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

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Искусственные нейронные сети»

Тема: Распознавание рукописных символов

Студент гр. 7382 _____ Гаврилов А.В.

Преподаватель _____ Жукова Н.А.

Санкт-Петербург

2020

Цель работы.

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

Порядок выполнения работы.

1. Ознакомиться с представлением графических данных.
2. Ознакомиться с простейшим способом передачи графических данных нейронной сети.
3. Создать модель.
4. Настроить параметры обучения.
5. Написать функцию, позволяющая загружать изображение пользователи и классифицировать его.

Требования к выполнению задания.

1. Найти архитектуру сети, при которой точность классификации будет не менее 95%.
2. Исследовать влияние различных оптимизаторов, а также их параметров, на процесс обучения.
3. Написать функцию, которая позволит загружать пользовательское изображение не из датасета.

Ход работы.

Была создана и обучена модель искусственной нейронной сети. Код предоставлен в приложении А.

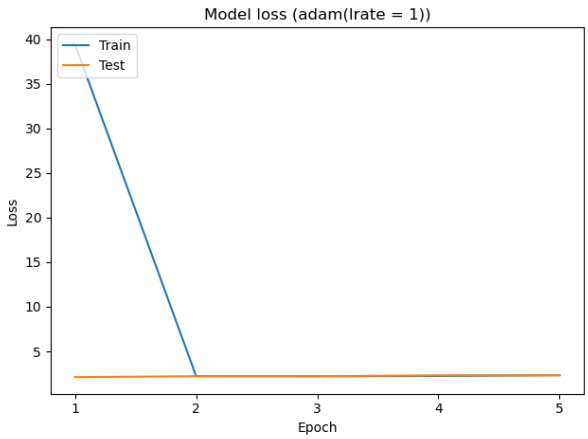
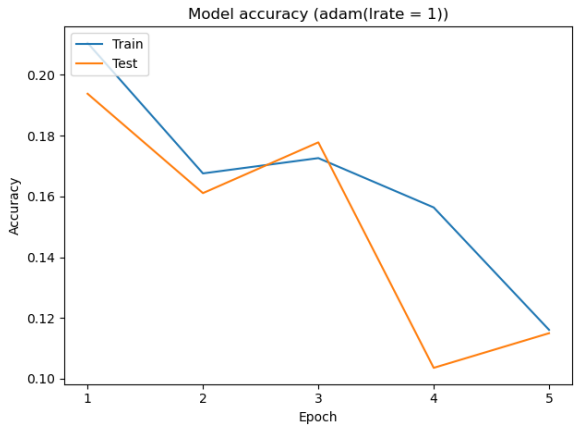
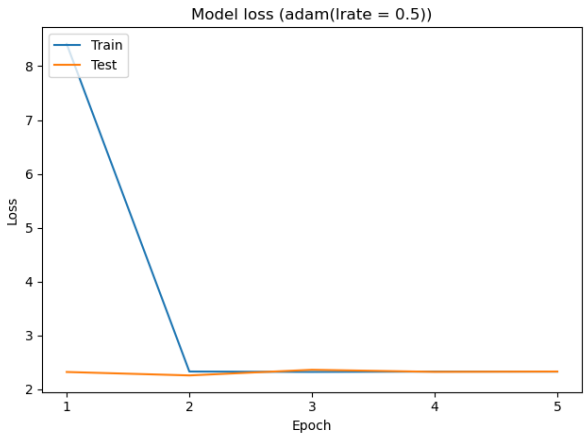
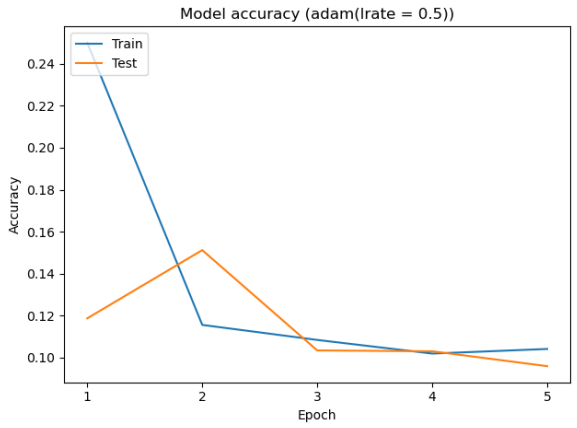
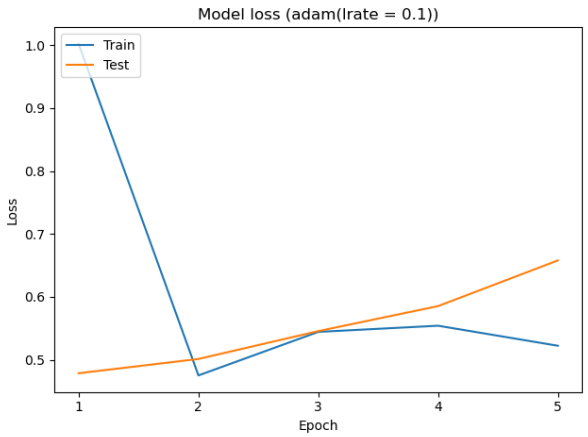
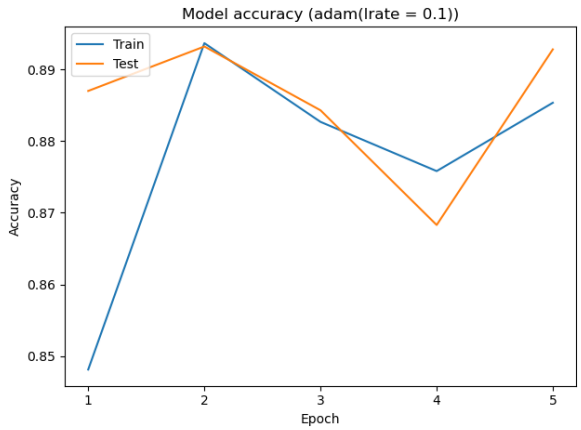
Все оптимизаторы были исследованы на одинаковой модели нейронной сети (естественно за исключением самого оптимизатора модели).

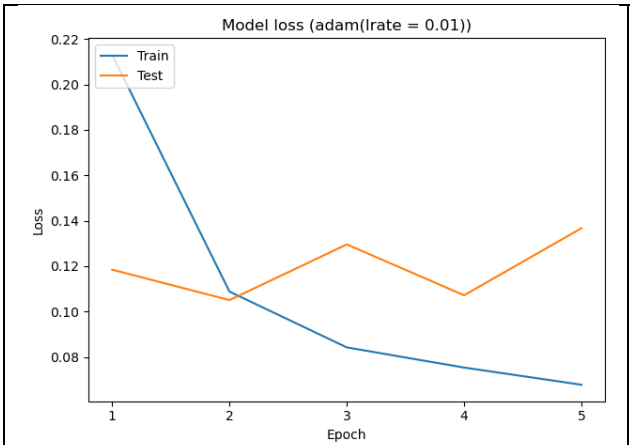
- Инициализация весов – normal,
- Epochs = 5,
- batch_size = 100,

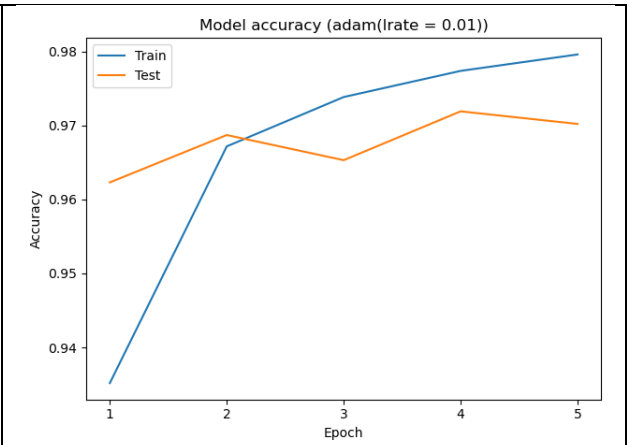
- loss = categorical_crossentropy.

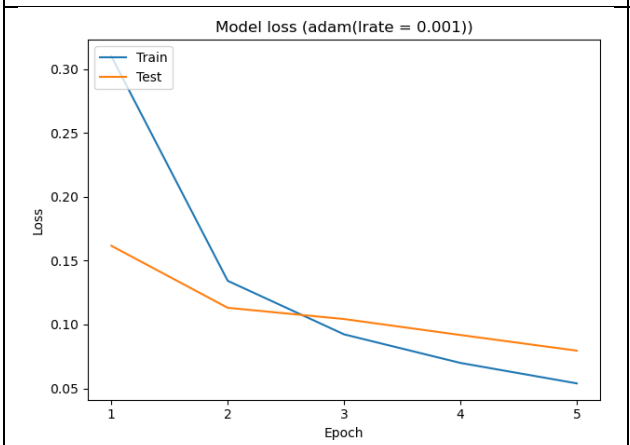
Каждый оптимизатор тестировался на разных значениях скорости обучения и все графики занесены в таблицы для каждого оптимизатора.

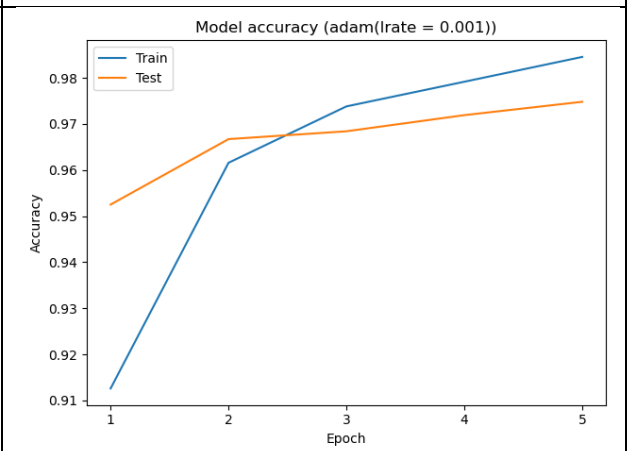
1. **Оптимизатор adam.**

График ошибки	График точности
	
	
	



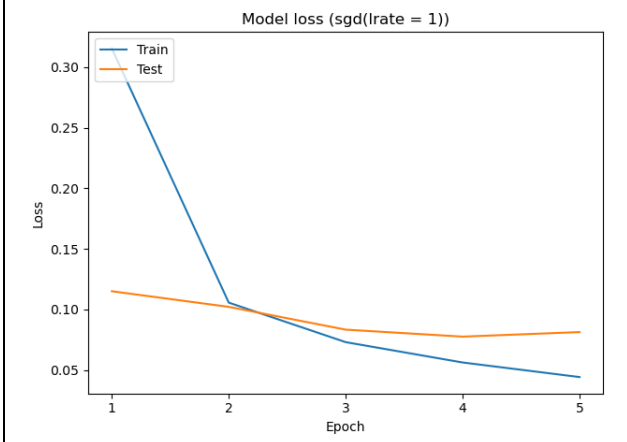


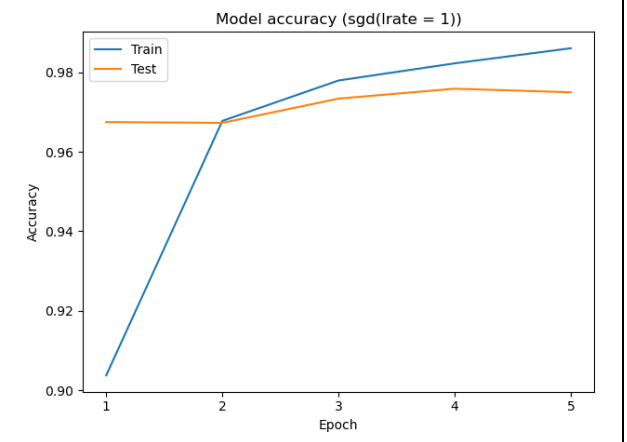


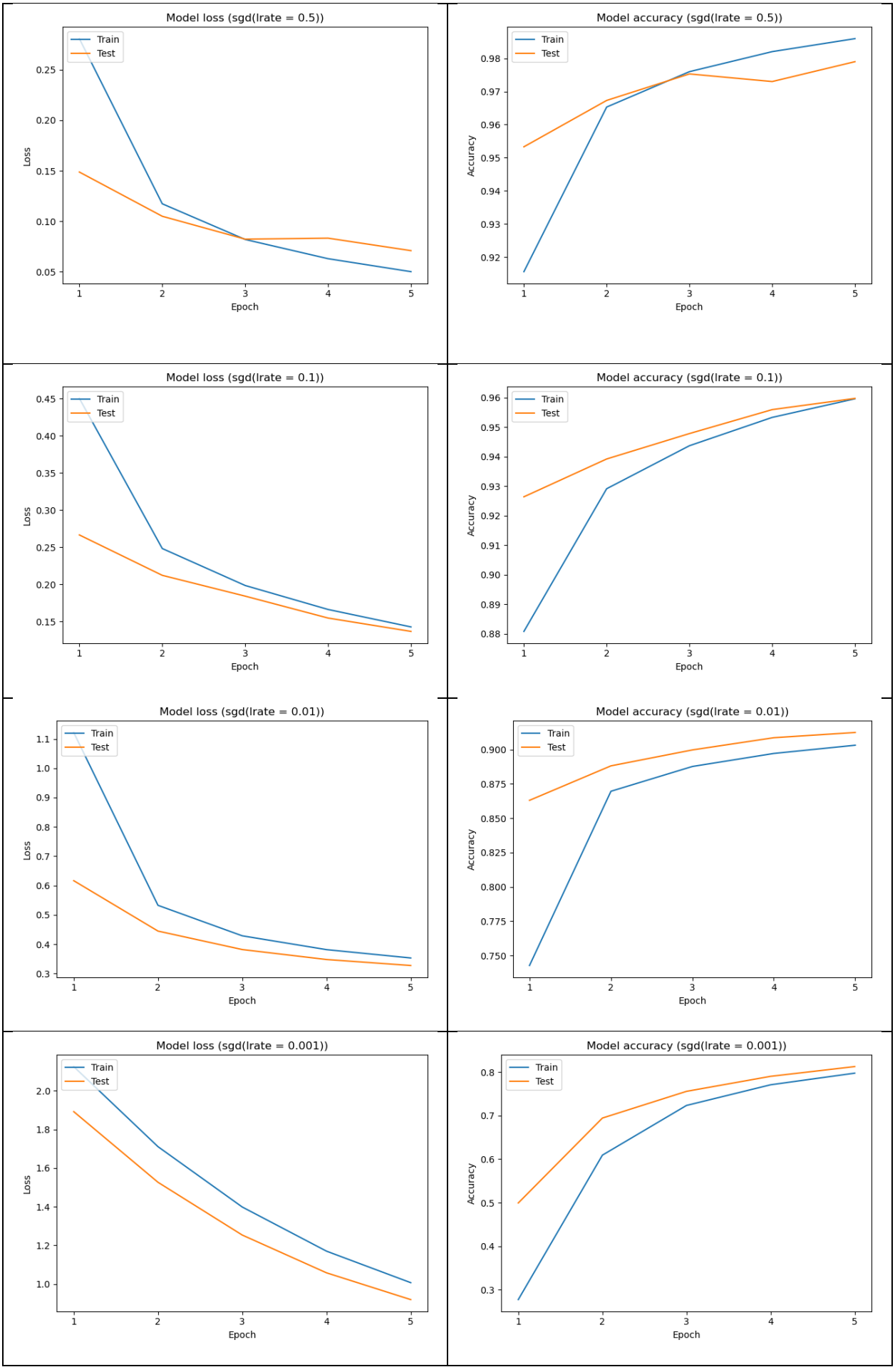


2. **Оптимизатор SGD.**

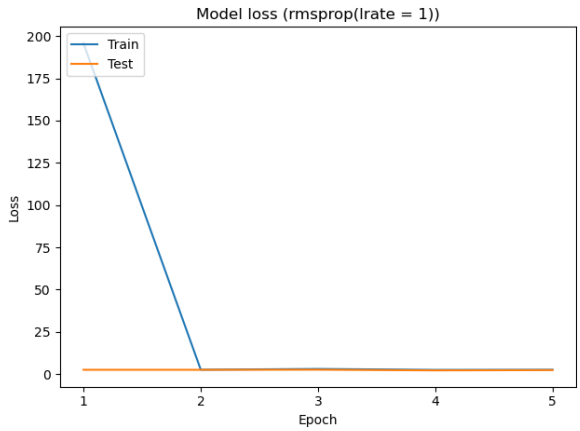
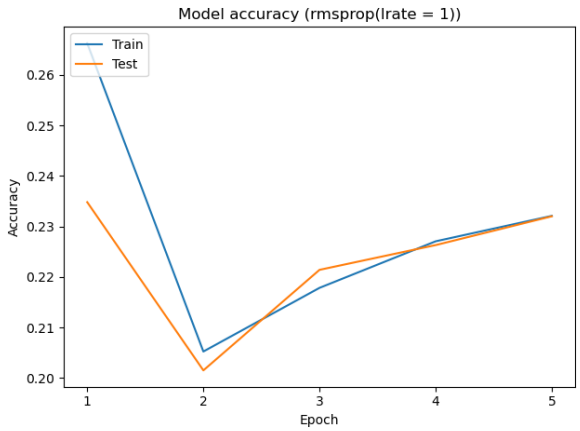
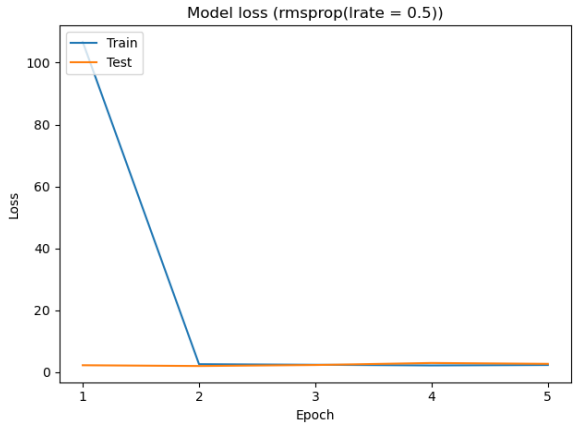
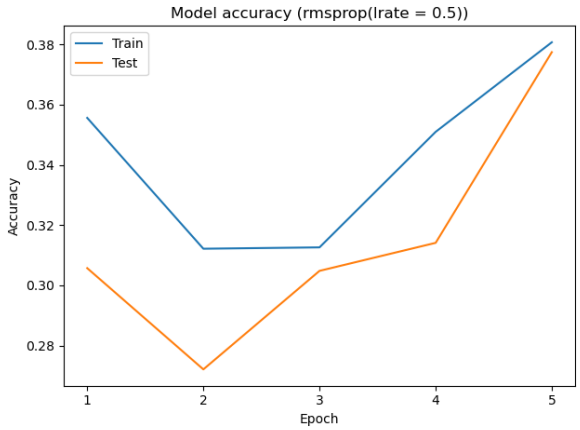
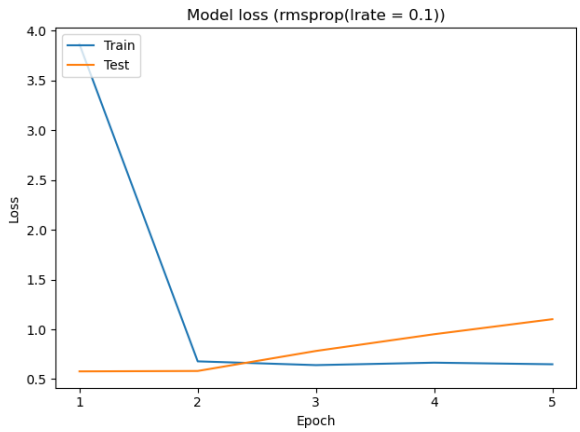
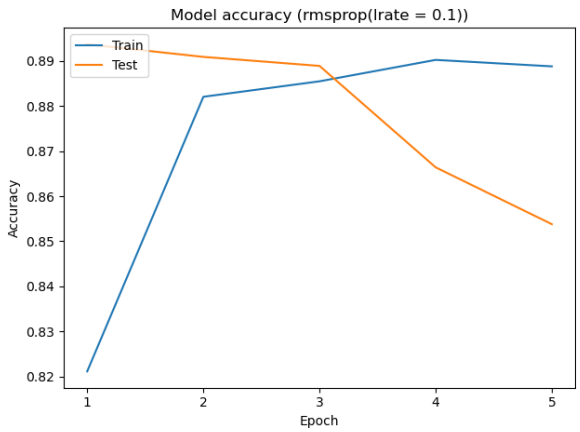
График ошибки	График точности

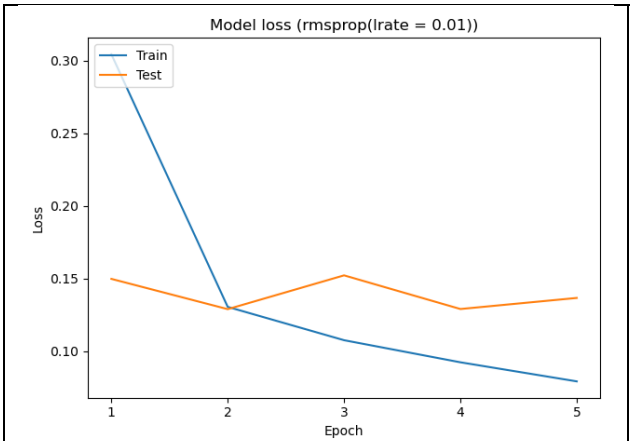


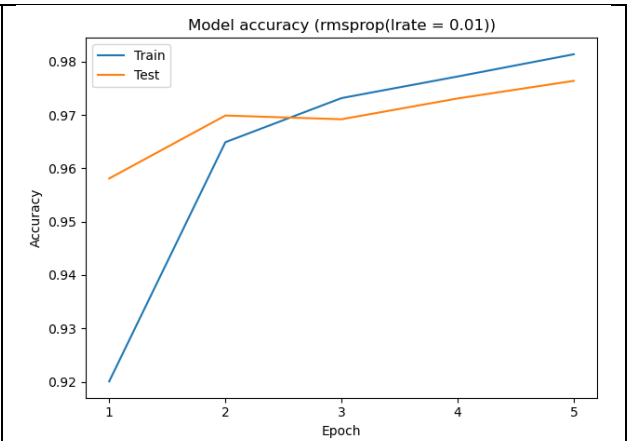


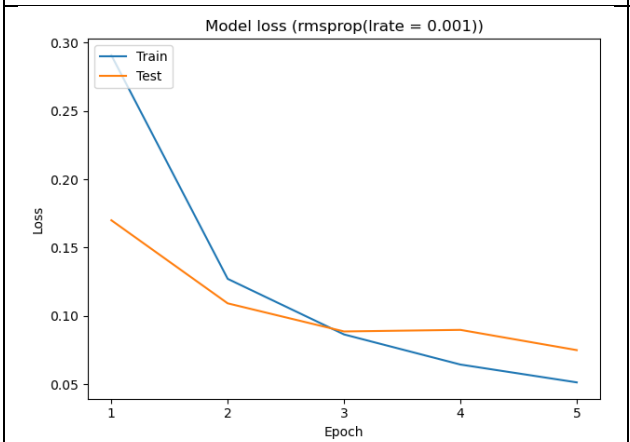

Model loss (sgd(lrate = 0.5))
Train
Test
Loss
Epoch
Model accuracy (sgd(lrate = 0.5))
Train
Test
Accuracy
Epoch
Model loss (sgd(lrate = 0.1))
Train
Test
Loss
Epoch
Model accuracy (sgd(lrate = 0.1))
Train
Test
Accuracy
Epoch
Model loss (sgd(lrate = 0.01))
Train
Test
Loss
Epoch
Model accuracy (sgd(lrate = 0.01))
Train
Test
Accuracy
Epoch
Model loss (sgd(lrate = 0.001))
Train
Test
Loss
Epoch
Model accuracy (sgd(lrate = 0.001))
Train
Test
Accuracy
Epoch

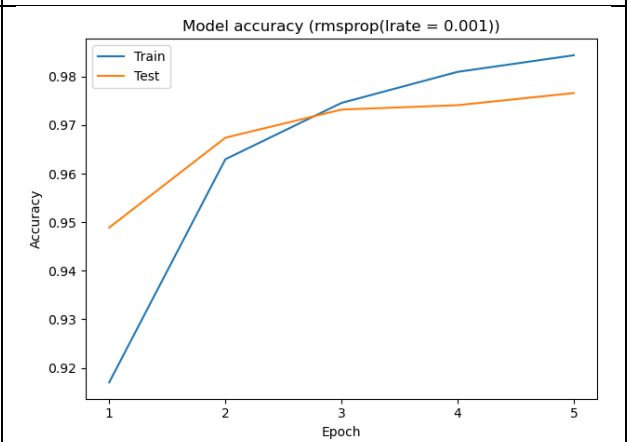
3. **Оптимизатор RMSprop.**

График ошибки	График точности
	
	
	



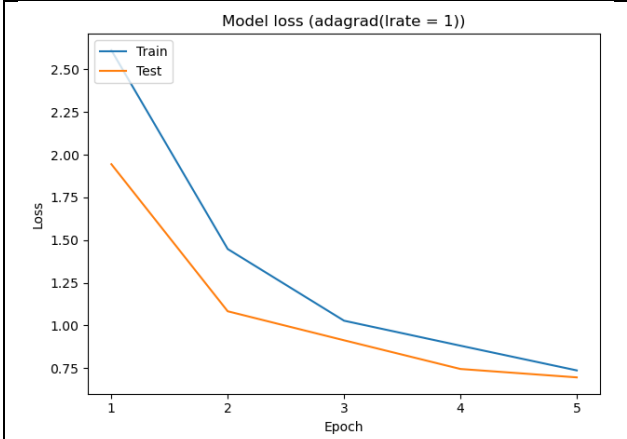


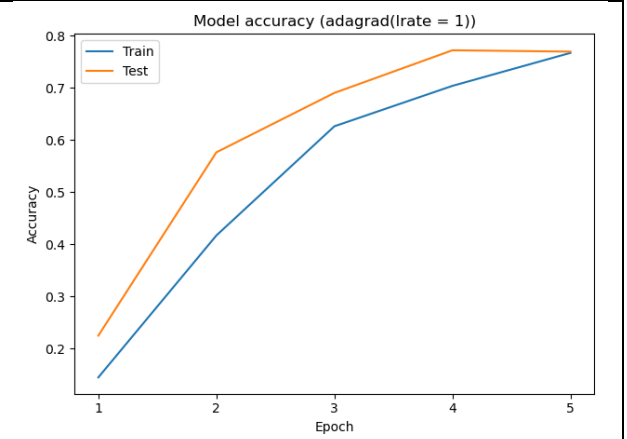


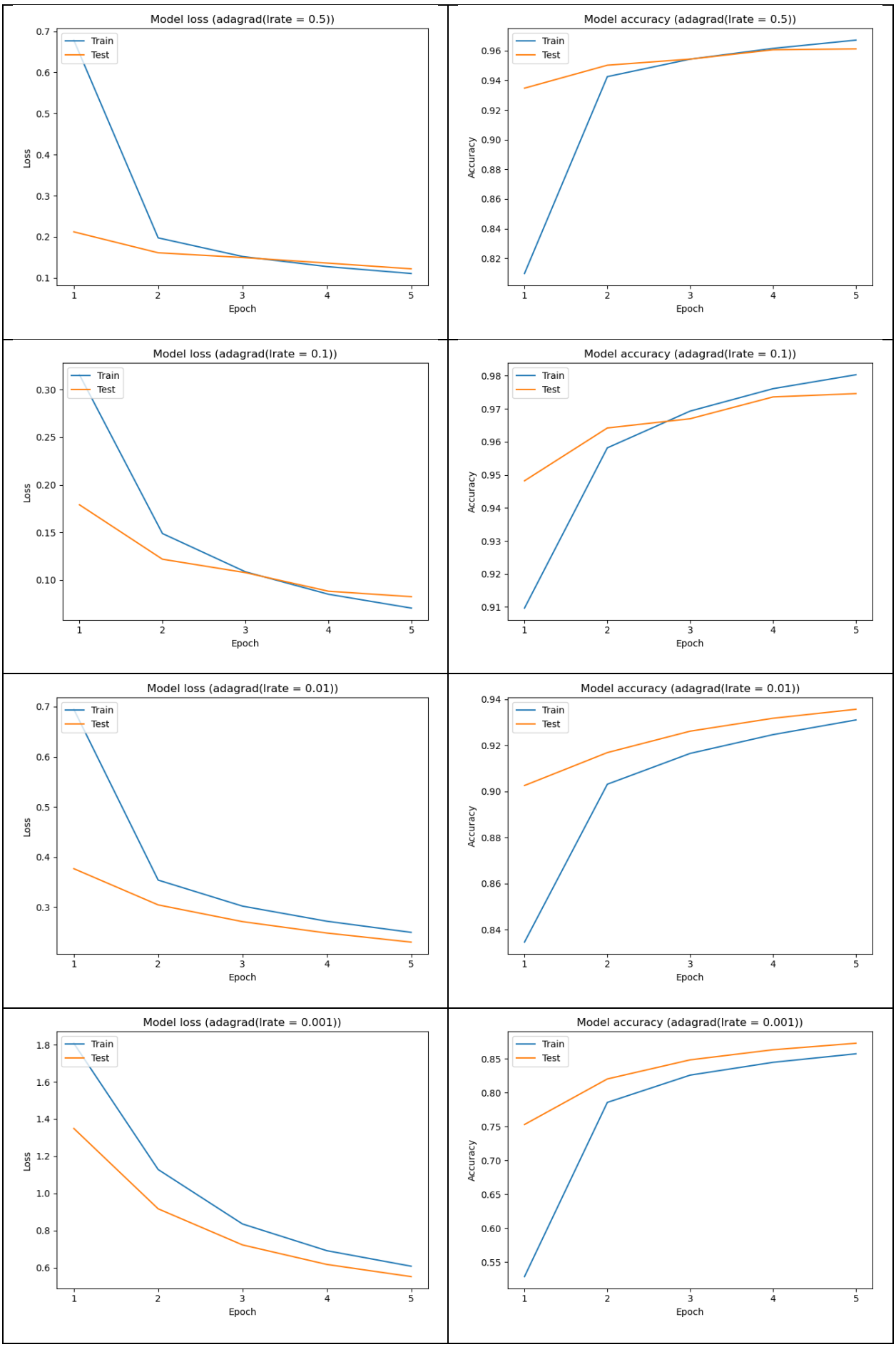


4. Оптимизатор Adagrad.

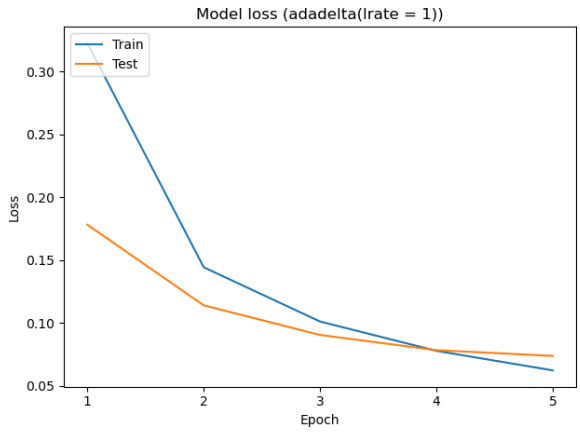
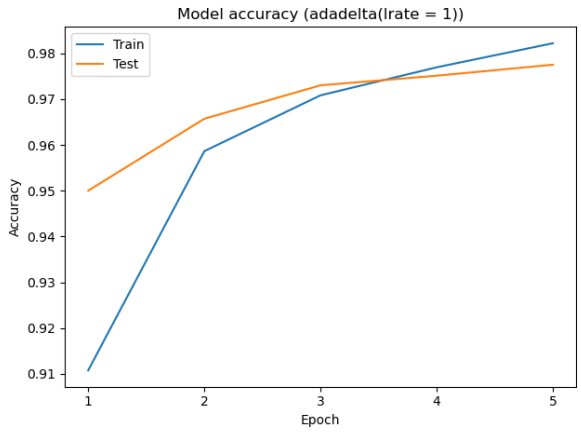
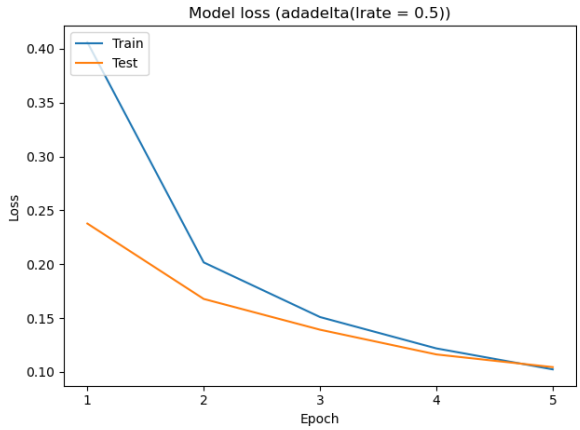
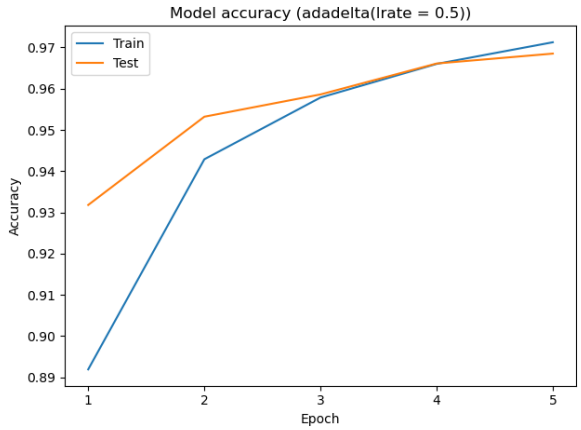
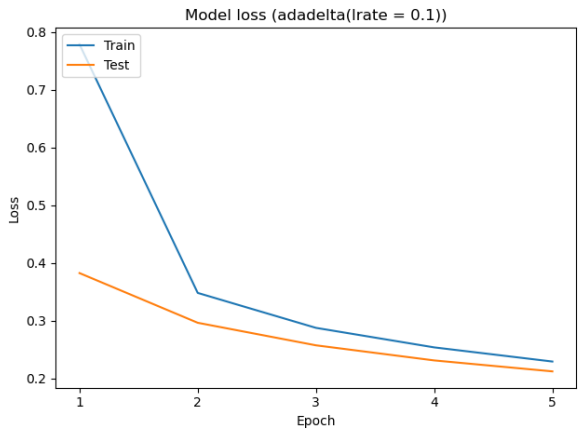
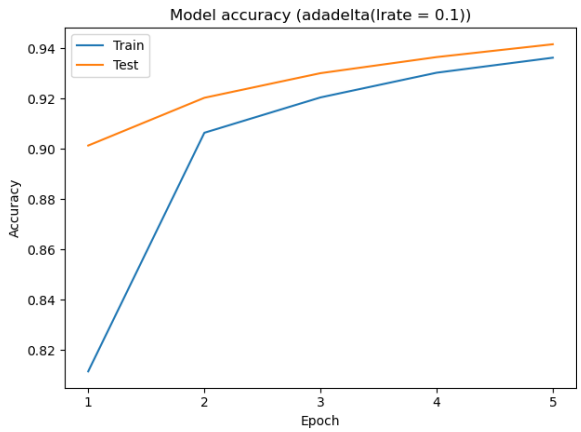
График ошибки	График точности

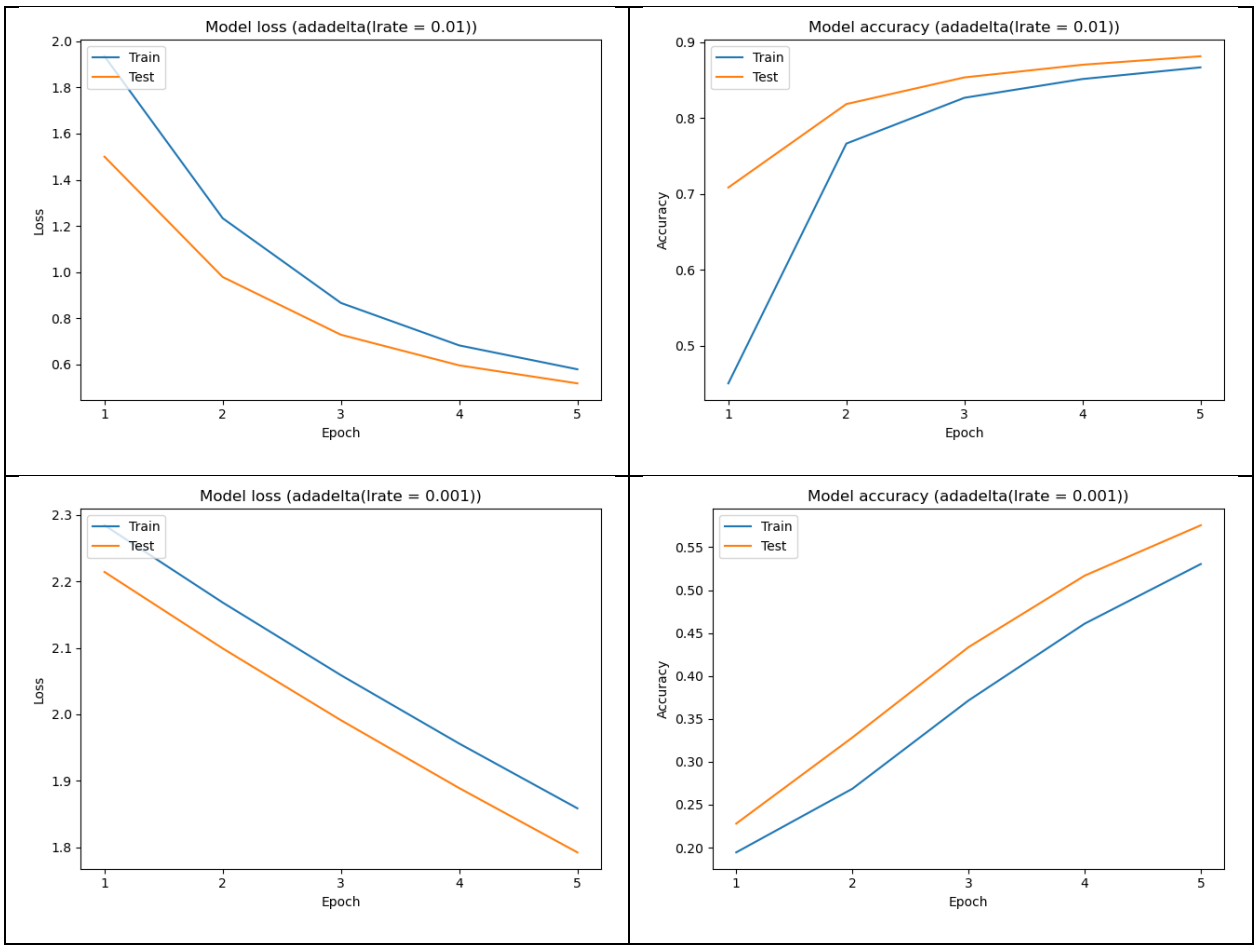





Model loss (adagrad(lrate = 0.5))
Train
Test
Loss
Epoch
Model accuracy (adagrad(lrate = 0.5))
Train
Test
Accuracy
Epoch
Model loss (adagrad(lrate = 0.1))
Train
Test
Loss
Epoch
Model accuracy (adagrad(lrate = 0.1))
Train
Test
Accuracy
Epoch
Model loss (adagrad(lrate = 0.01))
Train
Test
Loss
Epoch
Model accuracy (adagrad(lrate = 0.01))
Train
Test
Accuracy
Epoch
Model loss (adagrad(lrate = 0.001))
Train
Test
Loss
Epoch
Model accuracy (adagrad(lrate = 0.001))
Train
Test
Accuracy
Epoch

5. **Оптимизатор Adadelta.**

График ошибки	График точности
	
	
	



На каждый оптимизатор параметр скорости обучения влияет по-разному. Например, у *Adam* с понижением скорости обучения потери уменьшаются, а точность увеличивается, в то время как у *Adadelta* ситуация противоположная.

В итоге для достижения необходимой точности нам могут подойти следующие конфигурации:

1. Оптимизатор *Adam*, *lrate* = 0.0001.
2. Оптимизатор *SGD*, *lrate* = 1 или 0.5 или 0.1.
3. Оптимизатор *RMSprop*, *lrate* = 0.001 или 0.01.
4. Оптимизатор *Adagrad*, *lrate* = 0.1.
5. Оптимизатор *Adadelta*, *lrate* = 1 или 0.5.

Напишем функцию, которая позволит загружать пользовательское изображение не из датасета. Для загрузки будем использовать функцию image_load из библиотеки Keras:

```
def read_img(self, path):
      img = image.load_img(path=path, color_mode = "grayscale",
target_size=(28, 28, 1))
        img = image.img_to_array(img)
        img = 1 - img/255
        plt.imshow(img.reshape((28, 28)), cmap=plt.cm.binary)
        plt.show()
        img = img.reshape((1, 28, 28))
        return img
```

Данная функция считывает изображение в оттенках серого, затем переводит изображение в негатив и нужный размер массива, выводит полученное изображение и возвращает массив деленный на 255.

Выводы.

В ходе работы была изучена задача классификации рукописных цифр с помощью базы данных MINIST. Подобраны архитектуры, дающие точность свыше 95%. Также была написана функция загрузки изображения в память программы.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

```
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten, Dense
from keras.preprocessing import image
from tensorflow.keras.optimizers import *
import pylab

class lab4:
    def __init__(self):
        mnist = tf.keras.datasets.mnist
        (self.train_images, self.train_labels), (self.test_images,
self.test_labels) = mnist.load_data()
        self.train_images = self.train_images.reshape(60000, 784)
        self.test_images = self.test_images.reshape(10000, 784)
        self.train_images = self.train_images / 255.0
        self.test_images = self.test_images / 255.0
        self.train_labels = to_categorical(self.train_labels)
        self.test_labels = to_categorical(self.test_labels)
        self.epochs = 7
        self.optimizers = {'adam': Adam, 'sgd': SGD, 'rmsprop':
RMSprop, 'adagrad': Adagrad, 'adadelta': Adadelta}

    def read_img(self, path):
        img = image.load_img(path=path, color_mode = "grayscale",
target_size=(28, 28, 1))
        img = image.img_to_array(img)
        img = 1 - img/255
        plt.imshow(img.reshape((28, 28)), cmap=plt.cm.binary)
        plt.show()
        img = img.reshape((1, 28, 28))
        return img

    def build_model(self, optimizer, opt_name, rate=0.001):
        model = Sequential()
        model.add(Flatten())
        model.add(Dense(256, activation='relu'))
        model.add(Dense(10, activation='softmax'))
        model.compile(optimizer=optimizer,
loss='categorical_crossentropy', metrics=['accuracy'])
        history = model.fit(self.train_images, self.train_labels,
epochs=5, batch_size=128,
                            validation_data=(self.test_images,
self.test_labels))
        x = range(1, self.epochs-1)
        plt.plot(x, history.history['loss'])
```

```
        plt.plot(x, history.history['val_loss'])
        plt.title('Model loss' + ' ({}(lrate = {}))'.format(opt_name,
rate))
        plt.ylabel('Loss')
        plt.xlabel('Epoch')
        pylab.xticks(x)
        plt.legend(['Train', 'Test'], loc='upper left')
        plt.show()
        plt.plot(x, history.history['acc'])
        plt.plot(x, history.history['val_acc'])
        plt.title('Model accuracy' + ' ({}(lrate =
{}))'.format(opt_name, rate))
        plt.ylabel('Accuracy')
        plt.xlabel('Epoch')
        pylab.xticks(x)
        plt.legend(['Train', 'Test'], loc='upper left')
        plt.show()
        return model

    def fimd_best_optimizer(self):
        rates = [1, 0.5, 0.1, 0.01, 0.001]
        for opt in self.optimizers.keys():
            for rate in rates:
                model =
self.build_model(self.optimizers[opt](learning_rate=rate), opt, rate)


    def start(self):
        #model = self.build_model(self.optimizers['adam'](), 'adam')
        self.fimd_best_optimizer()
        #image = self.read_img('0.bmp')
        #res = model.predict_classes(image)
        #print(res)

lab = lab4()
lab.start()
```