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»

Студентка гр. 7383 _____ Маркова А.В.

Преподаватель _____ Жукова Н.А.

Санкт-Петербург

2020

Цель работы.

Реализовать классификацию черно-белых изображений рукописных цифр (28 × 28) 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, 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 также их параметров, на процесс обучения;
3. Написать функцию, которая позволит загружать пользовательское изображение не из датасета.

Ход работы.

Классификация – один из разделов машинного обучения, посвященный решению следующей задачи. Имеется множество объектов (ситуаций), разделённых некоторым образом на классы. Задано конечное множество объектов, для которых известно, к каким классам они относятся. Это множество называется обучающей выборкой. Классовая принадлежность остальных объектов не известна. Требуется построить алгоритм, способный классифицировать произвольный объект из исходного множества.

Классифицировать объект – значит, указать номер (или наименование класса), к которому относится данный объект.

MNIST – набор данных для оценки моделей машинного обучения по задаче классификации рукописных цифр. Изображения цифр были взяты из различных отсканированных документов, нормализованы по размеру и центрированы.

Каждое изображение представляет собой квадрат 28 на 28 пикселей (всего 784). Стандартный набор данных используется для оценки и сравнения моделей, где 60000 изображений используется для обучения модели (тренировочный датасет), а отдельный набор из 10000 изображений используется для её проверки.

Значения пикселей – это шкала серого в диапазоне от 0 до 255. Почти всегда рекомендуется выполнять некоторое масштабирование входных значений при использовании моделей нейронных сетей. Поскольку масштаб хорошо известен и хорошо себя ведет, можно очень быстро нормализовать значения пикселей в диапазоне 0 и 1, разделив каждое значение на максимум 255.

Была построена нейронная сеть, разработанный код представлен в приложении А.

Запустим программу с базовой архитектурой сети, которая была дана в указаниях к лабораторной работе. Архитектура представлена на рис. 1.

```
model = Sequential()
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dense(10, activation='softmax'))

# Инициализация параметров обучения
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Обучение сети
hist = model.fit(X_train, y_train, epochs=5, batch_size=128)
```

Рисунок 1 – Архитектура нейронной сети.

Графики потери и точности обучения для начальных параметров приведены на рис. 2.

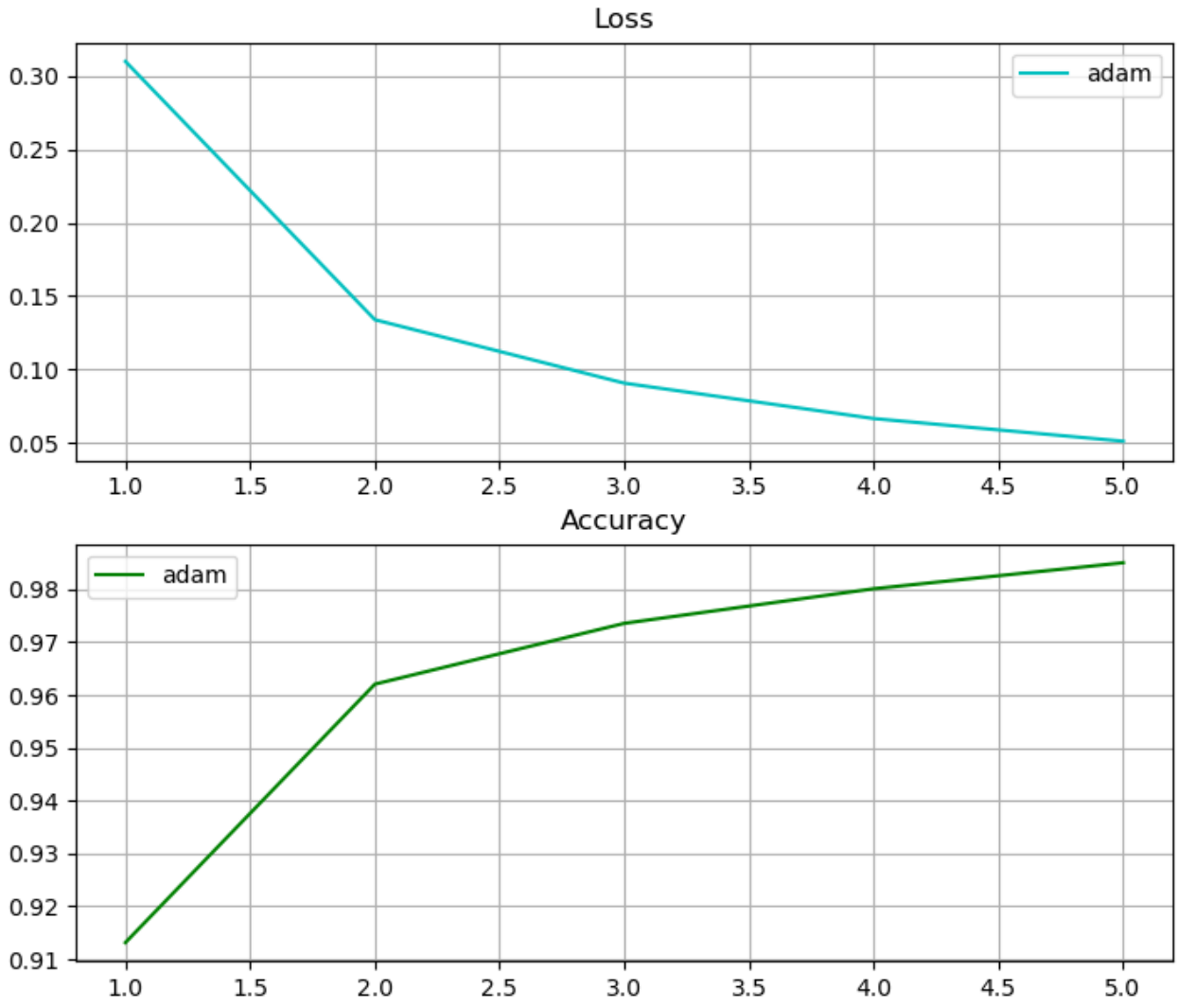

Рисунок 2 – Графики потери и точности.

Точность обучения модели равна 0.9785 при начальных параметрах, что говорит о их приемлемости для поставленной задачи.

Исследуем влияние различных оптимизаторов, а также их параметров, на процесс обучения. Для этого возьмём оптимизаторы Adagrad, Adam, RMSprop и SGD с разными входными параметрами.

Исследуемые оптимизаторы и их конфигурации представлены на рис. 3.

```
# sgd
optimazers_list.append(optimizers.SGD())
optimazers_list.append(optimizers.SGD(learning_rate=0.1, momentum=0.0))
optimazers_list.append(optimizers.SGD(learning_rate=0.1, momentum=0.8))
optimazers_list.append(optimizers.SGD(learning_rate=0.01, momentum=0.8))
baseline_model(optimazers_list, ("SGD(default)", "SGD(learning_rate=0.1, momentum=0.0)", "SGD(learning_rate=0.1, momentum=0.8)",

# adagrad
optimazers_list.append(optimizers.Adagrad())                              # default (0.01)
optimazers_list.append(optimizers.Adagrad(learning_rate=0.1))
optimazers_list.append(optimizers.Adagrad(learning_rate=0.001))
baseline_model(optimazers_list, ("Adagrad(default)", "Adagrad(learning_rate=0.1)", "Adagrad(learning_rate=0.001)"))

# rmsprop
optimazers_list.append(optimizers.RMSprop())                              # default learning_rate=0.001, rho=0.9
optimazers_list.append(optimizers.RMSprop(learning_rate=0.01, rho = 0.9))
optimazers_list.append(optimizers.RMSprop(learning_rate=0.01, rho = 0.5))
optimazers_list.append(optimizers.RMSprop(learning_rate=0.001, rho = 0.5))
baseline_model(optimazers_list, ("RMSprop(default)", "RMSprop(learning_rate=0.01, rho = 0.9)", "RMSprop(learning_rate=0.01, rho =

# adam
optimazers_list.append(optimizers.Adam())                                 # default (lr=0.001, beta_1=0.9, beta_2=0.999)
optimazers_list.append(optimizers.Adam(learning_rate=0.01, beta_1=0.9, beta_2=0.999, amsgrad=True))
optimazers_list.append(optimizers.Adam(learning_rate=0.1, beta_1=0.9, beta_2=0.999, amsgrad=True))
optimazers_list.append(optimizers.Adam(learning_rate=0.01, beta_1=0.1, beta_2=0.5, amsgrad=True))
baseline_model(optimazers_list, ("Adam(default)", "Adam(learning_rate=0.01, beta_1=0.9, beta_2=0.999, amsgrad=True)", "Adam(learni
```

Рисунок 3 – Исследуемые оптимизаторы.

Сначала сравним графики потери и точности для оптимизаторов с их изначальными параметрами, то есть которые заданы по умолчанию. Графики показаны на рис. 4.

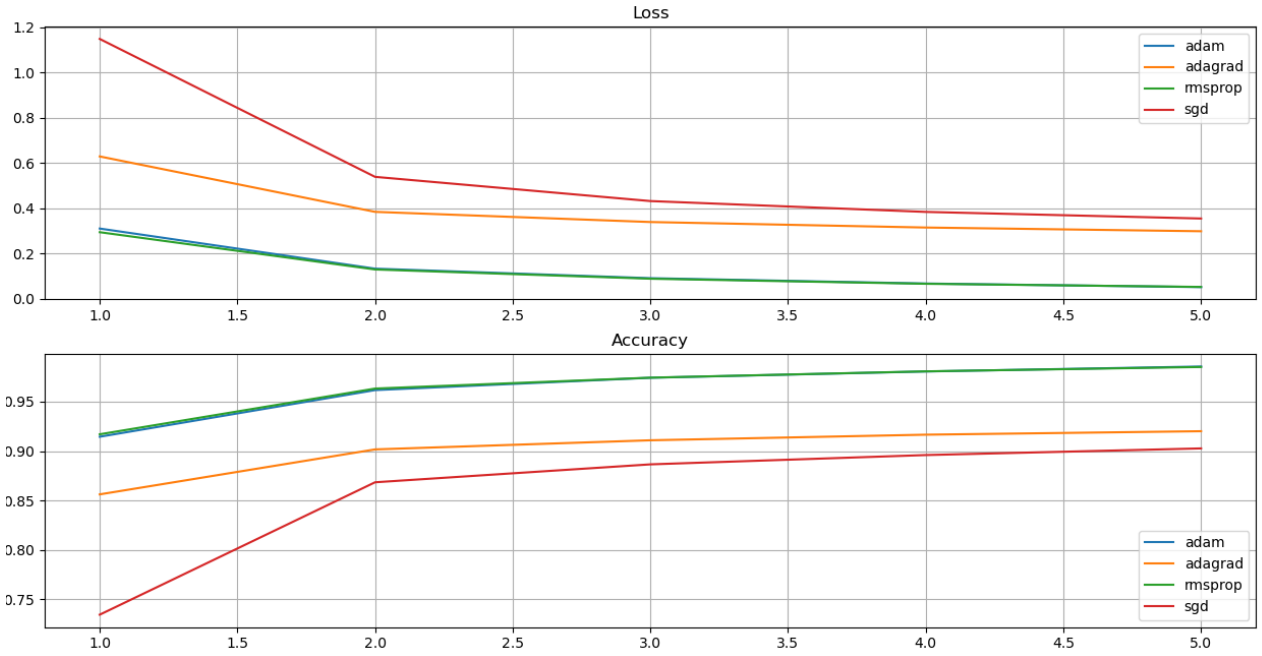

Рисунок 4 – Сравнение оптимизаторов с их изначальными параметрами.

По графикам видно, что оптимизаторы Adam и RMSProp обучаются почти одинаково, в то время как Adagrad и SGD заметно отстают.

Результаты исследования для оптимизатора Adagrad представлены на рис. 5.

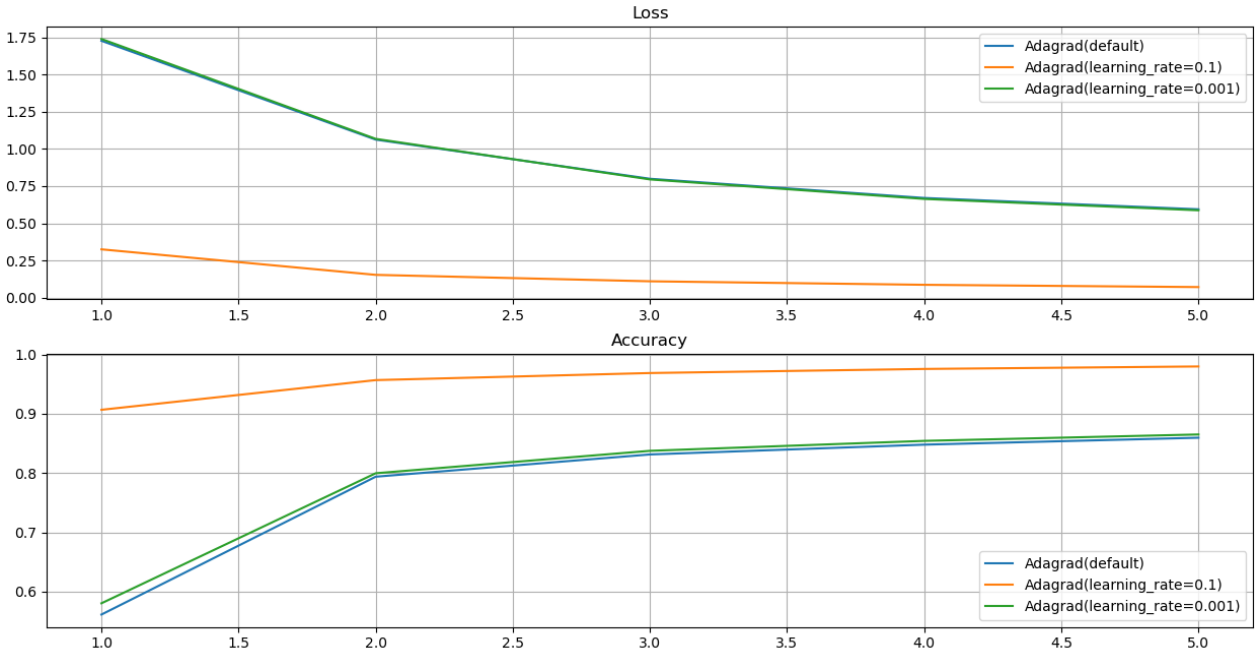

Рисунок 5 – Оптимизатор Adagrad.

Из графиков видно, что лучше всего сеть обучается при более высокой скорости. В данном случае наилучшая точность была достигнута при скорости 0.1.

Результаты исследования для оптимизатора Adam представлены на рис. 6 - 7.

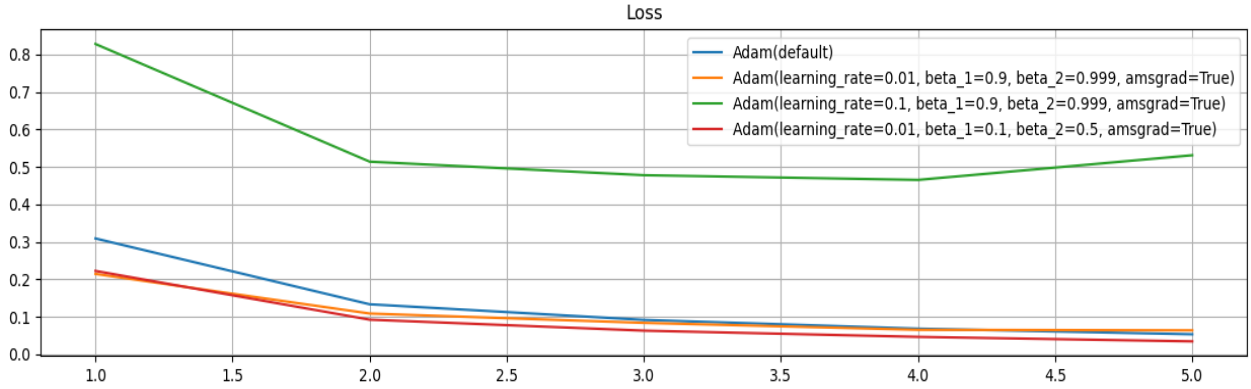

Рисунок 6 – Потери оптимизатора Adam.

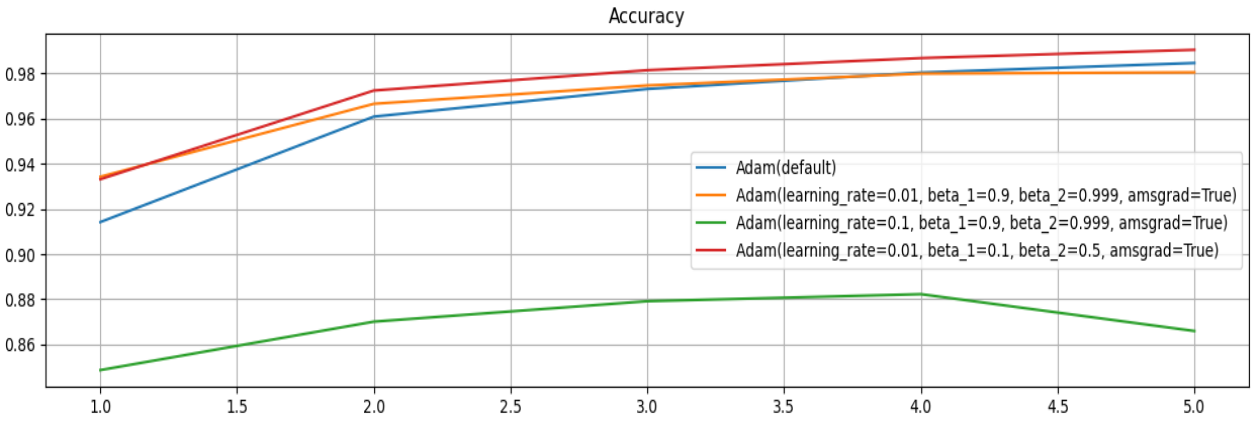

Рисунок 7 – Точность оптимизатора Adam.

Из полученных результатов заметно, что наилучшая обучаемость сети достигнута при меньшей скорости 0.01 и `beta_1` = 0.1, `beta_2` = 0.5 и `amsgrad` = `True`.

Результаты исследования для оптимизатора RMSprop показаны на рис. 8.

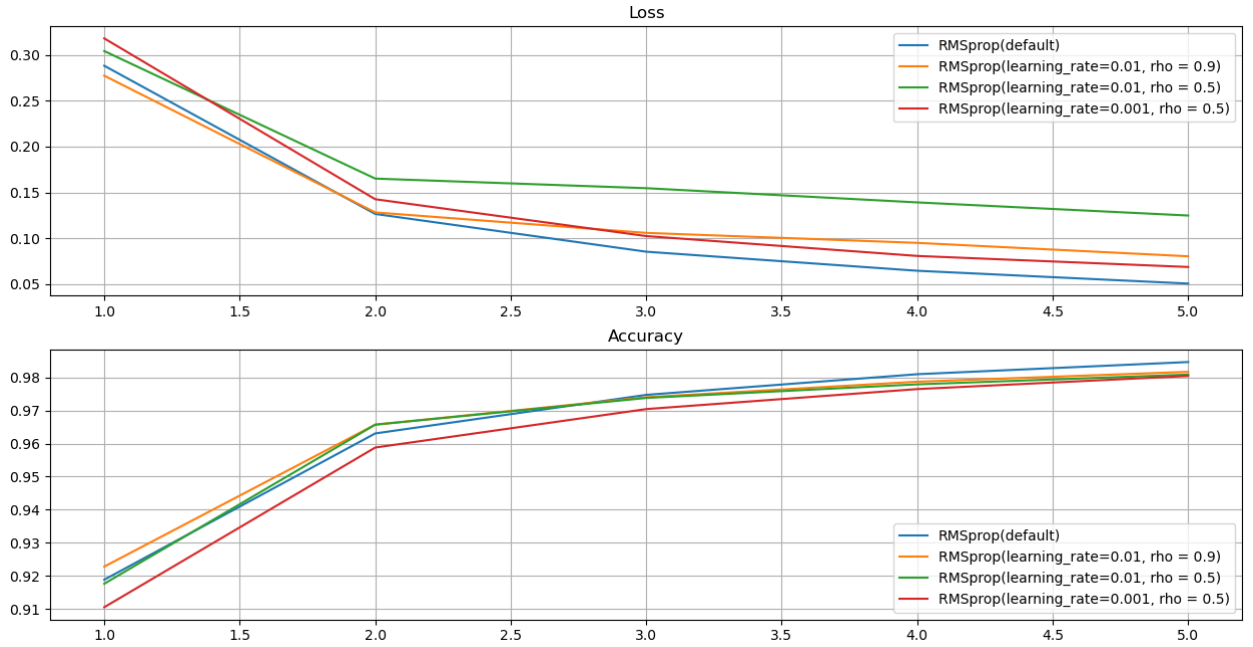

Рисунок 8 – Оптимизатор RMSprop.

Из графиков делаем вывод, что лучший результат достигается при значениях по умолчанию: скорости 0.001 и `rho` = 0.9.

Результаты исследования для оптимизатора SGD представлены на рис. 9.

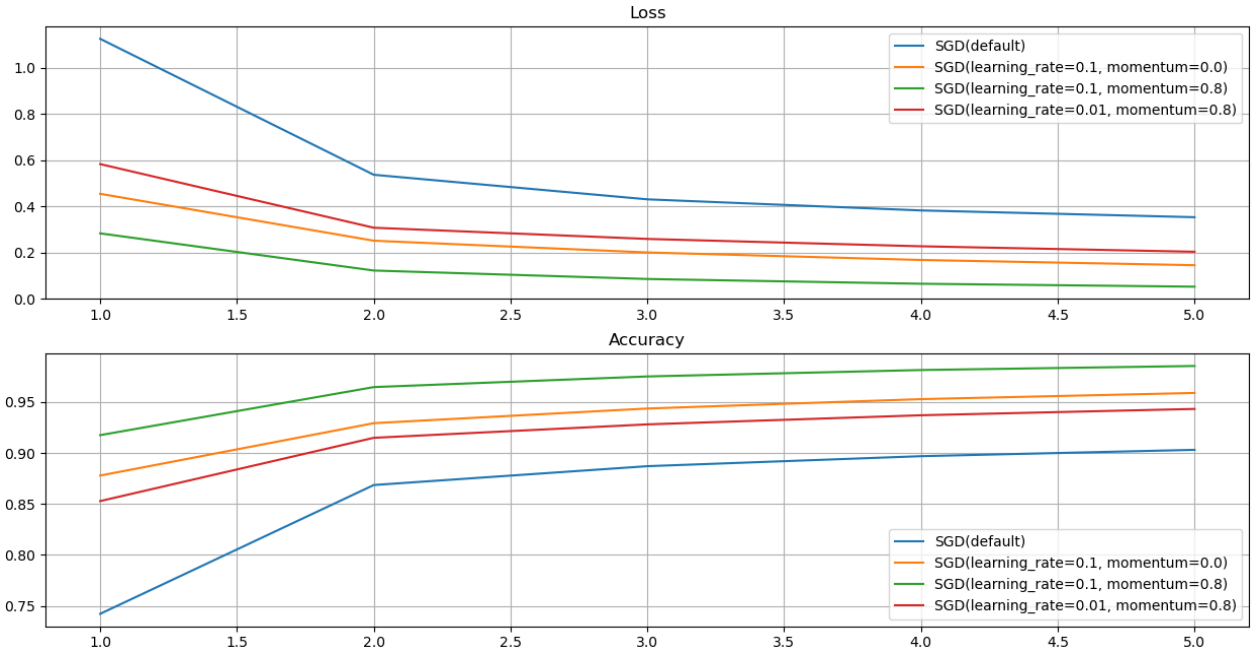

Рисунок 9 – Оптимизатор SGD.

Из графиков видно, что обучаемость улучшается при увеличении скорости и момента. Наилучшая точность для данного оптимизатора была достигнута при скорости 0.1 и моменте 0.8.

Для лучшего выбора оптимизатора сравним самые удачные варианты конфигураций, результат показан на рис. 10.

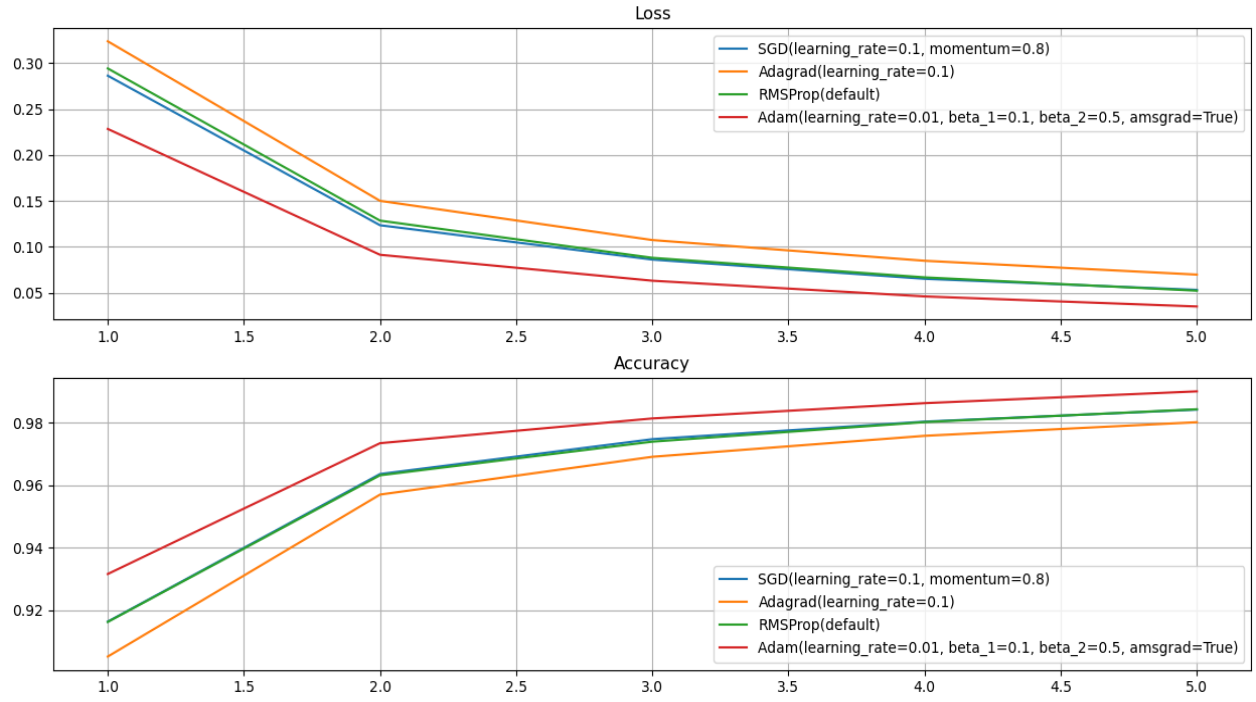

Рисунок 10 – Выбор лучшего оптимизатора из лучших.

Из графиков видно, что лучший результат достигается с использованием оптимизатора Adam. Для большей наглядности сравним численные значения точностей, данные приведены в табл. 1.

Таблица 1 – Сравнение точности оптимизаторов.

Optimizers	SGD	RMSprop	Adagrad	Adam
Accuracy	0.9773	0.9781	0.9743	0.9789

Также в лабораторной работе была написана функция `read_and_predict` для считывания картинки, содержащей рукописную цифру. Данная функция позволяет протестировать обученную нейронную сеть на пользовательских изображениях, проверив правильность её предсказания. Тестовые рисунки представлены на рис. 11.

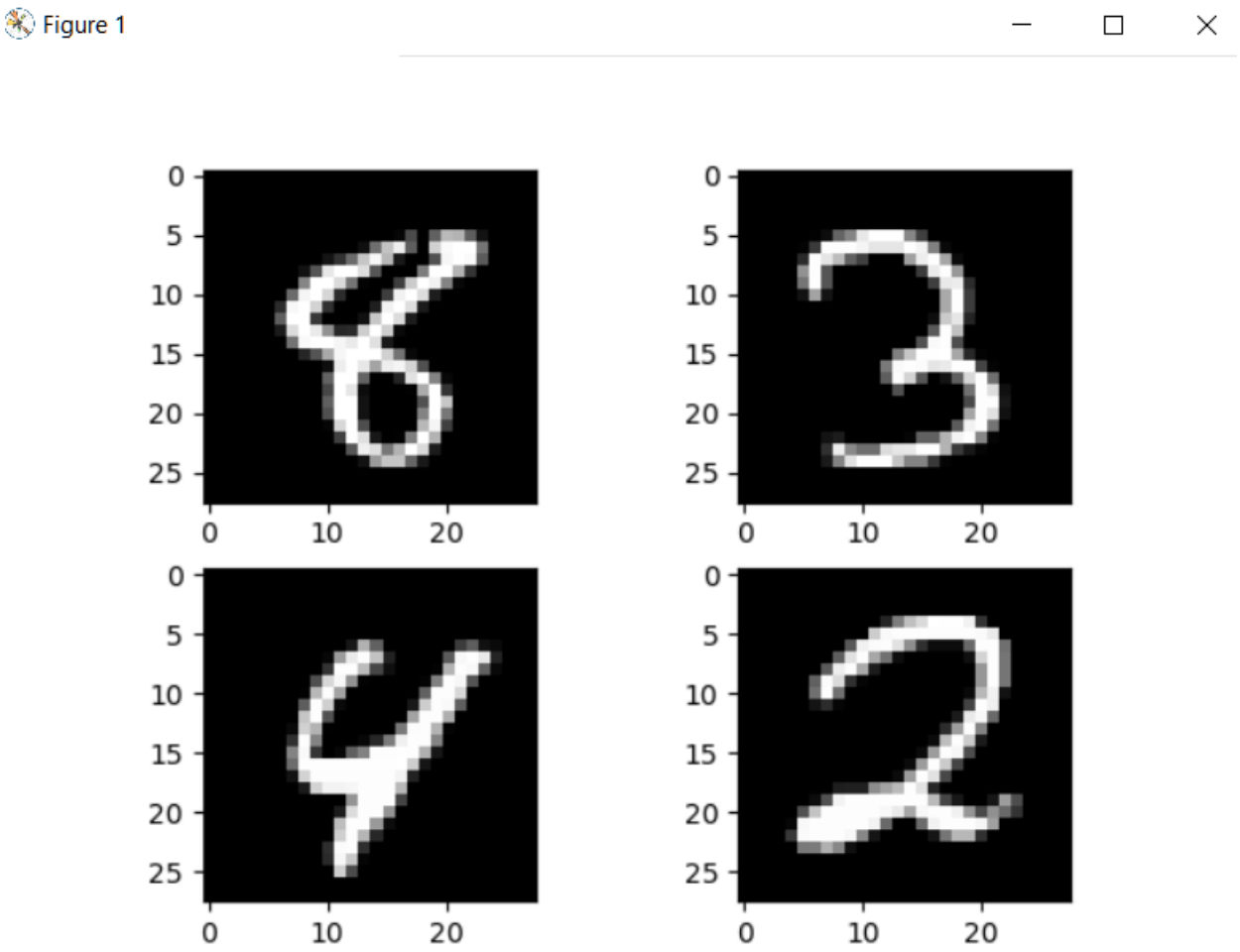

Рисунок 11 – Изображения для тестирования.

Результат работы разработанной нейронной сети на тестовых изображениях (см. рис. 11) показан на рис. 12.

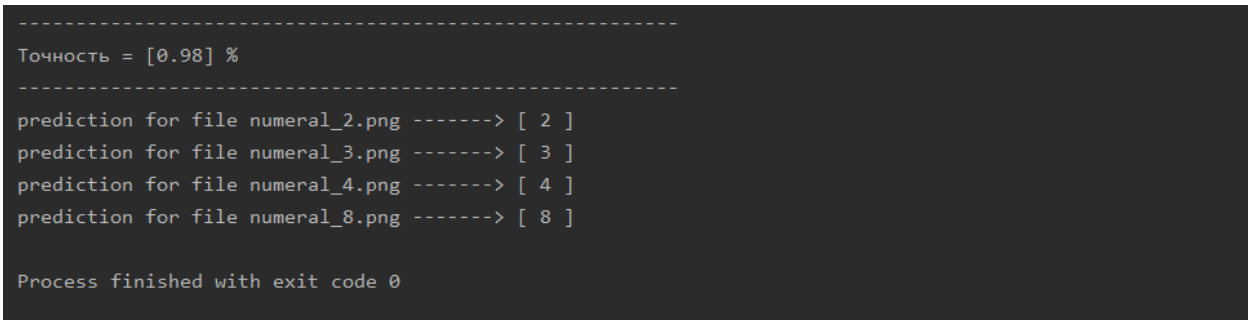

Рисунок 12 – Вывод программы.

С предсказанием модель справляется, но несмотря на большую точность нейронная сеть может давать и ошибочные результаты, что может быть связано со стилем написания цифр, то есть почерк на изображении сильно отличается от почерка на тренировочных данных.

Выводы.

В ходе выполнения лабораторной работы было изучено представление и обработка графических данных, простейших черно-белых изображений из базы данных MNIST, была написана функция для загрузки пользовательского изображения, был выявлен лучший оптимизатор из Adagrad, Adam, RMSprop и SGD для построения модели искусственной нейронной сети, распознающей рукописные цифры, была разработана и протестирована на пользовательских изображениях модель.

Были выбраны оптимальные параметры для оптимизатора Adam: скорость 0.01 и `beta_1` = 0.1, `beta_2` = 0.5 и `amsgrad` = `True`.

ПРИЛОЖЕНИЕ А

```
# Подключение модулей
import numpy as np
from PIL import Image
from numpy import asarray
import matplotlib.pyplot as plt
from keras.datasets import mnist
from keras.utils import to_categorical
from tensorflow.keras import optimizers
from tensorflow.keras.models import Sequential        # Для создания простых моделей используют Sequential,
from tensorflow.keras.layers import Dense, Flatten

# Загрузка набора данных MNIST
(X_train, y_train), (X_test, y_test) = mnist.load_data()

# Вывод изображений для тестов в серых оттенках
plt.subplot(221)
plt.imshow(X_test[84], cmap=plt.get_cmap('gray'))
plt.subplot(222)
plt.imshow(X_test[142], cmap=plt.get_cmap('gray'))
plt.subplot(223)
plt.imshow(X_test[139], cmap=plt.get_cmap('gray'))
plt.subplot(224)
plt.imshow(X_test[35], cmap=plt.get_cmap('gray'))
plt.show()

# Сохранение изображений
# temp = np.reshape(X_test[84], (28, 28))
# im = Image.fromarray(temp).convert('L')
# im.save("numeral_8.png")
# temp = np.reshape(X_test[142], (28, 28))
# im = Image.fromarray(temp).convert('L')
# im.save("numeral_3.png")
```

```
    # temp = np.reshape(X_test[139], (28, 28))
    # im = Image.fromarray(temp).convert('L')
    # im.save("numeral_4.png")
    # temp = np.reshape(X_test[35], (28, 28))
    # im = Image.fromarray(temp).convert('L')
    # im.save("numeral_2.png")

    # Нормализовать входные данные от 0-255 до 0-1
    X_train = X_train / 255.0
    X_test = X_test / 255.0

    # Горячее кодирование значений класса, преобразовывая вектор
целых чисел класса в двоичную матрицу
    y_train = to_categorical(y_train)
    y_test = to_categorical(y_test)

    # Определение базовой модели
    def baseline_model(optimazers_list, labels):
        acc_list = []
        history_loss_list = []
        history_acc_list = []

        # Создание модели
        for opt in optimazers_list:
            model = Sequential()
            model.add(Flatten(input_shape=(28, 28)))
            model.add(Dense(256, activation='relu'))
            model.add(Dense(10, activation='softmax'))

            # Инициализация параметров обучения
            model.compile(optimizer=opt,
loss='categorical_crossentropy', metrics=['accuracy'])

            # Обучение сети
```

```
            hist = model.fit(X_train, y_train, epochs=5,
batch_size=128)
            history_loss_list.append(hist.history['loss'])
            history_acc_list.append(hist.history['acc'])
            test_loss, test_acc = model.evaluate(X_test, y_test)
            acc_list.append(test_acc)

        # Построение графиков ошибки и точности
        print("-------------------------------------------------------")
        print("Точность = " + str(np.round(acc_list, 2)) + " %")
        x = range(1, 6)

        plt.subplot(211)
        plt.title('Loss')
        for loss in history_loss_list:
            plt.plot(x, loss, 'c')
        plt.legend(labels)
        plt.grid()

        plt.subplot(212)
        plt.title('Accuracy')
        for acc in history_acc_list:
            plt.plot(x, acc, 'g')
        plt.legend(labels)
        plt.grid()
        plt.show()

        return model

    optimazers_list = []

    ### Исследование влияния различных оптимизаторов, а также их
параметров, на процесс обучения ###
```

```
    # Оптимизаторы с параметрами по умолчанию
    # optimazers_list = ("adam", "adagrad", "rmsprop", "sgd")
    # baseline_model(optimazers_list, optimazers_list)

    # sgd
    # optimazers_list.append(optimizers.SGD())
    # optimazers_list.append(optimizers.SGD(learning_rate=0.1,
momentum=0.0))
    # optimazers_list.append(optimizers.SGD(learning_rate=0.1,
momentum=0.8))
    # optimazers_list.append(optimizers.SGD(learning_rate=0.01,
momentum=0.8))
    # baseline_model(optimazers_list, ("SGD(default)",
"SGD(learning_rate=0.1, momentum=0.0)", "SGD(learning_rate=0.1,
momentum=0.8)", "SGD(learning_rate=0.01, momentum=0.8)"))

    # adagrad
    # optimazers_list.append(optimizers.Adagrad())
    # optimazers_list.append(optimizers.Adagrad(learning_rate=0.1))
    # optimazers_list.append(optimizers.Adagrad(learning_rate=0.001))
    # baseline_model(optimazers_list, ("Adagrad(default)",
"Adagrad(learning_rate=0.1)", "Adagrad(learning_rate=0.001)"))

    # rmsprop
    # optimazers_list.append(optimizers.RMSprop())
    # optimazers_list.append(optimizers.RMSprop(learning_rate=0.01,
rho = 0.9))
    # optimazers_list.append(optimizers.RMSprop(learning_rate=0.01,
rho = 0.5))
    # optimazers_list.append(optimizers.RMSprop(learning_rate=0.001,
rho = 0.5))
    # baseline_model(optimazers_list, ("RMSprop(default)",
"RMSprop(learning_rate=0.01, rho = 0.9)", "RMSprop(learning_rate=0.01,
rho = 0.5)", "RMSprop(learning_rate=0.001, rho = 0.5)"))
```

```
    # adam
    # optimazers_list.append(optimizers.Adam())
    # optimazers_list.append(optimizers.Adam(learning_rate=0.01,
beta_1=0.9, beta_2=0.999, amsgrad=True))
    # optimazers_list.append(optimizers.Adam(learning_rate=0.1,
beta_1=0.9, beta_2=0.999, amsgrad=True))
    # optimazers_list.append(optimizers.Adam(learning_rate=0.01,
beta_1=0.1, beta_2=0.5, amsgrad=True))
    # baseline_model(optimazers_list, ("Adam(default)",
"Adam(learning_rate=0.01, beta_1=0.9, beta_2=0.999, amsgrad=True)",
"Adam(learning_rate=0.1, beta_1=0.9, beta_2=0.999, amsgrad=True)",
"Adam(learning_rate=0.01, beta_1=0.1, beta_2=0.5, amsgrad=True)"))

    # Поиск лучшего из лучших
    # optimazers_list.append(optimizers.SGD(learning_rate=0.1,
momentum=0.8))
    # optimazers_list.append(optimizers.Adagrad(learning_rate=0.1))
    # optimazers_list.append(optimizers.RMSprop())
    # optimazers_list.append(optimizers.Adam(learning_rate=0.01,
beta_1=0.1, beta_2=0.5, amsgrad=True))
    # baseline_model(optimazers_list, ("SGD(learning_rate=0.1,
momentum=0.8)", "Adagrad(learning_rate=0.1)", "RMSProp(default)",
"Adam(learning_rate=0.01, beta_1=0.1, beta_2=0.5, amsgrad=True)"))

    def read_and_predict(path):
        # Загрузка изображения
        image = Image.open(path).convert('L')
        # Преобразовать изображение в массив numpy
        data = asarray(image)
        data = data.reshape((1, 28, 28))
        Y = model.predict_classes(data)
        print("prediction for file " + path + " -------> " "[ " +
np.array2string(Y[0]) + " ]")
```

```
        return data

    model = baseline_model([optimizers.Adam(learning_rate=0.01,
beta_1=0.1, beta_2=0.5, amsgrad=True)], ["Adam"]) #[0.9789]
    print("-------------------------------------------------------")
    read_and_predict("numeral_2.png")
    read_and_predict("numeral_3.png")
    read_and_predict("numeral_4.png")
    read_and_predict("numeral_8.png")
```