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. 7383 _____ Рудоман В. А.

Преподаватель _____ Жукова Н.А.

Санкт-Петербург
2020

Цель работы:

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

Задачи.

1. Ознакомиться с представлением графических данных
2. Ознакомиться с простейшим способом передачи графических данных нейронной сети
3. Создать модель ИНС в tf.Keras
4. Настроить параметры обучения
5. Написать функцию, позволяющую загружать изображение пользователя и классифицировать его

Требования.

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

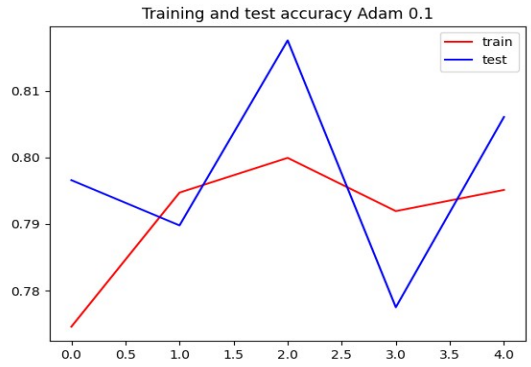
Ход работы.

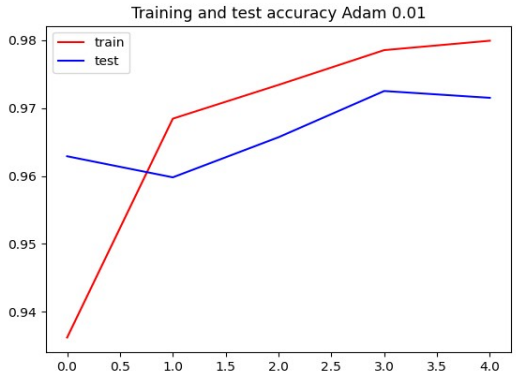
Для изучения влияния различных оптимизаторов на процесс обучения, была разработана и использована программа из приложения А.

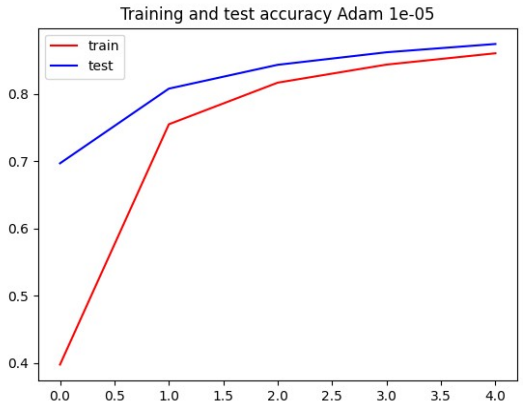
Модель сети: первый слой с 256 нейронами и функцией активации relu, второй слой с 10 нейронами и функцией активации softmax. Чтобы исследовать влияние различных оптимизаторов попробуем их следующие разновидности: Adam, RMSprop, SGD, Adagrad, Adadelta.

Ниже приведены графики точности различных оптимизаторов с разным значением параметра learning rate (0.1, 0.01, 0.001, 0.0001, 0.00001).

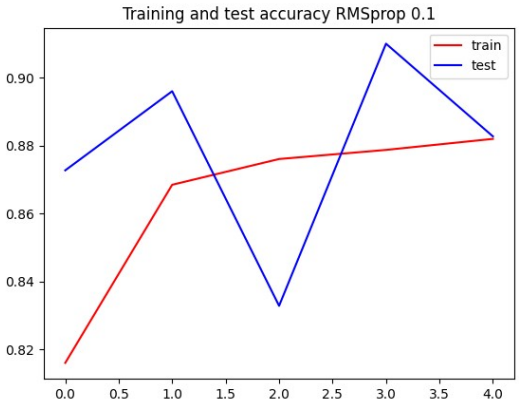
Графики точности для оптимизатора Adam

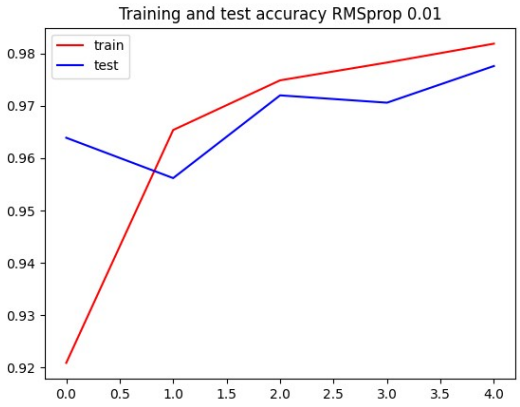


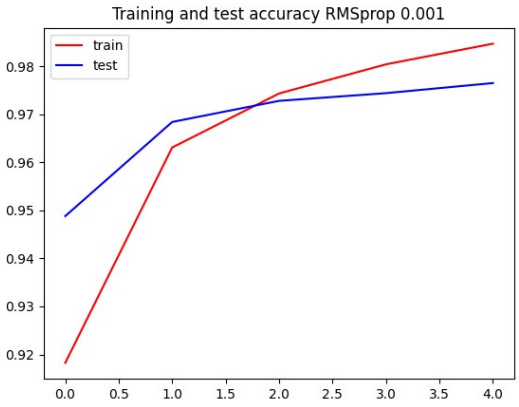


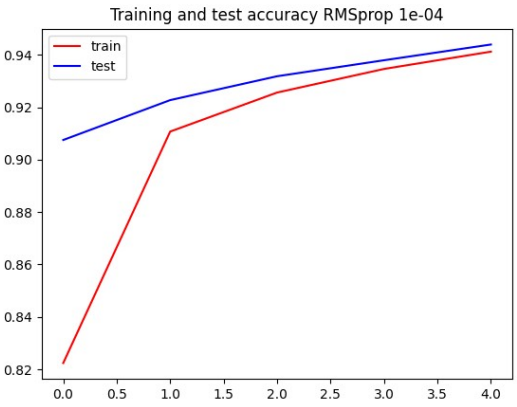


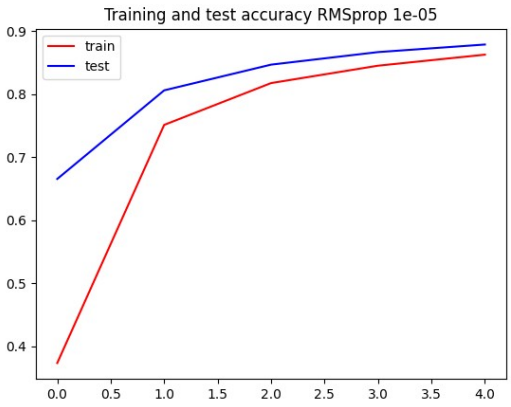
Графики точности для оптимизатора RMSprop



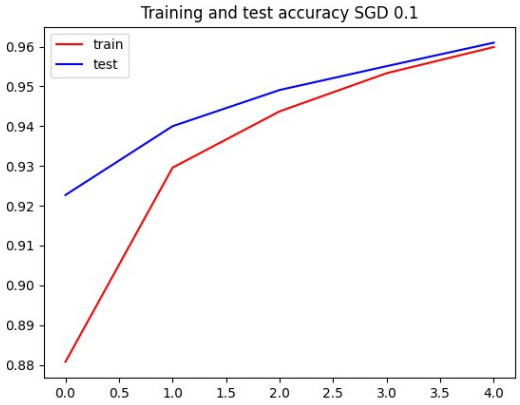


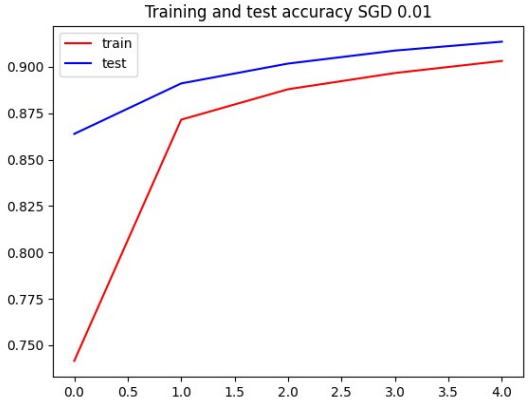


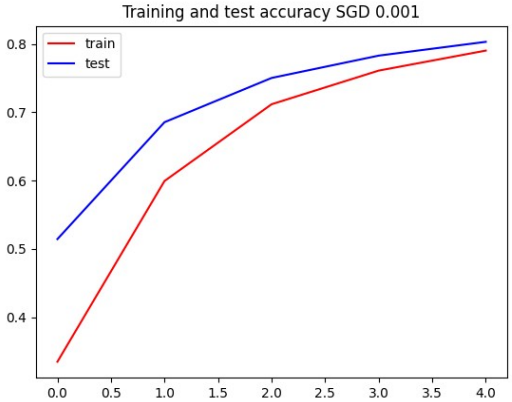


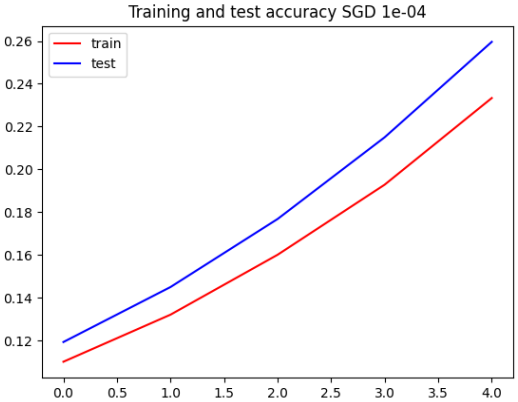


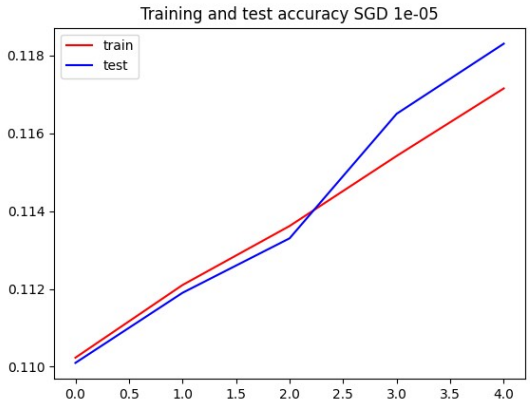
Графики точности для оптимизатора SGD



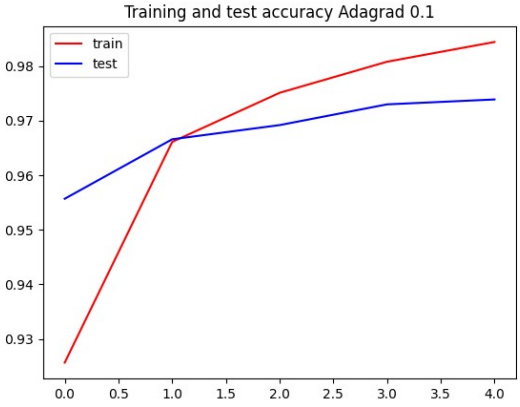


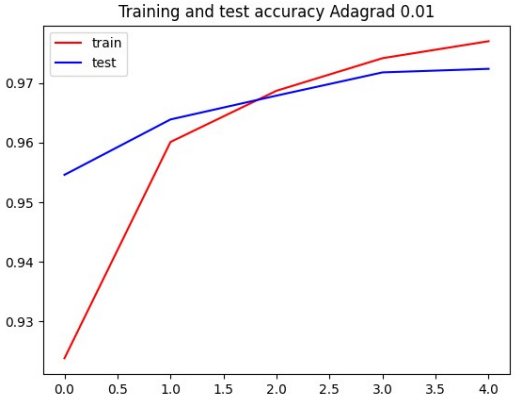


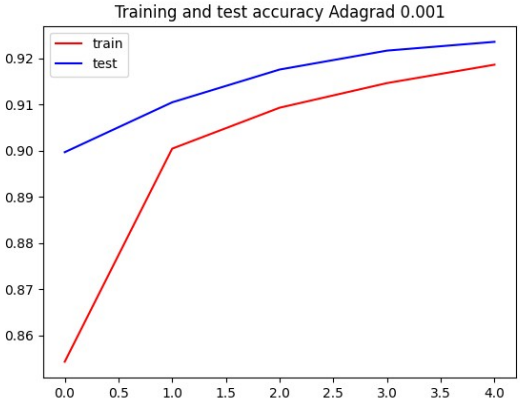


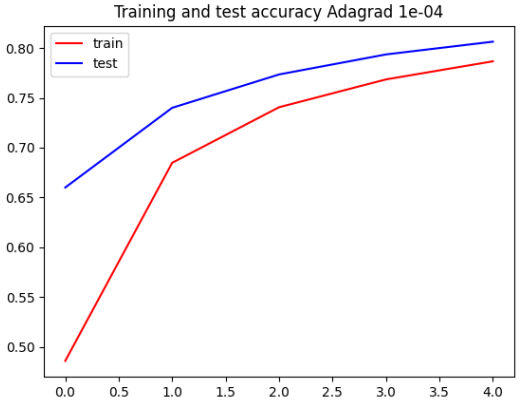


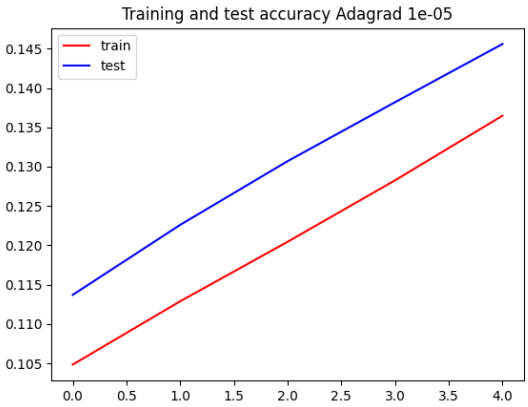
Графики точности для оптимизатора Adagrad



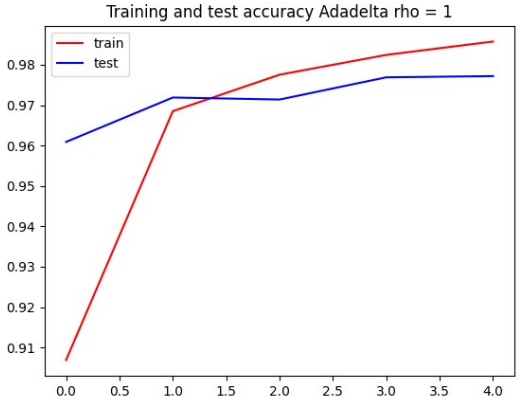


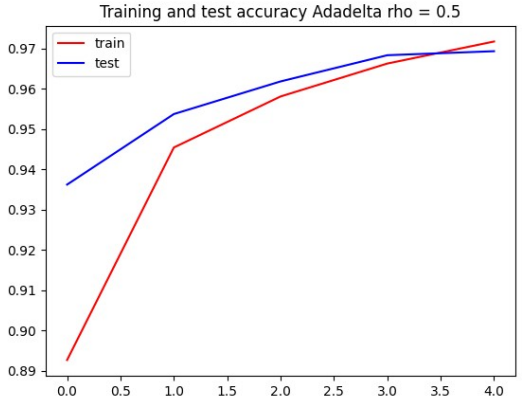


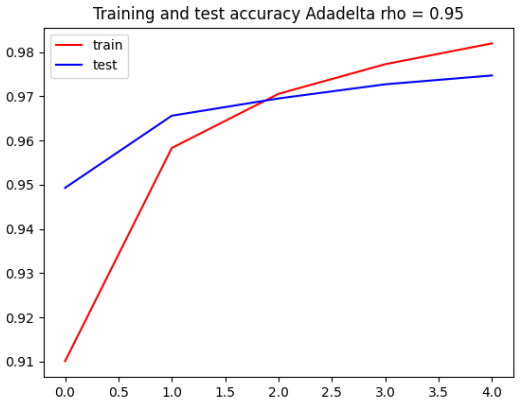


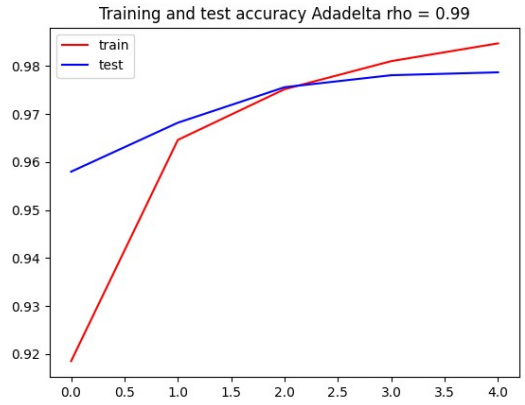


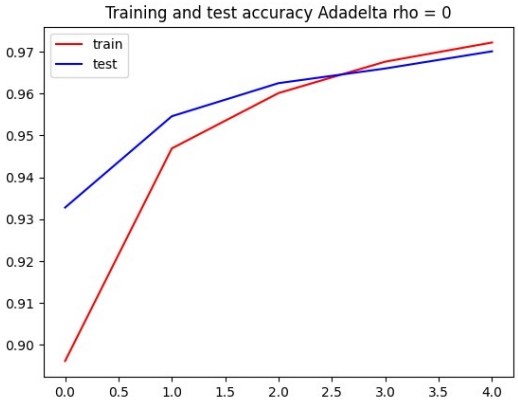
Для оптимизатора Adadelta рекомендуется менять параметр rho. Но при его изменении не было выявлено значительных различий в точности. При этом, каждая конфигурация оптимизатора выдала точность свыше 95%











Вывод.

В ходе выполнения лабораторной работы была создана простейшая сеть, распознающая рукописные символы. Было исследовано влияние оптимизаторов и их параметров на обучение сети. Были выявлены архитектуры сети, при которых точность классификации не менее 95%.

Приложение А

```
import tensorflow as tf
import keras.backend as kb
import matplotlib.pyplot as plt
import pylab
from keras.utils import to_categorical
from keras.layers import Dense, Activation, Flatten
from keras.models import Sequential
import matplotlib
import numpy as np
from PIL import Image
from keras import optimizers

mnist = tf.keras.datasets.mnist
(train_images, train_labels),(test_images, test_labels) =
mnist.load_data()



train_images = train_images / 255.0
test_images = test_images / 255.0

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

def build_model():
    model = Sequential()
    model.add(Flatten())
    model.add(Dense(256, activation='relu'))
    model.add(Dense(10, activation='softmax'))
    return model

def test_optimizer(optimizer, name):
    model = build_model()
    #optimizer = optimizers.Adadelta(rho=0.9)
    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))
    test_loss, test_acc = model.evaluate(test_images, test_labels)
```

```
    print('test_acc:', test_acc)
    print('test_loss:', test_loss)
    res_train_data.append(history.history['accuracy'][-1])
    res_test_data.append(test_acc)
    plt.title('Training and test accuracy '+name+"
"+str(kb.eval(model.optimizer.lr)))
    plt.plot(history.history['accuracy'], 'r', label='train')
    plt.plot(history.history['val_accuracy'], 'b', label='test')
    plt.legend()
    plt.savefig(name + '_' + "rho"
+str(kb.eval(model.optimizer.lr))
+"_acc"+str(kb.eval(model.optimizer.lr)) + '.png')
    plt.clf()

    plt.title('Training and test loss '+name+"
"+str(kb.eval(model.optimizer.lr)))
    plt.plot(history.history['loss'], 'r', label='train')
    plt.plot(history.history['val_loss'], 'b', label='test')
    plt.legend()
    plt.savefig(name + '_' + "rho"
+str(kb.eval(model.optimizer.lr))
+"_loss"+str(kb.eval(model.optimizer.lr)) + '.png')
    plt.clf()

def get_img(filename):
    image = Image.open(filename).convert('L')
    image = image.resize((28, 28))
    image = np.array(image)
    image = image/255
    return np.expand_dims(image, axis=0)

optimizerslist = [optimizers.Adam, optimizers.RMSprop,
optimizers.SGD, optimizers.Adagrad, optimizers.Adadelta]
learning_rates = [0.1, 0.01, 0.001, 0.0001, 0.00001]
for learn_rt in learning_rates:
    for optimizer in optimizerslist:
        test_optimizer(optimizer(learning_rate=learn_rt),
optimizer.__name__)
```