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

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Искусственные нейронные сети»

Тема: Распознавание рукописных символов

Студент гр. 7383 _____ Власов Р.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).

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

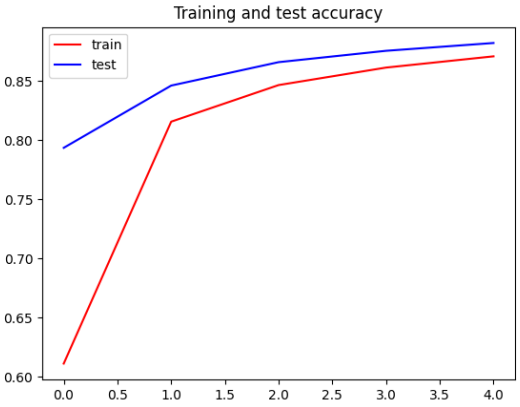
Ход работы.

Для исследования была разработана и использована программа, код которой приведен в приложении А.

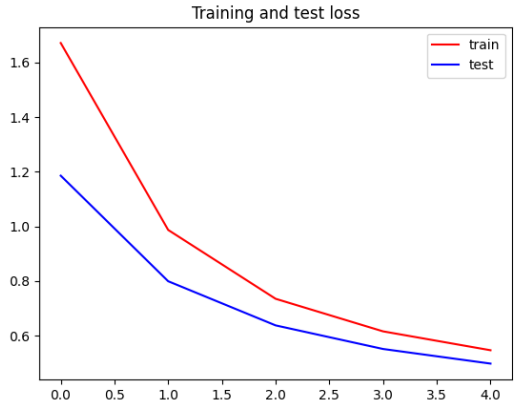
Для исследования будут рассмотрены оптимизаторы Adagrad, Adam, RMSprop и SGD со скоростью обучения 0.001, 0.01 и 0.1. Все исследования будут проходить при 5 эпохах обучения.

Оптимизатор Adagrad

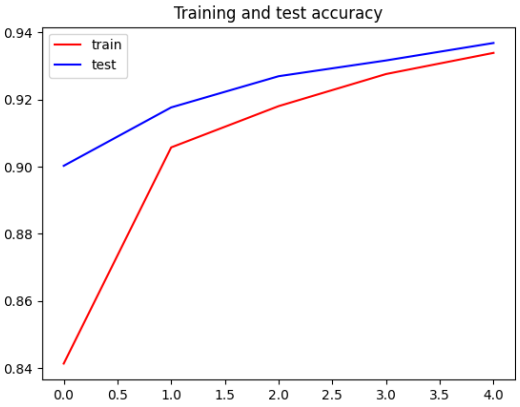
Результаты исследования для оптимизатора Adagrad представлены на рис. 1. Из графиков видно, что результаты работы сети улучшаются с увеличением скорости обучения. Наилучшая точность, равная 0.975, для оптимизатора Adagrad была достигнута при скорости обучения, равной 0.1.



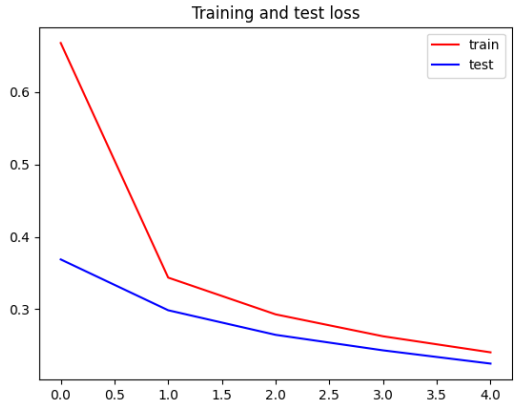
a) accuracy learning_rate 0.001



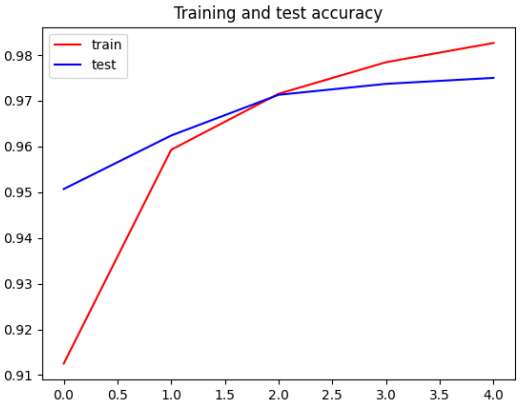
b) loss learning_rate 0.001



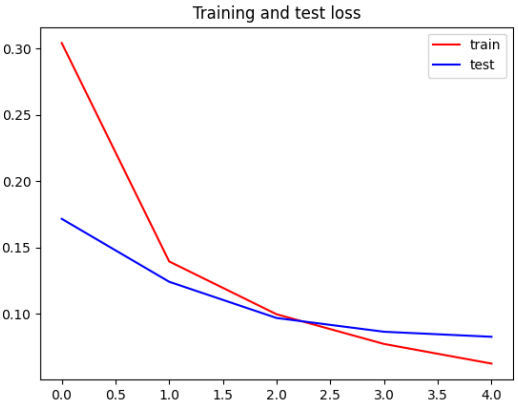
c) accuracy learning_rate 0.01



d) loss learning_rate 0.01



e) accuracy learning_rate 0.1



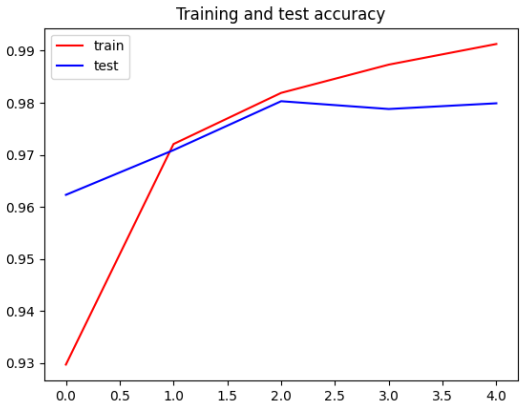
f) loss learning_rate 0.1

Рисунок 1 — Результаты исследования оптимизатора Adagrad

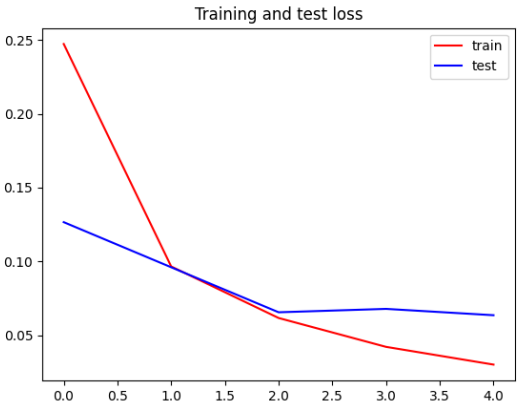
Оптимизатор Adam

Результаты исследования для оптимизатора Adam представлены на рис. 2. Из результатов видно, что результаты работы сети ухудшаются с

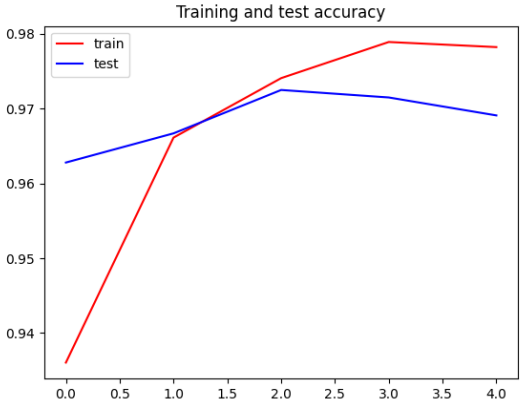
увеличением скорости обучения. Наилучшая точность, равная 0.9799, для оптимизатора Adam была достигнута при скорости обучения, равной 0.001.



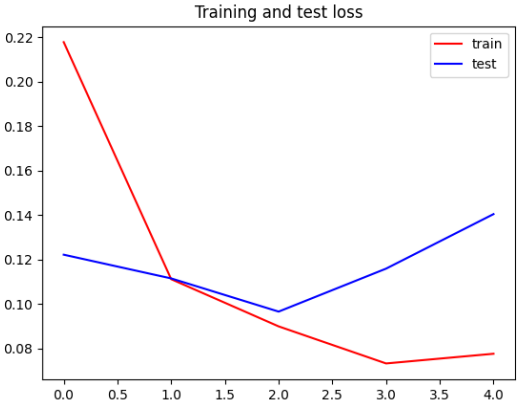
a) accuracy learning_rate 0.001



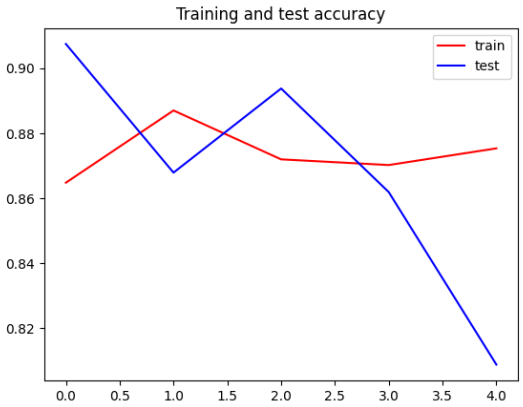
b) loss learning_rate 0.001



c) accuracy learning_rate 0.01



d) loss learning_rate 0.01



e) accuracy learning_rate 0.1

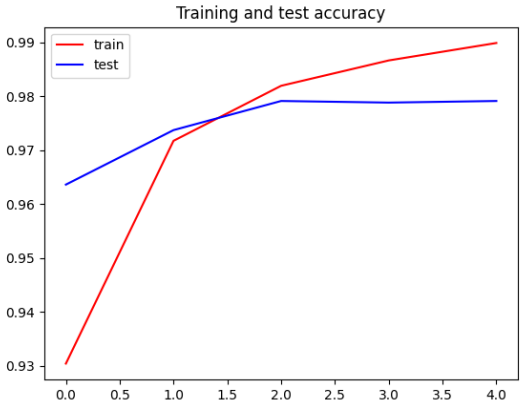
f) loss learning_rate 0.1

Рисунок 2 — Результаты исследования оптимизатора Adam

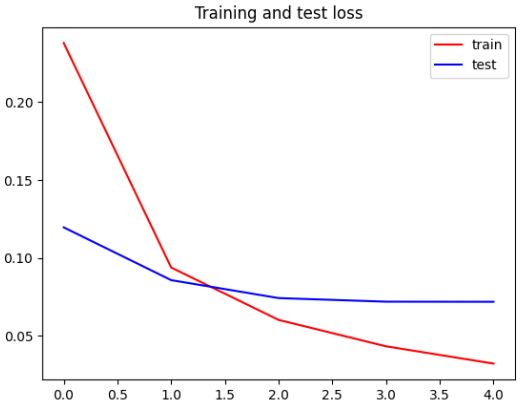
Оптимизатор RMSprop

Результаты исследования для оптимизатора RMSprop представлены на рис. 3. Из результатов видно, что результаты работы сети ухудшаются с

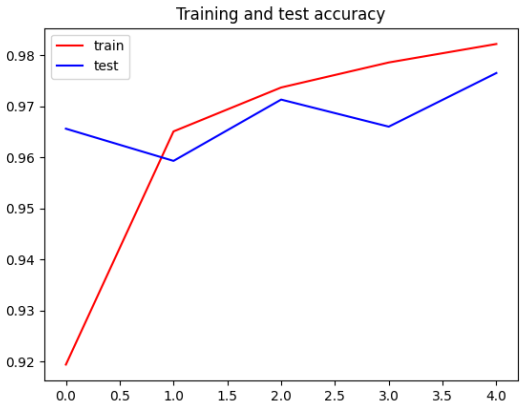
увеличением скорости обучения. Наилучшая точность, равная 0.9791, для оптимизатора RMSprop была достигнута при скорости обучения, равной 0.001.



a) accuracy learning_rate 0.001

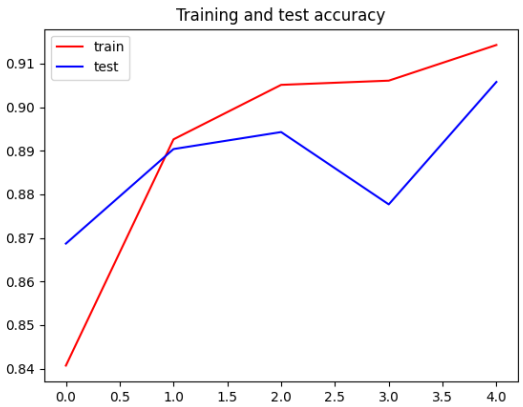


b) loss learning_rate 0.001

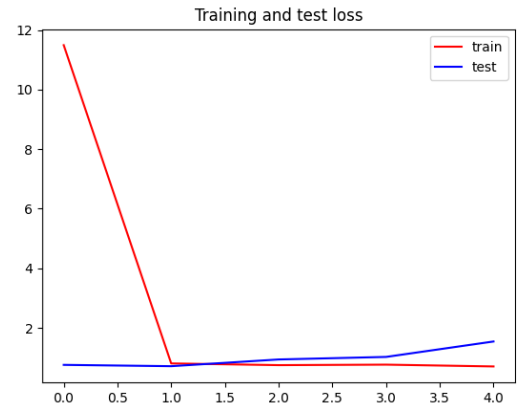


c) accuracy learning_rate 0.01

d) loss learning_rate 0.01



e) accuracy learning_rate 0.1



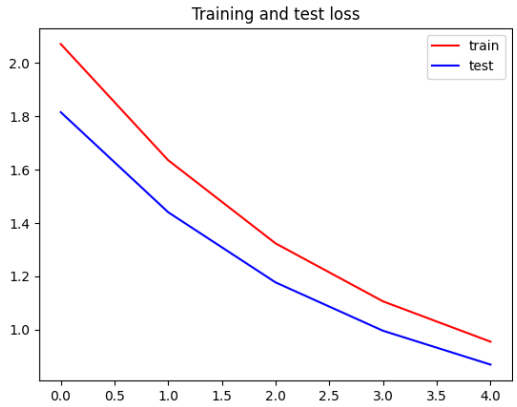
f) loss learning_rate 0.1

Рисунок 3 — Результаты исследования оптимизатора RMSprop

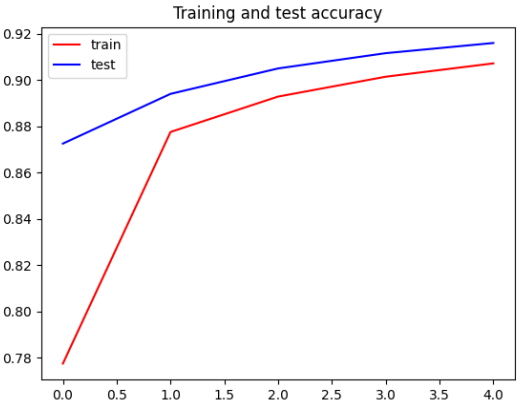
Оптимизатор SGD

Результаты исследования для оптимизатора SGD представлены на рис. 4. Из результатов видно, что результаты работы сети улучшаются с увеличением скорости обучения. Наилучшая точность, равная 0.9626, для оптимизатора SGD была достигнута при скорости обучения, равной 0.1.

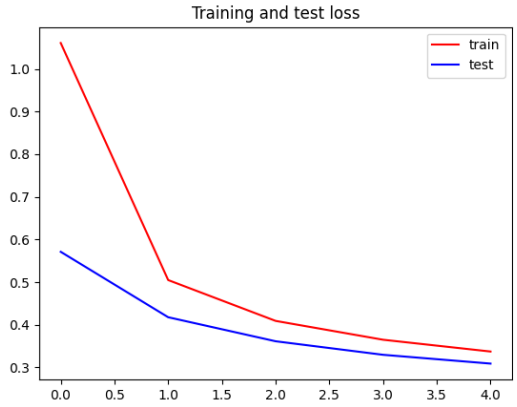
a) accuracy learning_rate 0.001



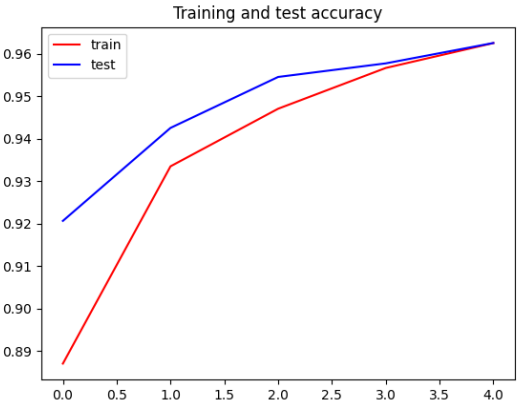
b) loss learning_rate 0.001



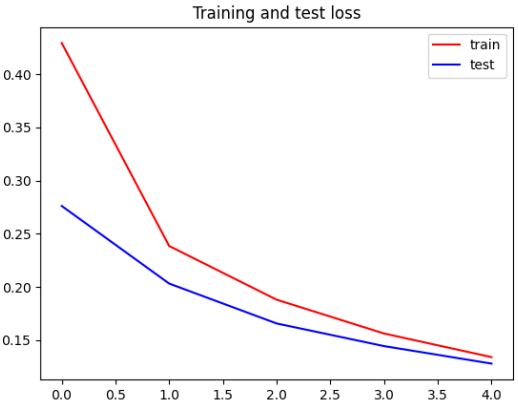
c) accuracy learning_rate 0.01



d) loss learning_rate 0.01



e) accuracy learning_rate 0.1



f) loss learning_rate 0.1

Рисунок 4 — Результаты исследования оптимизатора SGD

Из исследований видно, что показатели сетей с оптимизаторами Adagrad и SGD улучшаются, а показатели сетей с оптимизаторами Adam и RMSprop ухудшаются при увеличении скорости обучения. Наилучшим образом себя показала сеть с оптимизатором Adam и скоростью обучения 0.001 — её точность составила 0.9799

Для предсказания по пользовательскому изображению была реализована функция predict, принимающая путь до изображения и модель, с помощью которой необходимо сделать предсказание. Чтение и преобразоание в нужный формат изображения осуществляет функция loadImage.

Выводы.

В ходе выполнения лабораторной работы была создана сеть для распознавания рукописных символов. Было исследовано влияние оптимизаторов Adagrad, Adam, RMSprop и SGD и их параметров на результат обучения. Лучший результат был достигнут с оптимизатором Adam со скоростью обучения 0.001. Также была реализована функция, позволяющая загрузить пользовательское изображение и получить предсказание от обученной модели.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

```
import tensorflow as tf
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.layers import Dense
from tensorflow.keras.models import Sequential
import matplotlib.pyplot as plt
from tensorflow.keras import optimizers
from PIL import Image
import numpy as np

EPOHS = 5
IMAGE_WIDTH = 28
IMAGE_HEIGHT = 28


def buildModel(optimizer):
    model = Sequential()
    model.add(Dense(IMAGE_HEIGHT * IMAGE_WIDTH,
activation='relu', input_shape=(IMAGE_HEIGHT * IMAGE_WIDTH,)))
    model.add(Dense(10, activation='softmax'))
    model.compile(optimizer=optimizer,
loss='categorical_crossentropy', metrics=['accuracy'])
    return model


def loadData():
    mnist = tf.keras.datasets.mnist
    (train_images, train_labels), (test_images, test_labels) =
mnist.load_data()
    train_images = train_images.reshape((60000, IMAGE_HEIGHT *
IMAGE_WIDTH))
    train_images = train_images / 255.0
    test_images = test_images.reshape((10000, IMAGE_HEIGHT *
IMAGE_WIDTH))
    test_images = test_images / 255.0
    train_labels = to_categorical(train_labels)
    test_labels = to_categorical(test_labels)
    return train_images, train_labels, test_images, test_labels


def predict(path, model):
    return np.argmax(model.predict(loadImage(path)))


def loadImage(path):
    image = Image.open(path)
    image = image.resize((IMAGE_HEIGHT, IMAGE_WIDTH))
```

```
    image = np.dot(np.asarray(image), np.array([1 / 3, 1 / 3,
1 / 3]))
    image /= 255
    image = 1 - image
    image = image.reshape((1, IMAGE_HEIGHT * IMAGE_WIDTH))
    return image


def trainModel(optimizer, epohs):
    optimizerConf = optimizer.get_config()
    print(
        "Researching with optimizer %s with learning rate %s" %
(optimizerConf["name"], optimizerConf["learning_rate"]))

    model = buildModel(optimizer)
    history = model.fit(train_images, train_labels,
epochs=epohs,
                        batch_size=128,
validation_data=(test_images, test_labels))

    test_loss, test_acc = model.evaluate(test_images,
test_labels)

    plt.title('Training and test accuracy')
    plt.plot(history.history['accuracy'], 'r', label='train')
    plt.plot(history.history['val_accuracy'], 'b', label='test')
    plt.legend()
    plt.savefig("Graphics/%s%s_acc.png" %
(optimizerConf["name"], optimizerConf["learning_rate"]),
format='png')
    plt.clf()

    plt.title('Training and test loss')
    plt.plot(history.history['loss'], 'r', label='train')
    plt.plot(history.history['val_loss'], 'b', label='test')
    plt.legend()
    plt.savefig("Graphics/%s%s_loss.png" %
(optimizerConf["name"], optimizerConf["learning_rate"]),
format='png')
    plt.clf()

    result["%s%s" % (optimizerConf["name"],
optimizerConf["learning_rate"])] = test_acc
    return model


train_images, train_labels, test_images, test_labels =
loadData()

result = dict()
```

```
for learning_rate in [0.001, 0.01, 0.1]:
    trainModel(optimizers.Adagrad(learning_rate=learning_rate),
EPOHS)
    trainModel(optimizers.Adam(learning_rate=learning_rate),
EPOHS)
    trainModel(optimizers.RMSprop(learning_rate=learning_rate),
EPOHS)
    trainModel(optimizers.SGD(learning_rate=learning_rate),
EPOHS)

for res in result:
    print("%s: %s" % (res, result[res]))
```