

Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт перспективной инженерии
Департамент цифровых, робототехнических систем и электроники

ОТЧЕТ
По лабораторной работе №1
Дисциплины «Основы нейронных сетей»

Выполнил:

Евдаков Евгений Владимирович

3 курс, группа ИВТ-б-о-22-1,

09.03.01 «Информатика и
вычислительная техника (профиль)
«Программное обеспечение средств
вычислительной
техники и автоматизированных
систем», очная форма обучения

(подпись)

Руководитель практики:

Воронкин Р. А., доцент департамента
цифровых и робототехнических
систем и электроники и института
перспективной инженерии

(подпись)

Ставрополь, 2024 г.

Тема: Введение в нейронные сети, линейный слой.

Цель: научиться создавать модель простой нейронной сети, а также научиться обучать нейронную сеть с использованием библиотеки tensorflow.

Ход работы:

Необходимо создать модель нейронной сети. Была выбрана новая среда выполнения (Графический процессор T4).

Для начала подключим класс создания последовательной модели «Sequential» из библиотеки «tensorflow»:

```
from tensorflow.keras.models import Sequential
```

Рисунок 1. Подключение класса «Sequential»

Затем создадим экземпляр этого класса:

```
model = Sequential()
```

Рисунок 2. Создание экземпляра

Далее используем модуль «Dense», был создан первый слой из 32 нейронов и сеть была настроена на вход из 10 чисел:

```
from tensorflow.keras.layers import Dense
model.add(Dense(32, input_dim=10))
```

/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an 'input_shape'/'input_dim' argument to a layer. When using Sequential models, prefer super().__init__(activity_regularizer=activity_regularizer, **kwargs)

Рисунок 3. Использование модуля «Dense»

Затем зададим оптимизатор «Adam» и функцию потерь. Выведем структуру полученной сети.

```
[5] model.compile(loss='categorical_crossentropy',
optimizer='adam')

[6] model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 32)	352

Total params: 352 (1.38 KB)
Trainable params: 352 (1.38 KB)
Non-trainable params: 0 (0.00 B)

Рисунок 4. Структура сети

Далее подобным образом создадим новую сеть, добавим в нее 3 слоя из 32, 5 и 1 нейрона, а затем выведем ее структура.

```
model = Sequential()
model.add(Dense(32, input_dim=10))
model.add(Dense(5))
model.add(Dense(1))

model.summary()
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
dense_1 (Dense)	(None, 32)	352
dense_2 (Dense)	(None, 5)	165
dense_3 (Dense)	(None, 1)	6

Total params: 523 (2.04 KB)
Trainable params: 523 (2.04 KB)
Non-trainable params: 0 (0.00 B)

Рисунок 5. Структура сети из 3-х слоев

Задание 1. Распознавание рукописных цифр MNIST

Для создания нейронной сети для распознавания рукописных цифр добавим все необходимые библиотеки, модули и данные для обучения сети:

```
from tensorflow.keras.datasets import mnist # Библиотека с базой рукописных цифр
from tensorflow.keras.models import Sequential # Подключение класса создания модели Sequential
from tensorflow.keras.layers import Dense # Подключение класса Dense - полносвязный слой
from tensorflow.keras import utils # Утилиты для подготовки данных
import numpy as np # Работа с массивами
import matplotlib.pyplot as plt # Отрисовка изображений
```

Рисунок 6. Добавление необходимых библиотек

Затем загрузим набор данных для обучения и тестирования сети из облака:

```
# Загрузка из облака данных Mnist
(x_train_org, y_train_org), (x_test_org, y_test_org) = mnist.load_data()
```

Downloading data from <https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz>
11490434/11490434 — 0s 0us/step

Рисунок 7. Загрузка данных

Отсюда загруженные данные - 6 000 картинок, каждая 28 на 28 пикселей.

```
x_train_org.shape
```

(60000, 28, 28)

Рисунок 8. Данные для обучения

Далее из массива картинок была получена картинка и выведена с помощью «matplotlib»:

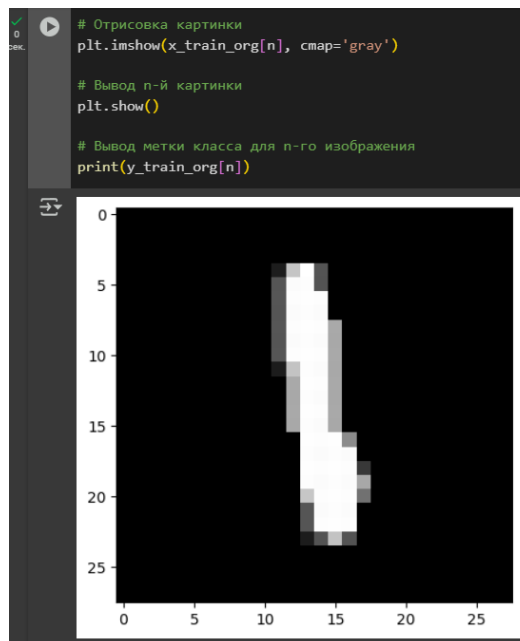


Рисунок 9. Вывод картинки

Далее переведем данные в одномерную последовательность чисел с помощью метода «reshape()»:

```
# Изменение формы входных картинок с 28x28 на 784
# первая ось остается без изменения, остальные складываются в вектор
x_train = x_train_org.reshape(x_train_org.shape[0], -1)
x_test = x_test_org.reshape(x_test_org.shape[0], -1)

# Проверка результата
print(f'Форма обучающих данных: {x_train_org.shape} -> {x_train.shape}')
print(f'Форма тестовых данных: {x_test_org.shape} -> {x_test.shape}')

Форма обучающих данных: (60000, 28, 28) -> (60000, 784)
Форма тестовых данных: (10000, 28, 28) -> (10000, 784)
```

Рисунок 10. Метод reshape

Далее выполним нормализацию (в данном случае, деление на 255):

```
# Нормализация входных картинок
# Преобразование x_train в тип float32 (числа с плавающей точкой) и нормализация
x_train = x_train.astype('float32') / 255.

# Преобразование x_test в тип float32 (числа с плавающей точкой) и нормализация
x_test = x_test.astype('float32') / 255.
```

Рисунок 11. Нормализация

Затем зададим количество распознаваемых классов, а также переведем данные в векторы «one hot encoding»:

```
CLASS_COUNT = 10

# Преобразование ответов в формат one_hot_encoding
y_train = utils.to_categorical(y_train_org, CLASS_COUNT)
y_test = utils.to_categorical(y_test_org, CLASS_COUNT)
```

Рисунок 12. Перевод данных в векторы

Задание 2. Создание нейронной сети.

Для начала создадим трехслойную модель сети:

```
# Создание последовательной модели
model = Sequential()

# Добавление полносвязного слоя на 800 нейронов с relu-активацией
model.add(Dense(800, input_dim=784, activation='relu'))

# Добавление полносвязного слоя на 400 нейронов с relu-активацией
model.add(Dense(400, activation='relu'))

# Добавление полносвязного слоя с количеством нейронов по числу классов с softmax-активацией
model.add(Dense(CLASS_COUNT, activation='softmax'))
```

/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass

Рисунок 13. Создание трехслойной модели

Далее данной модели назначим функцию ошибки и оптимизатор, модель была скомпилирована и выведена ее структура:

```
# Компиляция модели
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

# Вывод структуры модели
print(model.summary())
```

Model: "sequential_2"

Layer (type)	Output Shape	Param #
dense_4 (Dense)	(None, 800)	628,000
dense_5 (Dense)	(None, 400)	320,400
dense_6 (Dense)	(None, 10)	4,010

Total params: 952,410 (3.63 MB)
Trainable params: 952,410 (3.63 MB)
Non-trainable params: 0 (0.00 B)
None

Рисунок 14. Структура сети для распознавания цифр

Далее с помощью функции «plot_model()» нарисуем граф созданной сети:

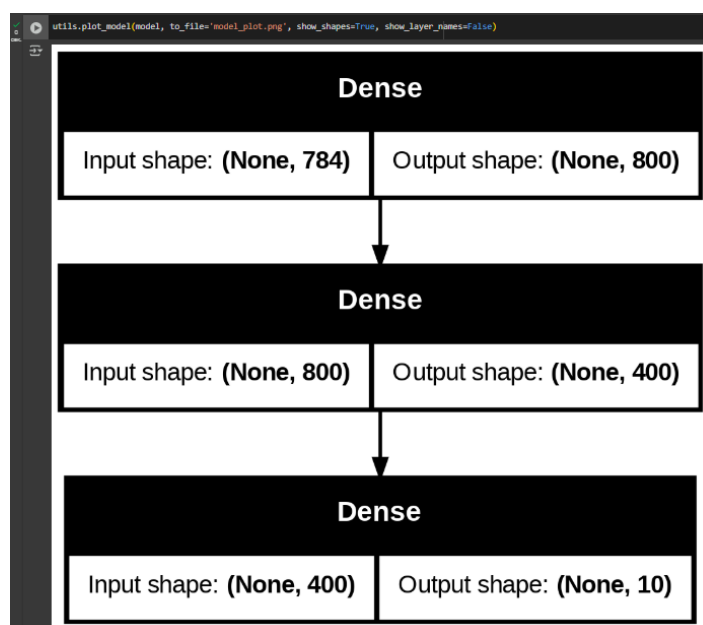


Рисунок 15. Граф сети для распознавания цифр

Далее выполним обучение нейронной сети. С помощью метода «fit()» модели передадим данные для обучения и запустим обучение нейронной сети, к концу обучения точность сети составила 0.9967.

```
model.fit(x_train, y_train, batch_size=128, epochs=15, verbose=1)
# обучающая выборка, входные данные
# обучающая выборка, выходные данные
# кол-во примеров, которое обрабатывает нейронка перед одним изменением весов
# количество эпох, когда нейронка обучается на всех примерах выборки
# 0 - не визуализировать ход обучения, 1 - визуализировать

Epoch 1/15
469/469 ————— 5s 5ms/step - accuracy: 0.8888 - loss: 0.3787
Epoch 2/15
469/469 ————— 1s 2ms/step - accuracy: 0.9752 - loss: 0.0803
Epoch 3/15
469/469 ————— 1s 2ms/step - accuracy: 0.9862 - loss: 0.0469
Epoch 4/15
469/469 ————— 1s 2ms/step - accuracy: 0.9889 - loss: 0.0324
Epoch 5/15
469/469 ————— 1s 3ms/step - accuracy: 0.9927 - loss: 0.0240
Epoch 6/15
469/469 ————— 1s 3ms/step - accuracy: 0.9940 - loss: 0.0188
Epoch 7/15
469/469 ————— 1s 3ms/step - accuracy: 0.9945 - loss: 0.0154
Epoch 8/15
469/469 ————— 1s 3ms/step - accuracy: 0.9944 - loss: 0.0180
Epoch 9/15
469/469 ————— 2s 3ms/step - accuracy: 0.9959 - loss: 0.0134
Epoch 10/15
469/469 ————— 2s 3ms/step - accuracy: 0.9957 - loss: 0.0121
Epoch 11/15
469/469 ————— 1s 3ms/step - accuracy: 0.9970 - loss: 0.0082
Epoch 12/15
469/469 ————— 1s 3ms/step - accuracy: 0.9974 - loss: 0.0082
Epoch 13/15
469/469 ————— 1s 3ms/step - accuracy: 0.9979 - loss: 0.0074
Epoch 14/15
469/469 ————— 3s 3ms/step - accuracy: 0.9976 - loss: 0.0070
Epoch 15/15
469/469 ————— 1s 3ms/step - accuracy: 0.9967 - loss: 0.0108
<keras.src.callbacks.history.History at 0x7ff8b40589d0>
```

Рисунок 16. Обучение модели

Далее сохраним настроенные веса модели:

```
model.save_weights('model.weights.h5')
model.load_weights('model.weights.h5')
```

Рисунок 17. Сохранение весов

Далее из тестового набора данных выберем изображение цифры 7:

```
# Номер тестовой цифры, которую будем распознавать
n_rec = np.random.randint(x_test_org.shape[0])

# Отображение картинки из тестового набора под номером n_rec
plt.imshow(x_test_org[n_rec], cmap='gray')
plt.show()
```

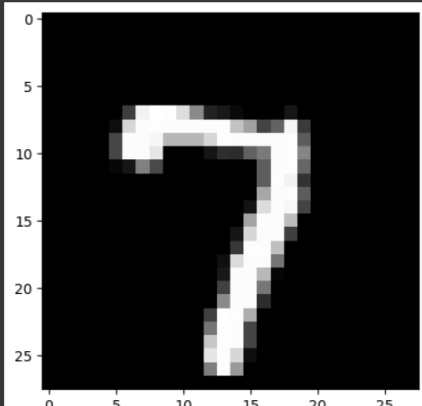
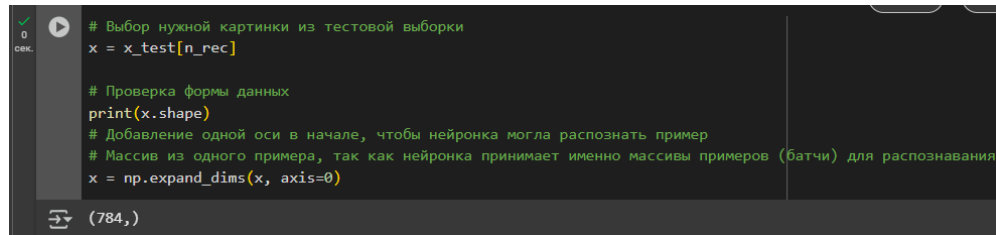


Рисунок 18. Изображение для распознавания

Данное изображение было сохранено в переменной для того, чтобы сделать предсказание:



```
# Выбор нужной картинки из тестовой выборки
x = x_test[n_rec]

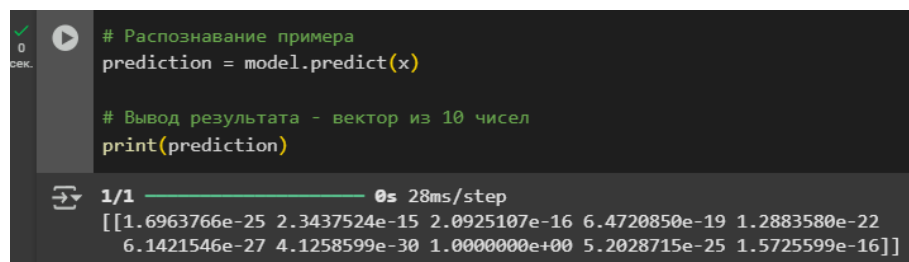
# Проверка формы данных
print(x.shape)

# Добавление одной оси в начале, чтобы нейронка могла распознать пример
# Массив из одного примера, так как нейронка принимает именно массивы примеров (батчи) для распознавания
x = np.expand_dims(x, axis=0)
```

(784,)

Рисунок 19. Сохранение изображения

Использование метода «predict()» и результат последовательности вероятностей, с которыми пример относится к тому или иному классу.



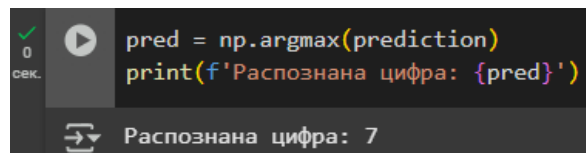
```
# Распознавание примера
prediction = model.predict(x)

# Вывод результата - вектор из 10 чисел
print(prediction)
```

1/1 — 0s 28ms/step
[[1.6963766e-25 2.3437524e-15 2.0925107e-16 6.4720850e-19 1.2883580e-22
6.1421546e-27 4.1258599e-30 1.0000000e+00 5.2028715e-25 1.5725599e-16]]

Рисунок 20. Предсказание модели

Далее определим индекс самой большой вероятности:

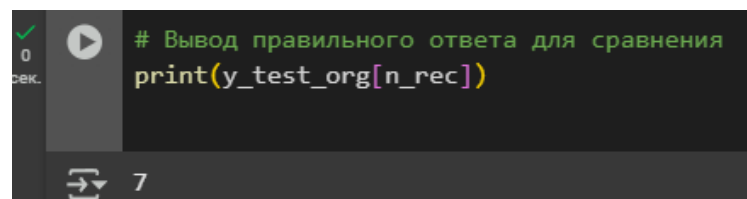


```
pred = np.argmax(prediction)
print(f'Распознана цифра: {pred}')
```

Распознана цифра: 7

Рисунок 21. Определение индекса

Далее проверим ответ нейронной сети на правильность:



```
# Вывод правильного ответа для сравнения
print(y_test_org[n_rec])
```

7

Рисунок 22. Ответ нейронной сети

Выполнение практических заданий:

Задание 1.

Условие: Дана модель сети следующей структуры:

- input_dim = 3 – размерность входных данных;
- Dense(3) – первый полносвязный слой с тремя нейронами;
- Dense(1) – второй полносвязный слой с одним нейроном.

Создайте модель заданной структуры, для этого: импортируйте библиотеку для создания модели импортируйте библиотеку для создания необходимых слоев создайте модель полносвязной сети добавьте заданные слои в модель. Выведите структуру модели с помощью функции `.summary()`.

```
# Ваше решение
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

model = Sequential()
model.add(Dense(3, input_dim = 3))
model.add(Dense(1))
model.summary()

weights = model.get_weights() # пока что веса случайные
print(weights)
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 3)	12
dense_1 (Dense)	(None, 1)	4

Total params: 16 (64.00 B)
 Trainable params: 16 (64.00 B)
 Non-trainable params: 0 (0.00 B)
 [array([[0.9113934, -0.5712757, -0.0407815],
 [-0.10088134, -0.896276, 0.1630857],
 [-0.42919683, -0.9949379, 0.44967675]], dtype=float32), array([0., 0., 0.], dtype=float32), array([[-1.1710387],
 [0.7663387],
 [-0.62915003]], dtype=float32), array([0.], dtype=float32))]

Рисунок 23. Структура сети, ее параметры и веса

Задание 2.

Условие: Создайте такую же нейронную сеть, как в первом задании, отключив нейрон смещения - параметр `use_bias=False`, используемый при создании полносвязного слоя. Выведите структуру модели и веса. Посмотрите, что изменилось.

```
model = Sequential()
model.add(Dense(3, input_dim = 3, use_bias=False))
model.add(Dense(1, use_bias=False))
model.summary()

weights = model.get_weights()
print(weights)
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
dense_2 (Dense)	(None, 3)	9
dense_3 (Dense)	(None, 1)	3

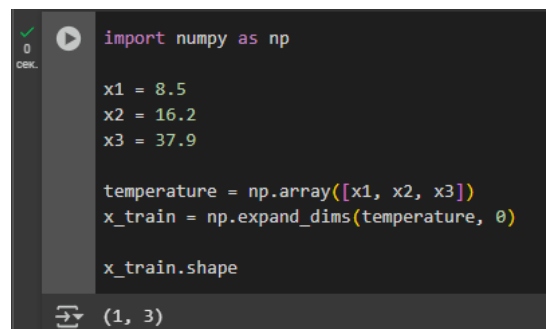
Total params: 12 (48.00 B)
 Trainable params: 12 (48.00 B)
 Non-trainable params: 0 (0.00 B)
 [array([[0.22415233, 0.30488324, 0.563298],
 [-0.6563654, 0.3810737, -0.775527],
 [-0.738878, 0.7671449, 0.42358208]], dtype=float32), array([[0.0882076],
 [-0.2509941],
 [0.72517276]], dtype=float32)]

Рисунок 24. Структура сети без нейрона смещения

Задание 3.

Условие: Создайте набор числовых данных размерностью (1, 3) для обучения нейронной сети.

- импортируйте библиотеку для работы с массивами
- задайте три числовых значения
- с помощью функции `.array()` создайте массив из трёх заданных значений
- с помощью функции `.expand_dims()` получите требуемую размерность входных данных - (1, 3)
- выведите размерность получившегося массива с помощью метода `.shape`



```
import numpy as np

x1 = 8.5
x2 = 16.2
x3 = 37.9

temperature = np.array([x1, x2, x3])
x_train = np.expand_dims(temperature, 0)

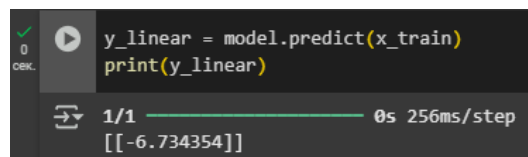
x_train.shape
```

(1, 3)

Рисунок 25. Выполнение 3 задания

Задание 4.

Условие: С помощью функции `.predict()` получите значение выхода сети, передав на вход вектор из трёх элементов, полученный в предыдущем задании.



```
y_linear = model.predict(x_train)
print(y_linear)
```

1/1 ————— 0s 256ms/step
[[-6.734354]]

Рисунок 26. Выполнение 4 задания

Задание 5.

Условие: Самостоятельно посчитайте выход сети, воспользовавшись массивом, полученным в задании 3, используя правила матричного перемножения.

```
# Ваше решение
N1 = x1 * weights[0][0, 0] + x2 * weights[0][1, 0] + x3 * weights[0][2, 0]
N2 = x1 * weights[0][0, 1] + x2 * weights[0][1, 1] + x3 * weights[0][2, 1]
N3 = x1 * weights[0][0, 2] + x2 * weights[0][1, 2] + x3 * weights[0][2, 2]

# Расчет значения выхода сети
Y_linear = N1 * weights[1][0, 0] + N2 * weights[1][1, 0] + N3 * weights[1][2, 0]

# Вывод выхода сети
print(Y_linear)
```

-6.7343535886270605

Рисунок 27. Выполнение 5 задания

Задание 6.

Условие: Создайте нейронную сеть следующей структуры:

- размер входных данных: 8
- полносвязный слой из 100 нейронов
- полносвязный слой из 10 нейронов
- полносвязный слой из 2 нейронов.
- Выведите summary модели.

```
model = Sequential()
model.add(Dense(100, input_dim = 8))
model.add(Dense(10))
model.add(Dense(2))
model.summary()
```

/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Model: "sequential_2"

Layer (type)	Output Shape	Param #
dense_4 (Dense)	(None, 100)	900
dense_5 (Dense)	(None, 10)	1,010
dense_6 (Dense)	(None, 2)	22

Total params: 1,932 (7.55 KB)
Trainable params: 1,932 (7.55 KB)
Non-trainable params: 0 (0.00 B)

Рисунок 28. Выполнение шестого задания

Задание 7.

Условие: Создайте нейронную сеть с такой же структурой, как в задаче 6, но без нейрона смещения во всех слоях.

```
model = Sequential()
model.add(Dense(100, input_dim = 8, use_bias=False))
model.add(Dense(10, use_bias=False))
model.add(Dense(2, use_bias=False))

model.summary()
```

Model: "sequential_3"

Layer (type)	Output Shape	Param #
dense_7 (Dense)	(None, 100)	800
dense_8 (Dense)	(None, 10)	1,000
dense_9 (Dense)	(None, 2)	20

Total params: 1,820 (7.11 KB)
Trainable params: 1,820 (7.11 KB)
Non-trainable params: 0 (0.00 B)

Рисунок 29. Выполнение седьмого задания

Задание 8.

Условие: Выведите веса модели из задачи 7 с помощью функции `.get_weights()`.

```
wei = model.get_weights()
print(wei)
```

```
[array([[ 1.84008434e-01, -2.60228664e-02, -1.22422263e-01,
        -2.44883299e-02,  2.23471075e-02, -8.46669525e-02,
         1.24659374e-01,  4.66223508e-02, -1.60698742e-01,
        -1.98254049e-01, -6.35957122e-02,  5.84216565e-02,
        -1.45900473e-01,  1.47430584e-01,  1.60486266e-01,
        -2.31012046e-01, -1.92247346e-01,  1.98688462e-01,
         1.10945329e-01,  9.58482176e-02, -1.85065240e-01,
        -3.20619494e-02,  1.67224556e-02,  1.00115016e-01,
        -1.91476852e-01,  1.72089800e-01, -2.24260464e-01,
         1.86410174e-01,  8.88578147e-02,  1.71723351e-01,
        -1.02242351e-01,  3.43990177e-02,  1.22212991e-01,
        -1.32065117e-01,  1.61914602e-01, -5.59404492e-02,
        -1.70145601e-01,  1.94973454e-01,  2.11941734e-01,
        -3.12885344e-02,  2.15826079e-01, -4.08474356e-02,
         1.55957118e-01,  1.41255781e-01,  9.68402475e-02,
```

Рисунок 30. Выполнение восьмого задания

Задание 9.

Условие: Задайте значения весов для модели следующей структуры:

- размерность входных данных равна 2
- количество нейронов на первом скрытом слое равно 2
- количество нейронов на втором скрытом слое равно 2
- количество нейронов на выходном слое равно 1
- нейрон смещения отключен на всех слоях.

```
model = Sequential()
model.add(Dense(2, input_dim=2, use_bias=False))
model.add(Dense(2, use_bias=False))
model.add(Dense(1, use_bias=False))

model.summary()

plot_model(model,
            dpi=60,
            show_shapes=True,
            show_layer_names=True)
```

```
Model: "sequential"
Layer (type)                Output Shape              Param #
-----
dense (Dense)                (None, 2)                  4
dense_1 (Dense)              (None, 2)                  4
dense_2 (Dense)              (None, 1)                  2
Total params: 10
Trainable params: 10
Non-trainable params: 0
```

Рисунок 31. Структура

```

# Ваше решение
import numpy as np
w1 = 0.7
w2 = 0.1
w3 = 0.28
w4 = 0.9
w5 = 0.7
w6 = 0.6
w7 = 0.15
w8 = 0.9
w9 = 0.28
w10 = 0.1
new_weight = [np.array([w1, w3], [w2, w4]), np.array([w5, w7], [w6, w8]), np.array([w9], [w10])]
print(new_weights)

```

```

array([[0.7, 0.28],
       [0.1, 0.9 ]]), array([[0.7, 0.15],
       [0.6, 0.9 ]]), array([0.28, 0.1 ])

```

Рисунок 32. Выполнение девятого задания

Задание 10.

Условие: Создайте модель для реализации структуры из задачи 9.

```

[10] model = Sequential()
      model.add(Dense(2, input_dim = 2, use_bias=False))
      model.add(Dense(2, use_bias=False))
      model.add(Dense(1, use_bias=False))
      model.summary()

```

Model: "sequential_4"

Layer (type)	Output Shape	Param #
dense_10 (Dense)	(None, 2)	4
dense_11 (Dense)	(None, 2)	4
dense_12 (Dense)	(None, 1)	2

Total params: 10 (40.00 B)
 Trainable params: 10 (40.00 B)
 Non-trainable params: 0 (0.00 B)

Рисунок 33. Выполнение десятого задания

Задание 11.

Условие: Создайте входной вектор из числовых значений, который можно использовать для формирования модели из задачи 10.

Пример создания входного вектора размерностью (1, 3): $x_1 = 5$ $x_2 = 1$ $x_3 = 6$
`x_train = np.expand_dims(np.array([x1, x2, x3]), 0)`

```

x1 = 4
x2 = 8
x_train = np.expand_dims(np.array([x1, x2]), 0)
x_train.shape

```

```

(1, 2)

```

Рисунок 34. Выполнение одиннадцатого задания

Задание 12.

Условие: Задайте созданные в задаче 9 веса в модель из задания 10 с помощью функции `.set_weights()`.

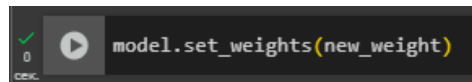


Рисунок 35. Выполнение двенадцатого задания

Задание 13.

Условие: Получите значения выхода сети с помощью функции `.predict()`, воспользовавшись вектором из задачи 11.

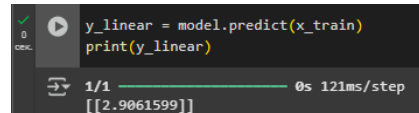


Рисунок 36. Выполнение тринадцатого задания

Задание 14.

Условие: Создайте нейронную сеть, содержащую три слоя, для классификации цифр от 0 до 5 включительно, с размерностью входных данных 256. Отобразите структуру модели.

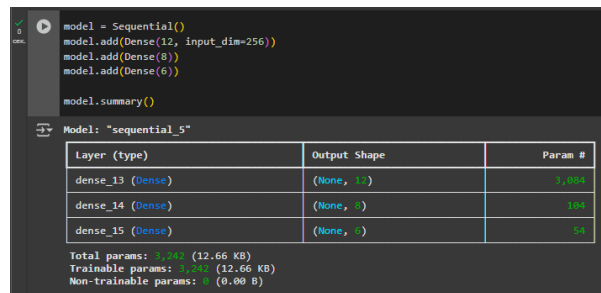


Рисунок 37. Выполнение четырнадцатого задания

Задание 15.

Условие: Создайте нейронную сеть для классификации 5-и видов диких животных по фотографии 25x25. Постройте архитектуру нейронной сети, содержащую шесть слоев.

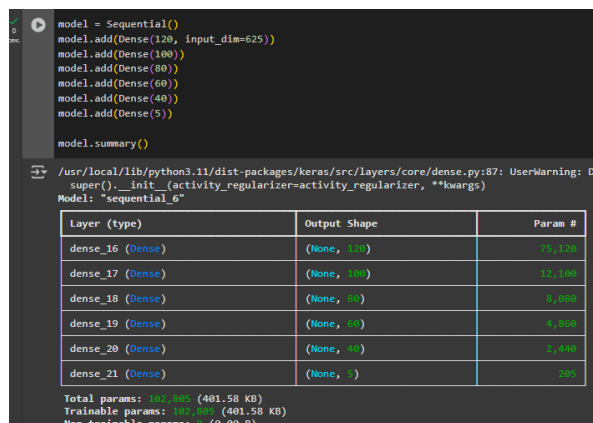


Рисунок 38. Выполнение пятнадцатого задания

Задание 16.

Условие: Создайте нейронную сеть, использующую температуру тела и давление для отличия больного человека от здорового. Постройте архитектуру нейронной сети, содержащую четыре слоя, на выходном слое используйте функцию активации linear.



```
model = Sequential()
model.add(Dense(12, input_dim=2))
model.add(Dense(10))
model.add(Dense(8))
model.add(Dense(1, activation='linear'))

model.summary()
```

Model: "sequential_8"

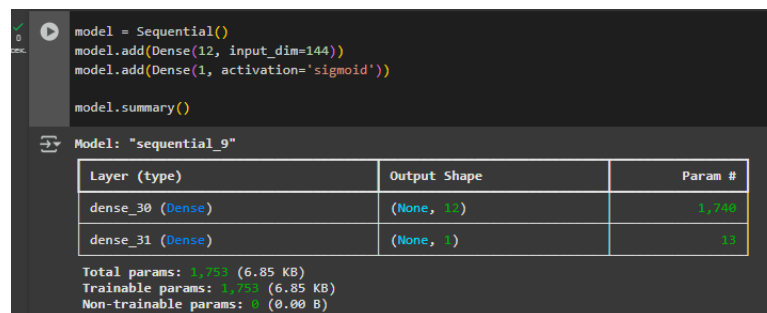
Layer (type)	Output Shape	Param #
dense_26 (Dense)	(None, 12)	36
dense_27 (Dense)	(None, 10)	130
dense_28 (Dense)	(None, 8)	88
dense_29 (Dense)	(None, 1)	9

Total params: 263 (1.03 KB)
Trainable params: 263 (1.03 KB)
Non-trainable params: 0 (0.00 B)

Рисунок 39. Выполнение шестнадцатого задания

Задание 17.

Условие: Создайте нейронную сеть, отличающую мак от розы по изображению 12 на 12 пикселей. Постройте архитектуру нейронной сети, содержащую два слоя, на выходном слое используйте функцию активации sigmoid.



```
model = Sequential()
model.add(Dense(12, input_dim=144))
model.add(Dense(1, activation='sigmoid'))

model.summary()
```

Model: "sequential_9"

Layer (type)	Output Shape	Param #
dense_30 (Dense)	(None, 12)	1,740
dense_31 (Dense)	(None, 1)	13

Total params: 1,753 (6.85 KB)
Trainable params: 1,753 (6.85 KB)
Non-trainable params: 0 (0.00 B)

Рисунок 40. Выполнение семнадцатого задания

Задание 18.

Условие: Создайте нейронную сеть для классификации пресмыкающихся по трем категориям. Известно, что каждая категория характеризуется 8-ю числовыми признаками. Постройте архитектуру нейронной сети, содержащую три слоя с различными активационными функциями для решения поставленной задачи.

```
model = Sequential()
model.add(Dense(12, input_dim=8, activation='relu'))
model.add(Dense(10, activation='elu'))
model.add(Dense(3, activation='softmax'))
model.summary()
```

Model: "sequential_10"

Layer (type)	Output Shape	Param #
dense_32 (Dense)	(None, 12)	108
dense_33 (Dense)	(None, 10)	130
dense_34 (Dense)	(None, 3)	33

Total params: 271 (1.06 KB)
Trainable params: 271 (1.06 KB)
Non-trainable params: 0 (0.00 B)

Рисунок 41. Выполнение восемнадцатого задания

Выполнение индивидуальных заданий:

Задание 1.

Условие: Создайте систему компьютерного зрения, которая будет определять тип геометрической фигуры. Используя подготовленную базу и шаблон ноутбука проведите серию экспериментов по перебору гиперпараметров нейронной сети, распознающей три категории изображений (треугольник, круг, квадрат).

Поменяйте количество нейронов в сети, используя следующие значения:

- один слой 10 нейронов
- один слой 100 нейронов
- один слой 5000 нейронов.

Поменяйте активационную функцию в скрытых слоях с relu на linear.

Поменяйте размеры batch_size:

- 10
- 100
- 1000

Выведите на экран получившиеся точности.

Всего должно получиться 18 комбинаций указанных параметров.

Для начала создания нейронной сети подключим следующие библиотеки и загрузим датасет из облака:

```
12 OK. # Подключение класса для создания нейронной сети прямого распространения
from tensorflow.keras.models import Sequential
# Подключение класса для создания полносвязного слоя
from tensorflow.keras.layers import Dense
# Подключение оптимизатора
from tensorflow.keras.optimizers import Adam
# Подключение утилит для to_categorical
from tensorflow.keras import utils
# Подключение библиотеки для загрузки изображений
from tensorflow.keras.preprocessing import image
# Подключение библиотеки для работы с массивами
import numpy as np
# Подключение библиотек для отрисовки изображений
import matplotlib.pyplot as plt
# Подключение модуля для работы с файлами
import os
# Вывод изображения в ноутбуке, а не в консоли или файле
%matplotlib inline

[2] OK. # Загрузка датасета из облака
import gdown
gdown.download('https://storage.yandexcloud.net/aiueducation/Content/base/13/hw_light.zip', None, quiet=True)

'hw_light.zip'
```

Рисунок 42. Подключение библиотек

Далее выполним нормализацию данных для загрузки их в нейронную сеть. Так же добавим метки для изображений, изображения преобразуем в массив:

```
0 OK. # Распаковываем архив hw_light.zip в папку hw_light
!unzip -q hw_light.zip

0 OK. # Путь к директории с базой
base_dir = '/content/hw_light'
# Создание пустого списка для загрузки изображений обучающей выборки
x_train = []
# Создание списка для меток классов
y_train = []
# Задание высоты и ширины загружаемых изображений
img_height = 20
img_width = 20
# Перебор папок в директории базы
for patch in os.listdir(base_dir):
    # Перебор файлов в папках
    for img in os.listdir(base_dir + '/' + patch):
        # Добавление в список изображений текущей картинки
        x_train.append(image.img_to_array(image.load_img(base_dir + '/' + patch + '/' + img,
                                                         target_size=(img_height, img_width),
                                                         color_mode='grayscale'))))
        # Добавление в массив меток, соответствующих классам
        if patch == '0':
            y_train.append(0)
        elif patch == '3':
            y_train.append(1)
        else:
            y_train.append(2)

# Преобразование в numpy-массив загруженных изображений и меток классов
x_train = np.array(x_train)
y_train = np.array(y_train)
# Вывод размерностей
print('Размер массива x_train', x_train.shape)
print('Размер массива y_train', y_train.shape)

Размер массива x_train (302, 20, 20, 1)
Размер массива y_train (302,)
```

Рисунок 43. Нормализация данных

После обучения сетей была составлена сравнительная характеристика точности каждой сети.

Таблица 1 – Сравнительная характеристика сетей

Количество нейронов:	10		100		5000	
Количество батчей:	relu	linear	relu	linear	relu	linear
10	0.811	0.785	0.934	0.825	0.934	0.798
100	0.596	0.530	0.728	0.639	0.785	0.669
1000	0.417	0.487	0.762	0.487	0.550	0.490

Исходя из данных таблицы, самым оптимальным вариантом была бы сеть с количеством нейронов 5000, функцией активации «relu» и количеством батчей равным 10. Увеличение количества батчей не ведет к улучшению свойств сети, функция активации «relu» справилась с данной задачей лучше, чем «linear», чем больше нейронов у сети, тем лучше ее параметры.

Задание 2.

Условие: Самостоятельно напишите нейронную сеть, которая может стать составной частью системы бота для игры в "Крестики-нолики". Используя подготовленную базу изображений, создайте и обучите нейронную сеть, распознающую две категории изображений: крестики и нолики. Добейтесь точности распознавания более 95% (ассурасу)

```

# Подключение класса для создания нейронной сети прямого распространения
from tensorflow.keras.models import Sequential
# Подключение класса для создания полносвязного слоя
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Dense, Dropout
# Подключение оптимизатора
from tensorflow.keras.optimizers import Adam
# Подключение утилит для to_categorical
from tensorflow.keras import utils
# Подключение библиотеки для загрузки изображений
from tensorflow.keras.preprocessing import image
# Подключение библиотеки для работы с массивами
import numpy as np
# Подключение модуля для работы с файлами
import os
# Подключение библиотек для отрисовки изображений
import matplotlib.pyplot as plt
from PIL import Image
# Вывод изображения в ноутбуке, а не в консоли или файле
%matplotlib inline

[24] # Загрузка датасета из облака
import gdown
gdown.download('https://storage.yandexcloud.net/aiueducation/Content/base/l3/hw_pro.zip', None, quiet=True)

```

Рисунок 44. Загрузка необходимых библиотек

Далее распакуем архив и укажем путь к директории. Создадим необходимые списки и зависимости. Выполним вывод размерностей.

```
0 сек. # Распаковываем архив hw_light.zip в папку hw_light
!unzip -q hw_pro.

# Путь к директории с базой
base_dir = 'hw_pro'

unzip: cannot find or open hw_pro., hw_pro..zip or hw_pro..ZIP.

[27] # Путь к директории с базой
base_dir = '/content/hw_pro'
# Создание пустого списка для загрузки изображений обучающей выборки
x_train = []
# Создание списка для меток классов
y_train = []
# Задание высоты и ширины загружаемых изображений
img_height = 20
img_width = 20
# Перебор папок в директории базы
for patch in os.listdir(base_dir):
    # Перебор файлов в папках
    for img in os.listdir(base_dir + '/' + patch):
        # Добавление в список изображений текущей картинки
        x_train.append(image.img_to_array(image.load_img(base_dir + '/' + patch + '/' + img,
                                                         target_size=(img_height, img_width),
                                                         color_mode='grayscale'))))
        # Добавление в массив меток, соответствующих классам
        if patch == '0':
            y_train.append(0)
        else:
            y_train.append(1)
# Преобразование в numpy-массив загруженных изображений и меток классов
x_train = np.array(x_train)
y_train = np.array(y_train)
# Вывод размерностей
print('Размер массива x_train', x_train.shape)
print('Размер массива y_train', y_train.shape)

Размер массива x_train (102, 20, 20, 1)
Размер массива y_train (102,)
```

Рисунок 45. Распаковка архива

Далее выполним нормализацию отношений. Преобразуем данные в одномерный массив. И создадим модель нейронной сети, выполним компиляцию и обучение модели, выведем точность и выполним сохранение весов.

```
0 сек. [28] # Нормализация данных (приведение значений пикселей к диапазону [0, 1])
x_train = x_train / 255.0

0 сек. [29] # Преобразование данных в одномерный вектор
x_train = x_train.reshape(x_train.shape[0], -1)

13 сек. [30] # Создание модели нейронной сети
model = Sequential()
model.add(Dense(400, input_dim=400, activation='relu'))
model.add(Dropout(0.2)) # Добавляем Dropout для предотвращения переобучения
model.add(Dense(200, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(100, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(1, activation='sigmoid'))

# Компиляция модели
model.compile(loss='binary_crossentropy', optimizer=Adam(learning_rate=0.0005), metrics=['accuracy'])

# Обучение модели
history = model.fit(x_train, y_train, epochs=50, batch_size=32, validation_split=0.2, verbose=1)

# Вывод точности на последней эпохе
print(f"Финальная точность: {history.history['accuracy'][-1]}")

# Сохранение весов модели
model.save_weights("dz_pro.weights.h5")

# Визуализация процесса обучения
plt.plot(history.history['accuracy'], label='Точность на обучающей выборке')
plt.plot(history.history['val_accuracy'], label='Точность на валидационной выборке')
plt.xlabel('Эпоха')
plt.ylabel('Точность')
plt.legend()
plt.show()
```

Рисунок 46. Нормализация данных

Далее визуализируем процесс обучения.

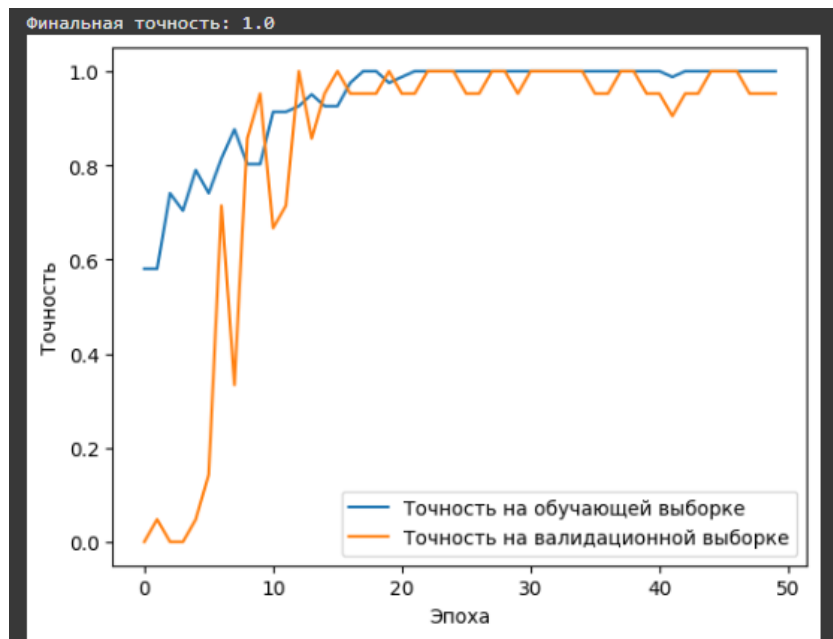


Рисунок 47. Визуализация процесса обучения

Задание 3.

Условие: Распознайте рукописную цифру, написанную на листе от руки. Последовательность шагов следующая:

- На бумаге рисуем произвольную цифру (желательно нарисовать цифру размером не более 5 * 5 мм и без наклона. В занятии нейронка обучалась на цифрах американских студентов. Эти цифры были написаны на тетрадных листах в клетку и имели схожий размер).
- Фотографируем. Загружаем фото в Collaboratory.
- С помощью функции `image.load_img(path, target_size=(28, 28), color_mode = 'grayscale')` загружаем картинку в переменную.
- С помощью функции `image.img_to_array(img)` преобразуем изображение в numpy-массив.
- Выполняем инверсию цветов, нормирование и ресейп массива.
- Выполняем распознавание собственной рукописной цифры.

Примечание: точность распознавания рукописных цифр может быть достаточно низкой, т.к. рукописные цифры после преобразований хоть и похожи на содержащиеся в базе, но могут отличаться по конфигурации, толщине линий и т.д.

Для начала выполним загрузку необходимых библиотек и выполним преобразование данных, а так же нормализацию данных.

```
[2] from tensorflow.keras.datasets import mnist
    from tensorflow.keras import utils
    from tensorflow.keras.models import Sequential
    from tensorflow.keras.layers import Dense
    import numpy as np

[3] (x_train_org, y_train_org), (x_test_org, y_test_org) = mnist.load_data()

x_train = x_train_org.reshape(x_train_org.shape[0], -1)
x_test = x_test_org.reshape(x_test_org.shape[0], -1)

# Преобразование x_train в тип float32 (числа с плавающей точкой) и нормализация
x_train = x_train.astype("float32") / 255.0

# Преобразование x_test в тип float32 (числа с плавающей точкой) и нормализация
x_test = x_test.astype("float32") / 255.0

CLASS_COUNT = 10

# Преобразование ответов в формат one_hot_encoding
y_train = utils.to_categorical(y_train_org, CLASS_COUNT)
y_test = utils.to_categorical(y_test_org, CLASS_COUNT)
```

Downloading data from <https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz>
11490434/11490434 — 1s 0us/step

Рисунок 48. Загрузка необходимых библиотек

Далее создадим нашу модель и выполним компиляцию:

```
# Создание последовательной модели
model = Sequential()

# Добавление полносвязного слоя на 800 нейронов с relu-активацией
model.add(Dense(800, input_dim=784, activation="relu"))

# Добавление полносвязного слоя на 400 нейронов с relu-активацией
model.add(Dense(400, activation="relu"))

# Добавление полносвязного слоя с количеством нейронов по числу классов с softmax-активацией
model.add(Dense(CLASS_COUNT, activation="softmax"))

# Компиляция модели
model.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["accuracy"])
```

/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass super().__init__(activity_regularizer=activity_regularizer, **kwargs)

Рисунок 49. Создание модели

Далее выполним обучение модели:

```
model.fit(
    x_train, # обучающая выборка, входные данные
    y_train, # обучающая выборка, выходные данные
    batch_size=128, # кол-во примеров, которые обрабатывает нейронка перед одним изменением весов
    epochs=15, # количество эпох, когда нейронка обучается на всех примерах выборки
    verbose=1,
) # 0 - не визуализировать ход обучения, 1 - визуализировать
```

Epoch 1/15
469/469 — 5s 5ms/step - accuracy: 0.8888 - loss: 0.3799
Epoch 2/15
469/469 — 3s 3ms/step - accuracy: 0.9762 - loss: 0.0807
Epoch 3/15
469/469 — 1s 3ms/step - accuracy: 0.9855 - loss: 0.0453
Epoch 4/15
469/469 — 3s 3ms/step - accuracy: 0.9894 - loss: 0.0323
Epoch 5/15
469/469 — 2s 3ms/step - accuracy: 0.9928 - loss: 0.0218
Epoch 6/15
469/469 — 3s 3ms/step - accuracy: 0.9935 - loss: 0.0193
Epoch 7/15
469/469 — 1s 3ms/step - accuracy: 0.9943 - loss: 0.0180
Epoch 8/15
469/469 — 1s 3ms/step - accuracy: 0.9966 - loss: 0.0110
Epoch 9/15
469/469 — 1s 3ms/step - accuracy: 0.9952 - loss: 0.0143
Epoch 10/15
469/469 — 1s 3ms/step - accuracy: 0.9961 - loss: 0.0118
Epoch 11/15
469/469 — 2s 4ms/step - accuracy: 0.9945 - loss: 0.0171
Epoch 12/15
469/469 — 2s 3ms/step - accuracy: 0.9959 - loss: 0.0124
Epoch 13/15
469/469 — 1s 3ms/step - accuracy: 0.9982 - loss: 0.0059
Epoch 14/15
469/469 — 3s 3ms/step - accuracy: 0.9975 - loss: 0.0091
Epoch 15/15
469/469 — 3s 3ms/step - accuracy: 0.9970 - loss: 0.0093
<keras.src.callbacks.history.History at 0x7a241bedb8d0>

Рисунок 50. Обучение модели

Далее фотографируем цифру и загружаем фото в Collaboratory.

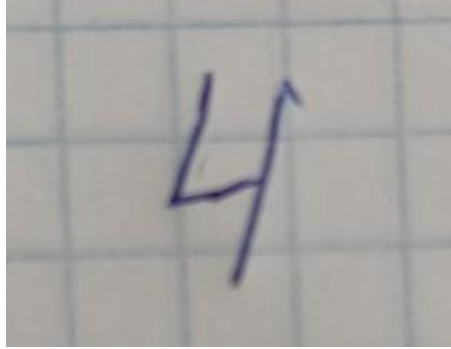


Рисунок 51. Написанная цифра

Затем с помощью функции `image.load_img(path, target_size=(28, 28), color_mode = 'grayscale')` загружаем картинку в переменную. И с помощью функции `image.img_to_array(img)` преобразуем изображение в numpy-массив.

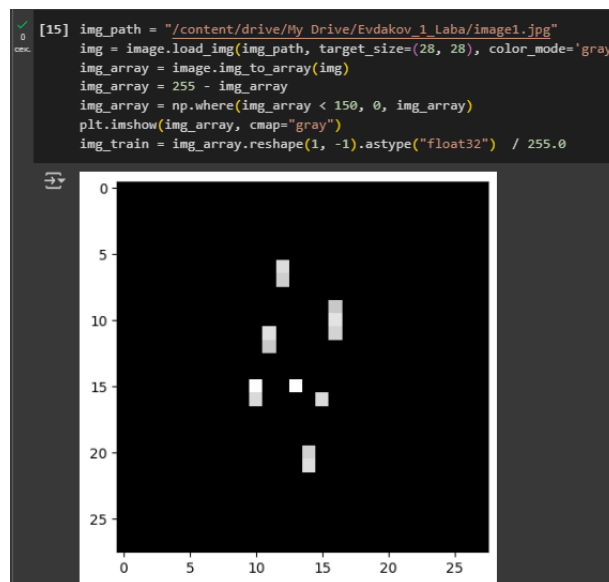


Рисунок 52. Загрузка изображения

Последним этапом выполняем распознавание собственной рукописной цифры.

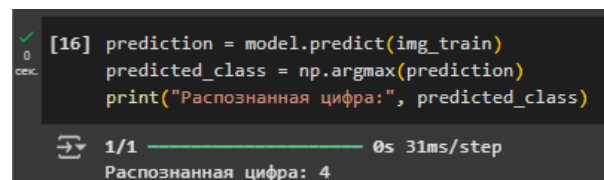


Рисунок 53. Распознавание цифры

Ссылка на папку с гугл диском, в которой содержатся все выполняемые файлы:

https://drive.google.com/drive/folders/18eEYkUGb5kwOrkfmFjmDcyFSo0aeVYIb?usp=drive_link

Вывод: научился создавать модель простой нейронной сети, а также научиться обучать нейронную сеть с использованием библиотеки tensorflow.