

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:

Говоров Егор Юрьевич

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.

Ссылка: https://github.com/Artorias1469/NN_2.git

Ход работы:

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.

```
# Сторонние библиотеки
import matplotlib.pyplot as plt # Отрисовка графиков
import numpy as np # Библиотека для работы с массивами
import pandas as pd # Библиотека для работы с таблицами
from google.colab import files # Связь с google-диск
from sklearn import preprocessing # Предварительная обработка данных
from sklearn.model_selection import train_test_split # Разделение данных на выборки

# TensorFlow / Keras
import tensorflow as tf # Основной фреймворк
from keras.datasets import fashion_mnist # Для загрузки датасета
from tensorflow.keras import utils # Утилиты для to_categorical()
from tensorflow.keras.layers import Activation, BatchNormalization, Dense, Dropout # Основные слои
from tensorflow.keras.models import Sequential # Последовательная модель НС
from tensorflow.keras.optimizers import Adam, Adadelta # Алгоритмы оптимизации для обучения модели
from tensorflow.keras.utils import to_categorical

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

Рисунок 1. 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)

Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-labels-idx1-ubyte.gz
29515/29515 ————— 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-images-idx3-ubyte.gz
26421880/26421880 ————— 2s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-labels-idx1-ubyte.gz
5148/5148 ————— 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-images-idx3-ubyte.gz
4422102/4422102 ————— 1s 0us/step
Размер x_train: (60000, 28, 28)
Размер y_train: (60000,)
Размер x_test: (10000, 28, 28)
Размер y_test: (10000,)
```

Рисунок 2. 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.

```
# Выбор 1 изображения каждого класса
imgs = np.array([x_train[y_train==i][0] for i in range(10)])

# Соединение изображения в одну линию
imgs = np.concatenate(imgs, axis=1)

# Создание поля для изображения
plt.figure(figsize=(30, 6))

# Отрисовка итогового изображения
plt.imshow(imgs, cmap='Greys_r')

# Без сетки
plt.grid(False)

# Без осей
plt.axis('off')

# Вывод результата
plt.show()
```



Рисунок 3. Отображение изображений

Далее выполним нормализацию, распределение и преобразование в векторы.

```
# Нормализация
x_train = x_train / 255.0
x_test = x_test / 255.0

# One-hot кодирование
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)

# Разделение
x_train_main = x_train[:50000]
y_train_main = y_train[:50000]
x_val = x_train[50000:]
y_val = y_train[50000:]

# Преобразование в векторы
x_train_main = x_train_main.reshape(-1, 784)
x_val = x_val.reshape(-1, 784)
x_test = x_test.reshape(-1, 784)
```

Рисунок 4. Нормализация и распределение

Далее выполним описание девяти архитектур.

```
# Описание архитектур
architectures = [
    [Dense(128, activation='relu'), Dense(10, activation='softmax')],
    [Dense(256, activation='relu'), Dense(10, activation='softmax')],
    [Dense(128, activation='relu'), Dropout(0.3), Dense(10, activation='softmax')],
    [Dense(64, activation='relu'), Dense(64, activation='relu'), Dense(10, activation='softmax')],
    [Dense(512, activation='relu'), Dense(256, activation='relu'), Dense(10, activation='softmax')],
    [Dense(128, activation='relu'), BatchNormalization(), Dense(10, activation='softmax')],
    [Dense(128, activation='relu'), Dropout(0.5), Dense(10, activation='softmax')],
    [Dense(64, activation='relu'), Dense(32, activation='relu'), Dense(10, activation='softmax')],
    [Dense(256, activation='relu'), Dropout(0.2), BatchNormalization(), Dense(10, activation='softmax')]
]
```

Рисунок 5. 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()
    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)

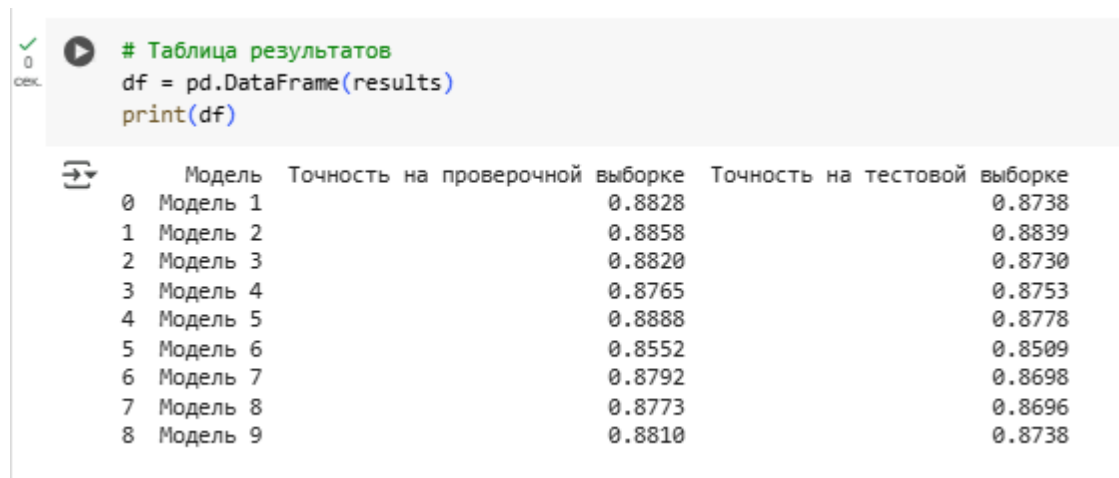
    val_acc = history.history['val_accuracy'][-1]
    test_loss, test_acc = model.evaluate(x_test, y_test, verbose=0)

    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 6. Обучение нейросетей

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



The screenshot shows a Jupyter Notebook cell with a green play button icon and a comment icon. The code in the cell is:

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

Below the code, the output of the DataFrame is displayed as a table with 4 columns: an index column, a 'Модель' column, a 'Точность на проверочной выборке' column, and a 'Точность на тестовой выборке' column. The data is as follows:

	Модель	Точность на проверочной выборке	Точность на тестовой выборке
0	Модель 1	0.8828	0.8738
1	Модель 2	0.8858	0.8839
2	Модель 3	0.8820	0.8730
3	Модель 4	0.8765	0.8753
4	Модель 5	0.8888	0.8778
5	Модель 6	0.8552	0.8509
6	Модель 7	0.8792	0.8698
7	Модель 8	0.8773	0.8696
8	Модель 9	0.8810	0.8738

Рисунок 7. 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:

```
# Работа с данными
import numpy as np
import matplotlib.pyplot as plt

# Scikit-learn: загрузка и разделение датасета
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split

# TensorFlow / Keras: модели, слои, оптимизаторы и утилиты
from tensorflow.keras import utils
from tensorflow.keras.layers import Activation, BatchNormalization, Dense, Dropout
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam

# Отрисовывать графики в ноутбуке, а не в консоли или файле
%matplotlib inline
```

Рисунок 8. 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.

```
[2] 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])
```

Размерность x_data - (178, 13)
Размерность y_data - (178,)

Данные по первому вину: [1.423e+01 1.710e+00 2.430e+00 1.560e+01 1.270e+02 2.800e+00 3.060e+00
2.800e-01 2.290e+00 5.640e+00 1.040e+00 3.920e+00 1.065e+03]
Класс вина: 0

Рисунок 9. 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 10. 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 11. 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 - 529ms/step - accuracy: 0.5625 - loss: 0.9069 - val_accuracy: 0.2500 - val_loss: 9.6286
Epoch 2/60
5/5 - 0s - 54ms/step - accuracy: 0.6597 - loss: 0.6905 - val_accuracy: 0.2500 - val_loss: 9.1191
Epoch 3/60
5/5 - 0s - 37ms/step - accuracy: 0.6667 - loss: 0.6587 - val_accuracy: 0.2500 - val_loss: 8.1196
Epoch 4/60
5/5 - 0s - 38ms/step - accuracy: 0.6806 - loss: 0.6215 - val_accuracy: 0.2500 - val_loss: 6.8748
Epoch 5/60
5/5 - 0s - 50ms/step - accuracy: 0.6806 - loss: 0.6052 - val_accuracy: 0.2500 - val_loss: 5.6060
Epoch 6/60
5/5 - 0s - 61ms/step - accuracy: 0.7222 - loss: 0.5615 - val_accuracy: 0.2500 - val_loss: 4.5974
Epoch 7/60
5/5 - 0s - 58ms/step - accuracy: 0.7153 - loss: 0.5247 - val_accuracy: 0.2500 - val_loss: 3.7286
Epoch 8/60
5/5 - 0s - 62ms/step - accuracy: 0.7708 - loss: 0.4974 - val_accuracy: 0.2500 - val_loss: 2.9920
Epoch 9/60
5/5 - 0s - 65ms/step - accuracy: 0.7708 - loss: 0.4873 - val_accuracy: 0.3125 - val_loss: 2.3449
Epoch 10/60
5/5 - 0s - 37ms/step - accuracy: 0.8194 - loss: 0.4648 - val_accuracy: 0.5000 - val_loss: 1.9303
Epoch 11/60
5/5 - 0s - 20ms/step - accuracy: 0.8056 - loss: 0.4720 - val_accuracy: 0.5000 - val_loss: 1.6992
Epoch 12/60
5/5 - 0s - 29ms/step - accuracy: 0.8472 - loss: 0.4218 - val_accuracy: 0.5000 - val_loss: 1.4056
Epoch 13/60
5/5 - 0s - 27ms/step - accuracy: 0.8958 - loss: 0.3697 - val_accuracy: 0.5000 - val_loss: 1.2868
Epoch 14/60
5/5 - 0s - 21ms/step - accuracy: 0.9097 - loss: 0.3666 - val_accuracy: 0.4375 - val_loss: 1.1775
Epoch 15/60
```

Рисунок 12. Обучение 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 62ms/step - accuracy: 0.2778 - loss: 2.6775
Точность на тестовых данных: 27.78%
```

Рисунок 13. 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, 3)	99
activation_3 (Activation)	(None, 3)	0

Total params: 37,707 (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 14. Использование метода .summary()

Далее построим график точности:

```
# График точности
plt.plot(history.history['accuracy'], label='Точность на обучении')
plt.plot(history.history['val_accuracy'], label='Точность на проверке')
plt.title('Точность модели')
plt.xlabel('Эпохи')
plt.ylabel('Точность')
plt.legend()
plt.show()
```

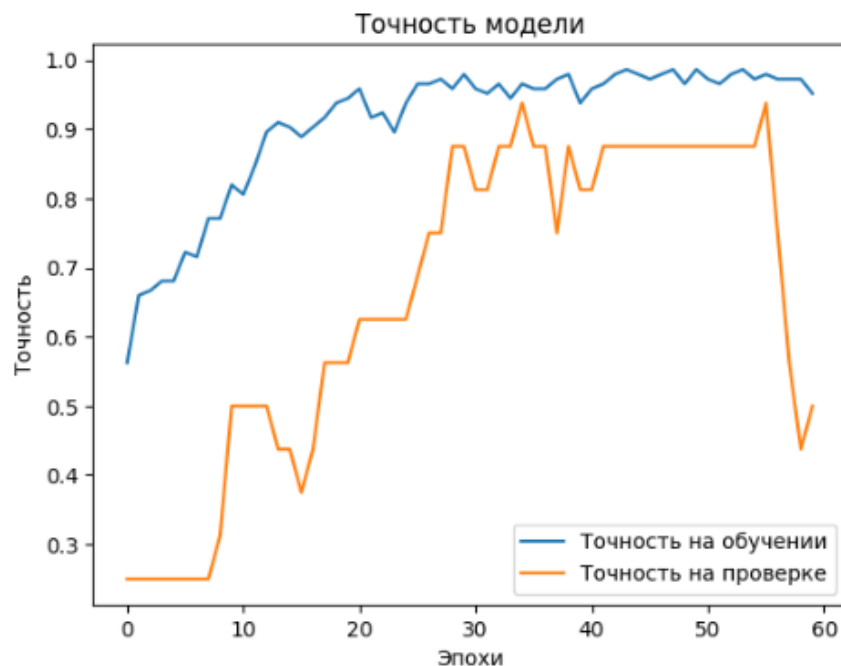


Рисунок 15. График точности

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.

```
# Стандартная библиотека
import os
import urllib.request
import zipfile
from pathlib import Path

# Внешние библиотеки
import matplotlib.pyplot as plt
import numpy as np
from sklearn.model_selection import train_test_split

# TensorFlow / Keras
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 16. Загрузка библиотек

Далее выполним скачивание и распаковку данных:

```
[16] # Скачиваем файл, если отсутствует
!wget https://storage.yandexcloud.net/aiueducation/Content/base/14/bus.zip
!unzip -q bus.zip -d bus

!ls bus

--2025-06-03 21:59:23-- 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.1'

bus.zip.1      100%[=====] 74.94M  19.5MB/s   in 4.7s

2025-06-03 21:59:28 (15.8 MB/s) - 'bus.zip.1' saved [78580527/78580527]

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

Рисунок 17. Распаковка данных

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

```

from sklearn.model_selection import train_test_split
from tensorflow.keras.datasets import mnist
import numpy as np

# 1. Загрузка данных MNIST
(x_all, y_all), (x_test_final, y_test_final) = mnist.load_data()

# 2. Преобразование и нормализация данных
x_all = x_all.reshape(-1, 28*28).astype('float32') / 255.0 # преобразуем в 1D и нормализуем

# 3. Разделение на обучающую и временную выборки (70%/30%)
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
)

# 4. Разделение временной выборки на валидационную и тестовую (15%/15%)
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
)

# 5. Параметры модели
drop_rate = 0.3
input_shape = 28 * 28 # для MNIST размер изображения 28x28

# Проверка размеров выборок
print(f"Обучающая выборка: {x_train.shape[0]} примеров")
print(f"Валидационная выборка: {x_val.shape[0]} примеров")
print(f"Тестовая выборка: {x_test.shape[0]} примеров")

```

 Downloading data from <https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz>
11490434/11490434 ————— 0s 0us/step
Обучающая выборка: 42000 примеров
Валидационная выборка: 9000 примеров
Тестовая выборка: 9000 примеров

Рисунок 18. 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"],
)

```

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

Рисунок 19. Создание модели

Далее выполним обучение модели:

```
# 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 1/40
1313/1313 ————— 24s 15ms/step - accuracy: 0.1310 - loss: -98.5502 - val_accuracy: 0.1144 - val_loss: -617.5780
Epoch 2/40
1313/1313 ————— 18s 13ms/step - accuracy: 0.1221 - loss: -1060.7671 - val_accuracy: 0.1192 - val_loss: -2444.8489
Epoch 3/40
1313/1313 ————— 21s 13ms/step - accuracy: 0.1312 - loss: -3114.1646 - val_accuracy: 0.1638 - val_loss: -5425.7568
Epoch 4/40
1313/1313 ————— 16s 12ms/step - accuracy: 0.1570 - loss: -6035.1519 - val_accuracy: 0.1812 - val_loss: -8950.0518
Epoch 5/40
1313/1313 ————— 23s 14ms/step - accuracy: 0.1627 - loss: -9697.4688 - val_accuracy: 0.1927 - val_loss: -13358.5986
Epoch 6/40
1313/1313 ————— 19s 13ms/step - accuracy: 0.1565 - loss: -14075.3535 - val_accuracy: 0.1906 - val_loss: -18396.1973
Epoch 7/40
1313/1313 ————— 21s 13ms/step - accuracy: 0.1723 - loss: -19206.0820 - val_accuracy: 0.1860 - val_loss: -23703.8359
Epoch 8/40
1313/1313 ————— 17s 13ms/step - accuracy: 0.1640 - loss: -24851.1641 - val_accuracy: 0.1918 - val_loss: -29969.4180
Epoch 9/40
1313/1313 ————— 19s 14ms/step - accuracy: 0.1687 - loss: -31257.2188 - val_accuracy: 0.1906 - val_loss: -36614.1875
Epoch 10/40
1313/1313 ————— 17s 13ms/step - accuracy: 0.1667 - loss: -38154.8320 - val_accuracy: 0.1939 - val_loss: -44224.5469
Epoch 11/40
1313/1313 ————— 16s 12ms/step - accuracy: 0.1684 - loss: -45962.1953 - val_accuracy: 0.2001 - val_loss: -51028.6211
Epoch 12/40
1313/1313 ————— 18s 14ms/step - accuracy: 0.1701 - loss: -54200.2852 - val_accuracy: 0.2044 - val_loss: -59600.5234
Epoch 13/40
1313/1313 ————— 17s 13ms/step - accuracy: 0.1719 - loss: -63025.6172 - val_accuracy: 0.1994 - val_loss: -71513.9297
Epoch 14/40
1313/1313 ————— 22s 14ms/step - accuracy: 0.1708 - loss: -72764.0625 - val_accuracy: 0.2024 - val_loss: -79927.8516

Рисунок 20. Обучение модели

Далее выполним оценку данных и выведем результаты.

```
# 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: "
    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 21. Оценка модели

Далее создадим графики точности и потерь.

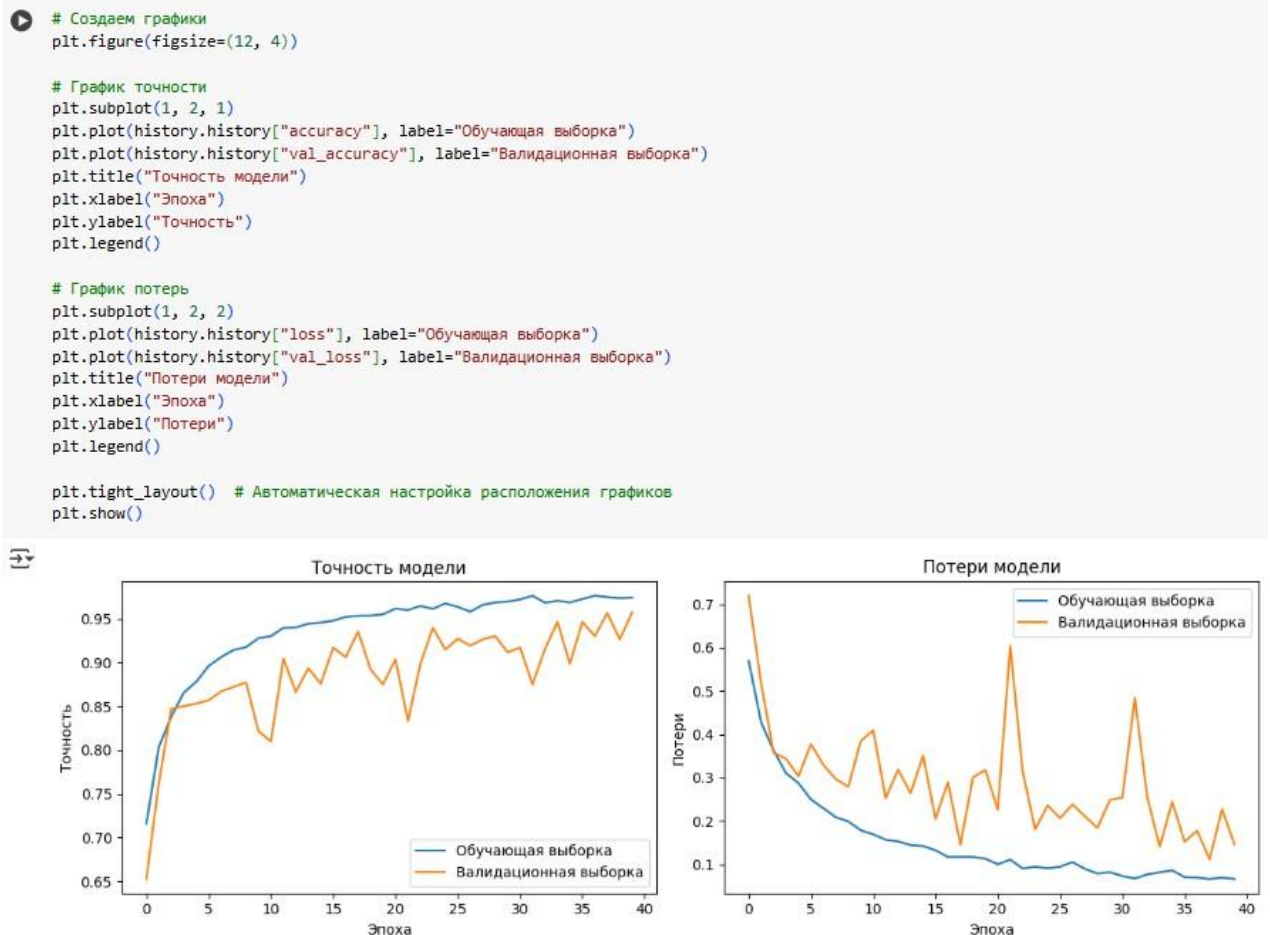


Рисунок 22. Создание графиков

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.