

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 №2
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: Обучающая, проверочная и тестовая выборки. Переобучение НС.

Цель: изучить способы разделения выборки на обучающую, проверочную и тестовую, изучить слои «Dropout» и «BatchNormalization» и их влияние на явление переобучения, изучить способы работы с параметрами загружаемых для обучения изображениями.

Ход работы:

Выполнение практических заданий:

Задание 1. Загрузка изображений.

Для начала загрузим датасет с изображениями.

```
[ ] # Импорт функции загрузки файлов
import gdown
# gdown.download('https://ia601509.us.archive.org/view_archive.php?archive=/2/items/CAT_DATASET/CAT_DATASET_01.zip&file=CAT_00%2F00000001_000.jpg', 'cat.jpg', quiet = True)
gdown.download('https://storage.yandexcloud.net/aiueducation/Content/knowledge/cat.jpg', 'cat.jpg', quiet = True)

path = '/content/cat.jpg'
```

Рисунок 1. Импорт функции загрузки файлов

Далее рассмотрим первый способ работы с изображениями, для этого необходимо сначала подключить необходимые библиотеки.

```
[ ] # Метод обработки изображений библиотеки Keras
from tensorflow.keras.preprocessing import image
import matplotlib.pyplot as plt
```

Рисунок 2. Подключение библиотек

После загрузки библиотек, можем выбрать любое изображение из загруженного ранее датасета.

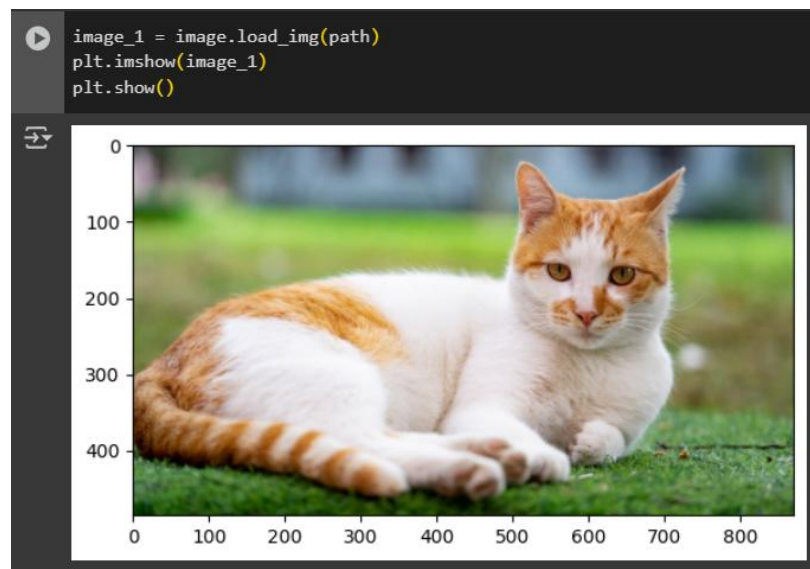


Рисунок 3. Получение изображения из датасета

Далее можем узнать размер и тип изображения, для этого выполнены следующие команды:

```
[ ] image_1.size
(870, 486)

[ ] type(image_1)
PIL.JpegImagePlugin.JpegImageFile
def __init__(fp: StrOrBytesPath | IO[bytes], filename: str | bytes | None=None) -> None
Base class for image file format handlers.
```

Рисунок 4. Определение размера изображения

Так же можно поменять цвет картинки, используя color_mode:

```
[ ] image_2 = image.load_img(path, target_size=(300, 300), color_mode = 'grayscale')
plt.imshow(image_2)
plt.show()
```



Рисунок 5. Изменение цвета картинки

Получить картинку из датасета, так же можно используя преобразование в NumPy-массив.

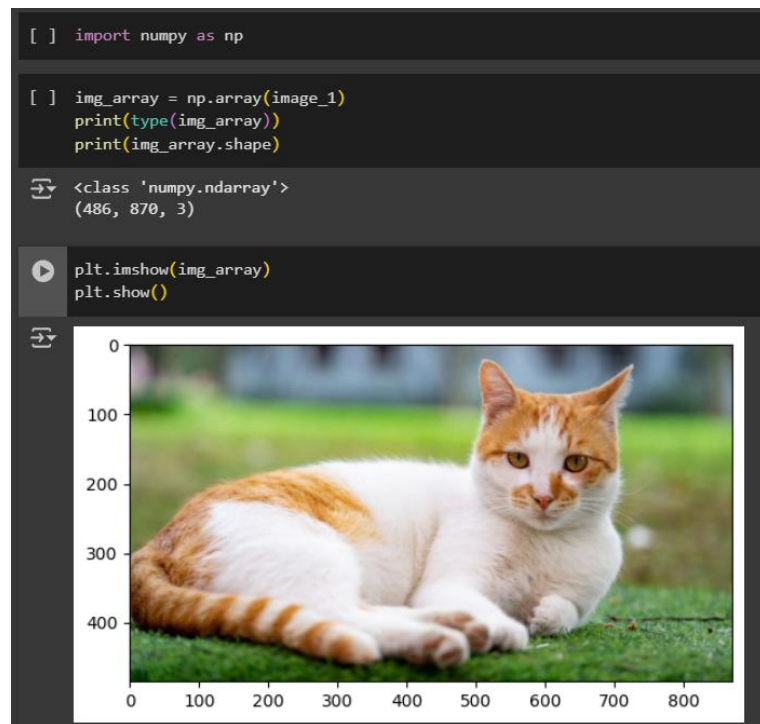


Рисунок 6. Преобразование в NumPy-массив

После чего также можем узнать размер и тип изображения, для этого выполнены следующие команды.

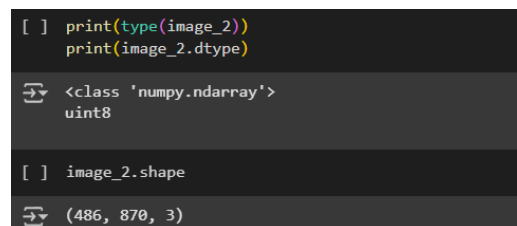


Рисунок 7. Информация о размере изображения

Далее рассмотрим второй способ, для этого подгрузим библиотеки и выполним загрузку тестового изображения и загрузим изображение в ноут и посмотрим размер изображения.

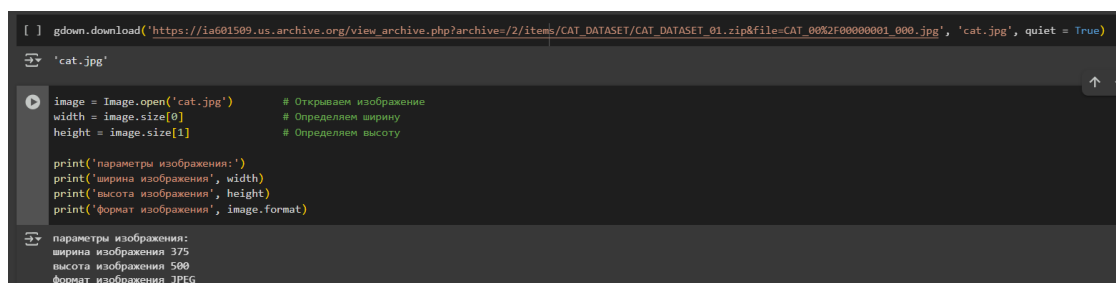


Рисунок 8. Размеры изображения

Далее посмотрим само изображение:

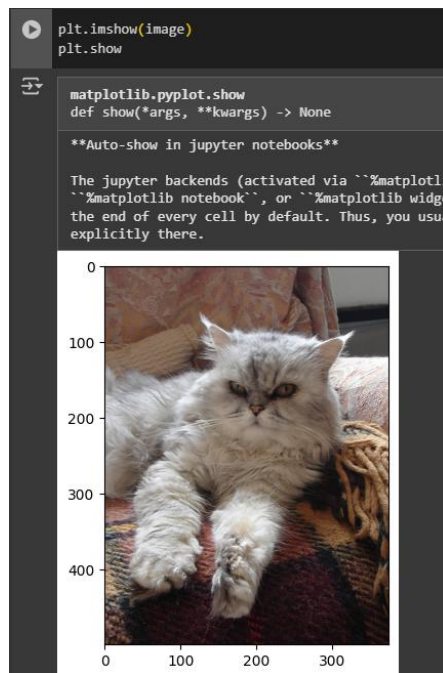


Рисунок 9. Полученное изображение

Далее изменим размер изображения:

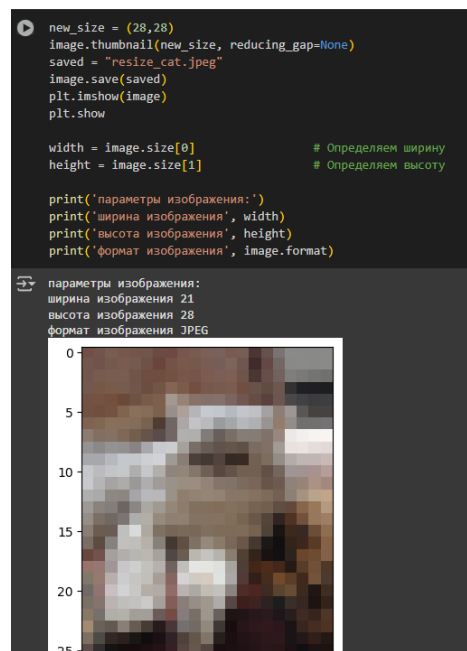


Рисунок 10. Изменение размера изображения

Далее выполним преобразование изображения в Grey.

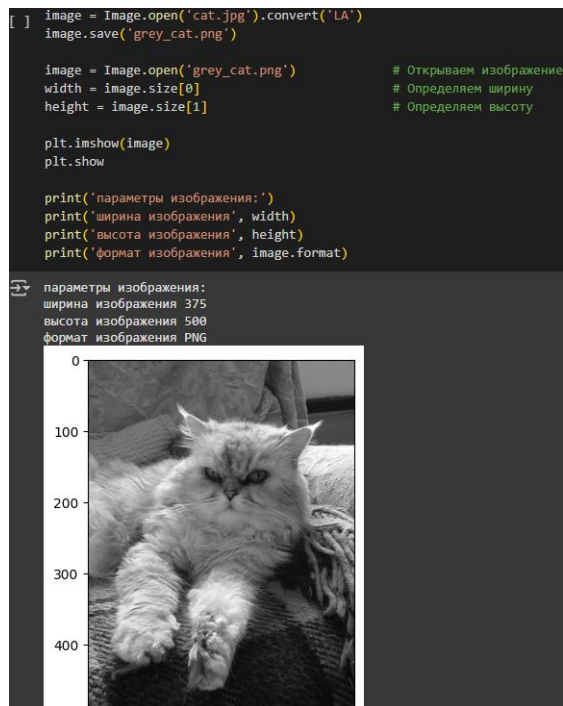


Рисунок 11. Преобразование в Grey

После преобразуем изображение в массив:

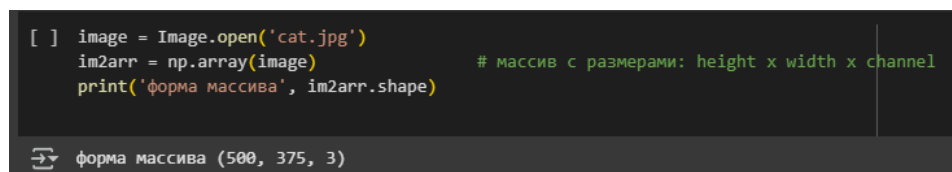


Рисунок 12. Добавление изображения в массив

Далее выполним обратное действие, то есть преобразуем массив в изображение:



Рисунок 13. Преобразование из массива в изображение

Задание 2. Слой BatchNormalization.

Рассмотрим пример работы с BatchNormalization. Для начала выполним загрузку необходимых библиотек.

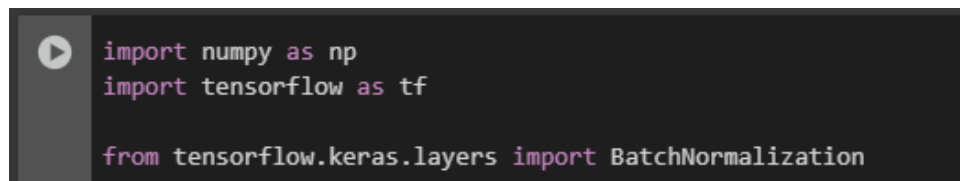


Рисунок 14. Загрузка библиотек

После выполним инициализацию входного тензора и инициализацию слоя, после инициализируем веса и вытащим веса, после чего отобразим веса:

```
# Инициализация входного тензора
x = np.ones((5, 2, 3,))

# Инициализация слоя
layer = BatchNormalization(epsilon=1e-8,
                           gamma_initializer='ones', # По умолчанию
                           beta_initializer='zeros', # По умолчанию
                           moving_mean_initializer='zeros', # По умолчанию
                           moving_variance_initializer='ones', # По умолчанию
                           )

# Инициализируем веса
layer(x)

# Выводим веса
w = np.array(layer.get_weights())

print('Входной тензор:')
print(x)

# Отобразим веса
print('Все веса:\n', str(w))
print('Форма:', w.shape)
```

Входной тензор:
[[[1. 1. 1.]
 [1. 1. 1.]

 [1. 1. 1.]
 [1. 1. 1.]

 [1. 1. 1.]
 [1. 1. 1.]

 [1. 1. 1.]
 [1. 1. 1.]]]]
Все веса:
[[1. 1. 1.]
 [0. 0. 0.]
 [0. 0. 0.]
 [1. 1. 1.]]
Форма: (4, 3)

Рисунок 15. Отображение весов

Далее выполним вывод режима теста и режима тренировки.

```
[ ] print('Выход слоя (режим теста):')
    print(layer(x)) # training=False

print('Выход слоя (режим тренировки):')
print(layer(x, training=True))
```

Выход слоя (режим теста):
tf.Tensor(
[[[1. 1. 1.]
 [1. 1. 1.]

 [1. 1. 1.]
 [1. 1. 1.]

 [1. 1. 1.]
 [1. 1. 1.]

 [1. 1. 1.]
 [1. 1. 1.]]]], shape=(5, 2, 3), dtype=float32)
Выход слоя (режим тренировки):
tf.Tensor(
[[[0. 0. 0.]
 [0. 0. 0.]

 [0. 0. 0.]
 [0. 0. 0.]

 [0. 0. 0.]
 [0. 0. 0.]

 [0. 0. 0.]
 [0. 0. 0.]]]], shape=(5, 2, 3), dtype=float32)

Рисунок 16. Режим теста и режим тренировки

Внутренние параметры `moving_mean` и `moving_variance` меняются при каждом вызове в режиме тренировки. Поэтому меняется выход сети в режиме теста:


```
print('Выход слоя (режим теста):')
print(layer(x)) # training=False
```

Выход слоя (режим теста):
tf.Tensor(
[[[0.9949874 0.9949874 0.9949874]
[0.9949874 0.9949874 0.9949874]]

[[0.9949874 0.9949874 0.9949874]
[0.9949874 0.9949874 0.9949874]]

[[0.9949874 0.9949874 0.9949874]
[0.9949874 0.9949874 0.9949874]]

[[0.9949874 0.9949874 0.9949874]
[0.9949874 0.9949874 0.9949874]]

[[0.9949874 0.9949874 0.9949874]
[0.9949874 0.9949874 0.9949874]]], shape=(5, 2, 3), dtype=float32)

Рисунок 17. Вывод слоя (режим теста)

В форме матрицы весов первое измерение всегда равно 4, так как у слоя всего 4 типа параметров (кроме тех случаев, когда мы выкидываем какой-либо из них, например при `scale=False`). Второе измерение – 3 – это размер пространства по оси `axis`. У каждого тензора это, конечно, свое число. По умолчанию (`axis = -1`) нормализация происходит перпендекулярно последней оси. То есть, если представить, что тензор `x` – изображение, а его последняя ось соответствует каналам, то слой батч-нормализации нормализует каждый канал независимо друг от друга. Выполним вывод параметров.

```
print('\nПараметры:')
print('w[0] =', w[0], '- gamma')
print('w[1] =', w[1], '- beta')
print('w[2] =', w[2], '- moving_mean')
print('w[3] =', w[3], '- moving_variance')
```

Параметры:
w[0] = [1. 1. 1.] - gamma
w[1] = [0. 0. 0.] - beta
w[2] = [0. 0. 0.] - moving_mean
w[3] = [1. 1. 1.] - moving_variance

Рисунок 18. Вывод параметров

После выполним изменение конкретного веса. Для этого поменяем значение веса `beta` на канале с индексом 1 на 10. Так как значение `beta` прибавляется в самой последней формуле при вычислении выхода слоя, то все значения канала 1 меняются на 10.

```
[ ] new_w = w.copy()

# Изменение значения beta, соответствующего каналу 1
new_w[1][1] = 10

# Установка новых весов
layer.set_weights(new_w)

# Проверка результата
new_w = np.array(layer.get_weights())

print('Новые веса:')
print(new_w)

print('Выход слоя:')
print(layer(x, training=True))

Новые веса:
[[ 1.  1.  1.]
 [ 0. 10.  0.]
 [ 0.  0.  0.]
 [ 1.  1.  1.]]
Выход слоя:
tf.Tensor(
[[[ 0. 10.  0.]
 [ 0. 10.  0.]]

[[ 0. 10.  0.]
 [ 0. 10.  0.]]

[[ 0. 10.  0.]
 [ 0. 10.  0.]]

[[ 0. 10.  0.]
 [ 0. 10.  0.]]

[[ 0. 10.  0.]
 [ 0. 10.  0.]]], shape=(5, 2, 3), dtype=float32)

[ ] print('epsilon:', layer.epsilon)

epsilon: 1e-08
```

Рисунок 19. Изменение конкретного веса

Задание 3. Слой Dropout.

Для начала выполним загрузку необходимых библиотек. Если dropout без режима обучения, то тензор проходит через слой без изменений.

```
[1] import tensorflow as tf

import numpy as np
from tensorflow.keras.layers import Dropout

# Входной тензор
X = np.ones((5, 2))

# Создание и вызов слоя в тестовом режиме
Dropout(0.2)(X)

<tf.Tensor: shape=(5, 2), dtype=float32, numpy=
array([[1., 1.],
       [1., 1.],
       [1., 1.],
       [1., 1.],
       [1., 1.]], dtype=float32)>
```

Рисунок 20. Загрузка библиотек

В режиме обучения dropout активируется. Выполним создание и вызов слоя в тренировочном режиме.

```
# Задание состояния
tf.random.set_seed(3)

# Создание и вызов слоя в тренировочном режиме
Dropout(0.2)(X, training=True)

<tf.Tensor: shape=(5, 2), dtype=float32, numpy=
array([[1.25, 0.  ],
       [1.25, 1.25],
       [1.25, 1.25],
       [1.25, 1.25],
       [1.25, 1.25]], dtype=float32)>
```

Рисунок 21. Тренировочный режим

Далее рассмотрим пример использования Dropout. Для начала выполним загрузку необходимых библиотек и выполним инициализацию входной матрицы.

```
[5] import numpy as np
    from keras.layers import Dropout

[6] # Инициализация входной матрицы
    X = np.ones((5, 2))
    X
array([[1., 1.],
       [1., 1.],
       [1., 1.],
       [1., 1.],
       [1., 1.]])
```

Рисунок 22. Инициализация входной матрицы

Далее выполним создание слоя Dropout с параметром 0.2. И вызовем слой с аргументом созданной ранее матрицы. Настроим слой на обучение, так как иначе тензор не изменится.

```
[7] # Создание слоя Dropout с параметром 0.2
    dropout_layer = Dropout(rate=0.5)

[8] dropout_layer(X, training=True)
<tf.Tensor: shape=(5, 2), dtype=float32, numpy=
array([[0., 0.],
       [0., 0.],
       [2., 2.],
       [2., 0.],
       [2., 2.]], dtype=float32)>
```

Рисунок 23. Создание слоя Dropout

Задание 4. Обучающая, проверочная и тестовая выборки. Переобучение НС.

Для начала выполним загрузку необходимых библиотек:

```
# Библиотека работы с массивами
import numpy as np

# Библиотека для работы с таблицами
import pandas as pd

# Последовательная модель НС
from tensorflow.keras.models import Sequential

# Основные слои
from tensorflow.keras.layers import Dense, Dropout, BatchNormalization

# Слой задания активационной функции <----- !!!
from tensorflow.keras.layers import Activation

# Утилиты предобработки данных
from tensorflow.keras import utils

# Оптимизаторы
from tensorflow.keras.optimizers import Adam

# Разделение на обучающую и проверочную/тестовую выборку
from sklearn.model_selection import train_test_split

# Рисование графиков
import matplotlib.pyplot as plt

%matplotlib inline
```

Рисунок 24. Загрузка библиотек

Далее импортируем библиотеку gdown и загрузим файл sonar.csv из репозитория при помощи метода .download(). После прочитаем и запишем его в переменную df, указывая, что в таблице нет заголовка (header=None), выведем размерность датафрейма при помощи метода .shape:

```
[ ] import gdown

gdown.download('https://storage.yandexcloud.net/aiueducation/Content/base/14/sonar.csv', None, quiet=True)

'sonar.csv'

df = pd.read_csv('sonar.csv', header=None)

print(df.shape)

(208, 61)
```

Рисунок 25. Загрузка файла sonar.csv

После выведем первые пять строчек таблицы, чтобы понимать, с какими данными придется работать:

```
[ ] df.head()

   0    1    2    3    4    5    6    7    8    9  ...  51  52  53  54  55  56  57  58  59  60
0  0.0200  0.0371  0.0428  0.0207  0.0954  0.0986  0.1539  0.1601  0.3109  0.2111  ...  0.0027  0.0065  0.0159  0.0072  0.0167  0.0180  0.0084  0.0090  0.0032  R
1  0.0453  0.0523  0.0843  0.0689  0.1183  0.2583  0.2156  0.3481  0.3337  0.2872  ...  0.0084  0.0089  0.0048  0.0094  0.0191  0.0140  0.0049  0.0052  0.0044  R
2  0.0282  0.0582  0.1099  0.1083  0.0974  0.2280  0.2431  0.3771  0.5598  0.6194  ...  0.0232  0.0166  0.0095  0.0100  0.0244  0.0316  0.0164  0.0095  0.0078  R
3  0.0100  0.0171  0.0623  0.0205  0.0205  0.0368  0.1098  0.1276  0.0598  0.1264  ...  0.0121  0.0036  0.0150  0.0085  0.0073  0.0050  0.0044  0.0040  0.0117  R
4  0.0782  0.0696  0.0481  0.0394  0.0590  0.0649  0.1209  0.2467  0.3564  0.4459  ...  0.0031  0.0054  0.0105  0.0110  0.0015  0.0072  0.0048  0.0107  0.0094  R

5 rows x 61 columns

[ ] dataset = df.replace('R', 1).replace('M', 0).astype(float).to_numpy()

<ipython-input-5-824386c2ed0b>:1: FutureWarning: Downcasting behavior in 'replace' is deprecated and will be removed in a future version. To retain the old behavior, explicitly call
dataset = df.replace('R', 1).replace('M', 0).astype(float).to_numpy()
```

Рисунок 26. Вывод первых пяти строчек таблицы

Затем в x_data добавим параметры объекта, в y_data – класс объекта (правильные ответы). После проверим форму данных и содержимое y_data. В выборке всего 208 примеров.

```
[ ] x_data = dataset[:, :60]
y_data = dataset[:, 60]

[ ] print('Размерность набора параметров объектов', x_data.shape)
print('Размерность набора меток класса', y_data.shape)
print()
print('Содержание y_data:', y_data)

Размерность набора параметров объектов (208, 60)
Размерность набора меток класса (208,)

Содержание y_data: [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

Рисунок 27. Содержимое y_data

Далее выполним создание и обучение тестовой выборки. Воспользуемся функцией train_test_split.

```
[ ] x_train, x_test, y_train, y_test = train_test_split(x_data, # набор параметров
                                                    y_data, # набор меток классов
                                                    test_size=0.2, # процент в тестовую
                                                    shuffle=True, # перемешивание
                                                    random_state=3) # воспроизводимость

# Выведем размерность полученных выборок

print('Обучающая выборка параметров', x_train.shape)
print('Обучающая выборка меток классов', y_train.shape)
print()
print('Тестовая выборка параметров', x_test.shape)
print('Тестовая выборка меток классов', y_test.shape)
```

Обучающая выборка параметров (166, 60)
Обучающая выборка меток классов (166,)

Тестовая выборка параметров (42, 60)
Тестовая выборка меток классов (42,)

Рисунок 28. Использование функции train_test_split

Далее выполним обучение нейросети. Скомпилируем НС и укажем binary_crossentropy в качестве функции ошибки, т.к. решается задача бинарной классификации:

```
def create_model():
    # Создание модели
    model = Sequential()

    # Добавление слоев
    model.add(Dense(60, input_dim=x_train.shape[1], activation='relu'))
    model.add(Dense(30))
    model.add(Activation('relu'))
    model.add(Dense(1, activation='sigmoid'))

    # Компиляция и возврат модели
    model.compile(loss='binary_crossentropy',
                  optimizer=Adam(learning_rate=0.001),
                  metrics=['accuracy'])

    return model

[ ] # Создание необученной модели при помощи функции create_model()
model = create_model()

# Обучение модели
history = model.fit(x_train, # Обучающая выборка параметров
                    y_train, # Обучающая выборка меток класса
                    batch_size=8, # Размер батча (пакета)
                    epochs=100, # Количество эпох обучения
                    verbose=1) # Отображение хода обучения
```

Рисунок 29. Обучение нейросети

Процесс выполнения обучения:

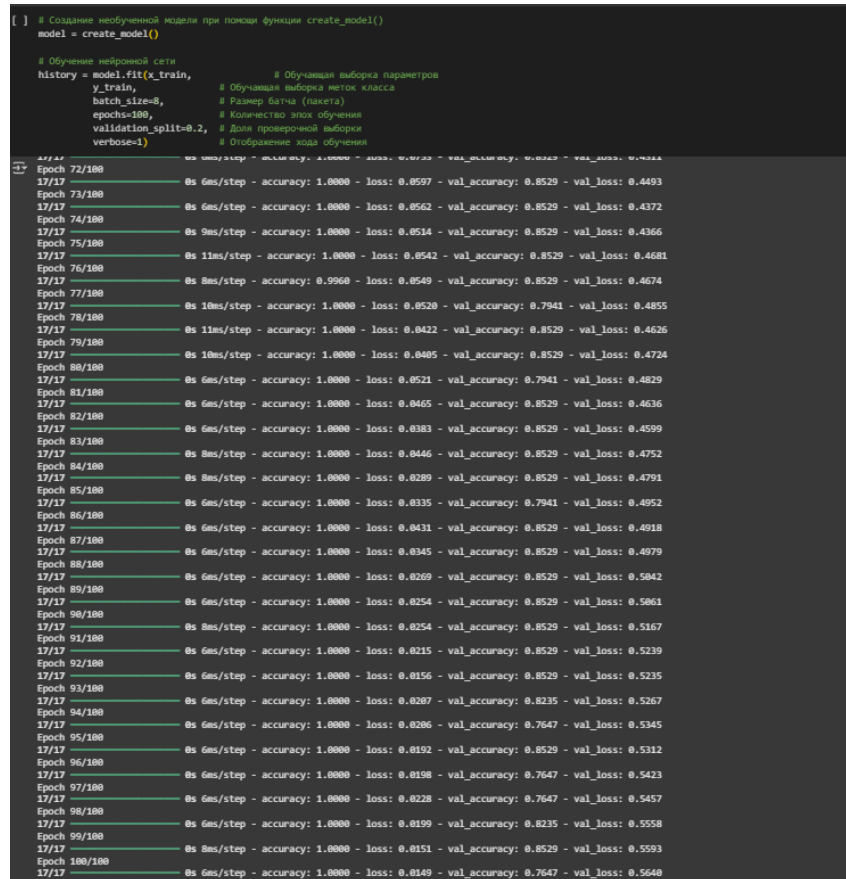
```
Epoch 83/100 — 0s 3ms/step - accuracy: 1.0000 - loss: 0.0314
21/21
Epoch 84/100 — 0s 3ms/step - accuracy: 0.9945 - loss: 0.0346
21/21
Epoch 85/100 — 0s 3ms/step - accuracy: 1.0000 - loss: 0.0277
21/21
Epoch 86/100 — 0s 3ms/step - accuracy: 1.0000 - loss: 0.0261
21/21
Epoch 87/100 — 0s 3ms/step - accuracy: 1.0000 - loss: 0.0250
21/21
Epoch 88/100 — 0s 3ms/step - accuracy: 1.0000 - loss: 0.0293
21/21
Epoch 89/100 — 0s 3ms/step - accuracy: 1.0000 - loss: 0.0238
21/21
Epoch 90/100 — 0s 3ms/step - accuracy: 1.0000 - loss: 0.0215
21/21
Epoch 91/100 — 0s 3ms/step - accuracy: 1.0000 - loss: 0.0312
21/21
Epoch 92/100 — 0s 3ms/step - accuracy: 1.0000 - loss: 0.0189
21/21
Epoch 93/100 — 0s 3ms/step - accuracy: 1.0000 - loss: 0.0244
21/21
Epoch 94/100 — 0s 3ms/step - accuracy: 1.0000 - loss: 0.0218
21/21
Epoch 95/100 — 0s 4ms/step - accuracy: 1.0000 - loss: 0.0220
21/21
Epoch 96/100 — 0s 3ms/step - accuracy: 1.0000 - loss: 0.0200
21/21
Epoch 97/100 — 0s 4ms/step - accuracy: 1.0000 - loss: 0.0211
21/21
Epoch 98/100 — 0s 3ms/step - accuracy: 1.0000 - loss: 0.0195
21/21
Epoch 99/100 — 0s 3ms/step - accuracy: 1.0000 - loss: 0.0194
21/21
Epoch 100/100 — 0s 3ms/step - accuracy: 1.0000 - loss: 0.0165
21/21
```

Рисунок 30. Выполнение обучения

Далее выполним оценку качества обучения. На тренировочной выборке НС достигает точности в 100%. Приятная глазу цифра, но, к сожалению, не отражающая истинное положение дел. Чтобы перепроверить нейросеть, а так же выяснить научилась она выявлять закономерности, или же просто заучила данные - создадим проверочную (валидационную) выборку.

```
[ ] # Создание необученной модели при помощи функции create_model()
model = create_model()

# Обучение нейронной сети
history = model.fit(x_train, y_train,
                    batch_size=8, epochs=100, validation_split=0.2, verbose=1)
# Обучающая выборка параметров
# Обучающая выборка меток класса
# Размер батча (пакета)
# Количество эпох обучения
# Доля проверочной выборки
# Отображение хода обучения
```

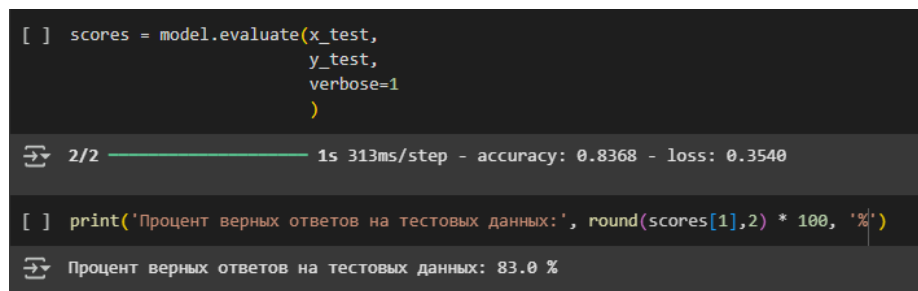


```
Epoch 72/100 1s 6ms/step - accuracy: 1.0000 - loss: 0.0597 - val_accuracy: 0.8529 - val_loss: 0.4493
Epoch 73/100 1s 6ms/step - accuracy: 1.0000 - loss: 0.0562 - val_accuracy: 0.8529 - val_loss: 0.4372
Epoch 74/100 1s 9ms/step - accuracy: 1.0000 - loss: 0.0514 - val_accuracy: 0.8529 - val_loss: 0.4366
Epoch 75/100 1s 11ms/step - accuracy: 1.0000 - loss: 0.0542 - val_accuracy: 0.8529 - val_loss: 0.4681
Epoch 76/100 1s 8ms/step - accuracy: 0.9960 - loss: 0.0549 - val_accuracy: 0.8529 - val_loss: 0.4674
Epoch 77/100 1s 10ms/step - accuracy: 1.0000 - loss: 0.0520 - val_accuracy: 0.7941 - val_loss: 0.4855
Epoch 78/100 1s 11ms/step - accuracy: 1.0000 - loss: 0.0422 - val_accuracy: 0.8529 - val_loss: 0.4626
Epoch 79/100 1s 10ms/step - accuracy: 1.0000 - loss: 0.0405 - val_accuracy: 0.8529 - val_loss: 0.4724
Epoch 80/100 1s 6ms/step - accuracy: 1.0000 - loss: 0.0521 - val_accuracy: 0.7941 - val_loss: 0.4829
Epoch 81/100 1s 6ms/step - accuracy: 1.0000 - loss: 0.0465 - val_accuracy: 0.8529 - val_loss: 0.4636
Epoch 82/100 1s 6ms/step - accuracy: 1.0000 - loss: 0.0383 - val_accuracy: 0.8529 - val_loss: 0.4599
Epoch 83/100 1s 8ms/step - accuracy: 1.0000 - loss: 0.0446 - val_accuracy: 0.8529 - val_loss: 0.4752
Epoch 84/100 1s 8ms/step - accuracy: 1.0000 - loss: 0.0289 - val_accuracy: 0.8529 - val_loss: 0.4791
Epoch 85/100 1s 6ms/step - accuracy: 1.0000 - loss: 0.0335 - val_accuracy: 0.7941 - val_loss: 0.4952
Epoch 86/100 1s 6ms/step - accuracy: 1.0000 - loss: 0.0431 - val_accuracy: 0.8529 - val_loss: 0.4918
Epoch 87/100 1s 6ms/step - accuracy: 1.0000 - loss: 0.0345 - val_accuracy: 0.8529 - val_loss: 0.4979
Epoch 88/100 1s 6ms/step - accuracy: 1.0000 - loss: 0.0269 - val_accuracy: 0.8529 - val_loss: 0.5042
Epoch 89/100 1s 6ms/step - accuracy: 1.0000 - loss: 0.0254 - val_accuracy: 0.8529 - val_loss: 0.5061
Epoch 90/100 1s 8ms/step - accuracy: 1.0000 - loss: 0.0254 - val_accuracy: 0.8529 - val_loss: 0.5167
Epoch 91/100 1s 6ms/step - accuracy: 1.0000 - loss: 0.0215 - val_accuracy: 0.8529 - val_loss: 0.5239
Epoch 92/100 1s 6ms/step - accuracy: 1.0000 - loss: 0.0156 - val_accuracy: 0.8529 - val_loss: 0.5235
Epoch 93/100 1s 6ms/step - accuracy: 1.0000 - loss: 0.0207 - val_accuracy: 0.8235 - val_loss: 0.5267
Epoch 94/100 1s 6ms/step - accuracy: 1.0000 - loss: 0.0206 - val_accuracy: 0.7647 - val_loss: 0.5345
Epoch 95/100 1s 6ms/step - accuracy: 1.0000 - loss: 0.0192 - val_accuracy: 0.8529 - val_loss: 0.5312
Epoch 96/100 1s 6ms/step - accuracy: 1.0000 - loss: 0.0198 - val_accuracy: 0.7647 - val_loss: 0.5423
Epoch 97/100 1s 6ms/step - accuracy: 1.0000 - loss: 0.0228 - val_accuracy: 0.7647 - val_loss: 0.5457
Epoch 98/100 1s 6ms/step - accuracy: 1.0000 - loss: 0.0199 - val_accuracy: 0.8235 - val_loss: 0.5558
Epoch 99/100 1s 6ms/step - accuracy: 1.0000 - loss: 0.0151 - val_accuracy: 0.8529 - val_loss: 0.5593
Epoch 100/100 1s 6ms/step - accuracy: 1.0000 - loss: 0.0149 - val_accuracy: 0.7647 - val_loss: 0.5640
```

Рисунок 31. Обучение нейронной сети

Далее применим метод `.evaluate()` к нашей модели, в качестве параметров передадим ему тестовые выборки и согласие на отображение хода вычисления. Поместим результат в переменную `scores`:

```
[ ] scores = model.evaluate(x_test, y_test, verbose=1)
```



```
2/2 1s 313ms/step - accuracy: 0.8368 - loss: 0.3540
```

```
[ ] print('Процент верных ответов на тестовых данных:', round(scores[1],2) * 100, '%')
```

```
Процент верных ответов на тестовых данных: 83.0 %
```

Рисунок 32. Использование метода `.evaluate()`

Далее выполним визуализацию качества обучения. Для этого обучим НС, при этом результаты процесса обучения запишес в переменную history:

```
# Создание модели
model = create_model()

# Обучение нейронной сети
history = model.fit(x_train,
                    y_train,
                    batch_size=8,
                    epochs=100,
                    validation_split=0.2,
                    verbose=1)
```

```
Epoch 72/100      6s 6ms/step - accuracy: 0.9983 - loss: 0.0669 - val_accuracy: 0.8529 - val_loss: 0.3917
Epoch 73/100      6s 6ms/step - accuracy: 0.9967 - loss: 0.0958 - val_accuracy: 0.8529 - val_loss: 0.3965
Epoch 74/100      6s 6ms/step - accuracy: 0.9855 - loss: 0.0984 - val_accuracy: 0.8529 - val_loss: 0.4081
Epoch 75/100      6s 6ms/step - accuracy: 0.9960 - loss: 0.0683 - val_accuracy: 0.8235 - val_loss: 0.4228
Epoch 76/100      6s 6ms/step - accuracy: 0.9937 - loss: 0.0658 - val_accuracy: 0.8235 - val_loss: 0.4211
Epoch 77/100      6s 6ms/step - accuracy: 0.9927 - loss: 0.0642 - val_accuracy: 0.8529 - val_loss: 0.4167
Epoch 78/100      6s 6ms/step - accuracy: 0.9884 - loss: 0.0754 - val_accuracy: 0.8824 - val_loss: 0.4888
Epoch 79/100      6s 6ms/step - accuracy: 0.9992 - loss: 0.0470 - val_accuracy: 0.8235 - val_loss: 0.4235
Epoch 80/100      6s 6ms/step - accuracy: 1.0000 - loss: 0.0506 - val_accuracy: 0.8235 - val_loss: 0.4288
Epoch 81/100      6s 6ms/step - accuracy: 0.9826 - loss: 0.0686 - val_accuracy: 0.8235 - val_loss: 0.4325
Epoch 82/100      6s 6ms/step - accuracy: 0.9861 - loss: 0.0604 - val_accuracy: 0.8235 - val_loss: 0.4411
Epoch 83/100      6s 6ms/step - accuracy: 1.0000 - loss: 0.0517 - val_accuracy: 0.7941 - val_loss: 0.4496
Epoch 84/100      6s 6ms/step - accuracy: 0.9884 - loss: 0.0509 - val_accuracy: 0.8529 - val_loss: 0.4309
Epoch 85/100      6s 6ms/step - accuracy: 0.9987 - loss: 0.0390 - val_accuracy: 0.8235 - val_loss: 0.4379
Epoch 86/100      6s 6ms/step - accuracy: 0.9992 - loss: 0.0341 - val_accuracy: 0.8529 - val_loss: 0.4430
Epoch 87/100      6s 6ms/step - accuracy: 1.0000 - loss: 0.0432 - val_accuracy: 0.7941 - val_loss: 0.5015
Epoch 88/100      6s 6ms/step - accuracy: 1.0000 - loss: 0.0415 - val_accuracy: 0.8529 - val_loss: 0.4588
Epoch 89/100      6s 6ms/step - accuracy: 1.0000 - loss: 0.0403 - val_accuracy: 0.8529 - val_loss: 0.4568
Epoch 90/100      6s 6ms/step - accuracy: 1.0000 - loss: 0.0532 - val_accuracy: 0.7647 - val_loss: 0.5080
Epoch 91/100      6s 6ms/step - accuracy: 0.9992 - loss: 0.0321 - val_accuracy: 0.8235 - val_loss: 0.4804
Epoch 92/100      6s 6ms/step - accuracy: 1.0000 - loss: 0.0335 - val_accuracy: 0.8235 - val_loss: 0.5518
Epoch 93/100      6s 6ms/step - accuracy: 1.0000 - loss: 0.0564 - val_accuracy: 0.8235 - val_loss: 0.4787
Epoch 94/100      6s 6ms/step - accuracy: 1.0000 - loss: 0.0362 - val_accuracy: 0.7941 - val_loss: 0.5035
Epoch 95/100      6s 6ms/step - accuracy: 1.0000 - loss: 0.0235 - val_accuracy: 0.8529 - val_loss: 0.5003
Epoch 96/100      6s 6ms/step - accuracy: 1.0000 - loss: 0.0259 - val_accuracy: 0.7941 - val_loss: 0.5034
Epoch 97/100      6s 6ms/step - accuracy: 1.0000 - loss: 0.0246 - val_accuracy: 0.8235 - val_loss: 0.5039
Epoch 98/100      6s 6ms/step - accuracy: 1.0000 - loss: 0.0286 - val_accuracy: 0.7941 - val_loss: 0.5207
Epoch 99/100      6s 6ms/step - accuracy: 1.0000 - loss: 0.0325 - val_accuracy: 0.8529 - val_loss: 0.4986
Epoch 100/100     6s 6ms/step - accuracy: 1.0000 - loss: 0.0311 - val_accuracy: 0.7941 - val_loss: 0.5234
```

Рисунок 33. Обучение нейронной сети

Далее построим график точности на протяжении всего обучения.

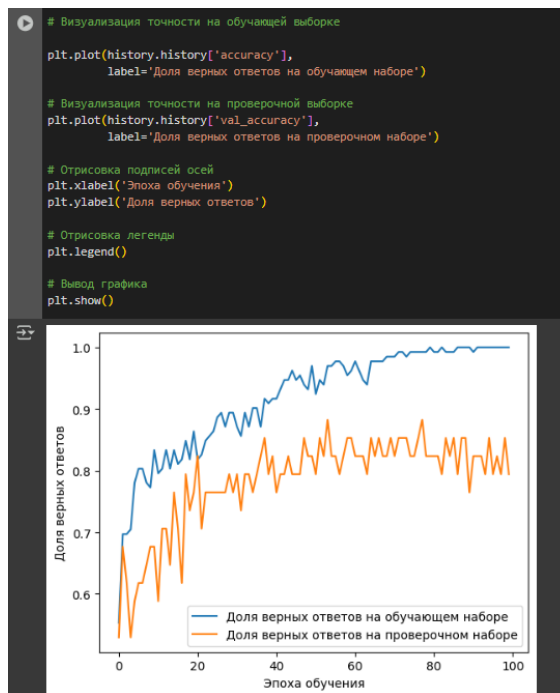


Рисунок 34. График точности на протяжении всего обучения

После выведем график ошибки:

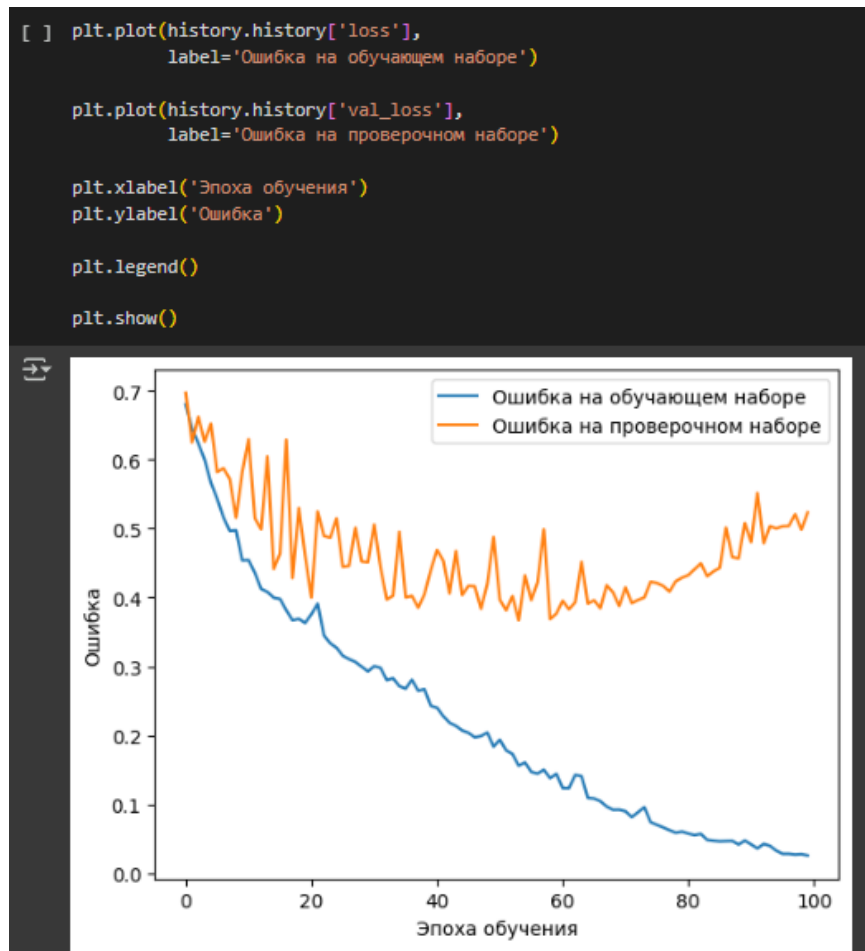


Рисунок 35. График ошибки

Выполнение индивидуальных заданий:

Задание 1. Обучающая, проверочная и тестовая выборки. Переобучение НС ДЗ Lite.

Условие: необходимо используя шаблон ноутбука для распознавания видов одежды и аксессуаров из набора `fashion_mnist`, выполнить следующие действия:

1. Создать 9 моделей нейронной сети с различными архитектурами и сравните в них значения точности на проверочной выборке (на последней эпохе) и на тестовой выборке. Необходимо использовать следующее деление: обучающая выборка - 50000 примеров, проверочная выборка - 10000 примеров, тестовая выборка - 10000 примеров.

2. Создать сравнительную таблицу в конце ноутбука.

Для начала необходимо выполнить загрузку необходимых библиотек.

```
# Последовательная модель HC
from tensorflow.keras.models import Sequential

# Основные слои
from tensorflow.keras.layers import Dense, Activation, Dropout, BatchNormalization

# Утилиты для to_categorical()
from tensorflow.keras import utils

# Алгоритмы оптимизации для обучения модели
from tensorflow.keras.optimizers import Adam, Adadelta

# Библиотека для работы с массивами
import numpy as np

# Библиотека для работы с таблицами
import pandas as pd

# Отрисовка графиков
import matplotlib.pyplot as plt

# Связь с google-дискон
from google.colab import files

# Предварительная обработка данных
from sklearn import preprocessing

# Разделение данных на выборки
from sklearn.model_selection import train_test_split

# Для загрузки датасета
from keras.datasets import fashion_mnist

from tensorflow.keras.utils import to_categorical

import tensorflow as tf

# Отрисовывать изображения в ноутбуке, а не в консоль или файл
%matplotlib inline
```

Рисунок 36. Загрузка библиотек

После чего выполним загрузку датасета и выполним вывод размерности выборок.

```
# Загрузка датасета
(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()

# Вывод размерностей выборок

print('Размер x_train:', x_train.shape)
print('Размер y_train:', y_train.shape)
print('Размер x_test:', x_test.shape)
print('Размер y_test:', y_test.shape)
```

```
Размер x_train: (60000, 28, 28)
Размер y_train: (60000,)
Размер x_test: (10000, 28, 28)
Размер y_test: (10000,)
```

Рисунок 37. Загрузка датасета

Далее выполним описание базы. База: одежда, обувь и аксессуары

- Датасет состоит из набора изображений одежды, обуви, аксессуаров и их классов.
- Изображения одного вида хранятся в numpy-массиве (28, 28) - x_train, x_test.
- База содержит 10 классов: (Футболка, Брюки, Пуловер, Платье, Пальто, Сандалии/Босоножки, Рубашка, Кроссовки, Сумочка, Ботильоны) - y_train, y_test.
- Примеров: train - 60000, test - 10000.



Рисунок 38. Отображение изображений

Далее выполним нормализацию, распределение и преобразование в векторы.

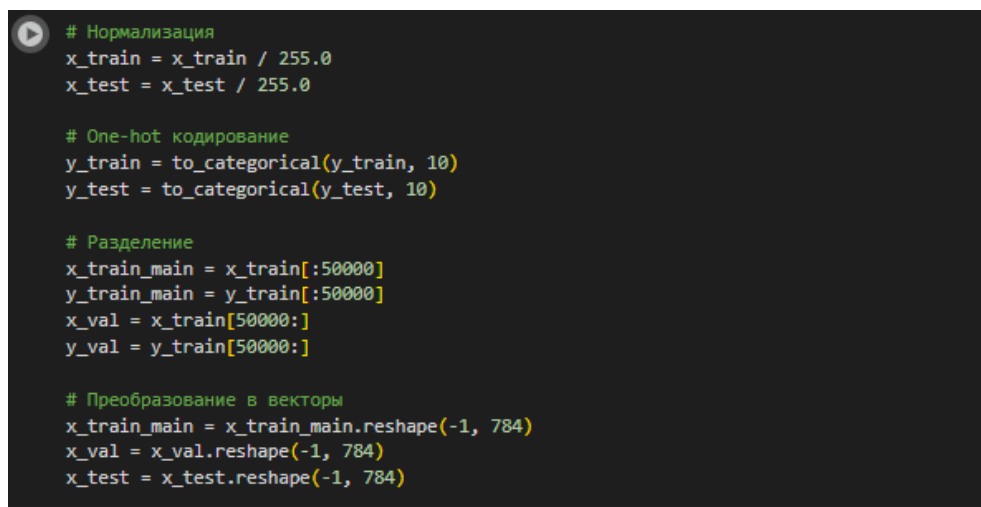


Рисунок 39. Нормализация и распределение

Далее выполним описание девяти архитектур.

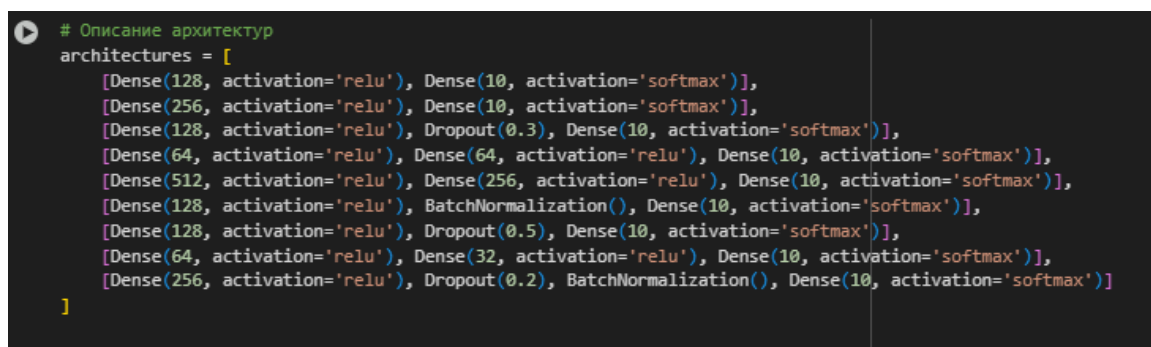


Рисунок 40. Описание архитектур

Далее выполним обучение всех архитектур нейросетей.

```
[ ] # Обучение и оценка
results = []

for i, layers in enumerate(architectures):
    print(f"Обучение модели {i+1}")
    model = Sequential()
    model.add(tf.keras.Input(shape=(784,)))
    for layer in layers:
        model.add(layer)
    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

    history = model.fit(x_train_main, y_train_main, epochs=10, batch_size=128,
                        validation_data=(x_val, y_val), verbose=0)

    val_acc = history.history['val_accuracy'][-1]
    test_loss, test_acc = model.evaluate(x_test, y_test, verbose=0)

    results.append({
        'Модель': f'Модель {i+1}',
        'Точность на проверочной выборке': round(val_acc, 4),
        'Точность на тестовой выборке': round(test_acc, 4)
    })
```

Обучение модели 1
Обучение модели 2
Обучение модели 3
Обучение модели 4
Обучение модели 5
Обучение модели 6
Обучение модели 7
Обучение модели 8
Обучение модели 9

Рисунок 41. Обучение нейросетей

Далее выполним вывод результатов всех таблиц. И сравним полученные результаты.

```
[ ] # Таблица результатов
df = pd.DataFrame(results)
print(df)
```

	Модель	Точность на проверочной выборке	Точность на тестовой выборке
0	Модель 1	0.8829	0.8749
1	Модель 2	0.8839	0.8790
2	Модель 3	0.8849	0.8748
3	Модель 4	0.8747	0.8729
4	Модель 5	0.8869	0.8829
5	Модель 6	0.8726	0.8644
6	Модель 7	0.8822	0.8711
7	Модель 8	0.8784	0.8691
8	Модель 9	0.8791	0.8738

Рисунок 42. Сравнение моделей

Модель 5 показала наилучший результат на тестовой выборке (0.8829), что говорит о её хорошей обобщающей способности. Это делает её наиболее предпочтительной среди всех протестированных моделей.

У большинства моделей, кроме Модели 3, точность на тестовой выборке немного ниже, чем на проверочной. Это ожидаемо, так как модель может переобучаться на проверочных данных. Однако разница незначительна, что указывает на устойчивость моделей.

Модель 6 имеет самые низкие показатели как на проверочной (0.8726), так и на тестовой выборке (0.8644). Её использование не рекомендуется.

Большинство моделей демонстрируют высокую точность (выше 0.87), что подтверждает их пригодность для решения задачи. Однако выбор конкретной модели должен основываться на тестовых данных, чтобы минимизировать риск переобучения.

Задание 2. Обучающая, проверочная и тестовая выборки. Переобучение НС ДЗ Pro.

Условие: используя модуль `datasets` библиотеки `sklearn`, необходимо загрузить базу вин (`.load_wine()`). Используя шаблон ноутбука, необходимо выполнить загрузку, подготовку и предобработку данных. Обязательное условие: разделение данных на три выборки осуществляется по шаблону (изменять параметры подготовки данных запрещается)!

Необходимо добиться максимальной точности классификации на тестовой выборке выше 94%.

С помощью метода `.summary()` необходимо зафиксировать количество параметров созданной вами нейронной сети.

Начнем с загрузки необходимых библиотек для работы:

```
[ ] from tensorflow.keras.models import Sequential
    from tensorflow.keras.layers import Dense, Activation, Dropout, BatchNormalization
    from tensorflow.keras import utils
    from tensorflow.keras.optimizers import Adam
    import numpy as np
    import matplotlib.pyplot as plt
    from sklearn.model_selection import train_test_split
    from sklearn.datasets import load_wine

    # Отрисовка изображений в ноутбуке, а не в консоли или файле
    %matplotlib inline
```

Рисунок 43. Загрузка библиотеки

Далее выполним описание базы:

1. Датасет состоит из набора данных о винах и их классах.
2. Данные по одному вину хранятся в `numpy`-массиве `x_data`: (13 параметров).
3. В датасете 3 класса вин: `y_data`.
4. Количество примеров: 178.

```
[ ] x_data = load_wine()['data']           # Загрузка набора данных о винах
    y_data = load_wine()['target']         # Загрузка классов вин

print('Размерность x_data -', x_data.shape)
print('Размерность y_data -', y_data.shape)
print()

# Вывод примера данных
print('Данные по первому вину:', x_data[0])
print('Класс вина:', y_data[0])
```

Рисунок 44. Описание базы

После выполним разбиение наборов на общую и тестовую выборки, а также разбиение общей выборки на обучающую и проверочную.

```
# Перевод в one hot encoding
y_data = utils.to_categorical(y_data, 3)

# Разбиение наборов на общую и тестовую выборки
x_all, x_test, y_all, y_test = train_test_split(x_data,
                                                y_data,
                                                test_size=0.1,
                                                shuffle=True,
                                                random_state = 6)

# Разбиение общей выборки на обучающую и проверочную
x_train, x_val, y_train, y_val = train_test_split(x_all,
                                                  y_all,
                                                  test_size=0.1,
                                                  shuffle=True,
                                                  random_state = 6)

print(x_train.shape)
print(y_train.shape)
print()
print(x_val.shape)
print(y_val.shape)
```

```
(144, 13)
(144, 3)

(16, 13)
(16, 3)
```

Рисунок 45. Разбиение на обучающую и проверочную выборки

Далее выполним построение модели нейронной сети.

```
[ ] # Построение модели нейронной сети
    model = Sequential()

    # Входной слой (с количеством нейронов, равным числу признаков)
    model.add(Dense(128, input_dim=x_train.shape[1]))
    model.add(Activation('relu'))
    model.add(BatchNormalization()) # Нормализация

    # Скрытые слои
    model.add(Dense(64))
    model.add(Activation('relu'))
    model.add(Dropout(0.3)) # Dropout для предотвращения переобучения

    model.add(Dense(32))
    model.add(Activation('relu'))

    # Выходной слой (3 нейрона для 3 классов)
    model.add(Dense(3))
    model.add(Activation('softmax'))

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

Рисунок 46. Построение модели нейронной сети

Далее выполним обучение модели:

```
[ ] # Обучение модели
history = model.fit(x_train, y_train,
                    epochs=60,
                    batch_size=32,
                    validation_data=(x_val, y_val),
                    verbose=2)
```

```
Epoch 1/60
5/5 - 3s - 51ms/step - accuracy: 0.3681 - loss: 1.1633 - val_accuracy: 0.3750 - val_loss: 1.9067
Epoch 2/60
5/5 - 8s - 19ms/step - accuracy: 0.6111 - loss: 0.7773 - val_accuracy: 0.2500 - val_loss: 4.8504
Epoch 3/60
5/5 - 8s - 18ms/step - accuracy: 0.6458 - loss: 0.7257 - val_accuracy: 0.2500 - val_loss: 5.5833
Epoch 4/60
5/5 - 8s - 28ms/step - accuracy: 0.6667 - loss: 0.6209 - val_accuracy: 0.2500 - val_loss: 4.9654
Epoch 5/60
5/5 - 8s - 36ms/step - accuracy: 0.6597 - loss: 0.6198 - val_accuracy: 0.2500 - val_loss: 4.4206
Epoch 6/60
5/5 - 8s - 28ms/step - accuracy: 0.7153 - loss: 0.6041 - val_accuracy: 0.2500 - val_loss: 3.8057
Epoch 7/60
5/5 - 8s - 19ms/step - accuracy: 0.7222 - loss: 0.5733 - val_accuracy: 0.2500 - val_loss: 3.2789
Epoch 8/60
5/5 - 8s - 19ms/step - accuracy: 0.7014 - loss: 0.5941 - val_accuracy: 0.2500 - val_loss: 2.6883
Epoch 9/60
5/5 - 8s - 19ms/step - accuracy: 0.7431 - loss: 0.5472 - val_accuracy: 0.2500 - val_loss: 2.1194
Epoch 10/60
5/5 - 8s - 21ms/step - accuracy: 0.7708 - loss: 0.5271 - val_accuracy: 0.3125 - val_loss: 1.6881
Epoch 11/60
5/5 - 8s - 28ms/step - accuracy: 0.8194 - loss: 0.5405 - val_accuracy: 0.3750 - val_loss: 1.2684
Epoch 12/60
5/5 - 8s - 28ms/step - accuracy: 0.8264 - loss: 0.4649 - val_accuracy: 0.4375 - val_loss: 1.0217
Epoch 13/60
5/5 - 8s - 27ms/step - accuracy: 0.7708 - loss: 0.5023 - val_accuracy: 0.3750 - val_loss: 0.9011
Epoch 14/60
5/5 - 8s - 27ms/step - accuracy: 0.8750 - loss: 0.4257 - val_accuracy: 0.5625 - val_loss: 0.8509
Epoch 15/60
5/5 - 8s - 21ms/step - accuracy: 0.8403 - loss: 0.3844 - val_accuracy: 0.5625 - val_loss: 0.8183
Epoch 16/60
5/5 - 8s - 26ms/step - accuracy: 0.8750 - loss: 0.3597 - val_accuracy: 0.6250 - val_loss: 0.8093
Epoch 17/60
5/5 - 8s - 18ms/step - accuracy: 0.9097 - loss: 0.3040 - val_accuracy: 0.6250 - val_loss: 0.7741
Epoch 18/60
5/5 - 8s - 19ms/step - accuracy: 0.9097 - loss: 0.2907 - val_accuracy: 0.6250 - val_loss: 0.7801
Epoch 19/60
5/5 - 8s - 19ms/step - accuracy: 0.9028 - loss: 0.2613 - val_accuracy: 0.6250 - val_loss: 0.7180
Epoch 20/60
5/5 - 8s - 19ms/step - accuracy: 0.9236 - loss: 0.2491 - val_accuracy: 0.6250 - val_loss: 0.8393
Epoch 21/60
5/5 - 8s - 19ms/step - accuracy: 0.9167 - loss: 0.2453 - val_accuracy: 0.6250 - val_loss: 0.8033
Epoch 22/60
5/5 - 8s - 18ms/step - accuracy: 0.9236 - loss: 0.1987 - val_accuracy: 0.6250 - val_loss: 0.6951
Epoch 23/60
5/5 - 8s - 18ms/step - accuracy: 0.9236 - loss: 0.1787 - val_accuracy: 0.6075 - val_loss: 0.5693
Epoch 24/60
5/5 - 8s - 28ms/step - accuracy: 0.8750 - loss: 0.2885 - val_accuracy: 0.7500 - val_loss: 0.5623
Epoch 25/60
5/5 - 8s - 27ms/step - accuracy: 0.9514 - loss: 0.1777 - val_accuracy: 0.6875 - val_loss: 0.4706
Epoch 26/60
5/5 - 8s - 28ms/step - accuracy: 0.9306 - loss: 0.2124 - val_accuracy: 0.6875 - val_loss: 0.5925
Epoch 27/60
5/5 - 8s - 28ms/step - accuracy: 0.9306 - loss: 0.1740 - val_accuracy: 0.8750 - val_loss: 0.4186
Epoch 28/60
5/5 - 8s - 19ms/step - accuracy: 0.9306 - loss: 0.2045 - val_accuracy: 0.8750 - val_loss: 0.4351
Epoch 29/60
5/5 - 8s - 28ms/step - accuracy: 0.9306 - loss: 0.1596 - val_accuracy: 0.7500 - val_loss: 0.5073
```

Рисунок 47. Обучение модели

Получим результат обучение модели:

```
Epoch 36/60
5/5 - 8s - 28ms/step - accuracy: 0.9722 - loss: 0.1073 - val_accuracy: 0.9375 - val_loss: 0.2038
Epoch 37/60
5/5 - 8s - 19ms/step - accuracy: 0.9444 - loss: 0.1601 - val_accuracy: 0.8125 - val_loss: 0.2997
Epoch 38/60
5/5 - 8s - 27ms/step - accuracy: 0.9514 - loss: 0.1038 - val_accuracy: 0.8750 - val_loss: 0.2408
Epoch 39/60
5/5 - 8s - 28ms/step - accuracy: 0.9653 - loss: 0.1186 - val_accuracy: 0.9375 - val_loss: 0.2029
Epoch 40/60
5/5 - 8s - 19ms/step - accuracy: 0.9375 - loss: 0.1786 - val_accuracy: 0.7500 - val_loss: 0.5330
Epoch 41/60
5/5 - 8s - 21ms/step - accuracy: 0.9722 - loss: 0.0944 - val_accuracy: 0.8750 - val_loss: 0.2410
Epoch 42/60
5/5 - 8s - 25ms/step - accuracy: 0.9375 - loss: 0.1754 - val_accuracy: 0.8750 - val_loss: 0.2245
Epoch 43/60
5/5 - 8s - 20ms/step - accuracy: 0.9653 - loss: 0.1400 - val_accuracy: 0.8750 - val_loss: 0.2089
Epoch 44/60
5/5 - 8s - 34ms/step - accuracy: 0.9444 - loss: 0.1316 - val_accuracy: 0.9375 - val_loss: 0.2710
Epoch 45/60
5/5 - 8s - 34ms/step - accuracy: 0.9722 - loss: 0.1026 - val_accuracy: 0.8125 - val_loss: 0.3397
Epoch 46/60
5/5 - 8s - 54ms/step - accuracy: 0.9861 - loss: 0.0876 - val_accuracy: 0.8125 - val_loss: 0.2712
Epoch 47/60
5/5 - 8s - 31ms/step - accuracy: 0.9861 - loss: 0.0736 - val_accuracy: 0.8750 - val_loss: 0.2529
Epoch 48/60
5/5 - 8s - 34ms/step - accuracy: 0.9583 - loss: 0.0953 - val_accuracy: 0.8750 - val_loss: 0.2477
Epoch 49/60
5/5 - 8s - 60ms/step - accuracy: 0.9653 - loss: 0.1078 - val_accuracy: 0.8125 - val_loss: 0.2919
Epoch 50/60
5/5 - 8s - 28ms/step - accuracy: 0.9583 - loss: 0.1157 - val_accuracy: 0.8125 - val_loss: 0.3038
Epoch 51/60
5/5 - 8s - 35ms/step - accuracy: 0.9792 - loss: 0.0651 - val_accuracy: 0.8750 - val_loss: 0.3014
Epoch 52/60
5/5 - 8s - 62ms/step - accuracy: 0.9653 - loss: 0.1026 - val_accuracy: 0.8750 - val_loss: 0.3032
Epoch 53/60
5/5 - 8s - 29ms/step - accuracy: 0.9444 - loss: 0.1033 - val_accuracy: 0.9375 - val_loss: 0.2211
Epoch 54/60
5/5 - 8s - 59ms/step - accuracy: 1.0000 - loss: 0.0376 - val_accuracy: 0.8750 - val_loss: 0.2068
Epoch 55/60
5/5 - 8s - 19ms/step - accuracy: 0.9722 - loss: 0.0678 - val_accuracy: 0.8750 - val_loss: 0.2103
Epoch 56/60
5/5 - 8s - 18ms/step - accuracy: 0.9861 - loss: 0.0436 - val_accuracy: 0.8750 - val_loss: 0.2188
Epoch 57/60
5/5 - 8s - 28ms/step - accuracy: 0.9861 - loss: 0.0475 - val_accuracy: 0.9375 - val_loss: 0.2462
Epoch 58/60
5/5 - 8s - 29ms/step - accuracy: 0.9722 - loss: 0.0614 - val_accuracy: 0.9375 - val_loss: 0.1821
Epoch 59/60
5/5 - 8s - 19ms/step - accuracy: 0.9792 - loss: 0.0661 - val_accuracy: 0.8125 - val_loss: 0.3864
Epoch 60/60
5/5 - 8s - 27ms/step - accuracy: 0.9861 - loss: 0.0601 - val_accuracy: 0.8750 - val_loss: 0.3970
```

Рисунок 48. Результат обучения модели

После чего выполним оценку модели на тестовых данных:

```
[ ] # Оценка модели на тестовых данных
test_loss, test_acc = model.evaluate(x_test, y_test)
print(f'Точность на тестовых данных: {test_acc * 100:.2f}%')
```

1/1 ————— 0s 43ms/step - accuracy: 0.9444 - loss: 0.1016
Точность на тестовых данных: 94.44%

Рисунок 49. Оценка модели

Оценка модели на тестовых данных составила 94.44%. После с помощью метода `.summary()` зафиксируем количество параметров созданной нейронной сети.

```
[ ] # Вывод сводки модели
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 128)	1,792
activation (Activation)	(None, 128)	0
batch_normalization (BatchNormalization)	(None, 128)	512
dense_1 (Dense)	(None, 64)	8,256
activation_1 (Activation)	(None, 64)	0
dropout (Dropout)	(None, 64)	0
dense_2 (Dense)	(None, 32)	2,080
activation_2 (Activation)	(None, 32)	0
dense_3 (Dense)	(None, 1)	98
activation_3 (Activation)	(None, 1)	0

Total params: 12,740 (147.30 KB)
Trainable params: 12,483 (48.76 KB)
Non-trainable params: 256 (1.00 KB)
Optimizer params: 24,968 (97.54 KB)

Рисунок 50. Использование метода `.summary()`

Далее построим график точности:

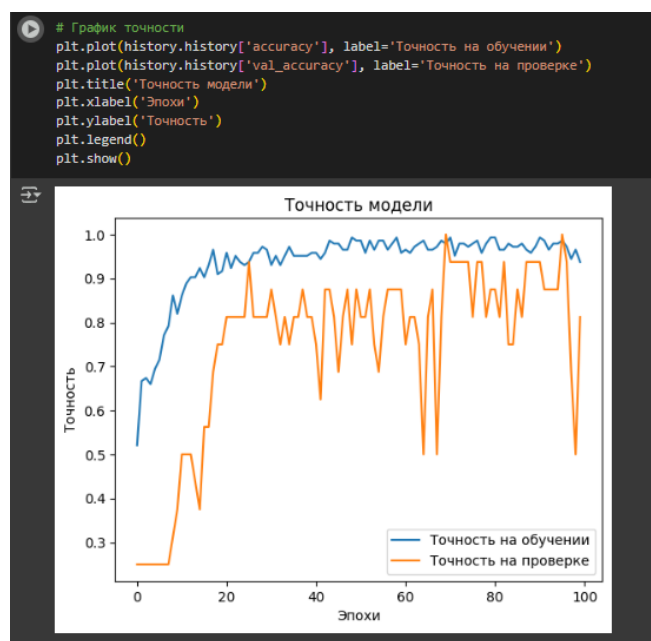


Рисунок 51. График точности

Задание 3.

Условие: необходимо используя базу "Пассажиры автобуса", подготовить данные для обучения нейронной сети, классифицирующей изображение на два класса:

- входящий пассажир
- выходящий пассажир

Необходимо добиться точности работы модели на проверочной выборке не ниже 85%

Для начала загрузим все необходимые библиотеки.

```
[28] import os
      from pathlib import Path

      import matplotlib.pyplot as plt
      import numpy as np
      from sklearn.model_selection import train_test_split
      from tensorflow.keras.layers import Activation, BatchNormalization, Dense, Dropout
      from tensorflow.keras.models import Sequential
      from tensorflow.keras.optimizers import Adam
      from tensorflow.keras.preprocessing import image
```

Рисунок 52. Загрузка библиотек

Далее выполним скачивание и распаковку данных:

```
# Скачиваем и распаковываем данные
!wget https://storage.yandexcloud.net/aiueducation/Content/base/14/bus.zip
!unzip -q bus.zip -d bus

# Проверка
!ls bus

--2025-04-13 16:40:22-- https://storage.yandexcloud.net/aiueducation/Content/base/14/bus.zip
Resolving storage.yandexcloud.net (storage.yandexcloud.net)... 213.180.193.243, 2a02:6b8::1d9
Connecting to storage.yandexcloud.net (storage.yandexcloud.net)[213.180.193.243]:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 78580527 (75M) [application/x-zip-compressed]
Saving to: 'bus.zip.8'

bus.zip.8      100%[=====] 74.94M  14.0MB/s   in 6.7s

2025-04-13 16:40:30 (11.2 MB/s) - 'bus.zip.8' saved [78580527/78580527]

replace bus/Входящий/01009.jpg? [y]es, [n]o, [A]ll, [N]one, [r]ename: A
Входящий  Выходящий
```

Рисунок 53. Распаковка данных

Далее разделим данные на обучающую и временную выборки и разделим временную выборку на валидационную тестовую.

```
# Разделяем данные на обучающую и временную выборки
x_train, x_temp, y_train, y_temp = train_test_split(
    x_all, y_all, test_size=0.3, random_state=42, stratify=y_all
)

# Разделяем временную выборку на валидационную и тестовую
x_val, x_test, y_val, y_test = train_test_split(
    x_temp, y_temp, test_size=0.5, random_state=42, stratify=y_temp
)

# Параметры модели
drop_rate = 0.3
input_shape = 128 * 128
```

Рисунок 54. Разделение данных

Далее создадим модель и выполним компиляцию.

```
# Создаем модель
model = Sequential()

# Первый скрытый слой
model.add(Dense(512, input_shape=(input_shape,))) # Полносвязный слой с 512 нейронами
model.add(BatchNormalization())
model.add(Activation("relu"))
model.add(Dropout(drop_rate))

# Второй скрытый слой
model.add(Dense(256)) # 256 нейронов
model.add(BatchNormalization())
model.add(Activation("relu"))
model.add(Dropout(drop_rate))

# Третий скрытый слой
model.add(Dense(128))
model.add(BatchNormalization())
model.add(Activation("relu"))
model.add(Dropout(drop_rate))

model.add(Dense(1, activation="sigmoid"))

# Компилируем модель
model.compile(
    loss="binary_crossentropy",
    optimizer=Adam(learning_rate=0.0005),
    metrics=["accuracy"],
)
```

Рисунок 55. Создание модели

Далее выполним обучение модели:

```
[43] # Обучаем модель
history = model.fit(
    x_train,
    y_train,
    epochs=40, # Количество эпох обучения
    batch_size=32, # Размер батча
    validation_data=(x_val, y_val), # Данные для валидации
    verbose=1, # Вывод информации об обучении
)
```

Epoch	Step	Time	Accuracy	Loss	Val Accuracy	Val Loss
Epoch 12/40	1s	5ms/step	0.9445	0.1534	0.9046	0.2528
Epoch 13/40	1s	5ms/step	0.9359	0.1611	0.8664	0.3189
Epoch 14/40	1s	5ms/step	0.9363	0.1534	0.8935	0.2645
Epoch 15/40	1s	6ms/step	0.9484	0.1403	0.8759	0.3513
Epoch 16/40	1s	7ms/step	0.9446	0.1370	0.9170	0.2058
Epoch 17/40	1s	6ms/step	0.9524	0.1173	0.9060	0.2900
Epoch 18/40	1s	5ms/step	0.9544	0.1183	0.9354	0.1457
Epoch 19/40	1s	5ms/step	0.9599	0.1005	0.8921	0.3008
Epoch 20/40	1s	5ms/step	0.9606	0.0994	0.8752	0.3183
Epoch 21/40	1s	5ms/step	0.9589	0.1117	0.9038	0.2266
Epoch 22/40	1s	5ms/step	0.9605	0.1096	0.8333	0.6043
Epoch 23/40	1s	5ms/step	0.9632	0.0955	0.8987	0.3142
Epoch 24/40	1s	5ms/step	0.9633	0.0919	0.9398	0.1807
Epoch 25/40	1s	5ms/step	0.9651	0.0963	0.9148	0.2363
Epoch 26/40	2s	7ms/step	0.9598	0.1034	0.9273	0.2066
Epoch 27/40	2s	5ms/step	0.9595	0.1078	0.9192	0.2385
Epoch 28/40	1s	5ms/step	0.9702	0.0877	0.9266	0.2110
Epoch 29/40	1s	5ms/step	0.9710	0.0748	0.9302	0.1846
Epoch 30/40	1s	5ms/step	0.9664	0.0901	0.9119	0.2489
Epoch 31/40	1s	5ms/step	0.9750	0.0700	0.9170	0.2545
Epoch 32/40	1s	5ms/step	0.9782	0.0645	0.8752	0.4846
Epoch 33/40	1s	5ms/step	0.9730	0.0670	0.9156	0.2550
Epoch 34/40	1s	5ms/step	0.9723	0.0751	0.9464	0.1411
Epoch 35/40	2s	7ms/step	0.9706	0.0769	0.8987	0.2447
Epoch 36/40	2s	6ms/step	0.9748	0.0666	0.9464	0.1525
Epoch 37/40	1s	5ms/step	0.9754	0.0766	0.9302	0.1779
Epoch 38/40	1s	5ms/step	0.9751	0.0656	0.9567	0.1122
Epoch 39/40	1s	5ms/step	0.9714	0.0716	0.9266	0.2273
Epoch 40/40	1s	5ms/step	0.9739	0.0715	0.9574	0.1464

Рисунок 56. Обучение модели

Далее выполним оценку данных и выведем результаты.

```
[44] # Оцениваем модель на тестовых данных
_, test_accuracy = model.evaluate(x_test, y_test, verbose=0)

# Выводим результаты
print(
    f"Точность на обучающей выборке: {history.history['accuracy'][-1] * 100:.2f}%,\n"
    "Точность на валидационной выборке: "\n"
    f"{history.history['val_accuracy'][-1] * 100:.2f}%,\n"
    f"Точность на тестовой выборке: {test_accuracy * 100:.2f}%"
)
```

Точность на обучающей выборке: 97.42%,
Точность на валидационной выборке: 95.74%,
Точность на тестовой выборке: 95.08%

Рисунок 57. Оценка модели

Далее создадим графики точности и потерь.

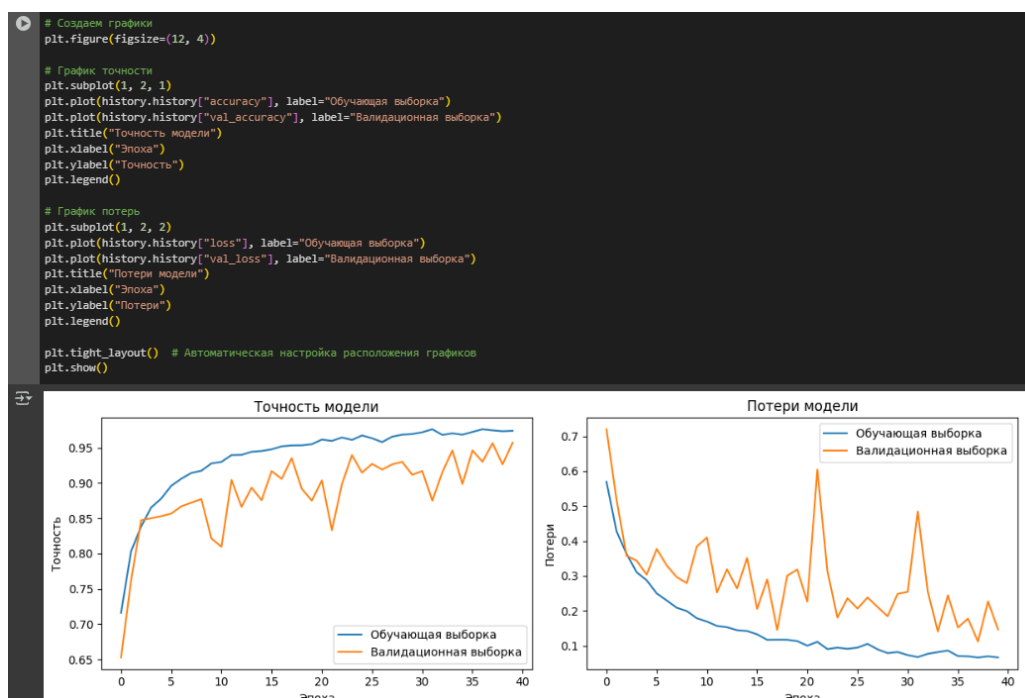


Рисунок 58. Создание графиков

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

https://drive.google.com/drive/folders/1VjaYvVBOhHc5Zivbb6mFRA2u5yNxNDqP?usp=drive_link

Вывод: в процессе выполнения были изучены способы разделения выборки на обучающую, проверочную и тестовую, изучить слои «Dropout» и «BatchNormalization» и их влияние на явление переобучения, также были изучены способы работы с параметрами загружаемых для обучения изображениями.