеющихся на видео. Нейронная сеть смогла обнаружить этих людей на кадрах видеопотока. Примеры с обнаруженными людьми находятся в приложении 4.

Можно убедиться, что нейросеть довольно часто угадывает имя человека, но иногда находит лица там, где их нет, или всем присутствующим в кадре приписывает одно и то же имя, но с разной вероятностью.

Заключение

Таким образом, подведя итоги работы, можно заключить, что цель исследования, а именно изучение некоторых алгоритмов идентификации лиц на изображениях средствами искусственных нейронных сетей и написание программной реализации одного из них, была достигнута.

Изучая некоторые алгоритмы идентификации лиц на изображениях, был выделен метод Виолы-Джонса, который обладает рядом преимуществ по сравнению с другими алгоритмами поиска лица. Его принцип скользящего окна даёт возможность обнаруживать более одного лица на изображении, а алгоритм распознавания через простые классификаторы имеет хорошую скорость, что позволяет использовать его в видеопотоке.

В ходе изучения этого метода было выявлено, что он является наиболее оптимальным алгоритмом для нахождения лиц на фотографиях. Выяснено, что метод Виолы-Джонса базируется на простых прямоугольных функциях-признаках, называемых функциями Хаара, и использует каскадный классификатор для эффективного совмещения множественных функций.

Перебирая различные варианты сравнения лиц было обнаружено, что для идентификации лиц оптимальнее использовать сиамскую нейронную сеть, которая после полного обучения может на основе небольшого количества фотографий нового объекта узнавать этот объект на изображениях.

В ходе написании программы было обнаружена проблема нехватки изображений для обучения нейронной сети. Для её решения была написана программа сбора фотографий из видеопотока. Фотографии автоматически обрабатывались и вручную сортировались по папкам, каждая из которых в итоге хранила примерно по 2000 фотографий одного человека.

При изучении различных архитектур сиамских нейронных сетей была найдена оптимальная для нашей задачи архитектура AlexNet. Изначальна она создавалась для обучении на компьютерах с двумя видеокартами, но на

имеющемся оборудовании присутствовала только одна видеокарта, из-за чего возникла проблема несовместимости архитектуры с оборудованием. Для решения этой проблемы AlexNet была модифицирована и приспособлена для решения поставленной задачи.

Полученная нейронная сеть была обучена на приблизительно 7000 фотографиях с результатом 82% угаданных лиц. На 3000 тестируемых фотографиях процент был ниже — 72 %. Также была написана программа, позволяющая распознавать людей на видео.

Литература

1. Виола П. М. Надежное обнаружение объектов в реальном времени: серия технических отчетов. / П. М. Виола, Дж. Джонс. — 1-е издание, Кембриджская исследовательская лаборатория, 2001. — 30 с.

2. Хайкин С. Нейронные сети: полный курс. / С. Хайкин. — 2-е издание, Издательский дом Вильямс, 2008. — 1104 с.

3. Брадски Г. Документация к библиотеке компьютерного зрения «OpenCV» [Электронный ресурс]. / Г. Брадски. — URL: https://docs.opencv.org/2.4/modules/contrib/doc/facerec/facerec_tutorial.html (дата обращения: 14.03.2021).

4. Савченко С. О. Детектирование лиц и выделение их контуров. / С. О. Савченко, А. Р. Семенова // Прикладная математика и фундаментальная информатика. — 2015. — №2. — С. 143-146.

5. Осовский С. Нейронные сети для обработки информации: учебное пособие. / С. Осовский. — 2002. — 344 с.

6. Великий Я. О. Анализ принципа распознавания объектов на изображении методом Виолы-Джонса / Я. О. Великий. // Открытые информационные и компьютерные интегрированные технологии. — 2015. — № 68. — С. 162-166.

7. Поликар Р. Введение в Вейвлет-преобразование: учебное пособие. / Р. Поликар. — АВТЭКС Санкт-Петербург, 2013. — 59 с.

8. Местецкий Л. М. Математические методы распознавания образов: курс лекций. / Л. М. Местецкий. — МГУ, ВМиК, Москва, 2004. — 85 с.

9. Траск Э. Глубокое обучение: учебное пособие. / Э. Траск. — 1-е издание, издательство Питер, 2019. — 352 с.

10. Свирневский Н. Распознавание лиц на основе OpenCV для C++ [Электронный ресурс]. / Н. Свирневский, С. Иващенко. — URL: https://api-2d3d-cad.com/face_recognition_with_opencv/ (дата обращения: 23.04.2021).

Приложения

Приложение 1

Код программы, вырезающей из пользовательского видео лица людей для базы данных.

```
import cv2
import os

cascadeFace = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
cascadeEye = cv2.CascadeClassifier('haarcascade_eye.xml')
SaveVideoPatch = 'video/video_1/'
prefix = '2_'
Play = True
VideoPatch = 'video_sources/SeroVolk/1.mp4'
vcap = cv2.VideoCapture(VideoPatch)
counter = 802

while (vcap.isOpened()) and Play:
    retCap, frameCap = vcap.read()
    grayColor = cv2.cvtColor(frameCap, cv2.COLOR_BGR2GRAY)
    foundFaces = cascadeFace.detectMultiScale(grayColor, scaleFactor=1.2,
minNeighbors=3, minSize=(100, 100))

    for (x, y, w, h) in foundFaces:
        grayRoi = grayColor[y:y + h, x:x + w]
        foundEyes = cascadeEye.detectMultiScale(grayRoi, scaleFactor=1.2,
minNeighbors=4, minSize=(10, 10))
        if len(foundEyes) > 0:
            name = prefix + str(counter) + '.jpg'
            counter += 1
            size = (100, 100)
            output = cv2.resize(grayRoi, size, interpolation=cv2.INTER_AREA)
            cv2.imwrite(os.path.join(SaveVideoPatch, name), output)
            cv2.rectangle(frameCap, (x, y), (x + w, y + h), (255, 0, 0), 2)

    cv2.imshow("camera", frameCap)
    if cv2.waitKey(10) == 27:
        Play = False

vcap.release()
cv2.destroyAllWindows()
```

Приложение 2

Код программы, собирающий и обучающий сиамскую нейронную сеть.

```
import keras
import cv2
import os
import numpy as np
import random

numberPeople = 5
epochsLear = 50
VideoPatch = "video/"

def createNeuralNetwork(inputShape):
    """Создаём модель нейросети"""
    neuralNetwork = keras.models.Sequential()
    neuralNetwork.add(keras.layers.Conv2D(48, kernel_size=11, activation='relu',
inputShape=inputShape))
    neuralNetwork.add(keras.layers.Conv2D(128, kernel_size=5,
activation='relu'))
    neuralNetwork.add(keras.layers.MaxPooling2D(pool_size=3))
    neuralNetwork.add(keras.layers.Dropout(.25))
    neuralNetwork.add(keras.layers.Conv2D(192, kernel_size=3,
activation='relu'))
    neuralNetwork.add(keras.layers.MaxPooling2D(pool_size=3))
    neuralNetwork.add(keras.layers.Dropout(.25))
    neuralNetwork.add(keras.layers.Conv2D(192, kernel_size=3,
activation='relu'))
    neuralNetwork.add(keras.layers.Conv2D(128, kernel_size=3,
activation='relu'))
    neuralNetwork.add(keras.layers.Flatten())
    neuralNetwork.add(keras.layers.Dense(2048, activation='relu'))
    neuralNetwork.add(keras.layers.Dense(2048, activation='relu'))
    neuralNetwork.add(keras.layers.Dense(500, activation='relu'))
    return neuralNetwork

def readDataSet():
    """Получаем список с преобразованными пикселями фотографии"""
    indexNumber = 0
    indexFoto = []
    listFoto = []
    listDir = os.listdir(VideoPatch)
    for i in listDir:
        if os.path.isdir(VideoPatch + i):
            next = os.listdir(VideoPatch + i)
            for a in next:
                imageAddress = cv2.imread(VideoPatch + i + '/' + a)
                listFoto.append(imageAddress)
                indexFoto.append(indexNumber)
            indexNumber += 1
    indexFoto = np.array(indexFoto)
    listFoto = np.array(listFoto)
    listFoto = np.resize(listFoto, (len(listFoto), 100, 100, 1))
    return listFoto, indexFoto
```

```
def createPairsOfPhotos(x, digitIndices):
    """Создаём позитивные и негативные пары"""
    listPairs = []
    listLabels = []
    length = len(digitIndices[0]) - 1
    for i in range(len(digitIndices) - 1):
        if (len(digitIndices[i]) < len(digitIndices[i+1])) and
(len(digitIndices[i]) > 1):
            length = len(digitIndices[i]) - 1
        else:
            length = len(digitIndices[i+1]) - 1
    for j in range(numberPeople):
        for i in range(length):
            a1, a2 = digitIndices[j][i], digitIndices[j][i + 1]
            listPairs += [[x[a1], x[a2]]]
            rand = random.randrange(1, numberPeople)
            jrand = (j + rand) % numberPeople
            a1, a2 = digitIndices[j][i], digitIndices[jrand][i]
            listPairs += [[x[a1], x[a2]]]
            listLabels += [1, 0]
    return np.array(listPairs), np.array(listLabels)

def energyFunction(vectors):
    """Функция расчета энергии, евклидово расстояние"""
    x, y = vectors
    sumSquare = keras.backend.sum(keras.backend.square(x - y), axis=1,
keepdims=True)
    return keras.backend.sqrt(keras.backend.maximum(sumSquare,
keras.backend.epsilon()))

def outputShapeEnergyFunction(inletShapes):
    """Функция для вывода расчета энергии"""
    inletShapes1, inletShapes2 = inletShapes
    return (inletShapes1[0], 1)

def lossContrastive(yTrue, yPrediction):
    """Функция контрастной потери"""
    profit = 1
    square = keras.backend.square(yPrediction)
    profitSquare = keras.backend.square(keras.backend.maximum(profit -
yPrediction, 0))
    return keras.backend.mean(yTrue * square + (1 - yTrue) * profitSquare)

def modelPerformanceEvaluation(yTrue, yPrediction):
    """Функция метрики, используется для оценки производительности модели"""
    prediction = yPrediction.ravel() < 0.5
    return np.mean(prediction == yTrue)

def classificationAccuracy(yTrue, yPrediction):
    """Функция, вычисляющая точность классификации с фиксированным порогом
расстояний"""
    return keras.backend.mean(keras.backend.equal(yTrue,
keras.backend.cast(yPrediction < 0.5, yTrue.dtype)))

xTrain, yTrain = readDataSet()
xTrain = xTrain.astype('float32')
xTrain /= 255
inputShape = xTrain.shape[1:]
```

```
digitIndices = [np.where(yTrain == i)[0] for i in range(numberPeople)]
trainPairs, trainY = createPairsOfPhotos(xTrain, digitIndices)
testPairs = trainPairs[0:200]
testY = trainY[0:200]
trainPairs = trainPairs[200::]
trainY = trainY[200::]

baseNetwork = createNeuralNetwork(inputShape)
input1 = keras.layers.Input(shape=inputShape)
input2 = keras.layers.Input(shape=inputShape)
processed1 = baseNetwork(input1)
processed2 = baseNetwork(input2)
distance = keras.layers.Lambda(energyFunction,
output_shape=outputShapeEnergyFunction)([processed1, processed2])
model = keras.models.Model([input1, input2], distance)

""" Запускаем обучение нейронной сети:"""
rmsprop = keras.optimizers.RMSprop()
model.compile(loss=lossContrastive, optimizer=rmsprop,
metrics=[classificationAccuracy])
history = model.fit([trainPairs[:, 0], trainPairs[:, 1]], trainY, batch_size=30,
epochsLear=1, validation_data=([testPairs[:, 0], testPairs[:, 1]], testY),
verbose=2)
model.save_weights('checkpoint')
yPrediction = model.predict([trainPairs[:, 0], trainPairs[:, 1]])
trainAcc = modelPerformanceEvaluation(trainY, yPrediction)
yPrediction = model.predict([testPairs[:, 0], testPairs[:, 1]])
testAcc = modelPerformanceEvaluation(testY, yPrediction)
print('* Точность на тренировочной выборке: %0.2f%%' % (100 * trainAcc))
print('* Точность на испытательном наборе: %0.2f%%' % (100 * testAcc))
```

Приложение 3

Код программы, тестирующий нейронную сеть на видеофайле.

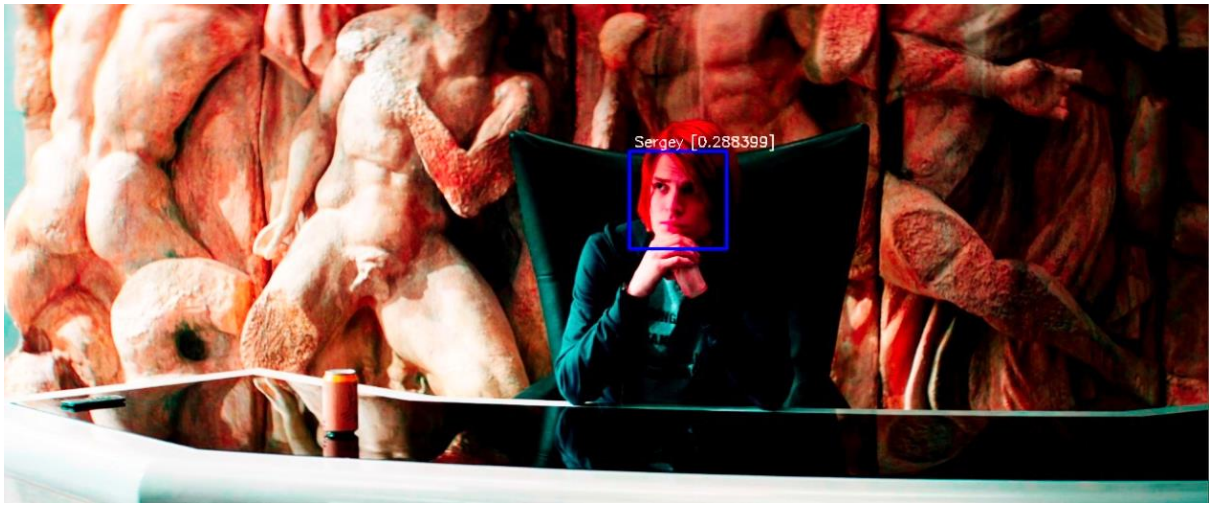
```
import keras
import cv2
import numpy as np
VideoPatch = 'video_sources/1.mp4'
Play = True
cv2font = cv2.FONT_HERSHEY_SIMPLEX
cascadeFace = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
cascadeEye = cv2.CascadeClassifier('haarcascade_eye.xml')

def giveOutSimilarityRate(videoImage):
    """Сверяет загруженные изображения с классами и выдаёт коэфф. сходства"""
    result1 = []
    Oleg = np.zeros([1, 2, 100, 100])
    Sergey = np.zeros([1, 2, 100, 100])
    tmp1 = cv2.imread('s/s_Oleg/2_426.jpg', cv2.IMREAD_UNCHANGED)
    tmp2 = cv2.imread('s/s_Sergey/2_134.jpg', cv2.IMREAD_UNCHANGED)
    Oleg[0, 0, :, :] = tmp1
    Oleg[0, 1, :, :] = videoImage
    Oleg /= 255
    Oleg = Oleg.reshape(1, 2, 100, 100, 1)
    Sergey[0, 0, :, :] = tmp2
    Sergey[0, 0, :, :] = videoImage
    Sergey /= 255
    Sergey = Sergey.reshape(1, 2, 100, 100, 1)
    yPrediction = model.predict([Oleg[:, 0], Oleg[:, 1]])
    result1.append(yPrediction[0])
    yPrediction = model.predict([Sergey[:, 0], Sergey[:, 1]])
    result1.append(yPrediction[0])
    names = ['Oleg', 'Sergey']
    result1 = np.array(result1)
    return names[np.argmin(result1)], result1[np.argmin(result1)]

def createNeuralNetwork(inputShape):
    """Создаём модель нейросети"""
    neuralNetwork = keras.models.Sequential()
    neuralNetwork.add(keras.layers.Conv2D(48, kernel_size=11, activation='relu',
inputShape=inputShape))
    neuralNetwork.add(keras.layers.Conv2D(128, kernel_size=5,
activation='relu'))
    neuralNetwork.add(keras.layers.MaxPooling2D(pool_size=3))
    neuralNetwork.add(keras.layers.Dropout(.25))
    neuralNetwork.add(keras.layers.Conv2D(192, kernel_size=3,
activation='relu'))
    neuralNetwork.add(keras.layers.MaxPooling2D(pool_size=3))
    neuralNetwork.add(keras.layers.Dropout(.25))
    neuralNetwork.add(keras.layers.Conv2D(192, kernel_size=3,
activation='relu'))
    neuralNetwork.add(keras.layers.Conv2D(128, kernel_size=3,
activation='relu'))
    neuralNetwork.add(keras.layers.Flatten())
    neuralNetwork.add(keras.layers.Dense(2048, activation='relu'))
    neuralNetwork.add(keras.layers.Dense(2048, activation='relu'))
    neuralNetwork.add(keras.layers.Dense(500, activation='relu'))
    return neuralNetwork
```

```
def energyFunction(vectors):
    """Функция расчета энергии, евклидово расстояние"""
    x, y = vectors
    sumSquare = keras.backend.sum(keras.backend.square(x - y), axis=1,
keepdims=True)
    return keras.backend.sqrt(keras.backend.maximum(sumSquare,
keras.backend.epsilon()))

def outputShapeEnergyFunction(inletShapes):
    """Функция для вывода расчета энергии"""
    inletShapes1, inletShapes2 = inletShapes
    return (inletShapes1[0], 1)

def lossContrastive(yTrue, yPrediction):
    """Функция контрастной потери"""
    profit = 1
    square = keras.backend.square(yPrediction)
    profitSquare = keras.backend.square(keras.backend.maximum(profit -
yPrediction, 0))
    return keras.backend.mean(yTrue * square + (1 - yTrue) * profitSquare)

inputShape = [100, 100, 1]
baseNetwork = createNeuralNetwork(inputShape)
input1 = keras.layers.Input(shape=inputShape)
input2 = keras.layers.Input(shape=inputShape)
processed1 = baseNetwork(input1)
processed2 = baseNetwork(input2)
distance = keras.layers.Lambda(energyFunction,
output_shape=outputShapeEnergyFunction)([processed1, processed2])
model = keras.models.Model([input1, input2], distance)
print(model.summary())
rmsprop = keras.optimizers.RMSprop()
model.compile(loss=lossContrastive, optimizer=rmsprop)
model.load_weights('checkpoint')
videoCap = cv2.VideoCapture(VideoPatch)
while (videoCap.isOpened()) and Play:
    v, vFrame = videoCap.read()
    grayColor = cv2.cvtColor(vFrame, cv2.COLOR_BGR2GRAY)
    foundFaces = cascadeFace.detectMultiScale(grayColor, scaleFactor=1.2,
minNeighbors=5, minSize=(100, 100))
    for (x, y, w, h) in foundFaces:
        grayRoi = grayColor[y:y + h, x:x + w]
        foundEyes = cascadeEye.detectMultiScale(grayRoi, scaleFactor=1.1,
minNeighbors=3, minSize=(10, 10))
        if len(foundEyes) > 0:
            cv2.rectangle(vFrame, (x, y), (x + w, y + h), (255, 0, 0), 2)
            size = (100, 100)
            out = cv2.resize(grayRoi, size, interpolation=cv2.INTER_AREA)
            text, ping = giveOutSimilarityRate(out)
            text = text + ' ' + str(ping)
            cv2.putText(vFrame, text, (x + 5, y - 5), cv2font, 0.5, (255, 255,
255), 1)
    if v == True: cv2.imshow('Frame', vFrame)
    else: break
    if cv2.waitKey(10) == 27: Play = False
videoCap.release()
cv2.destroyAllWindows()
```

Приложение 4

Примеры работы сиамской нейронной сети.


Sergey [0.24589884]


Sergey [0.288399]


Sergey [0.0790694]
Sergey [0.23478496]