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

ОТЧЕТ

по лабораторной работе №4
"Распознавание рукописных символов"
по дисциплине «Искусственные нейронные сети»

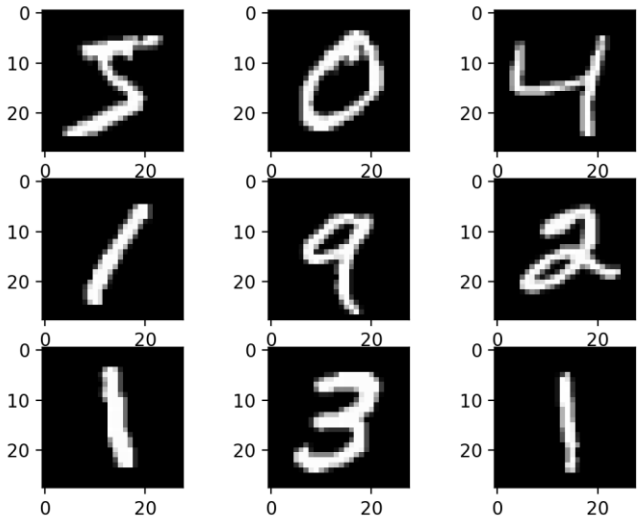
Студент гр. 8382 _____ Облизанов А.Д.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2021

Цель.

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



Задание.

- Ознакомиться с представлением графических данных
- Ознакомиться с простейшим способом передачи графических данных нейронной сети
- Создать модель
- Настроить параметры обучения
- Написать функцию, позволяющая загружать изображение пользователи и классифицировать его

Требования:

- 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.

Работа выполнялась на базе операционной системы Windows 10 в среде разработки PyCharm.

Для обучения использовался датасет из Tensorflow MNIST. Набор данных содержит 60,000 изображений для обучения и 10,000 изображений для тестирования. Было выполнено преобразование к категориальному виду классов изображения, а также нормализация значений в массивах (все значения были поделены на 255).

Реализация загрузки пользовательского изображения.

Изображение загружается как объект Image, предоставляемый библиотекой PIL. Изображение загружается в черно-белом формате. Проверяется разрешение рисунка, если оно не соответствует разрешению 28×28, то применяется метод resize, который изменяет разрешение на нужное. Изображение преобразуется в тензор нужного формата с помощью методов библиотеки Numpy. Листинг приведен ниже:

```
def load_image(path):
    img = Image.open(path).convert('L')
    img_data = np.array(img)
    if img_data.size != 784 or len(img_data) != 28:
        img = img.resize((28, 28), Image.ANTIALIAS)
        img_data = np.array(img)
    img_data = img_data / 255.0
    return np.expand_dims(img_data, axis=0)
```

Создание модели, параметры обучения

Была создана модель Sequental(). Добавленные слои (входной слой создается по умолчанию):

- Flatten – слой для «разворачивания» тензора – преобразует матрицу изображения в вектор.
- Dense, 256 нейронов, функция активации Relu

- Dense, 10 нейронов, функция активации Softmax – для классификации 10 цифр.

Для модели были выбраны следующие неизменяемые параметры:

- Функция потерь - categorical_crossentropy. Наиболее подходящая для задачи классификации. Имеет следующую формулу

$$CCE(y,p) = -\sum_{c=1}^{N} y_{i,c} \log(p_{i,c})$$

Здесь N – число классов, $y_{i,c}$ – индикатор верного класса (1 или 0), $p_{i,c}$ – предсказанная вероятность принадлежности классу.

- Число эпох – 10
- Размер батча – 256
- Метрика – точность

В тестировании будет проведено сравнение работы с разными оптимизаторами и их параметрами.

Тестирование.

Список использованных вариантов оптимизаторов и их параметров, а также условные обозначения на графиках для них представлены в табл. 1.

Таблица 1 – Оптимизаторы

Оптимизатор	**Параметры**	**Название графика**
RMSProp	Коэф. скорости обучения: 0.001	RMSProp,lr=.001
RMSProp	Коэф. скорости обучения: 0.01	RMSProp,lr=.01
RMSProp	Коэф. скорости обучения: 0.1	RMSProp,lr=.1
Adam	Коэф. ск. о.: 0.001, $\varepsilon = 10^{-7}$	Adam,lr=.001,e=1e-7
Adam	Коэф. ск. о.: 0.01, $\varepsilon = 10^{-7}$	Adam,lr=.01,e=1e-7
Adam	Коэф. ск. о.: 0.1, $\varepsilon = 10^{-7}$	Adam,lr=.1,e=1e-7
Adam	Коэф. ск. о.: 0.001, $\varepsilon = 10^{-4}$	Adam,lr=.001,e=1e-4
Adamax	Коэф. скорости обучения: 0.001	Adamax,lr=.001
Adamax	Коэф. скорости обучения: 0.01	Adamax,lr=.01

Adamax	Коэф. скорости обучения: 0.1	Adamax,lr=.1
SGD	Коэф. скорости обучения: 0.001	SGD,lr=.001
SGD	Коэф. скорости обучения: 0.01	SGD,lr=.01
SGD	Коэф. скорости обучения: 0.1	SGD,lr=.1

На рис. 1 показаны значения точности при обучении настроенной модели с различными оптимизаторами.

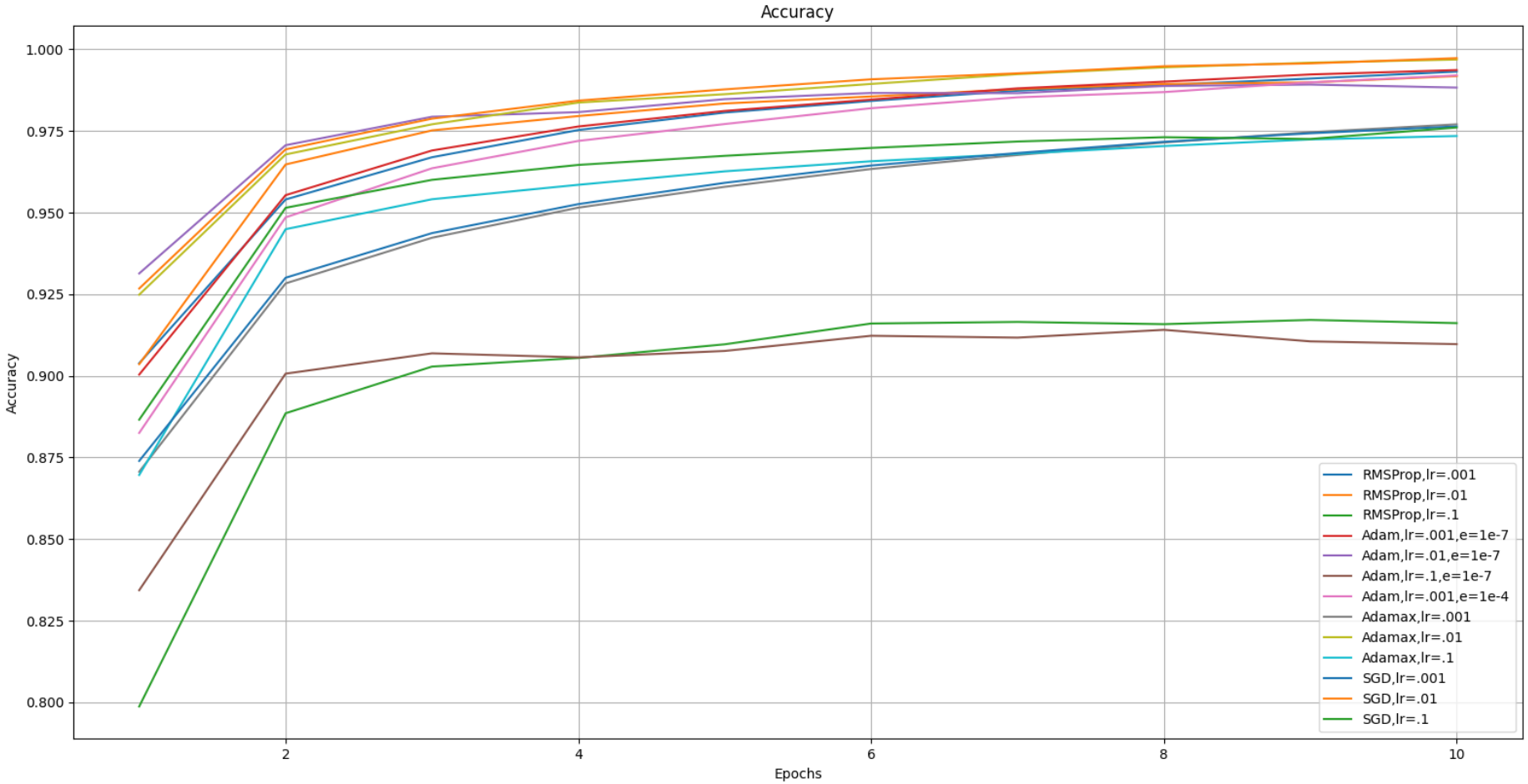


Рисунок 1 – Точность во время обучения при различных оптимизаторах и их параметрах

Из данного графика можно увидеть, то слишком высокое значение коэффициента скорости обучения (0.1) для оптимизаторов RMSProp и Adam приводит к переобучению.

В случае оптимизаторов Adamax и SGD при малом значении коэффициента скорости обучения (0.001) модель обучается медленнее, хоть и не переобучается.

На рис. 2 показаны значения потерь при обучении настроенной модели с различными оптимизаторами.

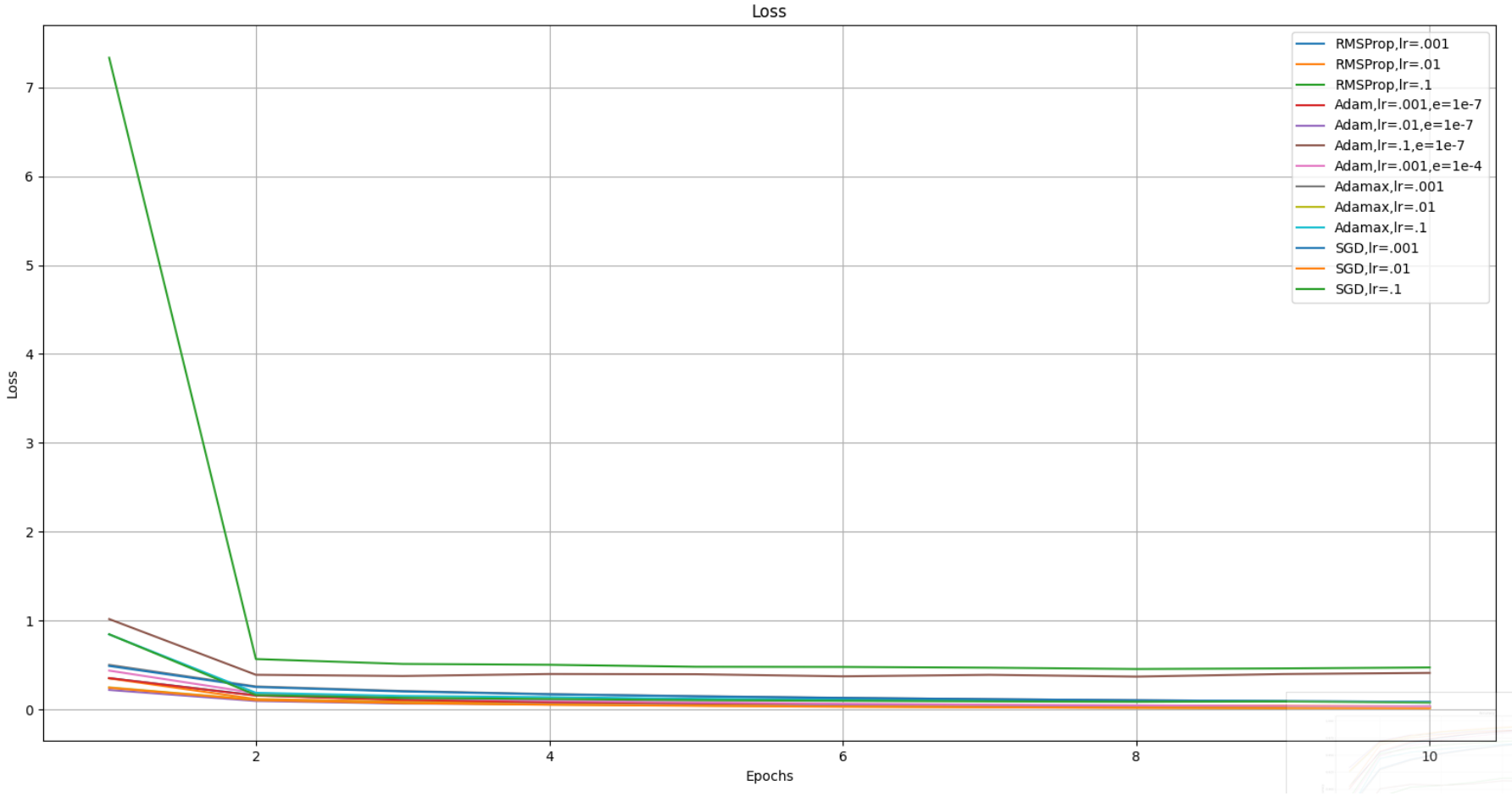


Рисунок 2 – Потери во время обучения при различных оптимизаторах и их параметрах

На рис. 3 приведена сравнительная столбчатая диаграмма точности обученной модели на данных для проверки при различных оптимизаторах и их параметрах.

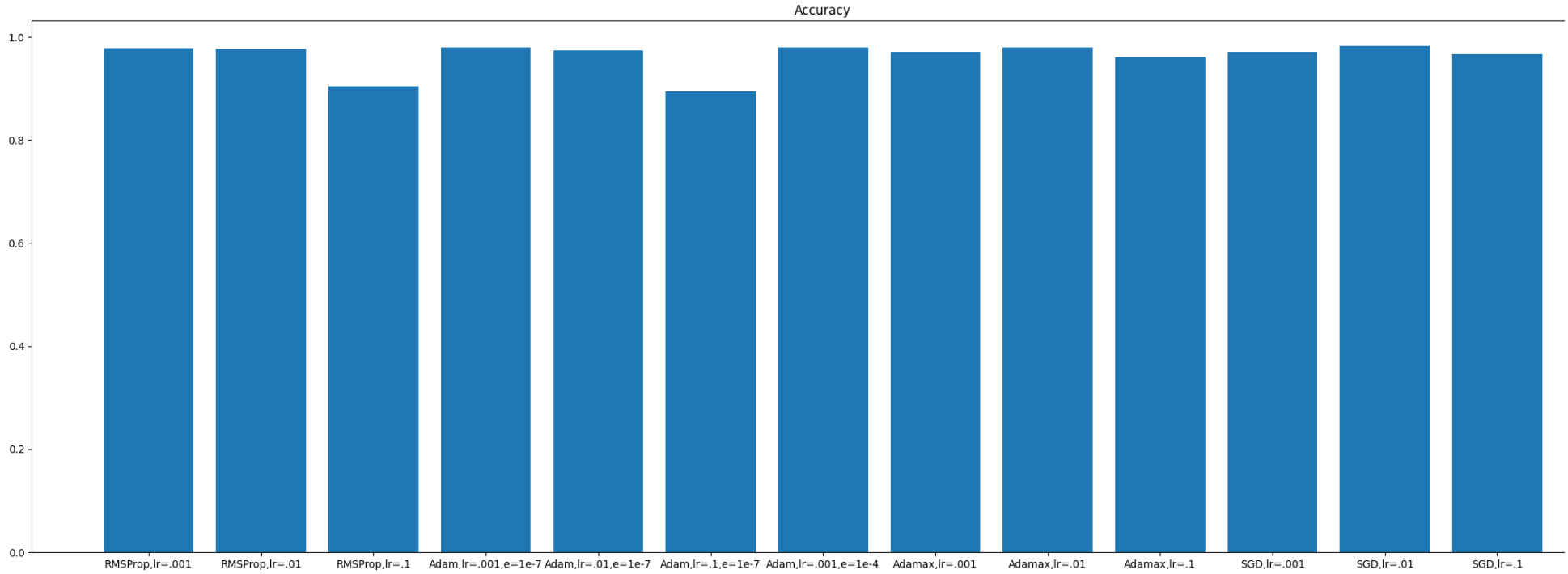


Рисунок 3 – Сравнение точности обученных моделей

В табл. 2. приведены показатели точности при проверке для каждого оптимизатора при различных параметрах и зеленым цветом отмечены наиболее оптимальные параметры.

Таблица 2 – Сравнение точности модели при различных оптимизаторах

RMSProp		**Adam**	
Lr=0.001	0.9786000251	Lr=0.001	0.97930002212
Lr=0.01	0.9768999814	Lr=0.01	0.97339999675
Lr=0.1	0.9039999842	Lr=0.1	0.89410001039
		Lr=0.01, e=1e-4	0.97970002889
Adamax		**SGD**	
Lr=0.001	0.97070002555	Lr=0.001	0.97089999914
Lr=0.01	0.97930002212	Lr=0.01	0.98259997367
Lr=0.1	0.96100002527	Lr=0.1	0.96640002727

Как видно из таблицы, для оптимизаторов RMSProp и Adam наилучше результаты модель показала при коэффициенте скорости обучения $Lr = 0.001$. Для оптимизаторов Adamax и SGD наиболее удачными оказались коэффициенты $Lr = 0.01$. По сути, данный параметр «сдерживает» изменение весов в сторону, противоположную градиенту. Стоит отметить, что для всех оптимизаторов слишком большое значение $Lr = 0.1$ повлекло за собой ухудшение точности. Также на рис. 1 заметно проявление переобучения.

Параметр epsilon для оптимизатора Adam практически не повлиял на показатели модели. Это можно объяснить тем, что данный параметр необходим для исключения деления на ноль, который может возникнуть при околонулевом значении градиента (в итоговой формуле к знаменателю прибавляется epsilon). Возможно, при увеличении числа эпох изменение данного параметра и пойдет на пользу, но в поставленных условиях его влияние минимально, если не выставлять большие значения.

Тестирование на пользовательских изображениях.

Изображения представлены на рис. 4. Изображение единицы выполнено в цвете, а также размер изображения – 48 на 48. Остальные изображения черно-белые с размером 24 на 24.

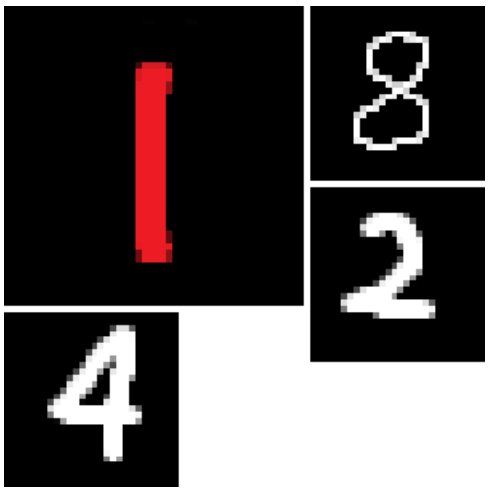


Рисунок 4 – Пользовательские изображения

Для получения ответа от модели был использован метод predict_classes. Моделью были верно классифицированы все изображения. Листинг вывода программы:

```
Loaded image of "1": [1]
Loaded image of "2": [2]
Loaded image of "4": [4]
Loaded image of "8": [8]
```

Выводы.

В ходе выполнения лабораторной работы было изучено решение задачи классификации небольших черно-белых изображений. Была создана и обучена модель, которая способна распознавать цифры на изображении с точностью более 98% на изображениях из MNIST. Также было проведено сравнение обучения модели при разных оптимизаторах и их параметрах. Была создана функция, позволяющая подготавливать для нейронной сети пользовательские изображения, проведено тестирование на собственных изображениях.

ПРИЛОЖЕНИЕ А

Исходный код программы. Файл lr4.py

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


def load_image(path):
    img = Image.open(path).convert('L')
    img_data = np.array(img)
    if img_data.size != 784 or len(img_data) != 28:
        img = img.resize((28, 28), Image.ANTIALIAS)
        img_data = np.array(img)
    img_data = img_data / 255.0
    return np.expand_dims(img_data, axis=0)


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 create_model(opt='adam'):
    model = Sequential()
    model.add(Flatten())
    model.add(Dense(256, activation='relu'))
    model.add(Dense(10, activation='softmax'))
    model.compile(optimizer=opt,               loss='categorical_crossentropy',
metrics=['accuracy'])
    history = model.fit(train_images, train_labels, epochs=10, batch_size=256,
verbose=0)
    return model, history


opts = {
    'RMSProp,lr=.001': optimizers.RMSprop(learning_rate=0.001),
    'RMSProp,lr=.01': optimizers.RMSprop(learning_rate=0.01),
    'RMSProp,lr=.1': optimizers.RMSprop(learning_rate=0.1),
```

```
    'Adam,lr=.001,e=1e-7': optimizers.RMSprop(learning_rate=0.001),
    'Adam,lr=.01,e=1e-7': optimizers.Adam(learning_rate=0.01),
    'Adam,lr=.1,e=1e-7': optimizers.Adam(learning_rate=0.1),
    'Adam,lr=.001,e=1e-4': optimizers.Adam(learning_rate=0.001, epsilon=0.0001),
    'Adamax,lr=.001': optimizers.Adamax(learning_rate=0.001),
    'Adamax,lr=.01': optimizers.Adamax(learning_rate=0.01),
    'Adamax,lr=.1': optimizers.Adamax(learning_rate=0.1),
    'SGD,lr=.001': optimizers.Adamax(learning_rate=0.001),
    'SGD,lr=.01': optimizers.Adamax(learning_rate=0.01),
    'SGD,lr=.1': optimizers.Adamax(learning_rate=0.1)
}


def test_optimizers(opts):
    stats = {}
    result = {}
    epochs = range(1, 11)
    res_model = None
    max_acc = 0
    best_opt = None
    for p in opts.keys():
        print(p)
        model, history = create_model(opts[p])
        test_loss, test_acc = model.evaluate(test_images, test_labels)
        if test_acc > max_acc:
            res_model = model
            max_acc = test_acc
            best_opt = p
        print("Accuracy:", test_acc)
        print("----------------------")
        stats[p] = [history.history["accuracy"], history.history["loss"]]
        result[p] = test_acc
    for p in stats.keys():
        plt.plot(epochs, stats[p][0], label=p)
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.title("Accuracy")
    plt.legend()
    plt.grid()
    plt.show()
    for p in stats.keys():
        plt.plot(epochs, stats[p][1], label=p)
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.title("Loss")
    plt.legend()
    plt.grid()
```

```
    plt.show()
    plt.bar(stats.keys(), result.values())
    plt.title("Accuracy")
    plt.show()
    print(result.values())
    return res_model, best_opt


model, best_opt = test_optimizers(opts)
print("Testing best model with optimizer", best_opt, "on custom images:")
image = load_image('test1.png')
print('Loaded image of "1":', model.predict_classes(image))
image = load_image('test2.png')
print('Loaded image of "2":', model.predict_classes(image))
image = load_image('test4.png')
print('Loaded image of "4":', model.predict_classes(image))
image = load_image('test8.png')
print('Loaded image of "8":', model.predict_classes(image))
```