Федеральное государственное бюджетное образовательное учреждение высшего образования

«Московский государственный технический университет имени Н.Э. Баумана (Национальный исследовательский университет)»   
(МГТУ им. Н.Э. Баумана)

Факультет Информатики и систем управления (ИУ)

Кафедра Информационные системы и телекоммуникации (ИУ-3)

**Отчёт**

**по лабораторной работе №3**

**«Разработка программы распознавания лица с использованием**

**свёрточной нейронной сети (CNN)»**

**по курсу**

**«Цифровая обработка изображений»,**

**1 курс, 2-й семестр.**

Выполнил: студент группы ИУ3-21М

Смирнова К.Д.,

Проверил: Алфимцев А.Н.

Москва, 2020

Оглавление

[Задание 3](#_Toc43123282)

[Теоретическая часть 4](#_Toc43123283)

[Свёрточные нейронные сети 4](#_Toc43123284)

[Выполнение 6](#_Toc43123285)

[Датасет 6](#_Toc43123286)

[Выбранные модели нейронных сетей 8](#_Toc43123287)

[Выполнение программы 9](#_Toc43123288)

[Модель 2: VGG16 12](#_Toc43123289)

[Список источников 17](#_Toc43123290)

[Листинг 18](#_Toc43123291)

**Задание**

1. Получить у преподавателя вариант изображений лиц (сами варианты внизу).
2. В среде Spyder (сборка Anaconda) на языке Python 3.х создать проект и

подключить библиотеку scikit-learn , tensorflow или pytorch.

1. Скачать базу данных лиц Yale Face Database https://www.kaggle.com/olgabelitskaya/yale-face-database
2. По аналогии с форматом Yale Face Database подготовить 10 изображений своего лица.
3. Выбрать реализацию и обучить сверточную нейронную сеть (CNN) для

распознавания в соответствии с вариантом лица человека с маскировкой очками, сбежавшего из-под карантина, в случае его попадания в поле зрения наружной видеокамеры с 10 альтернативными лицами (в том числе изображения из п. 4).

1. Провести эксперимент по распознаванию лиц с визуализацией

результатов.

1. Провести информационный поиск в Интернете. Попробовать несколько вариантов архитектур CNN. Выбрать лучшую архитектуру как минимум из 2. Построить график ошибок первого и второго рода по результатам распознавания.
2. Дать комментарий каждой строчке кода!
3. Прислать программу и подготовленные изображения лиц преподавателю.
4. Подготовить и прислать отчет (титульный лист, задание, теоретическая

часть, архитектуры сверточных нейронных сетей, принтскрины выборки лиц, диаграмма структуры программы, принтскрины основных шагов работы программы, заключение и выводы, листинг программы с комментариями, список использованной литературы).

**Вариант:**

|  |  |
| --- | --- |
| **№** | **Субъект** |
| 12 | Subject12 |

**Теоретическая часть**

**Свёрточные нейронные сети**

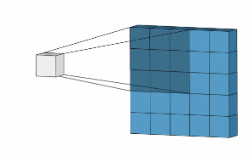
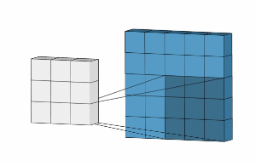
Обозначена задача классификации изображений. Задача классификации изображений — это приём начального изображения и вывод его класса (кошка, собака и т.д.) или группы вероятных классов, которая лучше всего характеризует изображение. Компьютер принимает на вход изображения в виде массива пикселей. Если есть цветное изображение в формате JPG, и его размер 480х480, то соответствующий массив будет 480х480х3. Каждому из этих чисел присваивается значение от 0 до 255, которое описывает интенсивность пикселя в этой точке. Идея в том, что при передаче программе этой матрицы, она выводит числа, которые описывают вероятность класса изображения.

Свёрточная нейронная сеть — специальная архитектура нейронных сетей, изначально нацеленная на эффективное распознавание изображений. Идея свёрточных нейронных сетей заключается в чередовании свёрточных слоёв и субдискретизирующих слоёв. Структура сети — однонаправленная (без обратных связей), принципиально многослойная. Для обучения используются стандартные методы, чаще всего метод обратного распространения ошибки. Функция активации нейронов (передаточная функция) — любая, по выбору исследователя.

В свёрточной нейронной сети выходы промежуточных слоев образуют матрицу (изображение) или набор матриц (несколько слоёв изображения). Основными видами слоев в свёрточной нейронной сети являются свёрточные слои, пулинговые слои и полносвязные слои.

Свёрточный слой нейронной сети представляет из себя применение операции свертки к выходам с предыдущего слоя, где веса ядра свертки являются обучаемыми параметрами. Еще один обучаемый вес используется в качестве константного сдвига.

Двумерная свертка (2D convolution) — это довольно простая операция: начинаем с ядра, представляющего из себя матрицу весов (*weight matrix*). Ядро “скользит” над двумерным изображением, поэлементно выполняя операцию умножения с той частью входных данных, над которой оно сейчас находится, и затем суммирует все полученные значения в один выходной пиксель .

Двумерная свертка

Ядро повторяет эту процедуру с каждой локацией, над которой оно “скользит”, преобразуя двумерную матрицу в другую все еще двумерную матрицу признаков. Признаки на выходе являются взвешенными суммами (где веса являются значениями самого ядра) признаков на входе, расположенных примерно в том же месте, что и выходной пиксель на входном слое.

Независимо от того, попадает ли входной признак в “примерно то же место”, он определяется в зависимости от того, находится он в зоне ядра, создающего выходные данные, или нет. Это значит, что размер ядра сверточной нейронной сети определяет количество признаков, которые будут объединены для получения нового признака на выходе.

Пулинговый слой призван снижать размерность изображения. Исходное изображение делится на блоки и для каждого блока вычисляется некоторая функция. Чаще всего используется функция максимума или (взвешенного) среднего. Обучаемых параметров у этого слоя нет. Основные цели пулингового слоя: уменьшение изображения, чтобы последующие свертки оперировали над большей областью исходного изображения; увеличение инвариантности выхода сети по отношению к малому переносу входа; ускорение вычислений.

После нескольких прохождений свёртки изображения и уплотнения с помощью пулинга система перестраивается от конкретной сетки пикселей с высоким разрешением к более абстрактным картам признаков, как правило на каждом следующем слое увеличивается число каналов и уменьшается размерность изображения в каждом канале. В конце концов остаётся большой набор каналов, хранящих небольшое число данных (даже один параметр), которые интерпретируются как самые абстрактные понятия, выявленные из исходного изображения.

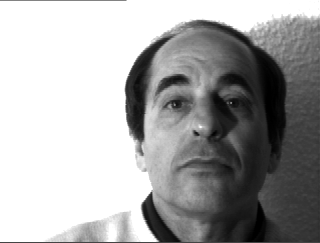
Эти данные объединяются и передаются на обычную полносвязную нейронную сеть, которая тоже может состоять из нескольких слоёв. При этом полносвязные слои уже утрачивают пространственную структуру пикселей и обладают сравнительно небольшой размерностью (по отношению к количеству пикселей исходного изображения).

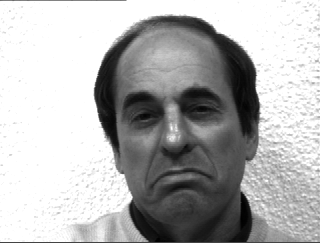
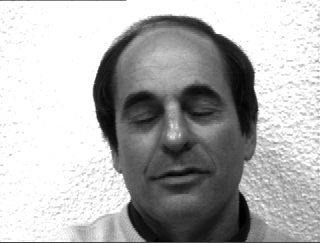
**Выполнение**

**Датасет**

Для выполнения лабораторной работы был подготовлен общий датасет, состоящий из одиннадцати фотографий лица человека из базы данных Yale Face Database согласно варианту (рисунок 1) и одиннадцати фотографий собственного лица (рисунок 2). На фотографиях человек расположен анфас, на преимущественно белом фоне. Все изображения специально подготовлены: полутоновое пространство и размеры 320x243. В обеих подборках первые десять фотографий используются для обучения нейронных сетей, а на одиннадцатой фотографии происходит тестирование их работы.

Фотографии и код доступны на <https://github.com/karinoizerr/Digital-image-processing/tree/master/lab3>

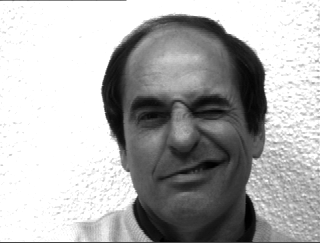
 

Рисунок 1 – Фотографии из базы данных Yale Face Database

Рисунок 2 – Фотографии меня, сбежавшей из-под карантина

**Выбранные модели нейронных сетей**

Первая модель свёрточной нейронной сети была создана по материалам TensorFlow Convolutional Neural Network [1]. Архитектура сети включает в себя 3 каскада слоёв двумерной свёртки layers.Conv2D и объединения layers.MaxPool2D для выделения важных признаков изображений, причём на первом и втором слоях свёртки работает ядро 3x3 с количеством карт признаков в 32 штуки, а у третьего слоя 64 карты признаков. Применяется функция активации «выпрямитель» или ReLu: A(x) = max(0,x). Далее слой layers.Flatten «выравнивает» двухмерный вывод MaxPooling2D в одномерный вектор.

Вторая модель имеет архитектуру VGG16, описанную в [2] на основе материалов в публикации [3]. Также при построении сети использовался материал [4]. В этой архитектуре на вход слоя layers.Conv2D\_1 подаются изображения размера 320х243. Далее изображения проходят через стек свёрточных слоев, в которых используются фильтры с полем размера 3х3. Пространственное дополнение (padding) входа свёрточного слоя выбирается, чтобы пространственное разрешение сохранялось после свертки, то есть дополнение равно 1 для 3х3 свёрточных слоев.  Пространственный пулинг осуществляется при помощи пяти layers.MaxPooling2D слоев. Операция layers.MaxPooling2D выполняется на окне размера 2х2 пикселей с шагом 2. После стека сверточных слоев идут три полносвязных слоя: первые два имеют по 4096 каналов, третий — 2 канала. Все скрытые слои снабжены ReLU.

**Выполнение программы**

В начале осуществляется создание генеаратора изображений и загрузка изображений в программа c использованием класса ImageDataGenerator.

Параллельно с этим генерируются сами модели нейронных сетей на основе Keras Sequential API. На рисунке 3 изображена сводка по архитекрутре модели 1 с помощью model.summary(). Тоже самое для модели 2 – на рисунке 4. Затем проиходит обучение моделей model.fit.

**Модель 1: TensorFlow Convolutional Neural Network**

Found 14 images belonging to 2 classes.

Found 8 images belonging to 2 classes.

Model: "sequential"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

conv2d (Conv2D) (None, 241, 318, 32) 320

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

max\_pooling2d (MaxPooling2D) (None, 120, 159, 32) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2d\_1 (Conv2D) (None, 118, 157, 32) 9248

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

max\_pooling2d\_1 (MaxPooling2 (None, 59, 78, 32) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2d\_2 (Conv2D) (None, 57, 76, 64) 18496

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

max\_pooling2d\_2 (MaxPooling2 (None, 28, 38, 64) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

flatten (Flatten) (None, 68096) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense (Dense) (None, 64) 4358208

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_1 (Dense) (None, 2) 130

=================================================================

Total params: 4,386,402

Trainable params: 4,386,402

Non-trainable params: 0

Рисунок 3 – Архитекрутра модели 1

Train for 14 steps, validate for 8 steps

Epoch 1/10

14/14 [==============================] - 2s 132ms/step - loss: 0.8549 - accuracy: 0.7143 - val\_loss: 0.5230 - val\_accuracy: 0.5000

Epoch 2/10

14/14 [==============================] - 2s 108ms/step - loss: 0.4807 - accuracy: 0.7857 - val\_loss: 0.0794 - val\_accuracy: 1.0000

Epoch 3/10

14/14 [==============================] - 1s 106ms/step - loss: 0.0561 - accuracy: 1.0000 - val\_loss: 0.0106 - val\_accuracy: 1.0000

Epoch 4/10

14/14 [==============================] - 2s 108ms/step - loss: 0.0060 - accuracy: 1.0000 - val\_loss: 1.1166e-04 - val\_accuracy: 1.0000

Epoch 5/10

14/14 [==============================] - 1s 106ms/step - loss: 1.6682e-04 - accuracy: 1.0000 - val\_loss: 2.2977e-05 - val\_accuracy: 1.0000

Epoch 6/10

14/14 [==============================] - 2s 107ms/step - loss: 9.8006e-06 - accuracy: 1.0000 - val\_loss: 7.0482e-06 - val\_accuracy: 1.0000

Epoch 7/10

14/14 [==============================] - 2s 110ms/step - loss: 7.6208e-06 - accuracy: 1.0000 - val\_loss: 5.2899e-06 - val\_accuracy: 1.0000

Epoch 8/10

14/14 [==============================] - 2s 109ms/step - loss: 6.1307e-06 - accuracy: 1.0000 - val\_loss: 4.3809e-06 - val\_accuracy: 1.0000

Epoch 9/10

14/14 [==============================] - 2s 108ms/step - loss: 4.9301e-06 - accuracy: 1.0000 - val\_loss: 3.8296e-06 - val\_accuracy: 1.0000

Epoch 10/10

14/14 [==============================] - 1s 106ms/step - loss: 4.1978e-06 - accuracy: 1.0000 - val\_loss: 3.5167e-06 - val\_accuracy: 1.0000

Рисунок 4 – Обучение модели 1

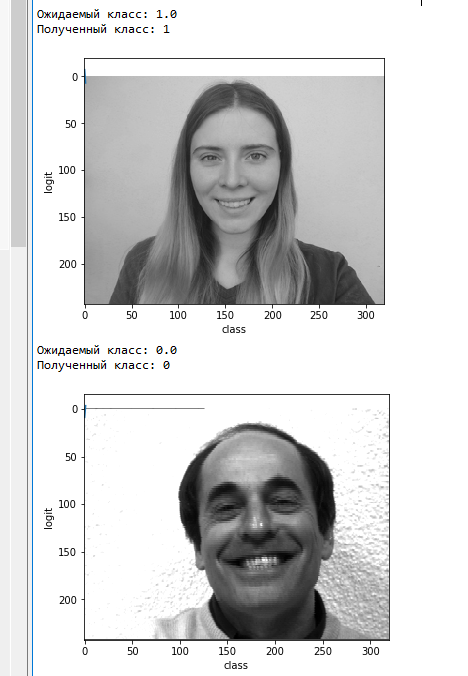


Рисунок 6 – Результат распознавания моделью 1

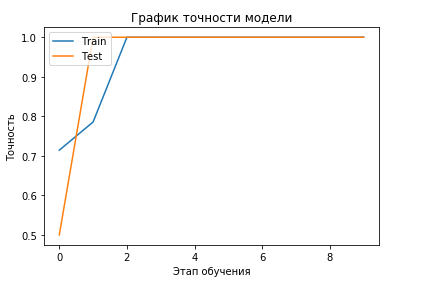


Рисунок 7 – График точности обучения модели 1

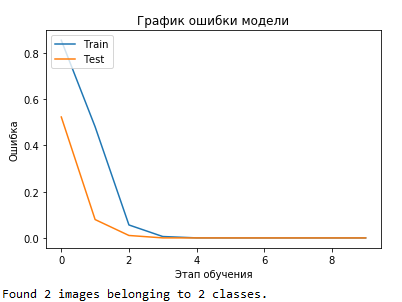


Рисунок 8 – График ошибки обучения модели 1

**Модель 2: VGG16**

Found 14 images belonging to 2 classes.

Found 8 images belonging to 2 classes.

Model: "sequential"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

conv2d (Conv2D) (None, 243, 320, 64) 640

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

max\_pooling2d (MaxPooling2D) (None, 121, 160, 64) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2d\_1 (Conv2D) (None, 121, 160, 128) 73856

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

max\_pooling2d\_1 (MaxPooling2 (None, 60, 80, 128) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2d\_2 (Conv2D) (None, 60, 80, 256) 295168

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2d\_3 (Conv2D) (None, 60, 80, 256) 590080

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

max\_pooling2d\_2 (MaxPooling2 (None, 30, 40, 256) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2d\_4 (Conv2D) (None, 30, 40, 512) 1180160

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2d\_5 (Conv2D) (None, 30, 40, 512) 2359808

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

max\_pooling2d\_3 (MaxPooling2 (None, 15, 20, 512) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2d\_6 (Conv2D) (None, 15, 20, 512) 2359808

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2d\_7 (Conv2D) (None, 15, 20, 512) 2359808

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

max\_pooling2d\_4 (MaxPooling2 (None, 7, 10, 512) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

flatten (Flatten) (None, 35840) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense (Dense) (None, 4096) 146804736

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_1 (Dense) (None, 4096) 16781312

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_2 (Dense) (None, 2) 8194

=================================================================

Total params: 172,813,570

Trainable params: 172,813,570

Non-trainable params: 0

Рисунок 9 – Архитектура модели 2

Train for 14 steps, validate for 8 steps

Epoch 1/10

14/14 [==============================] - 28s 2s/step - loss: 80.5357 - accuracy: 0.5000 - val\_loss: 0.6956 - val\_accuracy: 0.5000

Epoch 2/10

14/14 [==============================] - 28s 2s/step - loss: 0.7047 - accuracy: 0.3571 - val\_loss: 0.6931 - val\_accuracy: 0.5000

Epoch 3/10

14/14 [==============================] - 28s 2s/step - loss: 0.7138 - accuracy: 0.4286 - val\_loss: 0.6948 - val\_accuracy: 0.5000

Epoch 4/10

14/14 [==============================] - 28s 2s/step - loss: 0.6935 - accuracy: 0.5000 - val\_loss: 0.6932 - val\_accuracy: 0.5000

Epoch 5/10

14/14 [==============================] - 28s 2s/step - loss: 0.6961 - accuracy: 0.3571 - val\_loss: 0.6932 - val\_accuracy: 0.5000

Epoch 6/10

14/14 [==============================] - 29s 2s/step - loss: 0.6966 - accuracy: 0.5000 - val\_loss: 0.6932 - val\_accuracy: 0.5000

Epoch 7/10

14/14 [==============================] - 28s 2s/step - loss: 0.6931 - accuracy: 0.5000 - val\_loss: 0.6931 - val\_accuracy: 0.5000

Epoch 8/10

14/14 [==============================] - 27s 2s/step - loss: 0.6938 - accuracy: 0.4286 - val\_loss: 0.6931 - val\_accuracy: 0.5000

Epoch 9/10

14/14 [==============================] - 27s 2s/step - loss: 0.6939 - accuracy: 0.3571 - val\_loss: 0.6931 - val\_accuracy: 0.5000

Epoch 10/10

14/14 [==============================] - 27s 2s/step - loss: 0.6943 - accuracy: 0.3571 - val\_loss: 0.6931 - val\_accuracy: 0.5000

Рисунок 10 – Обучение модели 2

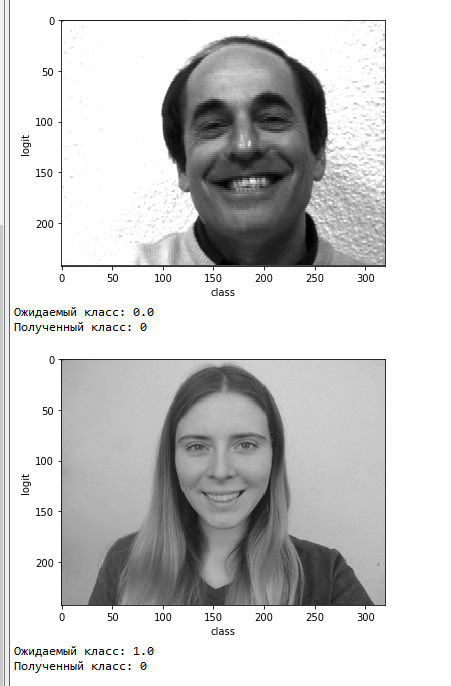


Рисунок 11 – Результат обучения модели 2

На рисунке 11 наблюдается ошибка распознавания.

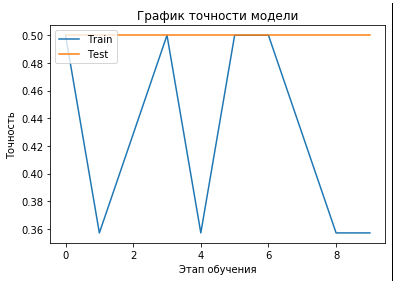


Рисунок 12 – График точности обучения модели 2

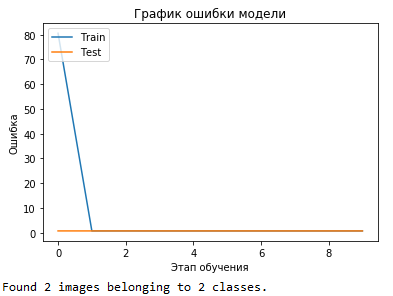
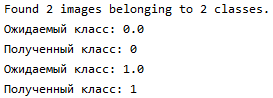


Рисунок 13 – График ошибки обучения модели 2

**Результаты и выводы**

Обе модели успешно запускаются и функционируют. Однако только первая модель обучилась и может использоваться для распознавания. Вторая модель показала низкую точность и высокую ошибку при обучении, что позволяет сделать вывод об её непригодности для поставленной задачи. Данная нейронная сеть не предназначена для работы со столь малым набором данных (22 фотографии).

Далее проихсодит тестирование первой модели нейросети на распознавание по одной фотографии каждого человека в очках с помощью model.predict, и определяя класс как argmax(predictions[i]).



Результатом выполнения лабораторной работы является программа создающая, обучающая и тестирующая свёрточную нейронную сеть. Для тестирования сети была поставлена задача распознавания лица человека, при обучении той сети на выборке лиц двух человек с разными эмоциями.

Работа была проделана над двумя моделями сетей, из которых успешно справилась с задачей модель 1.

**Список источников**

[1] TensorFlow tutorials Convoluational Neural Network [Электронный ресурс]. – Режим доступа: <https://www.tensorflow.org/tutorials/images/cnn>

[2] VGG16 — свёрточная сеть для выделения признаков изображений [Электронный ресурс]. – Режим доступа: <https://neurohive.io/ru/vidy-nejrosetej/vgg16-model/>

[3] Simonyan, K.; Zisserman, A. Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv:1409.1556 [cs], 2014.

[4] GitHub [Электронный ресурс]. – Режим доступа: <https://gist.github.com/baraldilorenzo/07d7802847aaad0a35d3>

**Листинг**

# -\*- coding: utf-8 -\*-

"""

Created on Mon Jun 15 12:50:27 2020

@author: karina

"""

import tensorflow as tf

from tensorflow.keras import datasets, layers, models

import matplotlib.pyplot as plt

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from skimage import io

import numpy as np

IMG\_HEIGHT = 243 # итоговая высота изображения

IMG\_WIDTH = 320 # итоговая ширина изображения

batch\_size = 1 # размер порции данных, считываемых в память

train\_image\_generator = ImageDataGenerator(rescale=1./255, validation\_split = 0.4) # создание генератора изображений с настроенной нормализацией и разделением на тестовую и тренировочную выборку (60% - тренировочная, 40% - тестовая)

train\_dir = 'Train' # корневая папка с изображениями для тренировки

# Создание генератора изображений, считывающего изображения из тренировочной выборки в папке train\_dir

train\_data\_gen = train\_image\_generator.flow\_from\_directory(batch\_size=batch\_size,

directory=train\_dir,

shuffle=True, # Перемешивание фотографий

color\_mode="grayscale", # Считывание в полутоновом пространстве

target\_size=(IMG\_HEIGHT, IMG\_WIDTH), # Конечный размер изображения

class\_mode="binary",

subset='training')

# Создание генератора изображений, считывающего изображения из тестовой выборки в папке train\_dir

validation\_generator = train\_image\_generator.flow\_from\_directory(batch\_size=batch\_size, color\_mode="grayscale", directory=train\_dir, target\_size=(IMG\_HEIGHT, IMG\_WIDTH), class\_mode='binary', subset='validation')

##########################

#СОЗДАНИЕ МОДЕЛИ 1

model = models.Sequential() # Последовательная сеть

model.add(layers.Conv2D(32, (3, 3), activation='relu', # Свёрточный слой, функция активации - ReLu

input\_shape=(IMG\_HEIGHT, IMG\_WIDTH, 1)))

model.add(layers.MaxPooling2D((2, 2))) # Пулинговый слой

model.add(layers.Conv2D(32, (3, 3), activation='relu')),

model.add(layers.MaxPooling2D((2,2))),

model.add(layers.Conv2D(64, (3, 3), activation='relu')),

model.add(layers.MaxPooling2D()),

model.add(layers.Flatten()) # Выравнивание в одномерный массив

model.add(layers.Dense(64, activation='relu')) # Полносвязанный слой

model.add(layers.Dense(2))

model.summary() # Вывод архитектуры

model.compile(optimizer='adam', # Компилирование сети

loss=tf.keras.losses.SparseCategoricalCrossentropy(from\_logits=True),

metrics=['accuracy'])

"""

##########################

#СОЗДАНИЕ МОДЕЛИ 2

model = models.Sequential()

model.add(layers.Conv2D(64, (3, 3), strides=(1,1),

padding='same',input\_shape=(IMG\_HEIGHT, IMG\_WIDTH, 1), activation='relu'))

model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Conv2D(128, (3, 3), strides = (1,1), padding='same',

activation='relu'))

model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Conv2D(256, (3, 3), strides = (1,1), padding='same',

activation='relu'))

model.add(layers.Conv2D(256, (3, 3), strides = (1,1), padding='same',

activation='relu'))

model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Conv2D(512, (3, 3), strides = (1,1), padding='same',

activation='relu'))

model.add(layers.Conv2D(512, (3, 3), strides = (1,1), padding='same',

activation='relu'))

model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Conv2D(512, (3, 3), strides = (1,1), padding='same',

activation='relu'))

model.add(layers.Conv2D(512, (3, 3), strides = (1,1), padding='same',

activation='relu'))

model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Flatten())

model.add(layers.Dense(4096, activation='relu'))

model.add(layers.Dense(4096, activation='relu'))

model.add(layers.Dense(2, activation='softmax'))

model.summary()

model.compile(optimizer='adam',

loss=tf.keras.losses.SparseCategoricalCrossentropy(from\_logits=False),

metrics=['accuracy'])

"""

##########################

#ОБУЧЕНИЕ МОДЕЛИ

history = model.fit(train\_data\_gen, epochs=10, validation\_data=validation\_generator) # Обучение сети

hh = history.history

# Построение графика точности

plt.plot(hh['accuracy'])

plt.plot(hh['val\_accuracy'])

plt.title('График точности модели')

plt.ylabel('Точность')

plt.xlabel('Этап обучения')

plt.legend(['Train', 'Test'], loc='upper left')

plt.show()

# Построение графика ошибки

plt.plot(history.history['loss'])

plt.plot(history.history['val\_loss'])

plt.title('График ошибки модели')

plt.ylabel('Ошибка')

plt.xlabel('Этап обучения')

plt.legend(['Train', 'Test'], loc='upper left')

plt.show()

##########################

#ТЕСТИРОВАНИЕ МОДЕЛИ

test\_image\_generator = ImageDataGenerator(rescale=1./255, validation\_split=0.3)

test\_dir = 'Test'

test\_data\_gen = test\_image\_generator.flow\_from\_directory(

batch\_size=batch\_size,

directory=test\_dir,

color\_mode="grayscale",

shuffle=True,

target\_size=(IMG\_HEIGHT, IMG\_WIDTH),

class\_mode='binary', subset='training'

)

for batch, labels in test\_data\_gen:

predictions = model.predict(batch) # Предсказание о классе

for i, image in enumerate(batch):

print(f'Ожидаемый класс: {labels[i]}')

print(f'Полученный класс: {np.argmax(predictions[i])}') # Метка с максимальным значением

plt.plot(predictions[i])

plt.ylabel('logit')

plt.xlabel('class')

plt.show()

io.imshow(image.reshape(IMG\_HEIGHT, IMG\_WIDTH))

plt.show()