МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ЛАБОРАТОРНАЯ РАБОТА №4

по дисциплине «Искусственные нейронные сети»

Тема: «Распознавание рукописных символов»

Студент гр. 7381 _____ Ильясов А.В.

Преподаватель _____ Жукова Н. А.

Санкт-Петербург
2020

Цели.

Реализовать классификацию черно-белых изображений рукописных цифр (28x28) по 10 категориям (от 0 до 9).

Задачи.

- Ознакомиться с представлением графических данных
- Ознакомиться с простейшим способом передачи графических данных нейронной сети
- Создать модель
- Настроить параметры обучения
- Найти архитектуру сети, при которой точность классификации будет не менее 95%
- Исследовать влияние различных оптимизаторов, а также их параметров, на процесс обучения
- Написать функцию, которая позволит загружать пользовательское изображение не из датасета

Выполнение работы.

Для решения данной задачи была выбрана архитектура модели, состоящая из 2 слоев:

```
Dense(512, activation='relu', input_shape=(28 * 28,))
Dense(10, activation='softmax')
```

Исходный размерность датасета была изменена с (60000, 28, 28) на (60000, 28*28).

Для возможности загрузки пользовательских изображений была написана функция:

```
def load_image(filepath):
    img = np.asarray(Image.open(filepath))
    img = cv2.resize(img, (28, 28),
interpolation=cv2.INTER_CUBIC)

    k = np.array([[[0.2989, 0.587, 0.114]]])
```

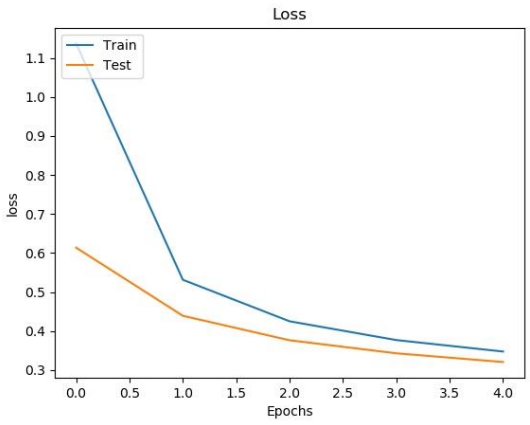
```
        img = np.sum(img * k, axis=2).reshape((1, 28 * 28)) /
255.0
        return img
```

Эта функция загружает изображение по адресу, затем сжимает или растягивает его под размер изображений с датасета (28, 28), переводит его из 3-х канального изображения в одноканальное и переводит в одномерный массив размера 28*28.

В ходе работы были сравнены следующие оптимизаторы с следующими параметрами:

1. SGD с дефолтными параметрами;
2. SGD с шагом learning rate = 0.001;
3. SGD с momentum = 0.9;
4. RMSProp с дефолтными параметрами;
5. RMSProp с шагом learning rate = 0.01;
6. RMSProp с коэффициентом затухания градиента rho = 0.5;
7. RMSProp с momentum = 0.9;
8. Adagrad с дефолтными параметрами;
9. Adagrad с шагом learning rate = 0.01;
10. Adam с дефолтными параметрами;
11. Adam с шагом learning rate = 0.01;

После обучения модели с каждым оптимизатором из списка, получились следующие результаты:



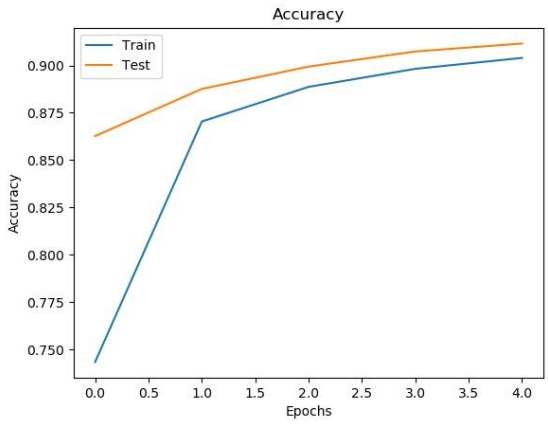


Рисунок 1 – графики функции ошибок и точности модели с оптимизатором 1

Рисунок 2 – графики функции ошибок и точности модели с оптимизатором 2

Рисунок 3 – графики функции ошибок и точности модели с оптимизатором 3

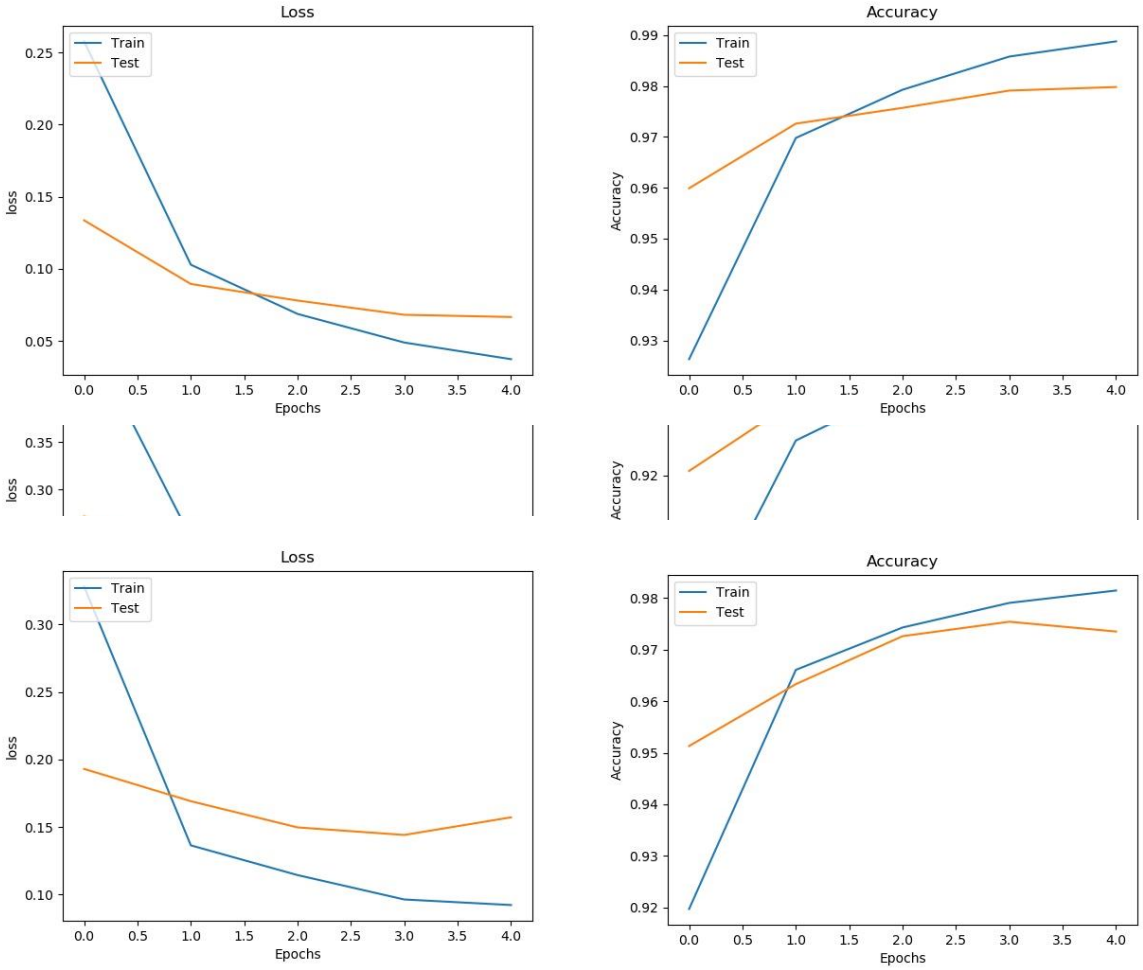


Рисунок 4 – графики функции ошибок и точности модели с оптимизатором 4

Рисунок 5 – графики функции ошибок и точности модели с оптимизатором 5

Рисунок 6 – графики функции ошибок и точности модели с оптимизатором 6

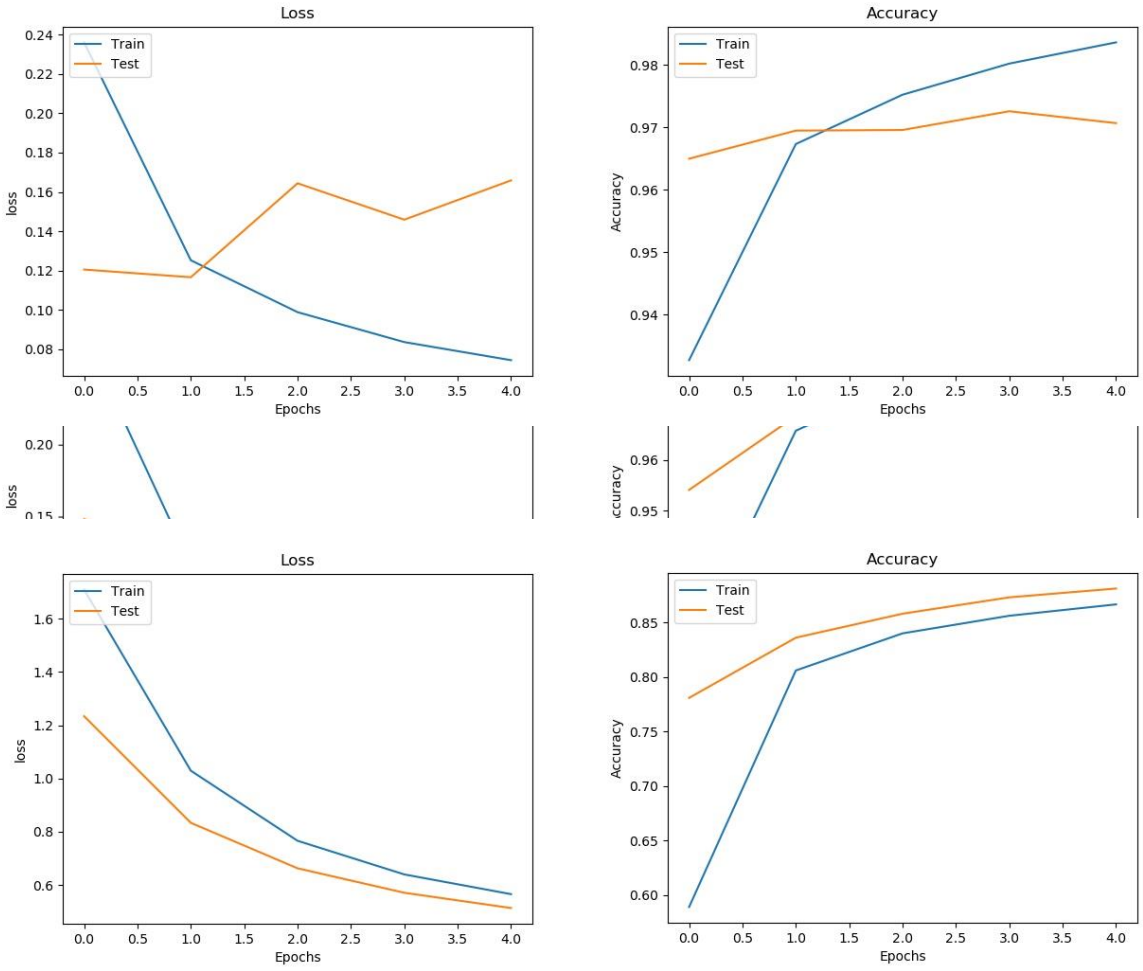


Рисунок 7 – графики функции ошибок и точности модели с оптимизатором 7

Рисунок 8 – графики функции ошибок и точности модели с оптимизатором 8

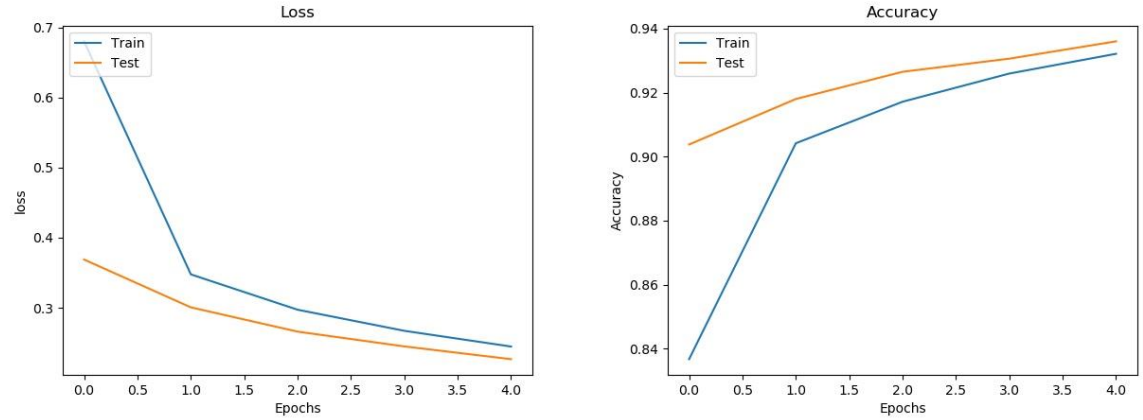


Рисунок 9 – графики функции ошибок и точности модели с оптимизатором 9

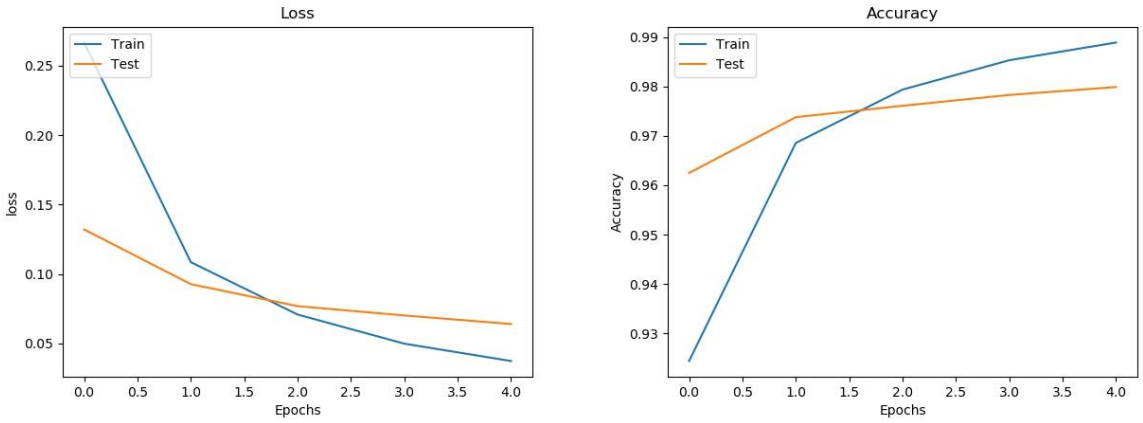


Рисунок 10 – графики функции ошибок и точности модели с оптимизатором 10

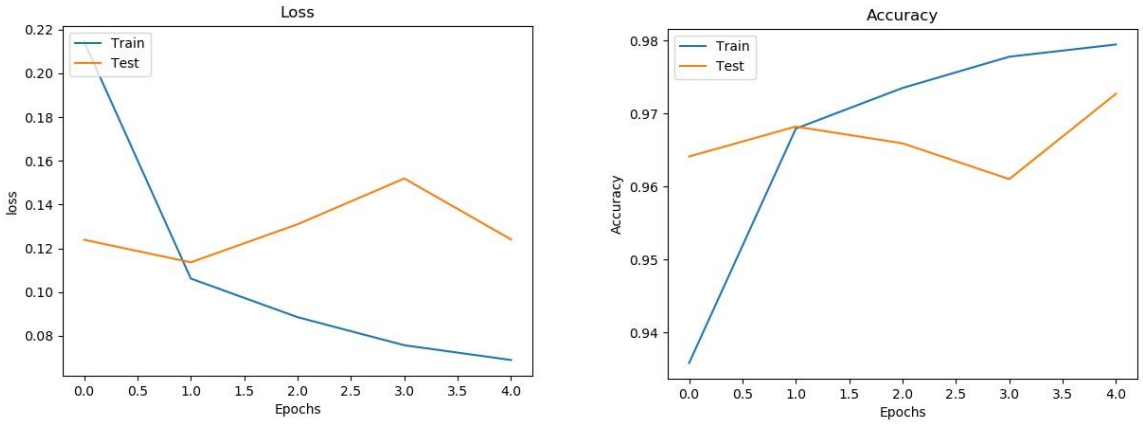


Рисунок 11 – графики функции ошибок и точности модели с оптимизатором 11

Ниже представлены графики и таблица сравнения точности и ошибок моделей с разными оптимизаторами.

Таблица 1 – сравнение результатов обучения моделей

№ модели	acc	val_acc	loss	val_loss
1	0.90400	0.91160	0.34768	0.32078
2	0.81032	0.83020	0.99229	0.90235
3	0.96103	0.96130	0.14096	0.13357
4	0.98877	0.97980	0.03739	0.06663
5	0.98145	0.97350	0.09218	0.15712

6	0.98460	0.97610	0.05526	0.08964
7	0.98363	0.97070	0.07449	0.16588
8	0.86655	0.88110	0.56590	0.51374
9	0.93212	0.93600	0.24480	0.22678
10	0.98892	0.97990	0.03736	0.06401
11	0.97943	0.97270	0.06895	0.12405

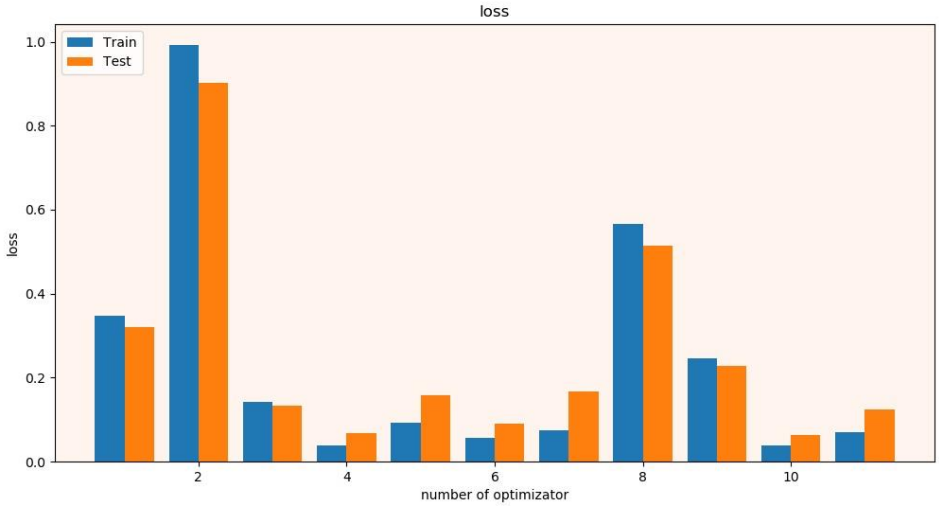


Рисунок 12 – график сравнения функции ошибок всех моделей

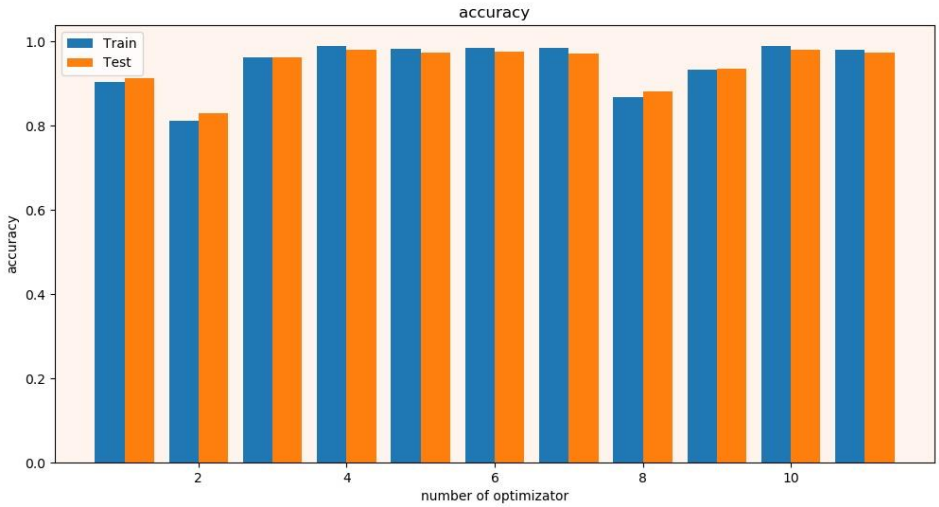


Рисунок 13 – график сравнения точности всех моделей

Сравнивая полученные цифры и смотря на графики, можно увидеть, что хуже других себя показали оптимизаторы SGD с шагом 0.001 и Adagrad с дефолтными параметрами. Лучше же справились оптимизаторы RMSProp и Adam, независимо от параметров.

Вывод.

В ходе выполнения данной работы была изучена задача распознавания рукописных цифр и исследовано влияние различных оптимизаторов на обучение моделей. Также была произведена работа по работе и обработке изображений.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

```
import cv2
from tensorflow.keras.layers import Dense
from tensorflow.keras import optimizers
from tensorflow.keras.models import Sequential
from tensorflow.keras.utils import to_categorical
import tensorflow as tf

import matplotlib.pyplot as plt
import numpy as np

from PIL import Image


optimizers = [optimizers.SGD(),
              optimizers.SGD(learning_rate=0.001),
              optimizers.SGD(momentum=0.9),
              optimizers.RMSprop(),
              optimizers.RMSprop(learning_rate=0.01),
              optimizers.RMSprop(rho=0.5),
              optimizers.RMSprop(momentum=0.9),
              optimizers.Adagrad(),
              optimizers.Adagrad(learning_rate=0.01),
              optimizers.Adam(),
              optimizers.Adam(learning_rate=0.01)]


def build_model():
    model = Sequential()
    model.add(Dense(512, activation='relu', input_shape=(28 * 28,)))
    model.add(Dense(10, activation='softmax'))
    return model


def load_image(filepath):
    img = np.asarray(Image.open(filepath))
    img = cv2.resize(img, (28, 28), interpolation=cv2.INTER_CUBIC)

    k = np.array([[[0.2989, 0.587, 0.114]]])
    img = np.sum(img * k, axis=2).reshape((1, 28 * 28)) / 255.0
    return img


def predict_numerals(model):
    for i in range(10):
        img = load_image(f'numerals/{i}.png')
        print(f'numeral: {i}')
        for j, predict in enumerate(model.predict(img)[0]):
```

```
            print(f'{j}: {predict: .4f}')


def plot_diagramm(y1, y2, metric):
    x1 = np.arange(1, 12) - 0.2
    x2 = np.arange(1, 12) + 0.2

    fig, ax = plt.subplots()

    ax.bar(x1, y1, width=0.4)
    ax.bar(x2, y2, width=0.4)

    ax.set_facecolor('seashell')
    fig.set_figwidth(12)  # ширина Figure
    fig.set_figheight(6)  # высота Figure
    fig.set_facecolor('floralwhite')
    plt.title(metric)
    plt.ylabel(metric)
    plt.xlabel('number of optimizator')
    plt.legend(['Train', 'Test'], loc='upper left')
    fig.savefig(f'images/{metric}.jpg')


if __name__ == '__main__':
    mnist = tf.keras.datasets.mnist
    (train_images, train_labels), (test_images, test_labels) =
mnist.load_data()

    train_images = train_images.reshape((60000,
28*28)).astype('float32') / 255.0
    test_images = test_images.reshape((10000,
28*28)).astype('float32') / 255.0

    train_labels = to_categorical(train_labels)
    test_labels = to_categorical(test_labels)

    acc = []
    val_acc = []
    loss = []
    val_loss = []

    for i, optimizer in enumerate(optimizers):
        model = build_model()
        model.compile(optimizer=optimizer,
loss='categorical_crossentropy', metrics=['accuracy'])

        history = model.fit(train_images, train_labels, epochs=5,
batch_size=128,
                            validation_data=(test_images,
test_labels), verbose=0)
        model.save_weights(filepath=f'models/{i+1}.h5')
```

```
            acc.append(history.history['acc'][-1])
            val_acc.append(history.history['val_acc'][-1])
            loss.append(history.history['loss'][-1])
            val_loss.append(history.history['val_loss'][-1])

            plt.plot(history.history['acc'])
            plt.plot(history.history['val_acc'])
            plt.title('Accuracy')
            plt.ylabel('Accuracy')
            plt.xlabel('Epochs')
            plt.legend(['Train', 'Test'], loc='upper left')
            plt.savefig(f'images/accuracy-{i + 1}.jpg')
            plt.clf()

            plt.plot(history.history['loss'])
            plt.plot(history.history['val_loss'])
            plt.title('Loss')
            plt.ylabel('loss')
            plt.xlabel('Epochs')
            plt.legend(['Train', 'Test'], loc='upper left')
            plt.savefig(f'images/loss-{i+1}.jpg')
            plt.clf()

        for a, va, l, vl in zip(acc, val_acc, loss, val_loss):
            print(f'acc = {a:.5f}, val_acc = {va:.5f}, loss = {l:.5f},
val_loss = {vl:.5f}')

        plot_diagramm(acc, val_acc, 'accuracy')
        plot_diagramm(loss, val_loss, 'loss')
```