

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

Институт перспективной инженерии  
Департамента цифровых, роботехнических систем и электроники

**ОТЧЕТ**  
**ПО ЛАБОРАТОРНОЙ РАБОТЕ №1**  
**дисциплины «Введение в нейронные сети»**

Выполнил:  
Гайчук Дарья Дмитриевна  
3 курс, группа ИВТ-б-о-22-1,  
09.03.01 «Информатика  
и вычислительная техника»,  
направленность (профиль)  
«Программное обеспечение средств  
вычислительной техники и  
автоматизированных систем», очная  
форма обучения

---

(подпись)

Руководитель практики:  
Воронкин Р.А.-доцент департамента  
цифровых, роботехнических систем и  
электроники института перспективной  
инженерии

---

(подпись)

Отчет защищен с оценкой \_\_\_\_\_ Дата защиты \_\_\_\_\_

Ставрополь, 2025 г.

Тема: «Линейный слой»

Ссылка на git: [https://github.com/Ichizuchi/NN\\_LR1](https://github.com/Ichizuchi/NN_LR1)

Порядок выполнения работы:

Задание №1. Задана модель нейронной 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()`.

Выведите веса модели с помощью функции `get_weights()`.

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

# Создание модели с использованием списка слоев
model = Sequential([
    Dense(3, input_shape=(3,)), # первый полносвязный слой: 3 нейрона, вход 3D
    Dense(1)                    # второй слой: 1 нейрон
])

# Вывод архитектуры модели
model.summary()

# Вывод веса (параметра) модели
print("Параметры модели:", model.get_weights())
```

/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  
super().\_\_init\_\_(activity\_regularizer=activity\_regularizer, \*\*kwargs)  
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)  
Параметры модели: [array([[ 0.4376979 , -0.7815907 , -0.61476684],  
[-0.8737745 , 0.56122446, 0.87490802],  
[-0.02951264, -0.7918103 , 0.25186086]], dtype=float32), array([0., 0., 0.], dtype=float32), array([[ 0.78446734],  
[-0.17965007],  
[ 0.35441244]], dtype=float32), array([0.], dtype=float32)]

Рисунок 1. Результат выполнения программы №1

Задание №2. 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.

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

# Модель без параметров bias в каждом слое
model_no_bias = Sequential([
    Dense(3, input_shape=(3,), use_bias=False),
    Dense(1, use_bias=False)
])

# Вывод структуры модели
model_no_bias.summary()

# Распечатка весов модели
print("Веса модели без bias:", model_no_bias.get_weights())
```

Model: "sequential\_1"

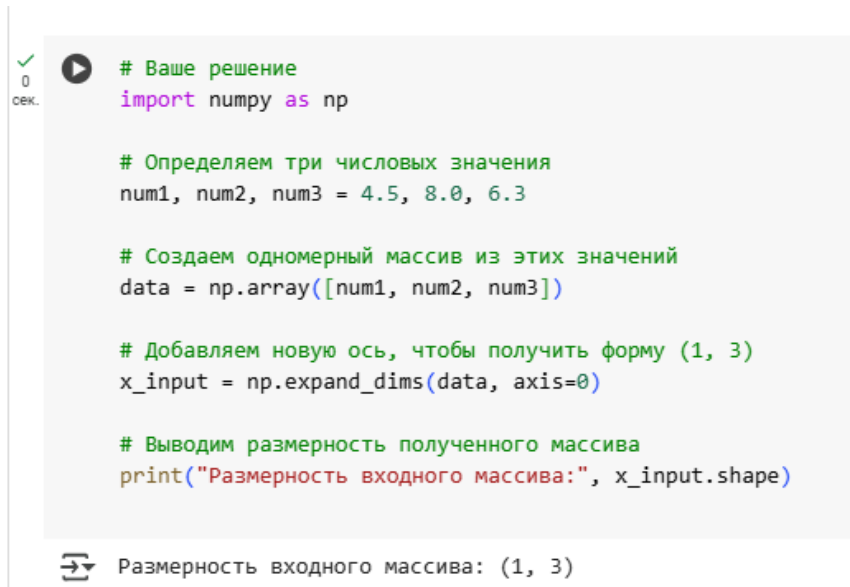
Layer (type)	Output Shape	Param #
dense_2 (Dense)	(None, 3)	9
dense_3 (Dense)	(None, 1)	3

Total params: 12 (48.00 B)  
Trainable params: 12 (48.00 B)  
Non-trainable params: 0 (0.00 B)  
Веса модели без bias: [array([[ -0.21431708, 0.6029453 , -0.6298449 ],  
[ 0.13221431, -0.5787082 , -0.18362308],  
[ 0.24758315, 0.43406105, -0.612777 ]], dtype=float32), array([[ 0.33649254],  
[ 0.95826113],  
[-1.0314991 ]], dtype=float32)]

Рисунок 2. Результат выполнения программы №2

Задание №3. 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`



```
# Ваше решение
import numpy as np

# Определяем три числовых значения
num1, num2, num3 = 4.5, 8.0, 6.3

# Создаем одномерный массив из этих значений
data = np.array([num1, num2, num3])

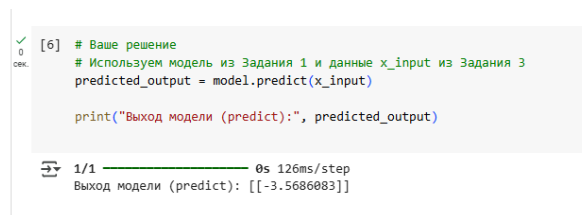
# Добавляем новую ось, чтобы получить форму (1, 3)
x_input = np.expand_dims(data, axis=0)

# Выводим размерность полученного массива
print("Размерность входного массива:", x_input.shape)
```

Размерность входного массива: (1, 3)

Рисунок 3. Результат выполнения программы №3

Задание №4. С помощью функции `.predict()` получите значение выхода сети, передав на вход вектор из трёх элементов, полученный в предыдущем задании.



```
[6]: # Ваше решение
# Используем модель из Задания 1 и данные x_input из Задания 3
predicted_output = model.predict(x_input)

print("Выход модели (predict):", predicted_output)
```

1/1 — 0s 126ms/step  
Выход модели (predict): [[-3.5686083]]

Рисунок 4. Результат выполнения программы №4

Задание №5. Самостоятельно посчитайте выход сети, воспользовавшись массивом, полученным в задании 3, используя правила матричного перемножения.

```

# Ваше решение

# weights = model.get_weights()

# Линейное преобразование
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]

# Вычисление итогового выхода (второй слой - линейное преобразование)
Y_linear = N1 * weights[1][0, 0] + N2 * weights[1][1, 0] + N3 * weights[1][2, 0]
print("Ручной расчет выхода сети:", Y_linear)

```

```

Traceback (most recent call last)
<ipython-input-11-1430ed6e541a> in <cell line: 0>()
      4
      5 # Линейное преобразование
----> 6 N1 = x1 * weights[0][0, 0] + x2 * weights[0][1, 0] + x3 * weights[0][2, 0]
      7 N2 = x1 * weights[0][0, 1] + x2 * weights[0][1, 1] + x3 * weights[0][2, 1]
      8 N3 = x1 * weights[0][0, 2] + x2 * weights[0][1, 2] + x3 * weights[0][2, 2]

NameError: name 'x1' is not defined

```

Решение

```

# Расчет значений нейронов скрытого слоя
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)

```

```

Traceback (most recent call last)
<ipython-input-9-0ab7c104153c> in <cell line: 0>()
      1 # Расчет значений нейронов скрытого слоя
----> 2 N1 = x1 * weights[0][0, 0] + x2 * weights[0][1, 0] + x3 * weights[0][2, 0]
      3 N2 = x1 * weights[0][0, 1] + x2 * weights[0][1, 1] + x3 * weights[0][2, 1]
      4 N3 = x1 * weights[0][0, 2] + x2 * weights[0][1, 2] + x3 * weights[0][2, 2]
      5 # Расчет значения выхода сети

NameError: name 'weights' is not defined

```

Рисунок 5. Ошибка в ходе выполнения задания

Задание №6. 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.

```

[7] # Ваше решение
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

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

```

```

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 6. Результат выполнения программы №6

Задание №7. Создайте нейронную сеть с такой же структурой, как в задаче 6, но без нейрона смещения во всех слоях.

```
[8] # Ваше решение
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 7. Результат выполнения программы №7

Задание №8. Выведите веса модели из задачи 7 с помощью функции `.get_weights()`.

```
# Ваше решение
weights_no_bias = model.get_weights()
print("Веса модели без bias:", weights_no_bias)
```

Веса модели без bias: [array([[ 0.00445671, 0.09470622, -0.21458973, 0.03009842, 0.11110561,  
-0.02347332, 0.17399843, 0.17638193, 0.19299878, -0.07543392,  
0.08264183, -0.03265701, -0.0543306 , -0.1428709 , 0.20797183,  
0.11076294, -0.20946793, 0.12806515, -0.20636147, 0.10981335,  
0.0371031 , 0.14816679, 0.11253922, 0.16367857, 0.03937308,  
0.10255496, -0.21940778, -0.03073364, -0.14706385, -0.0482187 ,  
0.22480352, -0.07878791, -0.20022069, -0.22200015, 0.02246381,  
0.12226172, 0.2268645 , 0.12267302, -0.05555652, -0.12152886,  
0.12837455, 0.1042182 , -0.113987 , 0.05696292, 0.0697396 ,  
-0.13868505, -0.2104196 , 0.10324077, -0.08849534, 0.03567271,  
-0.0500218 , -0.02768783, 0.02110134, -0.05608986, -0.08516389,  
0.01479612, 0.1166269 , 0.20854114, -0.0192991 , 0.19156592,  
-0.19285583, -0.1332607 , 0.07898448, -0.23152095, -0.08417585,  
0.10700212, -0.11970964, -0.01924251, 0.1599863 , 0.15661038,  
0.22534104, -0.23106645, -0.02921081, -0.02587642, 0.05567499,  
-0.09981829, -0.08186182, 0.04887693, 0.1905648 , 0.00820056,  
-0.22856079, 0.11956538, 0.09233351, -0.05350542, 0.10294665,  
-0.1650297 , 0.22267 , -0.2126385 , 0.11689536, 0.21639778,  
0.01912139, 0.06037812, 0.04042237, 0.0994464 , -0.14290598,  
0.05708773, 0.08446829, -0.1512652 , 0.16414945, 0.19872381],  
[-0.22831364, 0.15045105, -0.22905503, 0.02117004, 0.11516358,  
-0.22021514, -0.17625958, 0.08589949, 0.13279589, -0.06695685,  
-0.17529099, 0.11456491, 0.11546822, 0.01176706, -0.21047412,  
0.0497063 , -0.20804016, 0.16145737, -0.13638708, -0.0993512 ,  
0.17659749, 0.18880586, 0.10184415, 0.21784694, -0.17352037,  
-0.12341502, -0.23481847, -0.11121312, -0.13814512, 0.01283568,  
-0.03199109, 0.15692289, 0.22017945, 0.069984 , -0.04630405,  
0.13292615, 0.00481395, 0.1467352 , -0.10224669, 0.06729294,  
0.00940499, 0.2037826 , 0.21948226, 0.15787558, -0.18396407,  
-0.07254422, 0.00207222, -0.23309146, 0.17914505, 0.15570997,  
-0.05548498, -0.13140173, 0.23144104, -0.16183823, -0.23115462,  
-0.00296843, -0.12467628, -0.12097256, 0.17599614, 0.14276133,  
-0.21178403, -0.2003774 , 0.15923315, 0.16563578, -0.10538718,  
-0.2237646 , 0.20372151, -0.05548441, 0.16890107, -0.13574758,  
-0.01469217, 0.22416072, -0.17743452, -0.03365734, -0.02982935,  
0.23308988, -0.1939856 , 0.16653322, 0.2009006 , -0.1417937 ,  
-0.11082604, 0.08143519, -0.13410959, -0.23372006, 0.19456328,  
0.02863981, -0.14133878, -0.14007133, 0.07352827, -0.10070708,  
-0.04669933, 0.15349968, 0.23282887, -0.2061376 , -0.17039207,  
-0.01782817, 0.09217049, -0.1044658 , -0.07118331, -0.02005729],  
[-0.04302317, -0.19607067, -0.13985577, 0.16093121, 0.02155969,  
-0.20632552, 0.00340417, 0.13887353, 0.15202971, 0.16274111,  
0.11247714, -0.07436025, -0.2161854 , -0.13946842, -0.23016956,

Рисунок 8. Результат выполнения программы №8

Задание №9. 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.

```
[10] # Ваше решение
import numpy as np

# Веса для первого скрытого слоя (матрица 2x2)
w1, w2, w3, w4 = 0.7, 0.1, 0.28, 0.9
W1 = np.array([[w1, w3],
               [w2, w4]])

# Веса для второго скрытого слоя (матрица 2x2)
w5, w6, w7, w8 = 0.7, 0.6, 0.15, 0.9
W2 = np.array([[w5, w7],
               [w6, w8]])

# Веса для выходного слоя (матрица 2x1)
w9, w10 = 0.28, 0.1
W3 = np.array([[w9],
               [w10]])

custom_weights = [W1, W2, W3]
print("Заданные веса:", custom_weights)
```

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

Рисунок 9. Результат выполнения программы №9

Задание №10. Создайте модель для реализации структуры из задачи 9.

```
# Ваше решение
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 10. Результат выполнения программы №10

Задание №11. Создайте входной вектор из числовых значений, который можно использовать для формирования модели из задачи 10.

Пример создания входного вектора размерностью (1, 3):  $x_1 = 5$   $x_2 = 1$   $x_3 = 6$   
 $x\_train = np.expand\_dims(np.array([x_1, x_2, x_3]), 0)$

```
# Ваше решение
import numpy as np

# Создание входного вектора размером (1, 2)
x1, x2 = 8, 15
x_input = np.expand_dims(np.array([x1, x2]), axis=0)
print("Размерность входного вектора:", x_input.shape)
```

Размерность входного вектора: (1, 2)

Рисунок 11. Результат выполнения программы №11

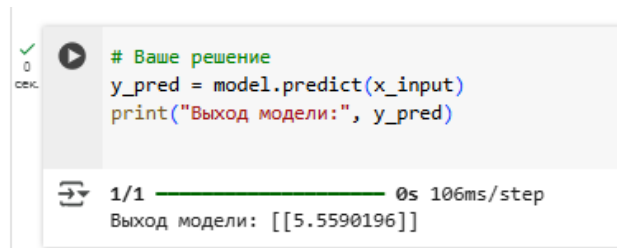
Задание №12. Задайте созданные в задаче 9 веса в модель из задания 10 с помощью функции `.set_weights()`.



```
# Ваше решение
model.set_weights(custom_weights)
```

Рисунок 12. Результат выполнения программы №12

Задание №13. Получите значения выхода сети с помощью функции `.predict()`, воспользовавшись вектором из задачи 11.

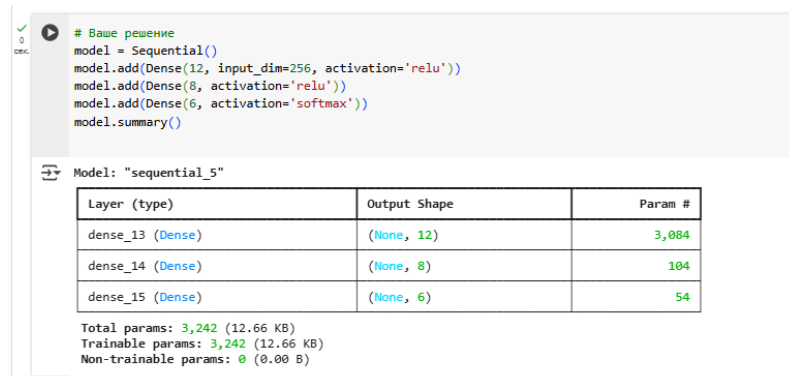


```
# Ваше решение
y_pred = model.predict(x_input)
print("Выход модели:", y_pred)
```

1/1 ————— 0s 106ms/step  
Выход модели: [[5.5590196]]

Рисунок 13. Результат выполнения программы №13

Задание №14. Создайте нейронную сеть, содержащую три слоя, для классификации цифр от 0 до 5 включительно, с размерностью входных данных 256. Отобразите структуру модели.



```
# Ваше решение
model = Sequential()
model.add(Dense(12, input_dim=256, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(6, activation='softmax'))
model.summary()
```

Model: "sequential\_5"

Layer (type)	Output Shape	Param #
dense_13 (Dense)	(None, 12)	3,084
dense_14 (Dense)	(None, 8)	104
dense_15 (Dense)	(None, 6)	54

Total params: 3,242 (12.66 KB)  
Trainable params: 3,242 (12.66 KB)  
Non-trainable params: 0 (0.00 B)

Рисунок 14. Результат выполнения программы №14

Задание №15. Создайте нейронную сеть для классификации 5-и видов диких животных по фотографии 25x25. Постройте архитектуру нейронной сети, содержащую шесть слоев.



```
# Ваше решение
model = Sequential()
model.add(Dense(120, input_dim=625, activation='relu'))
model.add(Dense(100, activation='relu'))
model.add(Dense(80, activation='relu'))
model.add(Dense(60, activation='relu'))
model.add(Dense(40, activation='relu'))
model.add(Dense(5, activation='softmax'))
model.summary()
```

Model: "sequential\_6"

Layer (type)	Output Shape	Param #
dense_16 (Dense)	(None, 120)	75,120
dense_17 (Dense)	(None, 100)	12,100
dense_18 (Dense)	(None, 80)	8,080
dense_19 (Dense)	(None, 60)	4,860
dense_20 (Dense)	(None, 40)	2,440
dense_21 (Dense)	(None, 5)	205

Total params: 102,805 (401.58 KB)  
Trainable params: 102,805 (401.58 KB)  
Non-trainable params: 0 (0.00 B)

Рисунок 15. Результат выполнения программы №15

Задание №16. 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.

```
# Ваше решение
model = Sequential()
model.add(Dense(12, input_dim=2, activation='relu'))
model.add(Dense(10, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='linear'))
model.summary()
```

Model: "sequential\_7"

Layer (type)	Output Shape	Param #
dense_22 (Dense)	(None, 12)	36
dense_23 (Dense)	(None, 10)	130
dense_24 (Dense)	(None, 8)	88
dense_25 (Dense)	(None, 1)	9

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

Рисунок 16. Результат выполнения программы №16

Задание №17. 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.

```
# Ваше решение
model = Sequential()
model.add(Dense(12, input_dim=144, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.summary()
```

Model: "sequential\_8"

Layer (type)	Output Shape	Param #
dense_26 (Dense)	(None, 12)	1,740
dense_27 (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 17. Результат выполнения программы №17

Задание №18. 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.

```
# Ваше решение
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\_9"

Layer (type)	Output Shape	Param #
dense_28 (Dense)	(None, 12)	108
dense_29 (Dense)	(None, 10)	130
dense_30 (Dense)	(None, 3)	33

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

Рисунок 18. Результат выполнения программы №18

В ходе выполнения задач были изучены основные принципы построения полносвязных нейронных сетей, включая работу с их архитектурой, весами и предсказаниями. В частности,

- 1) Реализованы различные структуры нейронных сетей с разным количеством слоев и нейронов.
- 2) Изучена разница между моделями с нейроном смещения и без него, что позволило увидеть, как этот параметр влияет на обучение сети.
- 3) Проведена работа с входными данными, их преобразованием и подачей в нейронную сеть.

4) Изучены методы вычисления выхода нейронной сети — как программно через `.predict()`, так и вручную с применением матричных операций.

5) Рассмотрены различные архитектуры нейронных сетей для решения прикладных задач, таких как классификация изображений, анализ медицинских данных и распознавание объектов.

### Выполнение домашнего задания

Уровень 1. 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).

1. Поменяйте количество нейронов в сети, используя следующие значения:

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

2. Поменяйте активационную функцию в скрытых слоях с `relu` на `linear`.

3. Поменяйте размеры `batch_size`:

- 10
- 100
- 1000

4. Выведите на экран получившиеся точности.

Всего должно получиться 18 комбинаций указанных параметров.

```

# Загрузка и подготовка данных
data_dir = "./hw_light"

train_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
    data_dir,
    target_size=(64, 64),
    batch_size=32,
    class_mode='categorical'
)

test_generator = test_datagen.flow_from_directory(
    data_dir,
    target_size=(64, 64),
    batch_size=32,
    class_mode='categorical'
)

# Гиперпараметры для экспериментов
neurons_options = [10, 100, 5000]
activation_options = ['relu', 'linear']
batch_sizes = [10, 100, 1000]

results = []

# Перебор всех комбинаций
for neurons in neurons_options:
    for activation in activation_options:
        for batch_size in batch_sizes:

            # Создание модели
            model = Sequential([
                Flatten(input_shape=(64, 64, 3)),
                Dense(neurons, activation=activation),
                Dense(3, activation='softmax')
            ])

            model.compile(optimizer=Adam(),
                          loss='categorical_crossentropy',
                          metrics=['accuracy'])

            # Обучение модели
            history = model.fit(
                train_generator,
                epochs=5,
                batch_size=batch_size,
                validation_data=test_generator,
                verbose=0
            )

            # Оценка модели
            _, test_acc = model.evaluate(test_generator, verbose=0)

            # Сохранение результатов
            results.append({
                'Neurons': neurons,
                'Activation': activation,
                'Batch Size': batch_size,
                'Accuracy': test_acc
            })

print(f"Neurons: {neurons}, Activation: {activation}, Batch Size: {batch_size}, Accuracy: {test_acc:.4f}")

```

Рисунок 19. Код для определения типа геометрической фигуры

```
Found 302 images belonging to 3 classes.
Found 302 images belonging to 3 classes.
/usr/local/lib/python3.11/dist-packages/keras/src/layers/reshaping/flatten.py:37: UserWarning: Do not pass an `in
super().__init__(**kwargs)
/usr/local/lib/python3.11/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: `
self._warn_if_super_not_called()
Neurons: 10, Activation: relu, Batch Size: 10, Accuracy: 0.5464
Neurons: 10, Activation: relu, Batch Size: 100, Accuracy: 0.3377
Neurons: 10, Activation: relu, Batch Size: 1000, Accuracy: 0.6358
Neurons: 10, Activation: linear, Batch Size: 10, Accuracy: 0.5166
Neurons: 10, Activation: linear, Batch Size: 100, Accuracy: 0.8377
Neurons: 10, Activation: linear, Batch Size: 1000, Accuracy: 0.7914
Neurons: 100, Activation: relu, Batch Size: 10, Accuracy: 0.8245
Neurons: 100, Activation: relu, Batch Size: 100, Accuracy: 0.7748
Neurons: 100, Activation: relu, Batch Size: 1000, Accuracy: 0.7682
Neurons: 100, Activation: linear, Batch Size: 10, Accuracy: 0.6821
Neurons: 100, Activation: linear, Batch Size: 100, Accuracy: 0.8113
Neurons: 100, Activation: linear, Batch Size: 1000, Accuracy: 0.6258
Neurons: 5000, Activation: relu, Batch Size: 10, Accuracy: 0.7152
Neurons: 5000, Activation: relu, Batch Size: 100, Accuracy: 0.6921
Neurons: 5000, Activation: relu, Batch Size: 1000, Accuracy: 0.7781
Neurons: 5000, Activation: linear, Batch Size: 10, Accuracy: 0.7351
Neurons: 5000, Activation: linear, Batch Size: 100, Accuracy: 0.7483
Neurons: 5000, Activation: linear, Batch Size: 1000, Accuracy: 0.8179
Neurons Activation Batch Size Accuracy
0 10 relu 10 0.546358
1 10 relu 100 0.337748
2 10 relu 1000 0.635762
3 10 linear 10 0.516556
4 10 linear 100 0.837748
5 10 linear 1000 0.791391
6 100 relu 10 0.824503
7 100 relu 100 0.774834
8 100 relu 1000 0.768212
9 100 linear 10 0.682119
10 100 linear 100 0.811258
11 100 linear 1000 0.625828
12 5000 relu 10 0.715232
13 5000 relu 100 0.692053
14 5000 relu 1000 0.778146
15 5000 linear 10 0.735099
16 5000 linear 100 0.748344
17 5000 linear 1000 0.817881
```

Рисунок 20. Результат выполнения

Уровень 2. 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% (accuracy).

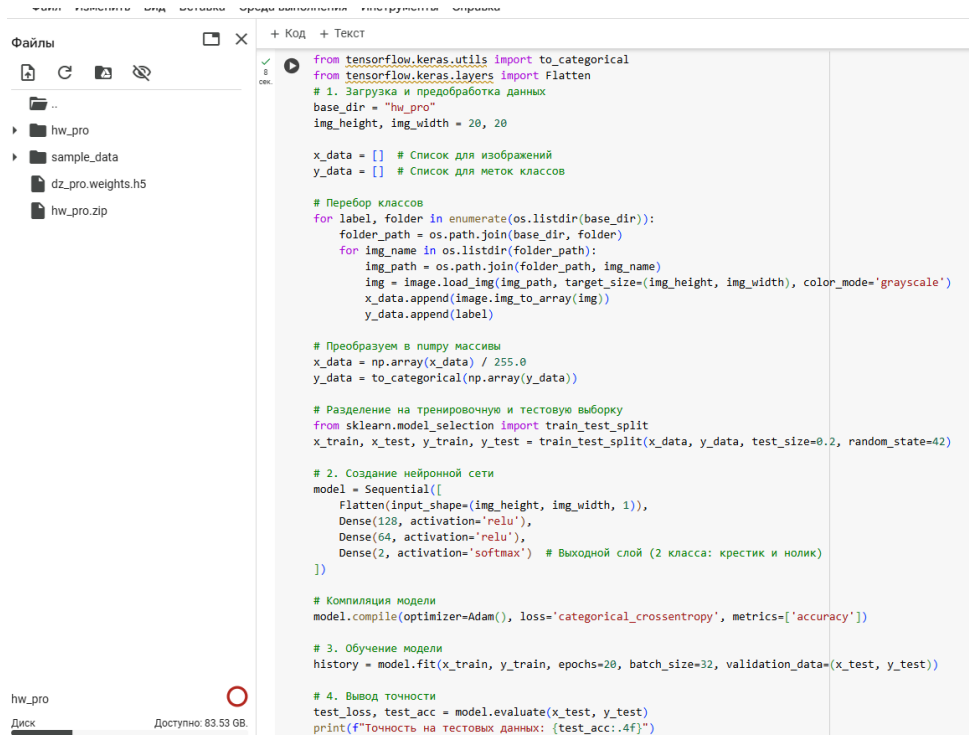


Рисунок 21. Работа с файлами и код программы

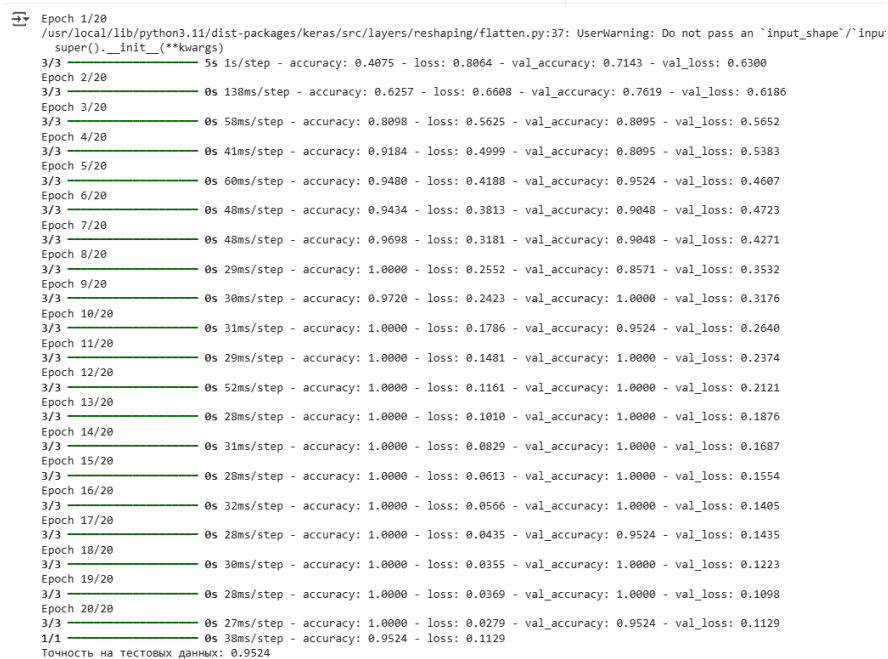
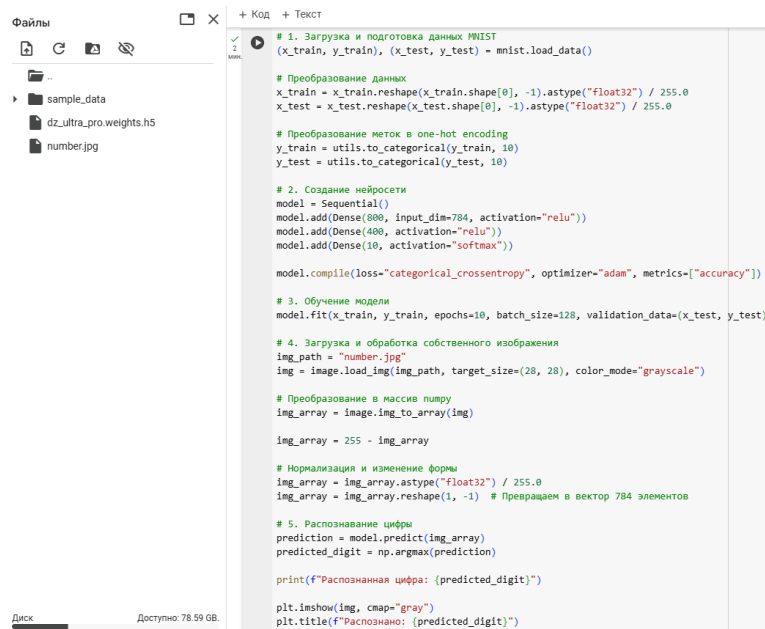


Рисунок 22. Результат выполнения

Уровень 3.



```
Файлы + Код + Текст
sample_data
dz_ultra_pro.weights.h5
number.jpg

# 1. Загрузка и подготовка данных MNIST
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Преобразование данных
x_train = x_train.reshape(x_train.shape[0], -1).astype("float32") / 255.0
x_test = x_test.reshape(x_test.shape[0], -1).astype("float32") / 255.0

# Преобразование меток в one-hot encoding
y_train = utils.to_categorical(y_train, 10)
y_test = utils.to_categorical(y_test, 10)

# 2. Создание нейросети
model = Sequential()
model.add(Dense(800, input_dim=784, activation="relu"))
model.add(Dense(400, activation="relu"))
model.add(Dense(10, activation="softmax"))

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

# 3. Обучение модели
model.fit(x_train, y_train, epochs=10, batch_size=128, validation_data=(x_test, y_test))

# 4. Загрузка и обработка собственного изображения
img_path = "number.jpg"
img = image.load_img(img_path, target_size=(28, 28), color_mode="grayscale")

# Преобразование в массив numpy
img_array = image.img_to_array(img)

img_array = 255 - img_array

# Нормализация и изменение формы
img_array = img_array.astype("float32") / 255.0
img_array = img_array.reshape(1, -1) # Превращаем в вектор 784 элементов

# 5. Распознавание цифры
prediction = model.predict(img_array)
predicted_digit = np.argmax(prediction)

print(f"Распознанная цифра: {predicted_digit}")

plt.imshow(img, cmap="gray")
plt.title(f"Распознано: {predicted_digit}")
```

Рисунок 23. Работа с файлами и код программы



Рисунок 24. Фото для первой попытки

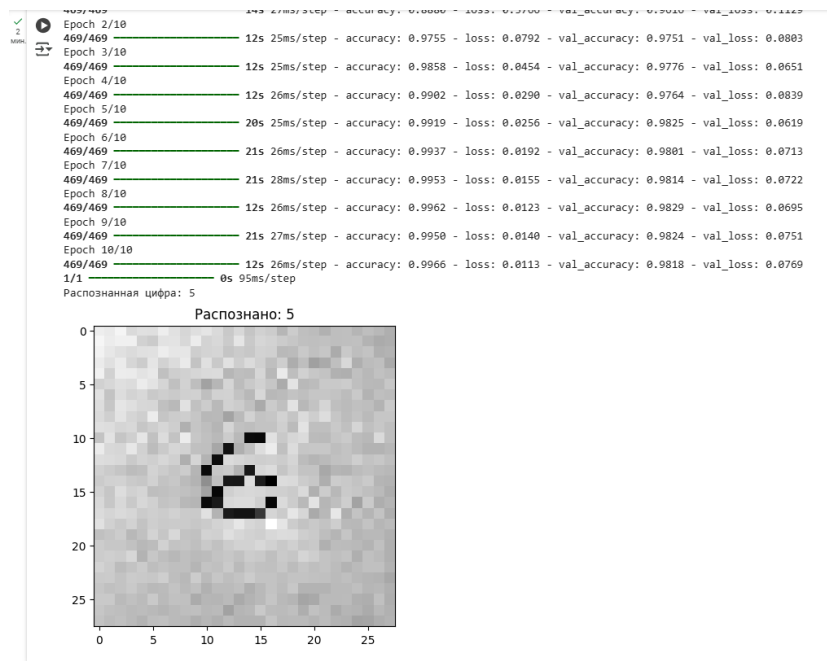


Рисунок 25. Результат

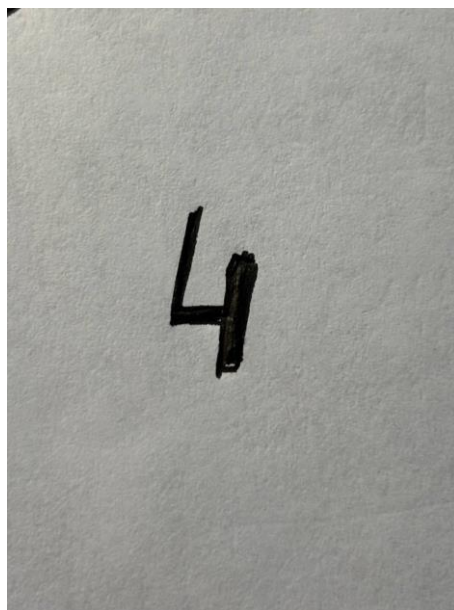


Рисунок 26. Фото для второй попытки



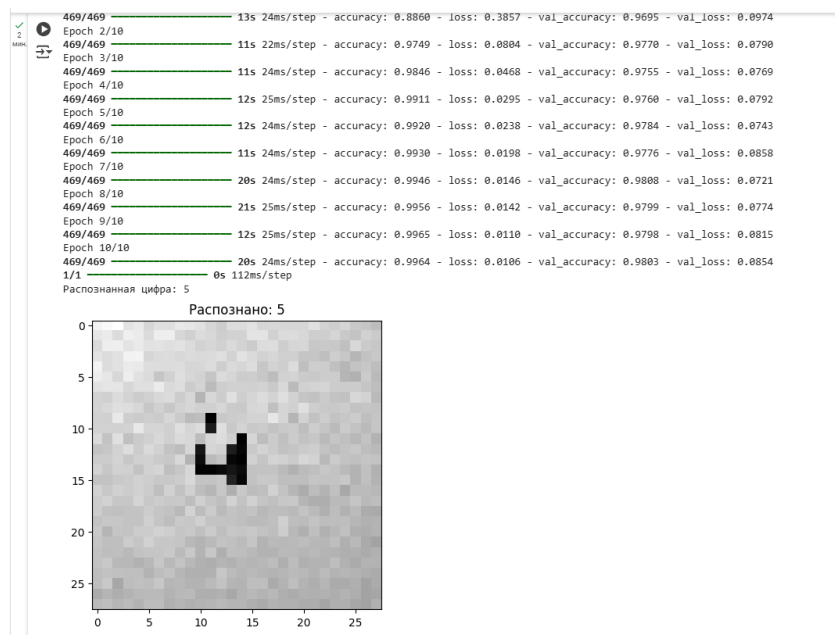


Рисунок 27. Результат

Домашняя работа позволила углубиться в экспериментирование с нейросетями и проанализировать влияние различных параметров на их точность.

1) В первом уровне выполнена серия экспериментов по распознаванию геометрических фигур. Изменение количества нейронов, активационной функции и `batch_size` показало, что гиперпараметры оказывают значительное влияние на точность модели.

2) Во втором уровне была создана и обучена нейросеть для распознавания крестиков и ноликов, где удалось достичь точности выше 95%, что говорит о хорошем качестве модели.

3) В третьем уровне проведен анализ изображений, на основе которого построена нейросеть. Были сделаны две попытки, в результате которых можно было заметить ошибки в первой попытке и исправления во второй.

В целом, домашняя работа позволила практически применить изученные методы, увидеть эффект от изменения гиперпараметров, а также разобраться в создании и обучении нейронных сетей для реальных задач.