

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ  
Федеральное государственное бюджетное образовательное учреждение  
высшего образования  
«Московский государственный технический университет  
имени Н.Э. Баумана  
(национальный исследовательский университет)»

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА**  
**по курсу**  
**«Data Science»**

Слушатель

Мягков Олег Вячеславович

Москва, 2022

## Содержание

1	Аналитическая часть.....	3
1.1	Постановка задачи.....	3
1.2	Описание используемых методов.....	5
1.3	Разведочный анализ данных.....	9
2	Практическая часть .....	15
2.1	Предобработка данных .....	15
2.2	Создание датасета для обучения классификатора с использованием TensorFlow из имеющегося набора изображений .....	17
2.3	Создание различных вариантов нейронных сетей для классификации изображений, оценка качества классификации .....	18
2.4	Разработка приложений .....	30
2.4.1	Утилита ImgPrep_V_1_0_0.py .....	30
2.4.2	Утилита Classifier_V_1_0_0.py.....	32
2.4.3	Утилита check_simbol_pic_V_1_0_0.py.....	32
2.5	Создание удаленного репозитория и загрузка результатов работы на него.....	34

# 1 Аналитическая часть

## 1.1 Постановка задачи

Современное производство микросхем - сложный процесс, в котором заняты тысячи инженеров и рабочих. При создании цифровых микросхем активно используются библиотеки цифровых ячеек. Что из себя представляют эти библиотеки: это своеобразные «блоки», из которых может быть создана микросхема любой сложности. Каждая ячейка представляет собой законченный логический элемент, начиная от инвертора и заканчивая триггером. Ячейка имеет несколько представлений: layout, schematic, symbol.

Layout содержит топологию ячейки и используется на этапе фотолитографии.

Schematic - принципиальная электрическая схема ячейки.

Symbol – «черный ящик» ссылающийся на schematic, используется в электрических схемах более высокого уровня. В данной работе будет проводиться анализ именно представления symbol (в дальнейшем будем называть ее символ).

На рисунке 1 можно увидеть изображение символа ячейки SDFRRS

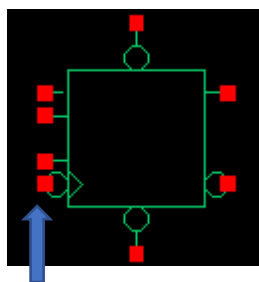


Рисунок 1 - Символ ячейки SDFRRS

На рисунке 2 изображён символ другой ячейки – SDFFRS

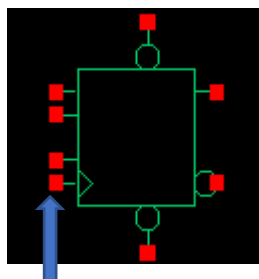


Рисунок 2 - Символ ячейки SDFFRS

Как видим изображения очень похожи, важная разница — это отсутствие окружности на одном из входов ячейки. Синяя стрелка указывает на что следует обратить внимание.

При разработке библиотеки инженер для создания новой ячейки может использовать уже спроектированную ячейку, при этом, если он в представлении символ изменит название вывода, но не исправит изображение, например, не удалит или не дорисует окружность, то все будет работать корректно, с точки зрения использования ячейки в системах автоматизированного проектирования, однако инженер, использующий библиотеку для разработки микросхемы, может запутаться, видя неверное изображение символа цифровой ячейки.

В данный момент проверка соответствия символа и логической функции ячейки осуществляется вручную. Инженер внимательно просматривает все символы библиотеки, стараясь обнаружить ошибки. Это занимает время и не гарантирует, что ошибки не будут пропущены.

Проверять символы, сравнивая их попиксельно, используя некий «идеальный» набор символов, невозможно, поскольку от библиотеки к библиотеке, от технологии к технологии символы одной и той же ячейки могут иметь незначительные отличия в пропорциях, длинах выводов и так далее.

При этом человек однозначно определит, что эти символы соответствуют всем необходимым требованиям.

Задача: Необходимо, используя методы машинного обучения, создать классификатор изображений, который сможет определять принадлежность изображения к определенному типу ячейки, сравнивать полученное

предсказание с названием ячейки, из которой было получено изображение. При несовпадении предсказанного и истинного типа ячейки записывать истинное имя ячейки в текстовый файл для дальнейшего анализа человеком.

Создать приложение для обучения нейронной сети, чтобы инженер мог легко создать новую модель при появлении новых символов или при изменении старых.

Создать приложение для подготовки данных, которые могут быть использованы для обучения новой модели, при появлении новых символов или при изменении старых.

Приложения должны работать в средах Линукс, интерфейс приложений командная строка.

Датасет, предоставленный для работы, это каталог, содержащий в себе 831 папку. Каждая папка имеет уникальное имя, в каждой папке находится файл с именем «thumbnail\_128x128.png». Файл является цветным изображением размером 128 на 128 пикселей. Файл есть в каждой папке. Папки с разными именами могут содержать одинаковые изображения.

## **1.2 Описание используемых методов**

В качестве метода распознавания изображения используется сверточная нейросеть – CNN. Данная архитектура нейросети была предложена французским ученым Яном Лекуном в 1988 году. Сверточная нейросеть ориентирована на эффективное распознавание образов. Ей удаётся значительно точнее распознавать объекты на изображениях, так как, в отличие от многослойного персептрона, учитывается двухмерная топология изображения. В настоящий момент архитектуры, основанные на свёрточных сетях, чаще всего занимают первые места в соревнованиях по распознаванию образов.

Хорошая точность распознавания изображения может быть достигнута в том числе с помощью обычных нейросетей прямого распространения. За примером не нужно ходить далеко, наверное, каждый студент, изучающий

машинное обучение сталкивался с набором данных MNIST, где используются изображения 28x28 пикселей, и классифицировал их с помощью многослойного персептрона. Но если мы хотим обрабатывать изображение с большим разрешением, то число параметров для нейронной сети увеличивается многократно.

Например, если требуется обработать черно белое изображение размером 128x128 пикселей, то для каждого нейрона первого слоя полносвязанной сети потребуется  $128 \times 128 = 16384$  параметров, из-за чего время, затрачиваемое на обучение нейросети многократно возрастет.

Простейшая сверточная нейросеть содержит 4 слоя расположенных последовательно.

Слой свёртки — это основной блок свёрточной нейронной сети. Слой свёртки включает в себя для каждого канала свой фильтр, ядро свёртки которого обрабатывает предыдущий слой по фрагментам (суммируя результаты поэлементного произведения для каждого фрагмента). Весовые коэффициенты ядра свёртки (небольшой матрицы) неизвестны и устанавливаются в процессе. Так например, если исходное изображение имеет размерность 100×100 пикселей по трём каналам (это значит 30 000 входных нейронов), а свёрточный слой использует фильтры с ядром 3×3 пикселя с выходом на 6 каналов, тогда в процессе обучения определяется только 9 весов ядра, однако по всем сочетаниям каналов, то есть  $9 \times 3 \times 6 = 162$ , в таком случае данный слой требует нахождения только 162 параметров. Это существенно меньше количества искомых параметров полносвязной нейронной сети.

Скалярный результат каждой свёртки попадает на функцию активации, которая представляет собой некую нелинейную функцию. Слой активации обычно логически объединяют со слоем свёртки (считают, что функция активации встроена в слой свёртки). Иллюстрацию сверточного слоя можно увидеть на рисунке 3.

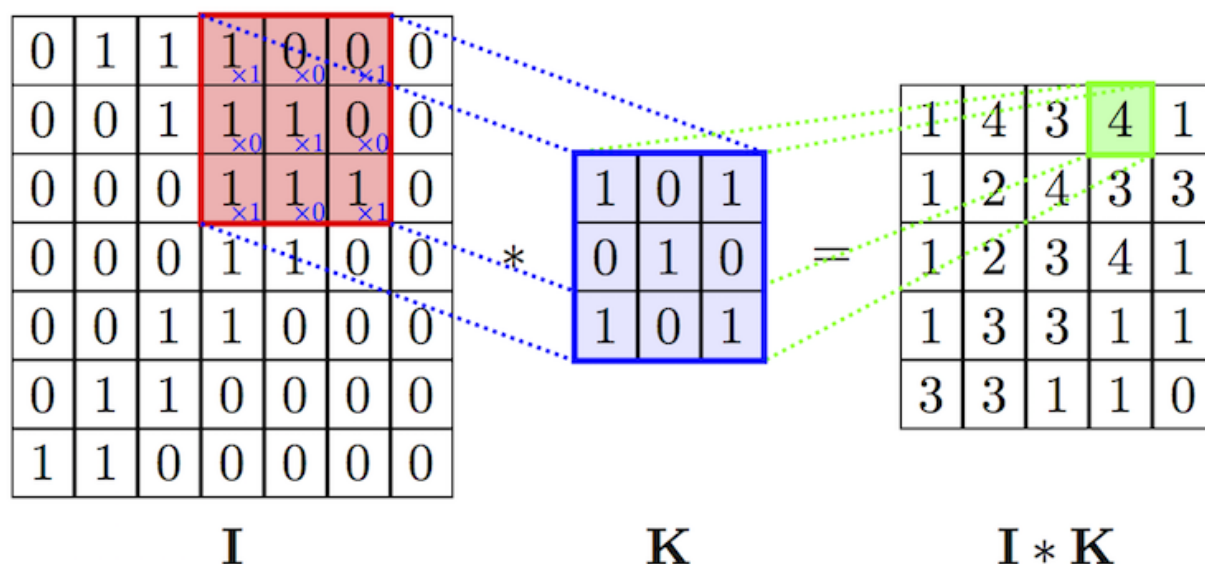


Рисунок 3 – Сверточный слой нейросети

Следующий слой в CNN это пулинг или слой субдискретизации. Этот слой представляет собой нелинейное уплотнение карты признаков, при этом группа пикселей (обычно размера  $2 \times 2$ ) уплотняется до одного пикселя, проходя нелинейное преобразование. Наиболее употребительна при этом функция максимума. Преобразования затрагивают непересекающиеся прямоугольники или квадраты, каждый из которых ужимается в один пиксель, при этом выбирается пиксель, имеющий максимальное значение. Операция пулинга позволяет существенно уменьшить пространственный объём изображения. Пулинг интерпретируется так: если на предыдущей операции свёртки уже были выявлены некоторые признаки, то для дальнейшей обработки настолько подробное изображение уже не нужно, и оно уплотняется до менее подробного. К тому же фильтрация уже ненужных деталей помогает не переобучаться. Слой пулинга, как правило, вставляется после слоя свёртки перед слоем следующей свёртки. На рисунке 4 можно увидеть пример пулинга с функцией максимума, размером фильтра  $2 \times 2$  и шагом 2.

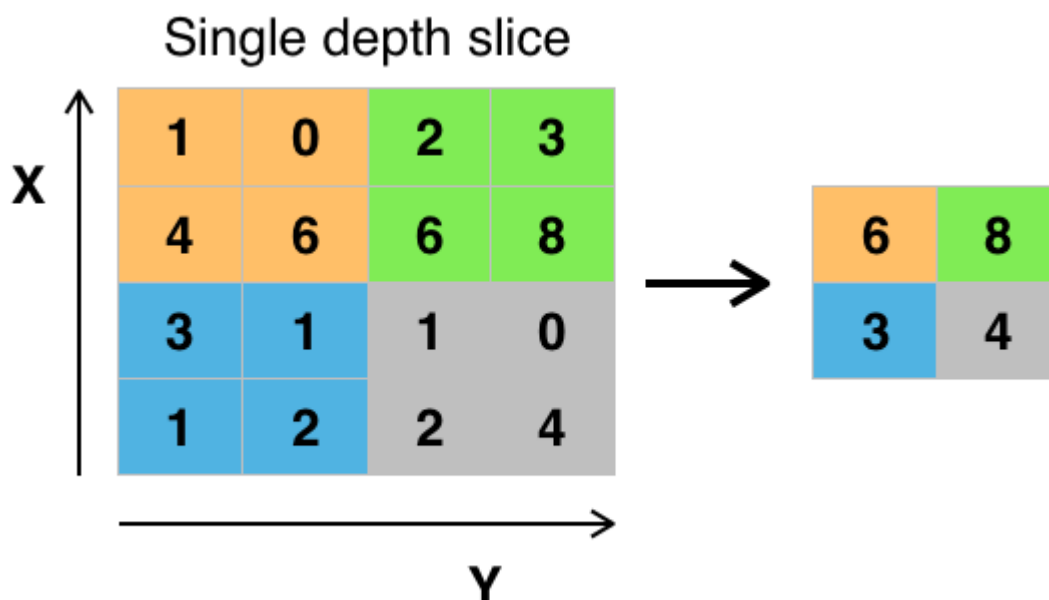


Рисунок 4 – Пример пулинга

Далее находится обычная полносвязная нейронная 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).

На рисунке 5 можно увидеть схематичное изображение сверточной нейронной сети.

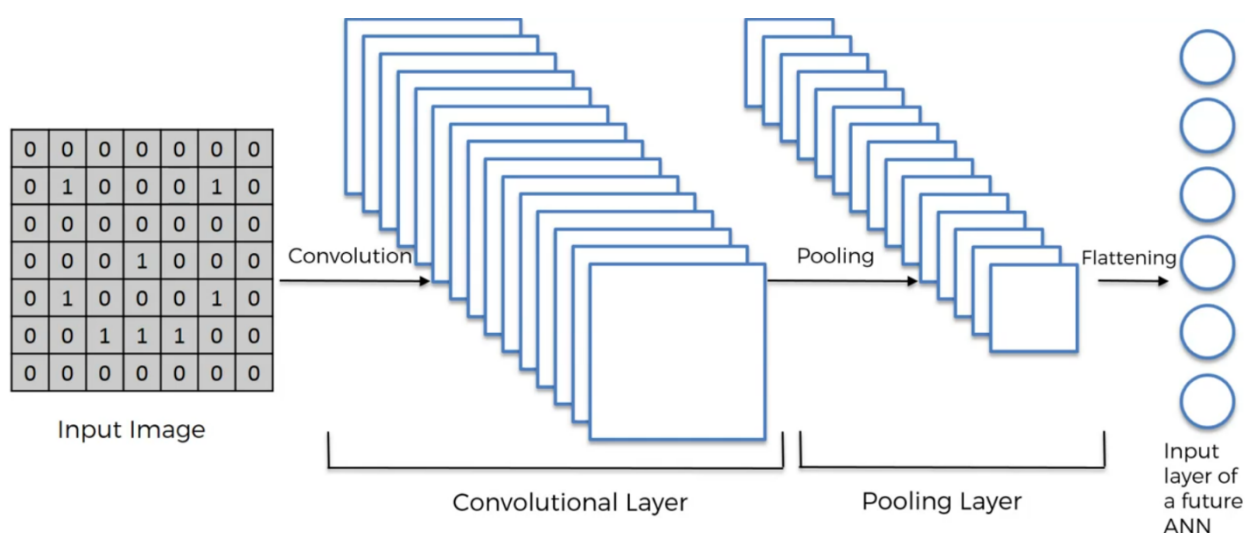


Рисунок 5 - Схематичное изображение сверточной нейронной сети



### 1.3 Разведочный анализ данных

Изучение предоставленных данных показывает, что все ячейки имеют уникальные названия. Например, «DFFRSQHDX0». Название состоит из:

- 1) буквенного или буквенно-цифрового индекса – имени «DFFRSQ», в котором закодирована логическая функция ячейки;
- 2) суффикса «HD», который показывает принадлежность ячейки к определенной библиотеке (каждая библиотека ячеек имеет свой уникальный суффикс);
- 3) последняя часть имени «X0» указывает силу выходного драйвера ячейки. Сила выходного драйвера присутствует не во всех именах ячеек.

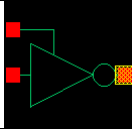
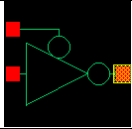
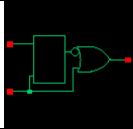
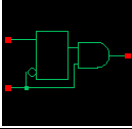
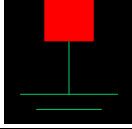
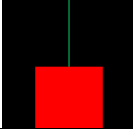
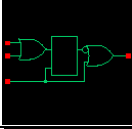
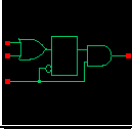
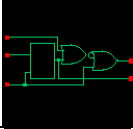
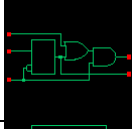
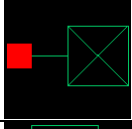
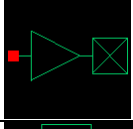
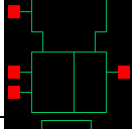
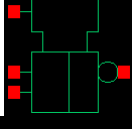
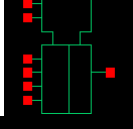
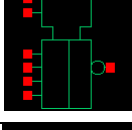
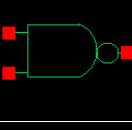
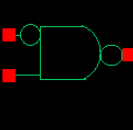
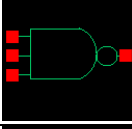
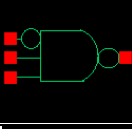
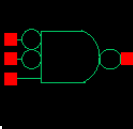
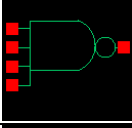
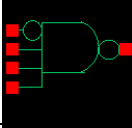
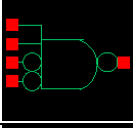
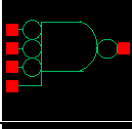
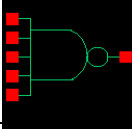
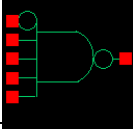
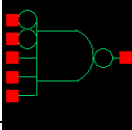
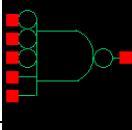
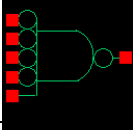
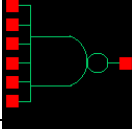
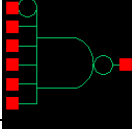
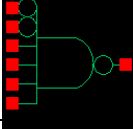
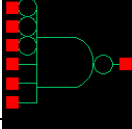
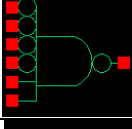
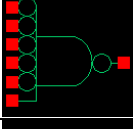
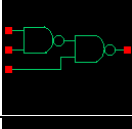
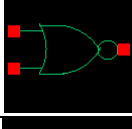
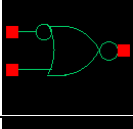
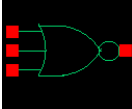
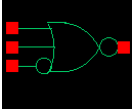
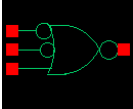
Более детальное изучение данных показало, что в библиотеке присутствует 208 уникальных изображений символа, некоторые ячейки с разными именами имеют одинаковый символ.

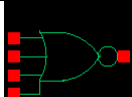
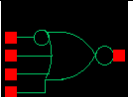
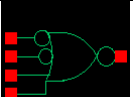
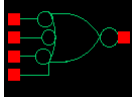
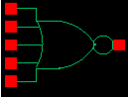
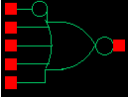
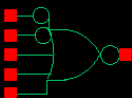
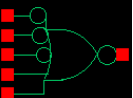
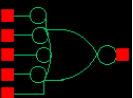
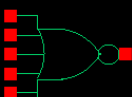
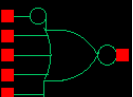

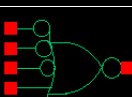
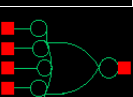
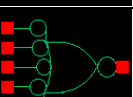








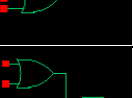










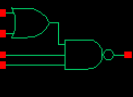
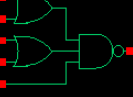

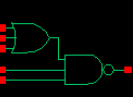

Все изображения собраны по классам в таблице, представленной ниже.

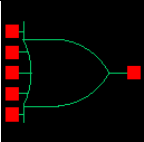
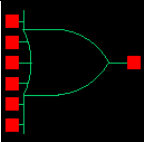
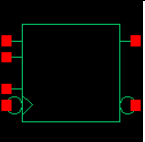
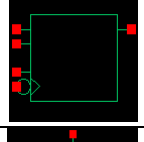
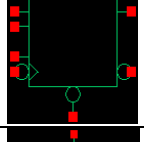
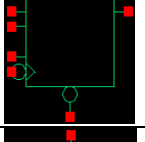
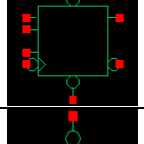
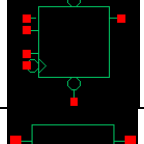
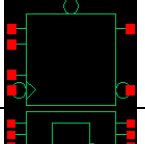
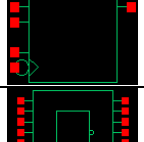
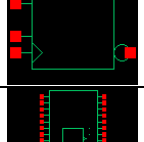
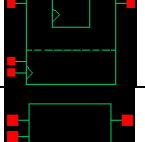
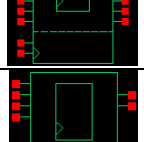
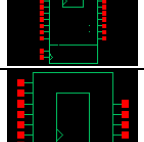
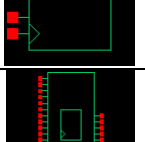
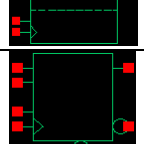
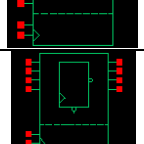
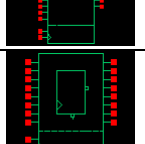
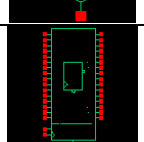
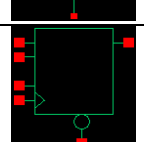
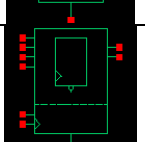
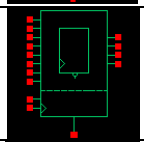
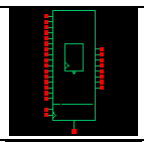
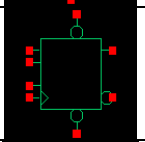
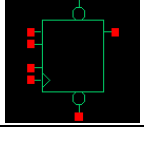
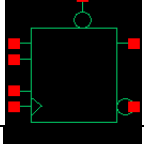
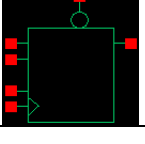

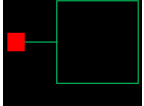

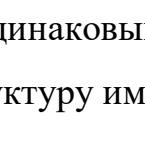
AN21		AN222		AND6	
AN22		AN311		ANTENNACELLN2 ANTENNACELLP2	
AN31		AN321		ANTENNACELLNP2	
AN32		AND2		AO21	
AN33		AND3		AO22	
AN211		AND4		AO31	

AN221		AND5		AO32	
AO33		AO211		AO221	
AO222		AO311		AO321	
BTH		BTL		BU STE CLKVBU DLY	
CAG		DFF		DFFQ	
DFFR		DFFRQ		DFFRS	
DFFRSQ		DFFS		DFFSQ	
DFR		DFR2		DFR4	
DFR8		DFRQ		DFRQ2	
DFRQ4		DFRQ8		DFRR	
DFRR2		DFRR4		DFRR8	
DFRRQ		DFRRQ2		DFRRQ4	
DFRRQ8		DFRRS		DFRRSQ	

DFRS		DFRSQ		DLH	
DLHQ		DLHR		DLHRQ	
DLHRS		DLHRSQ		DLHRT	
DLHS		DLHSQ		DLHST	
DLHT		DLL		DLLQ	
DLLR		DLLRQ		DLLRS	
DLLRSQ		DLLRT		DLLRT2	
DLLRT4		DLLRT8		DLLS	
DLLSQ		DLLST		DLLT	
DLLT2		DLLT4		DLLT8	
EN2		EN3		EO2	
EO3		FA		FCNE FCNED DECAP	
FEED		HA		IN	

ITH		ITL		LGCN	
LGCP		LOGIC0		LOGIC1	
LSGCN		LSGCP		LSOGCN	
LSOGCP		MPROBE		MPROBEBU	
MU2		MU2I		MU4	
MU4I		NA2		NA2I1	
NA3		NA3I1		NA3I2	
NA4		NA4I1		NA4I2	
NA4I3		NA5		NA5I1	
NA5I2		NA5I3		NA5I4	
NA6		NA6I1		NA6I2	
NA6I3		NA6I4		NA6I5	
NA22		NO2		NO2I1	
NO3		NO3I1		NO3I2	

NO4		NO4I1		NO4I2	
NO4I3		NO5		NO5I1	
NO5I2		NO5I3		NO5I4	
NO6		NO6I1		NO6I2	
NO6I3		NO6I4		NO6I5	
NO22		OA21		OA22	
OA31		OA32		OA33	
OA211		OA221		OA222	
OA311		OA321		ON21	
ON22		ON31		ON32	
ON33		ON211		ON221	
ON222		ON311		ON321	
OR2		OR3		OR4	

OR5		OR6		Sdff	
SdffQ		SdffR		SdffRQ	
SdffRS		SdffRSQ		SdffS	
SdffSQ		Sdfr		Sdfr2	
Sdfr4		Sdfr8		SdfrQ	
SdfrQ2		SdfrQ4		SdfrQ8	
SdfrR		SdfrR2		SdfrR4	
SdfrR8		SdfrRQ		SdfrRQ2	
SdfrRQ4		SdfrRQ8		SdfrRS	
SdfrRSQ		SdfrS		SdfrSQ	
		SIGNALHOLD			

- 1) Все изображения имеют одинаковый размер;
- 2) классы имеют четкую структуру имен;
- 3) изображения одинаково сориентированы;
- 4) на изображениях отсутствуют шумы.

## **2 Практическая часть**

### **2.1 Предобработка данных**

В предыдущем пункте я просмотрел и систематизировал данные, которые предстоит обработать и в дальнейшем использовать для обучения нейросети.

Для того чтобы качественно обучить нейросеть необходимо большое количество изображений, об этом пишут все авторы статей и книг, они же, всегда указывают, что количество необходимых изображений сильно зависит от количества классов и от сложности самих изображений. Зачастую возможно получить приемлемый результат используя очень скромный набор данных, они буквально толкают меня на эксперимент.

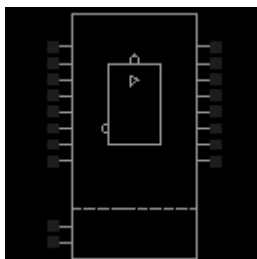
В моем случае есть всего лишь одно уникальное изображение для каждого класса ячеек, недостающие изображения для обучающего и проверочного датасетов придется получить методом аугментации. При этом нет смысла сильно искажать или обрезать изображение, потому что в дальнейшем, для анализа нейросеть будет получать, такие же или очень похожие картинки, а в мои задачи не входит усложнить условия обучения для нейросети, напротив мне необходимо получить максимальную точность классификации.

Совершенно наугад я принял решение, что в обучающем датасете мне будет достаточно 30 изображений для каждого класса, при этом одно изображение не будет иметь никаких искажений, а еще 29 будут подвергнуты аугментации. Для тестового датасета я буду использовать 3 аугментированных изображения для каждого класса, таким образом я получу соотношение 90% обучающий набор и 10% тестовый набор.

Кроме того, изображения, предоставленные для работы, являются цветными, но с точки зрения анализа изображений, цвет в данном случае является ненужными, пустыми данными, цвет никак не может помочь, дать уникальный признак, для классификации.

Перейдем к описанию процесса подготовки изображений. В начале обесцветим изображение:

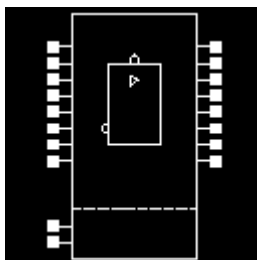
```
image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
```



Пока выглядит не очень. Тем более, если задуматься наличие градаций серого тоже ни как нам не поможет в дальнейшей классификации.

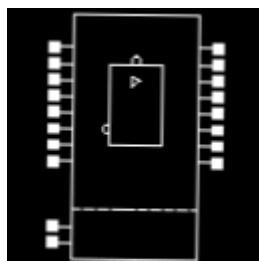
Поэтому, следующим шагом попробуем сделать изображение бинарным. Пусть все пиксели, что имеют значения яркости ниже 20 станут черными, примут значение 0, а все, что имеет значение яркости выше 20 станет белым, примет значение 255. Команда:

```
ret, image = cv2.threshold(image, 20, 255, 0)
```



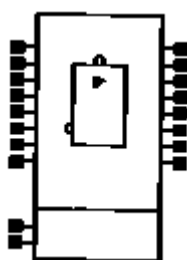
На мой взгляд изображение выглядит значительно интереснее. На этом этапе обработки изображений проведем аугментацию. Примененные изменения не должны сильно исказить первоначальное изображение, поэтому применим операцию *iaa.Affine(rotate=(-1, 1))* и *iaa.Crop(percent=(0, 0.005))* в случайном порядке, это обеспечит получение изображений незначительно отличающихся друг от друга.





С точки зрения машины этого будет достаточно, но я человек и для моего глаза более приятно темное изображение на светлом фоне, чем наоборот.

Поэтому проведем последнее изменение, проведем инверсию изображения, чтобы на него было приятнее смотреть.



Цепочка из этих преобразований производится в цикле для изображения каждого класса. Таким образом, мы получим наборы изображений, которые будут преобразованы в датасеты для обучения нейросети.

## 2.2 Создание датасета для обучения классификатора с использованием TensorFlow из имеющегося набора изображений

Мы подготовили необходимые изображения, но напрямую мы не можем передать их нейросети для обучения. К счастью, существует библиотека *tensorflow* позволяющая создать из папок с изображениями необходимый датасет. Создаем датасет для обучения:

```
train_dataset = image_dataset_from_directory('./training_set',
                                             subset='training',
                                             seed=42,
                                             validation_split=0.1,
                                             batch_size=batch_size,
                                             image_size=image_size,
                                             color_mode="grayscale")
```

Создаем датасет для валидации модели в процессе обучения:

```
validation_dataset = image_dataset_from_directory('./training_set',  
                                                  subset='validation',  
                                                  seed=42,  
                                                  validation_split=0.1,  
                                                  batch_size=batch_size,  
                                                  image_size=image_size,  
                                                  color_mode="grayscale")
```

Создаем датасет для проверки модели:

```
test_dataset = image_dataset_from_directory('./test_set',  
                                             seed=42,  
                                             batch_size=batch_size,  
                                             image_size=image_size,  
                                             color_mode="grayscale")
```

Все подготовительные работы сделаны, следующий шаг - создание и обучение нейронной сети для классификации изображений.

### **2.3 Создание различных вариантов нейронных сетей для классификации изображений, оценка качества классификации**

В самом начале работы я воспользовался готовой архитектурой нейросети которую нашел в интернете - Fruits\_360\_CNN. Статью автора этой нейросети можно найти по ссылке <https://github.com/Horea94/Fruit-Images-Dataset>

Автор создал датасет содержащий 90483 изображений овощей и фруктов, разделенных на 131 класс, а заодно он разработал нейросеть для работы с датасетом Fruits\_360. Именно с этой нейросетью я вступаю в заочное соревнование.

Нейросеть создана для работы с цветными изображениями, это не удивительно, цвет играет огромное значение в определении фруктов и овощей, в моем случае, изображения черно-белые, поэтому мне пришлось сделать незначительные изменения на первом входном слое.

```

# Создаем последовательную модель
model = Sequential()
# Сверточный слой
model.add(Conv2D(16, (5, 5), padding='same',
                 input_shape=(128, 128, 1), activation='relu'))
# Слой подвыборки
model.add(MaxPooling2D(pool_size=(2, 2)))
# Сверточный слой
model.add(Conv2D(32, (5, 5), activation='relu', padding='same'))
# Слой подвыборки
model.add(MaxPooling2D(pool_size=(2, 2)))
# Сверточный слой
model.add(Conv2D(64, (5, 5), activation='relu', padding='same'))
# Слой подвыборки
model.add(MaxPooling2D(pool_size=(2, 2)))
# Сверточный слой
model.add(Conv2D(128, (5, 5), activation='relu', padding='same'))
# Слой подвыборки
model.add(MaxPooling2D(pool_size=(2, 2)))
# Полносвязная часть нейронной сети для классификации
model.add(Flatten())
model.add(Dense(1024, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.2))
# Выходной слой, 208 нейронов по количеству классов
model.add(Dense(208, activation='softmax'))

```

Модель обучалась в течение 15 эпох, точность модели на тестовом датасете составила 99.04%. Результат меня обнадежил. Графики обучения модели можно увидеть на рисунках 6 и 7.

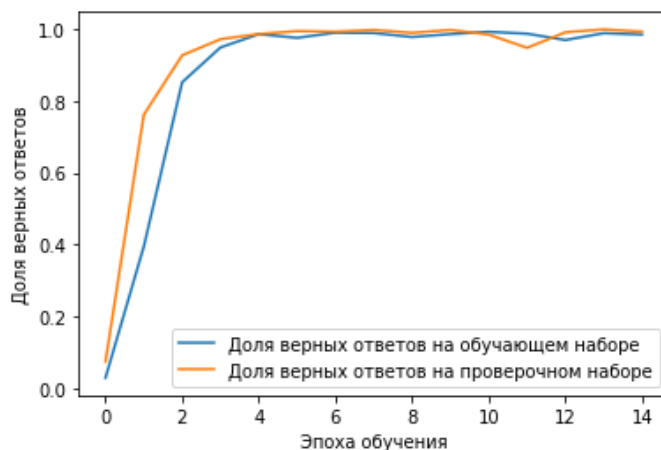


Рисунок 6 – График обучения модели

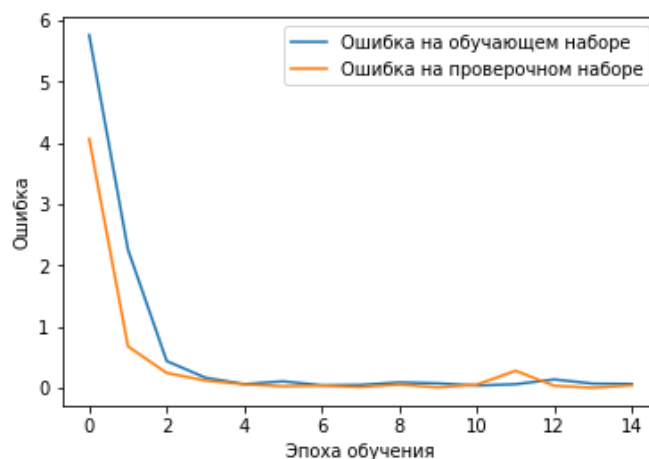


Рисунок 7 - График обучения модели

Переобучения не наблюдается, графики находятся очень близко друг к другу.

Меня не совсем устраивает полученная точность 99% - это очень хорошо, но мне бы хотелось получить точность максимально близкую к 100%, в идеале 100%.

Попробуем создать простейшую нейросеть, опираясь на полученные в процессе моего обучения знания. Итак, пришло время *model\_simple*.

```
# Создаем последовательную модель
model_simple = Sequential()
# Сверточный слой
model_simple.add(Conv2D(16, (5, 5), padding='same',
                        input_shape=(128, 128, 1), activation='relu'))
# Слой подвыборки
model_simple.add(MaxPooling2D(pool_size=(2, 2)))
# Полносвязная часть нейронной сети для классификации
model_simple.add(Flatten())
model_simple.add(Dense(256, activation='relu'))
# Выходной слой, 208 нейронов по количеству классов
model_simple.add(Dense(208, activation='softmax'))
model_simple.summary()
```

Нейросеть содержит один сверточный слой, один пулинг слой и полносвязанную часть. Точность на тестовом наборе данных составила 98.88%, что лишь немногим хуже, чем показала нейросеть *Fruits\_360\_CNN*.

На рисунке 8 и 9 можно увидеть как проходило обучение самой простой, но в то же время вполне работоспособной нейросети.

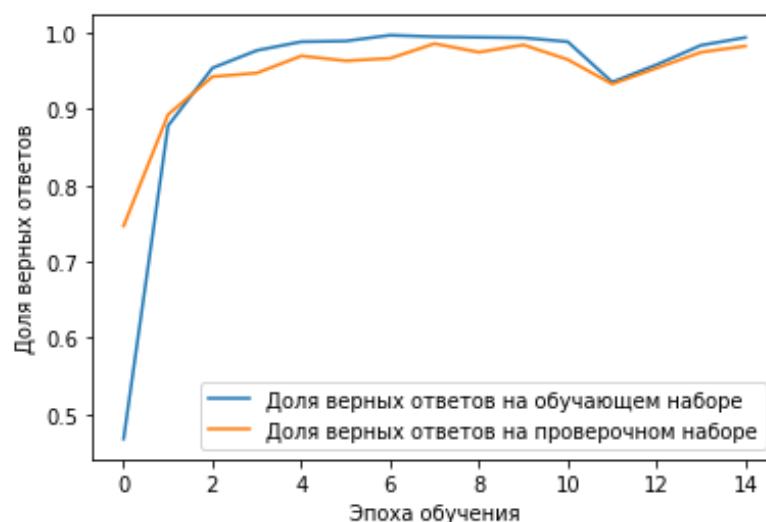


Рисунок 8 - График обучения модели

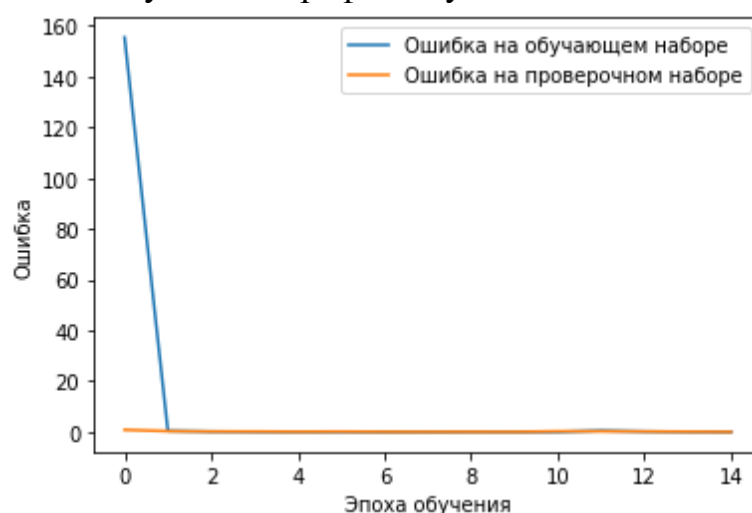


Рисунок 9 - График обучения модели

Если попробовать изменить количество фильтров на сверточном слое, это не приведет ни к каким положительным изменениям. Точность нейросети содержащей 32 фильтра на сверточном слое составляет 98.78%, что даже немного меньше, чем у нейросети с 16 фильтрами. На рисунке 10 и 11 представлены графики обучения этой нейросети.

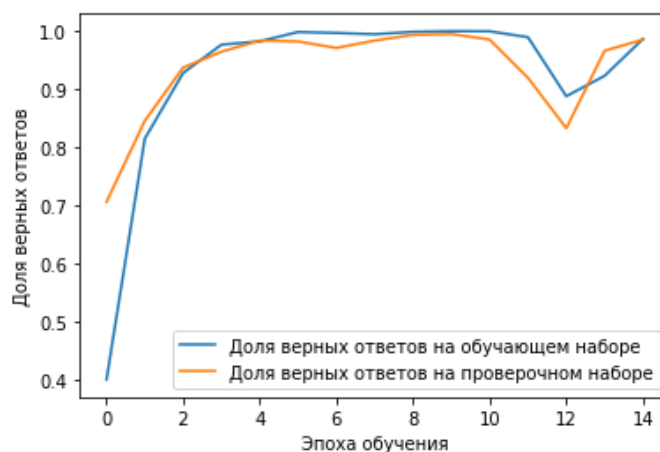


Рисунок 10 - График обучения модели

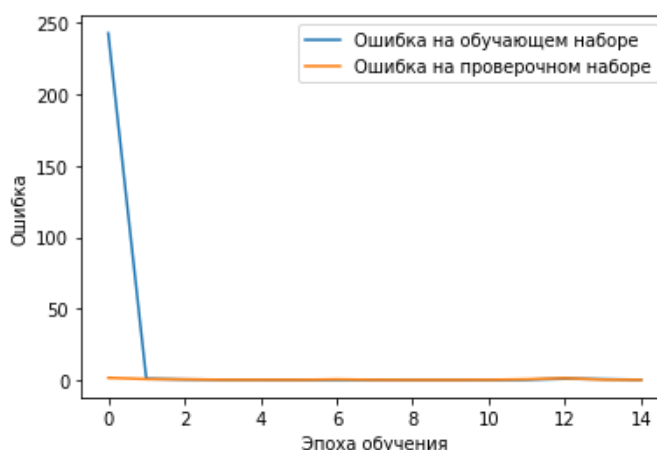


Рисунок 11 - График обучения модели

Попробуем усложнить нейросеть, добавив еще один слой свертки, на этом слое будет 32 фильтра размером 3x3 естественно появится еще один слой пулинга аналогичный первому. Назовем нейросеть `model_s16_32`

```
model_s16_32 = Sequential()
model_s16_32.add(Conv2D(16, (5, 5), padding='same',          #первый слой свертки
                        input_shape=(128, 128, 1), activation='relu'))
model_s16_32.add(MaxPooling2D(pool_size=(2, 2)))
model_s16_32.add(Conv2D(32, (3, 3), padding='same',          #второй слой свертки
                        input_shape=(128, 128, 1), activation='relu'))
model_s16_32.add(MaxPooling2D(pool_size=(2, 2)))
model_s16_32.add(Flatten())                                  #полносвязанная часть
model_s16_32.add(Dense(512, activation='relu'))
model_s16_32.add(Dense(208, activation='softmax'))
```

Результаты обучения, хороши. Точность выросла почти на один процент и сейчас составляет 99.68% это выше, чем у `Fruits_360_CNN`.

Графики обучения представлены на графиках рисунков 12 и 13.

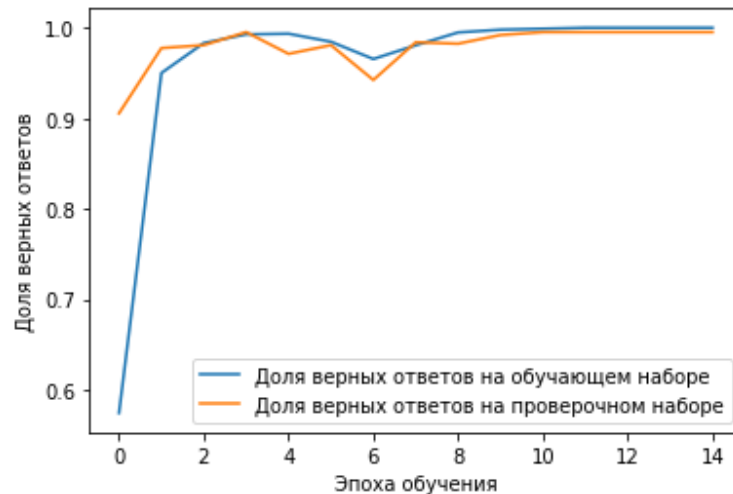


Рисунок 12 - График обучения модели

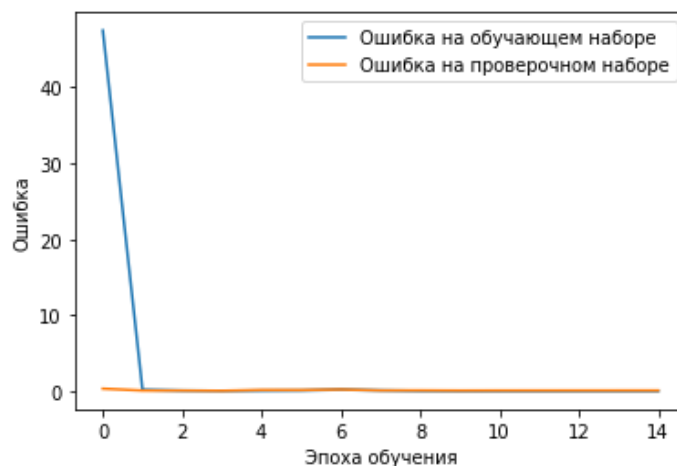


Рисунок 13 - График обучения модели

Добавим к нашей нейросети model\_s16\_32 еще один сверточный слой. Пусть в нем будет 64 фильтра размером 3x3 остальную часть сети оставим без изменений.

```
model_s16_32_64 = Sequential()
model_s16_32_64.add(Conv2D(16, (5, 5), padding='same', #первый слой свертки
    input_shape=(128, 128, 1), activation='relu'))
model_s16_32_64.add(MaxPooling2D(pool_size=(2, 2)))
model_s16_32_64.add(Conv2D(32, (3, 3), padding='same', #второй слой свертки
    input_shape=(128, 128, 1), activation='relu'))
model_s16_32_64.add(MaxPooling2D(pool_size=(2, 2)))
model_s16_32_64.add(Conv2D(64, (3, 3), padding='same', #третий слой свертки
    input_shape=(128, 128, 1), activation='relu'))
model_s16_32_64.add(MaxPooling2D(pool_size=(2, 2)))
model_s16_32_64.add(Flatten()) #полносвязанная часть
model_s16_32_64.add(Dense(512, activation='relu'))
model_s16_32_64.add(Dense(208, activation='softmax'))
```

После обучения, очевидно, целевой показатель достигнут. После обучения в течение 15 эпох был достигнут показатель точности 100% на тестовом наборе изображений. Графики обучения представлены на рисунке 14 и 15. Можно остановиться, но мне хотелось бы обучить еще 3 модели с разными установками, в качестве исследования.

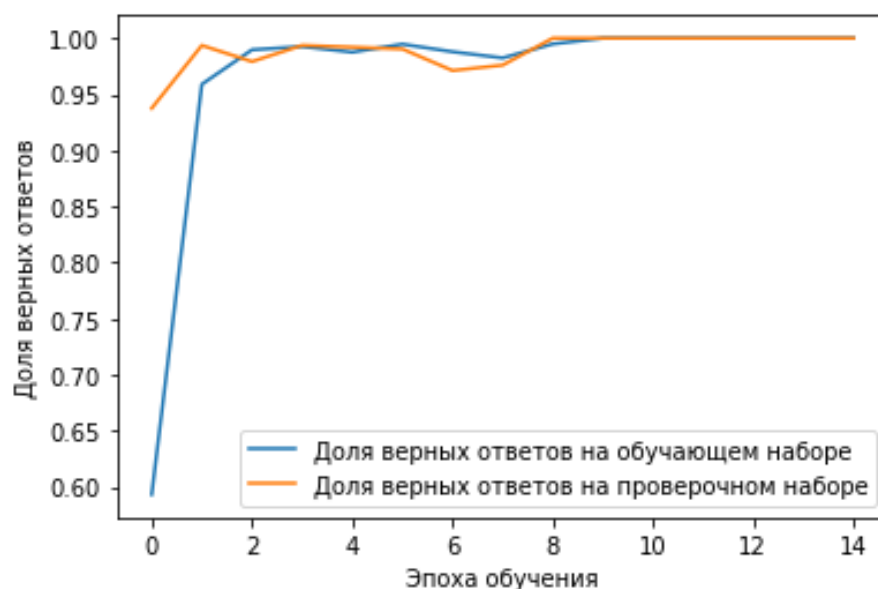


Рисунок 14 - График обучения модели

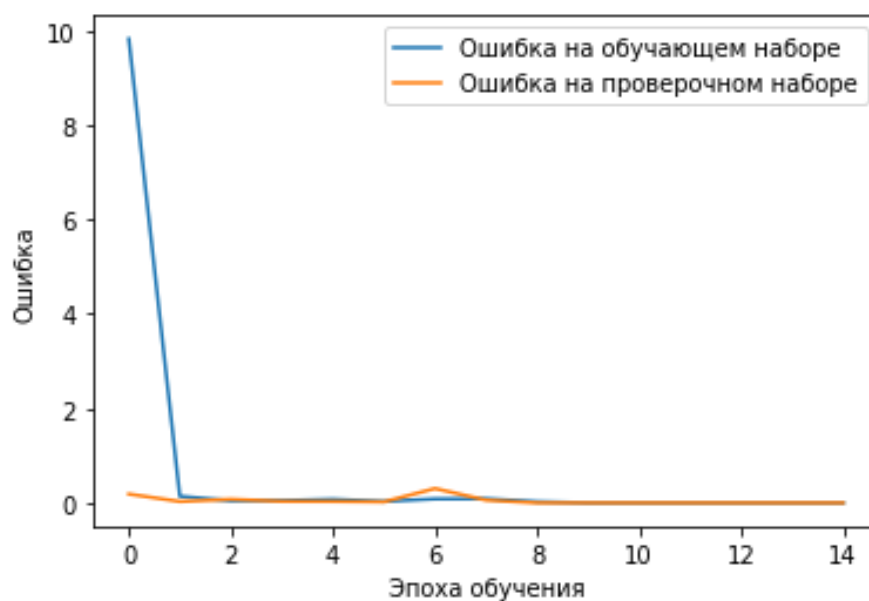


Рисунок 15 - График обучения модели



В следующей модели попробуем добавить слой дропаут. На графиках не наблюдается переобучения, вероятно этот слой будет избыточным, но необходимо попробовать. Назовем модель `model_s16_32_64_drop`

```
model_s16_32_64_drop = Sequential()
model_s16_32_64_drop.add(Conv2D(16, (5, 5), padding='same', # Сверточный слой
                               input_shape=(128, 128, 1), activation='relu'))
model_s16_32_64_drop.add(MaxPooling2D(pool_size=(2, 2)))
model_s16_32_64_drop.add(Conv2D(32, (3, 3), padding='same', # Сверточный слой
                               input_shape=(128, 128, 1), activation='relu'))
model_s16_32_64_drop.add(MaxPooling2D(pool_size=(2, 2)))
model_s16_32_64_drop.add(Conv2D(64, (3, 3), padding='same', # Сверточный слой
                               input_shape=(128, 128, 1), activation='relu'))
model_s16_32_64_drop.add(MaxPooling2D(pool_size=(2, 2)))
model_s16_32_64_drop.add(Flatten()) # Полносвязная часть
model_s16_32_64_drop.add(Dense(512, activation='relu'))
model.add(Dropout(0.1)) #Слой дропаут
# Выходной слой, 208 нейронов по количеству классов
model_s16_32_64_drop.add(Dense(208, activation='softmax'))
```

После обучения нейросети получаем точность на тестовой выборке 99.84% это ниже единицы, то есть показатель ниже, чем у предыдущей модели `model_s16_32_64`. Графики обучения представлены на рисунке 16 и 17.

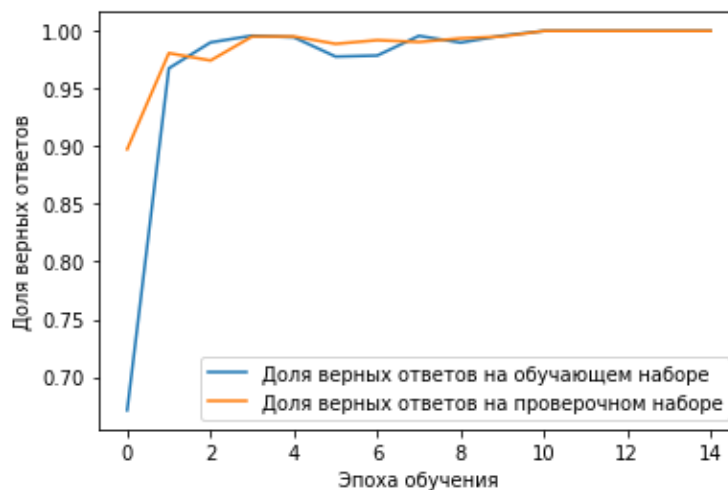


Рисунок 16 - График обучения модели

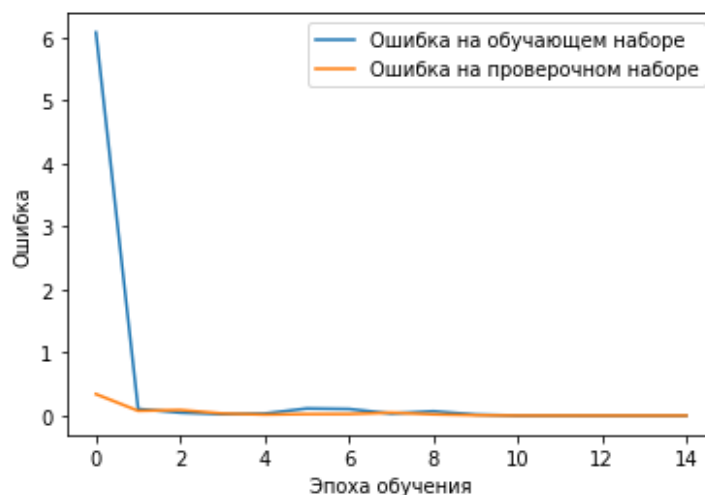


Рисунок 17 - График обучения модели

Попробуем изменить размеры фильтров на сверточных слоях, возможно это поможет увеличить точность и получить заветные 100% при наличии слоя дропаут. Установим следующие размеры фильтров на сверточных слоях. На первом слое 16 фильтров размером 7x7, на втором сверточном слое 32 фильтра размером 5x5 и на третьем слое 64 фильтра размером 3x3

```
model_f7_5_3_drop = Sequential()
model_f7_5_3_drop.add(Conv2D(16, (7, 7), padding='same', # Сверточный слой
                             input_shape=(128, 128, 1), activation='relu'))
model_f7_5_3_drop.add(MaxPooling2D(pool_size=(2, 2)))
model_f7_5_3_drop.add(Conv2D(32, (5, 5), padding='same', # Сверточный слой
                             input_shape=(128, 128, 1), activation='relu'))
model_f7_5_3_drop.add(MaxPooling2D(pool_size=(2, 2)))
model_f7_5_3_drop.add(Conv2D(64, (3, 3), padding='same', # Сверточный слой
                             input_shape=(128, 128, 1), activation='relu'))
model_f7_5_3_drop.add(MaxPooling2D(pool_size=(2, 2)))
model_f7_5_3_drop.add(Flatten()) # Полносвязная часть
model_f7_5_3_drop.add(Dense(512, activation='relu'))
model.add(Dropout(0.1)) #Слой дропаут
# Выходной слой, 208 нейронов по количеству классов
model_f7_5_3_drop.add(Dense(208, activation='softmax'))
```

После обучения получаем значение точности на тестовой выборке 100%.

Эта модель очень похожа на модель, Fruits\_360\_CNN. Но имеет на один сверточный слой меньше, кроме того, Fruits\_360\_CNN имеет два слоя дропаут, возможно именно это приводит к снижению точности модели взятой из интернета. Графики обучения можно увидеть на рисунке 18 и рисунке 19.

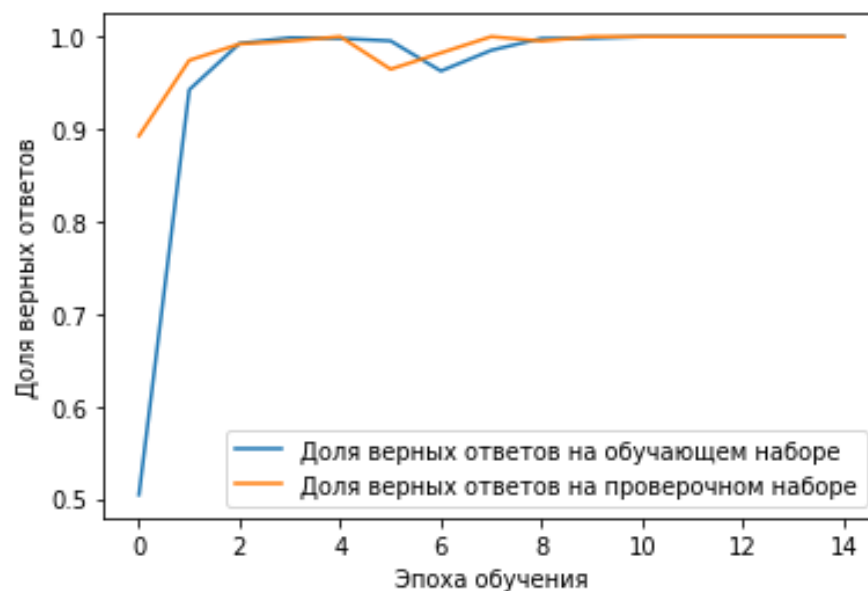


Рисунок 18 - График обучения модели

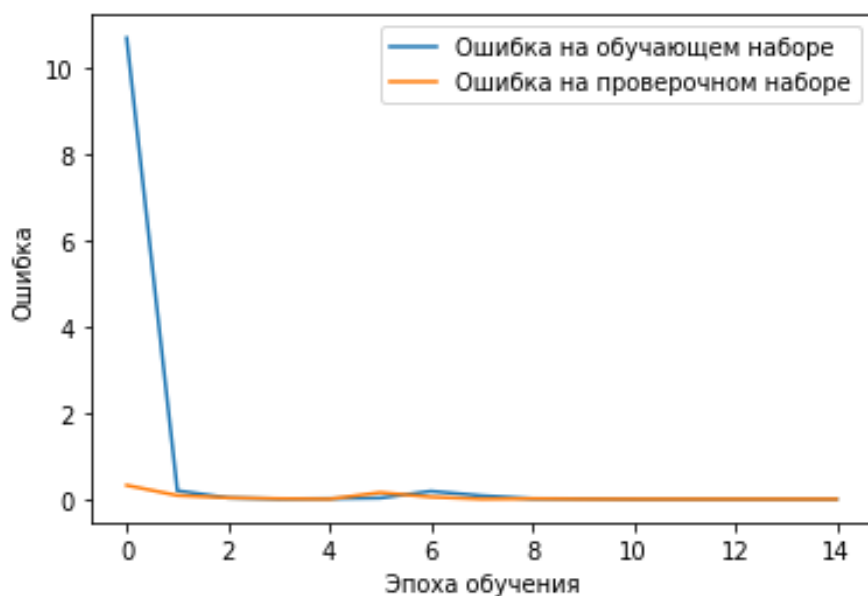


Рисунок 19 - График обучения модели

В последнем эксперименте я хочу изменить размеры фильтров на сверточных слоях. Сделаем фильтр первого слоя 11x11 пикселей, второго слоя 7x7 и третьего слоя 3x3 пикселя. Все остальное оставим неизменным как в предыдущей модели.

```

model_f11_7_3_drop = Sequential()
model_f11_7_3_drop.add(Conv2D(16, (11, 11), padding='same', # Сверточный слой
                             input_shape=(128, 128, 1), activation='relu'))
model_f11_7_3_drop.add(MaxPooling2D(pool_size=(2, 2)))
model_f11_7_3_drop.add(Conv2D(32, (7, 7), padding='same', # Сверточный слой
                             input_shape=(128, 128, 1), activation='relu'))
model_f11_7_3_drop.add(MaxPooling2D(pool_size=(2, 2)))
model_f11_7_3_drop.add(Conv2D(64, (3, 3), padding='same', # Сверточный слой
                             input_shape=(128, 128, 1), activation='relu'))
model_f11_7_3_drop.add(MaxPooling2D(pool_size=(2, 2)))
model_f11_7_3_drop.add(Flatten()) # Полносвязная часть
model_f11_7_3_drop.add(Dense(512, activation='relu'))
model_f11_7_3_drop.add(Dropout(0.1))
model_f11_7_3_drop.add(Dense(208, activation='softmax')) # Выходной слой, 208

```

Результат меня немного удивил. Точность на тестовой выборке составила всего 98.88% такой же результат был получен с помощью самой простой нейросети *model\_simple*, содержащей всего один слой свертки. Графики обучения этой модели можно увидеть на рисунке 20

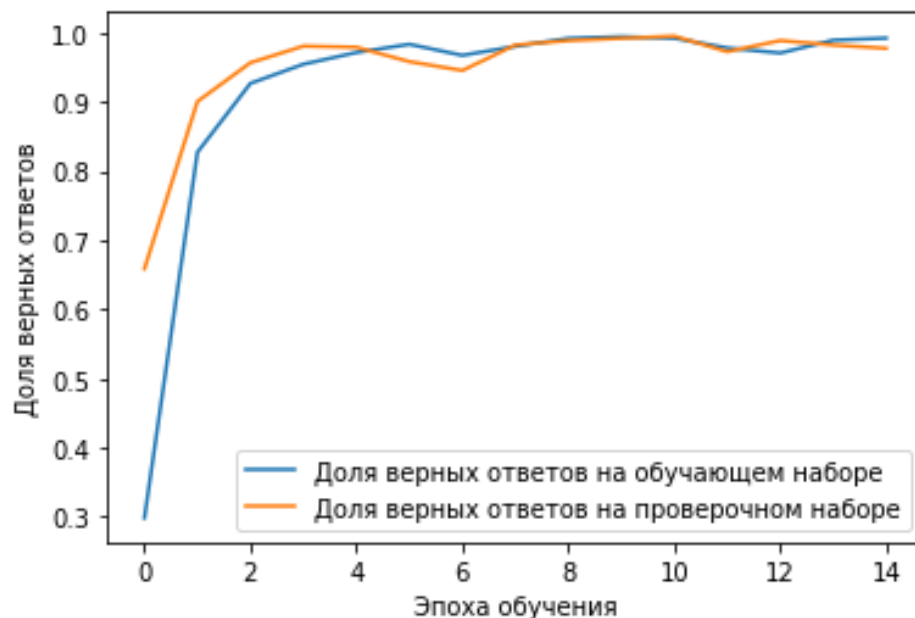


Рисунок 20 - График обучения модели

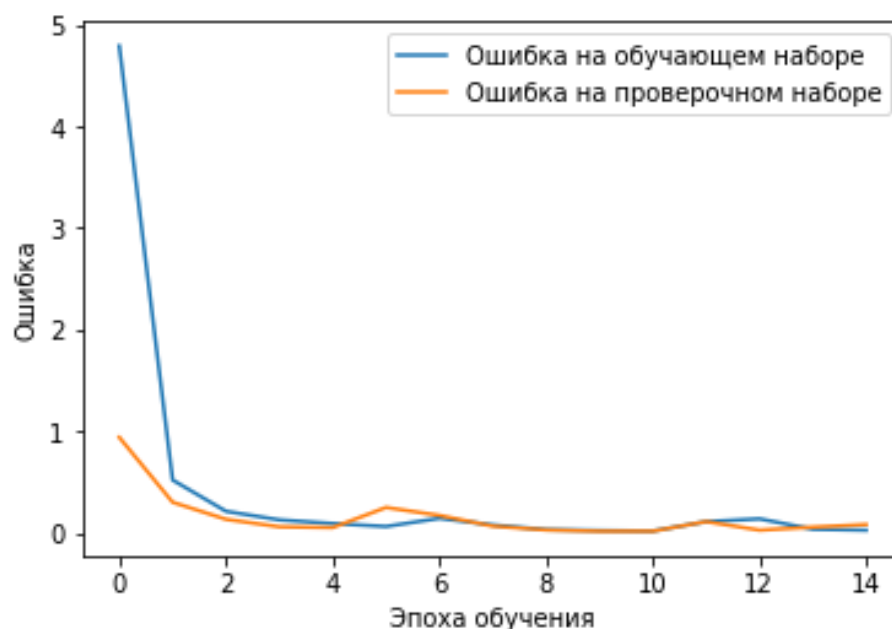


Рисунок 21 - График обучения модели

Таблица - Результаты исследований

Модель	Точность	Модель	Точность
Fruits_360_CNN	99.04%	model_s16_32_64	100%
model_simple	98.88%	model_s16_32_64_drop	99.84%
model_s32	98.72%	model_f7_5_3_drop	100%
model_s16_32	99.68%	model_f11_7_3_drop	98.88%

Очевидно, что использовать предстоит одну из двух моделей, **окрашенных зеленым**, я склоняюсь к модели: model\_f7\_5\_3\_drop. С точки зрения машинного обучения все модели показали очень высокие результаты, но следует учесть, я решаю узкоспециализированную задачу. Нейросеть должна распознавать достаточно простые изображения, при этом изображения должны распознаваться с максимально высокой точностью, желательно 100%, именно это является приоритетной целью.

## 2.4 Разработка приложений

В состав программного комплекса входит три приложения:

- 1) `ImgPrep_V_1_0_0.py`
- 2) `Classifier_V_1_0_0.py`
- 3) `check_simbol_pic_V_1_0_0.py`

Для работы приложений необходим Python 3.9.7. А также необходимо установить следующие библиотеки:

<code>pandas</code>	1.3.4
<code>tensorflow</code>	2.8.0
<code>imageio</code>	2.9.0
<code>imgaug</code>	0.4.0
<code>opencv-python</code>	4.5.5.62
<code>matplotlib</code>	3.4.3
<code>matplotlib-inline</code>	0.1.2
<code>numpy</code>	1.21.5
<code>keras</code>	2.8.0
<code>Keras-Preprocessing</code>	1.1.2

Запуск приложений с другими версиями Python или с другими версиями перечисленных библиотек не обязательно приведет к фатальным ошибкам, однако автор программ не гарантирует корректной работы приложений, в другом окружении. Все три утилиты должны запускаться в одной и той же директории.

### 2.4.1 Утилита `ImgPrep_V_1_0_0.py`

Приложение имеет интерфейс командной строки.

В директории, где запускается приложение необходимо создать две папки со следующей структурой:

```
./training_set
  /AN21
    / thumbnail_128x128.png
  ...
  /SIGNALHOLD
    / thumbnail_128x128.png
```

и

```
./test_set
  /AN21
    / thumbnail_128x128.png
```

```
...  
/SIGNALHOLD  
  / thumbnail_128x128.png
```

Данные папки должны содержать подпапки по количеству известных символов, в моем случае 208 папок, названия которых, совпадают с классами символов. В каждой подпапке находится необработанное изображение символа соответствующего класса.

При запуске приложения в терминале появится запрос, необходимо ввести имя папки, которую следует обрабатывать, затем появится второй запрос, ввести целочисленное значение, определяющее количество изображений, которые необходимо создать. Рекомендуется для training\_set создать 30 изображений, а для test\_set создать 3 изображения.

Приложение запускается отдельно для обработки папки training\_set и для обработки папки test\_set. Если папки уже присутствуют в рабочей директории и содержат обработанные изображения, в необходимом количестве, то нет необходимости производить вышеописанные действия, можно переходить к обучению классификатора.

Если мы хотим добавить новые символы, а затем переобучить модель классификатора, необходимо создать две директории add\_training\_set и add\_test\_set, поместить в них папки с именем и изображением нового класса.

```
./add_training_set  
  /new_class  
    / thumbnail_128x128.png  
и  
./add_test_set  
  /new_class  
    / thumbnail_128x128.png
```

Далее запустить приложение ImgPrep.py и произвести действия, аналогичные действиям при подготовке training\_set и test\_set.

После этого папки с подготовленными изображениями необходимо перенести из `add_training_set` и `add_test_set` в `training_set` и `test_set`, соответственно. Далее можно переходить к обучению классификатора.

**ВАЖНО:** в настоящий момент `training_set` и `test_set` содержат проверенный набор подготовленных изображений. Никакие действия с этими директориями не требуются.

#### **2.4.2 Утилита `Classifier_V_1_0_0.py`**

Данная утилита запускается в той же директории, что и утилита «`ImgPrep_V_1_0_0.py`». После запуска необходимо указать количество эпох, в течение которых следует обучать нейросеть, рекомендуемым значением для существующей версии утилит является 15 эпох. Далее утилита обращается к директориям `training_set` и `test_set`, происходит обучение нейросети. Описание используемой нейросети находится в главе 2.3. После окончания обучения модель сохраняется в файл с именем `Symbol_Check_model_f7_5_3_drop.h5`, кроме того утилита создает файл `class_names.csv` этот файл содержит имена классов. Данные файлы используются третьей утилитой `check_simbol_pic_V_1_0_0.py`. Никаких дополнительных действий от пользователя больше не требуется.

**ВАЖНО:** в настоящий момент существует обученная модель, которая сохранена в файле `Symbol_Check_model_f7_5_3_drop.h5` и файл `class_names.csv`. До появления новых классов символов, или обнаружения иных проблем с работой классификатора, нет необходимости переобучать существующую модель.

#### **2.4.3 Утилита `check_simbol_pic_V_1_0_0.py`**

Данная утилита является основной, именно она осуществляет проверку символов цифровой библиотеки. Утилита запускается в той же директории, что и предыдущие утилиты `ImgPrep_V_1_0_0.py` и `Classifier_V_1_0_0.py`.



Для работы программе необходимы три файла:

- 1) `class_names.csv`, который содержит имена классов,
- 2) `exemption.csv`, который содержит имена ячеек с нестандартными именами, имена ячеек у которых в имени отсутствует значение силы выходного драйвера X0, X1...X16, описание системы наименования ячеек находится в начале раздела 2.3.
- 3) `Symbol_Check_model_f7_5_3_drop.h5` в этом файле хранится обученная модель нейросети.

После запуска программа предложит ввести путь до проверяемой цифровой библиотеки. Далее появится предложение ввести суффикс характерный для библиотеки. Что такое суффикс описано в начале раздела 2.3.

В процессе работы утилита будет выводить в терминал имена ячеек, вызвавших у неё «подозрение». По окончании проверки программа выведет информацию о файле, в который были записаны все нераспознанные ячейки, так же в терминал будет выведена информация сколько всего ячеек было проверено.

## 2.5 Создание удаленного репозитория и загрузка результатов работы на него

Создан репозиторий <https://github.com/OlegMyagkov/GraduationWork>

1. Файл `ImgPrep.ipynb` юпитер ноутбук в котором разрабатывался скрипт для обработки и подготовки набора изображений.
2. Файл `Classifier.ipynb` юпитер ноутбук в котором проводилось исследование нейросетей для распознавания изображений.
3. Файл `check_simbol_pic.ipynb` юпитер ноутбук в котором разрабатывалось приложение для проверки символов цифровой библиотеки.
4. Файл `VKR_Myagkov_Oleg.docx` пояснительная записка (выпускная контрольная работа)
6. Директория `Check_Sym_Pic` содержит комплекс приложений и необходимых файлов для проверки символов цифровой библиотеки.