**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. 8383 _____ Бабенко Н.С.

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:

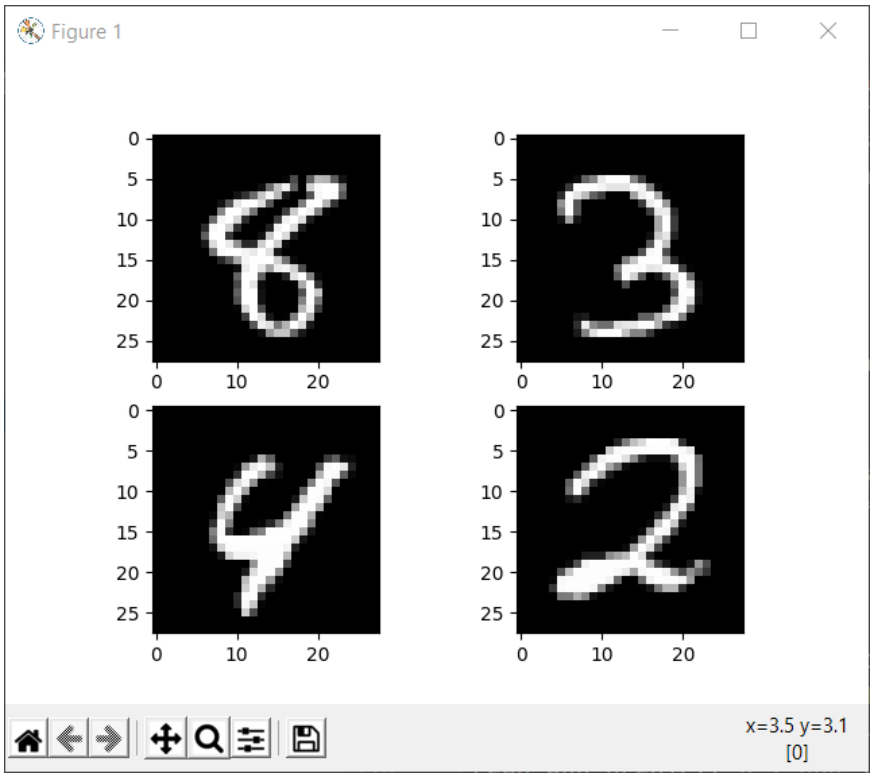
- Найти архитектуру сети, при которой точность классификации будет не менее 95%
- Исследовать влияние различных оптимизаторов, а также их параметров, на процесс обучения
- Написать функцию, которая позволит загружать пользовательское изображение не из датасета

**Выполнение работы.**

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. Требуется построить алгоритм, способный классифицировать произвольный объект из исходного множества.

MNIST – набор данных для оценки моделей машинного обучения по задаче классификации рукописных цифр.

Ознакомимся с изображениями из датасета. В программе с помощью matplotlib были выведены некоторые изображения из выборки, они представлены на рисунке ниже.




Создание модели приведено в листинге ниже:

```
model = Sequential()
model.add(Flatten(input_shape=(28, 28)))
model.add(Dense(256, activation='relu'))
model.add(Dense(10, activation='softmax'))
```

Слой Flatten преобразует двумерный тензор пикселей изображения в вектор. Так как цифр 10, то перед выходным слоем будет 10 значений, в сумме дающих 1 – вероятности.

Будет анализироваться три оптимизатора с различными параметрами:

- SGD
- RMSProp
- Adam

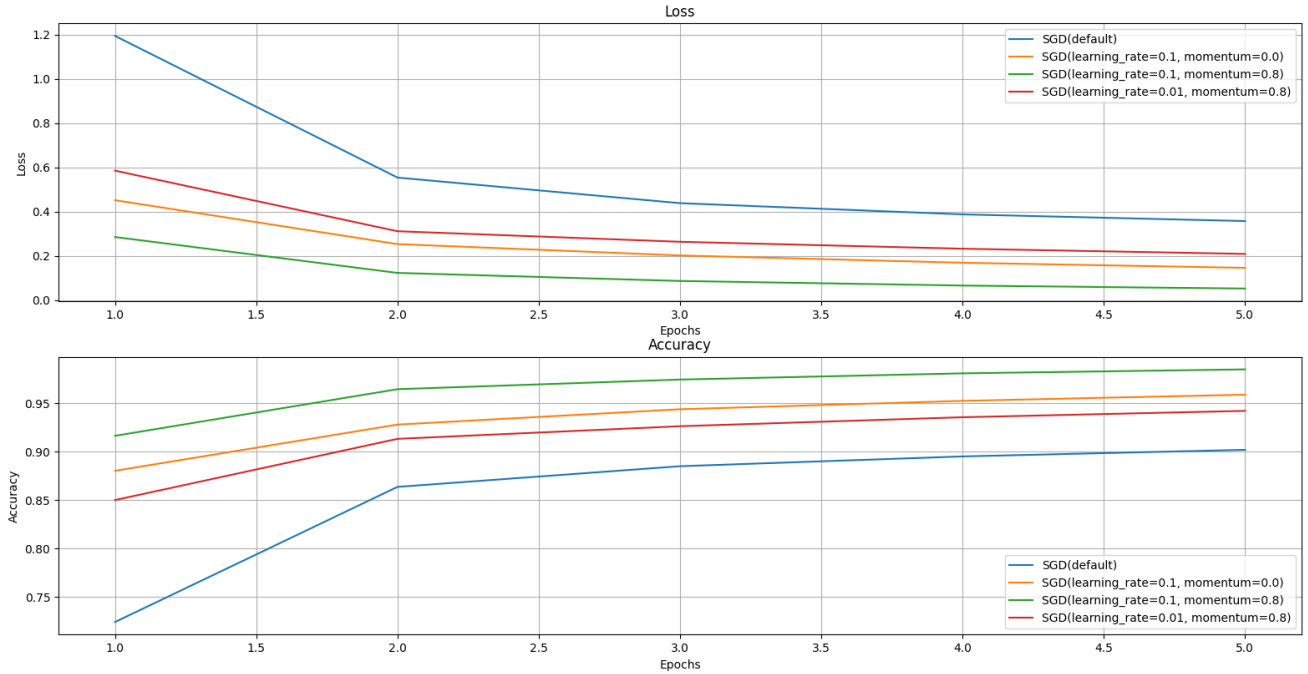
Функция потерь - categorical_crossentropy, подходит для задач классификации.

**Анализ оптимизатора SGD.**

Параметры при запусках:

- По умолчанию
- Скорость обучения: 0.1
- Скорость обучения: 0.1, momentum = 0.8
- Скорость обучения: 0.01, momentum = 0.8

Результат обучения модели представлен на рисунках ниже (графики точности и потерь):



Результаты на проверочных данных:

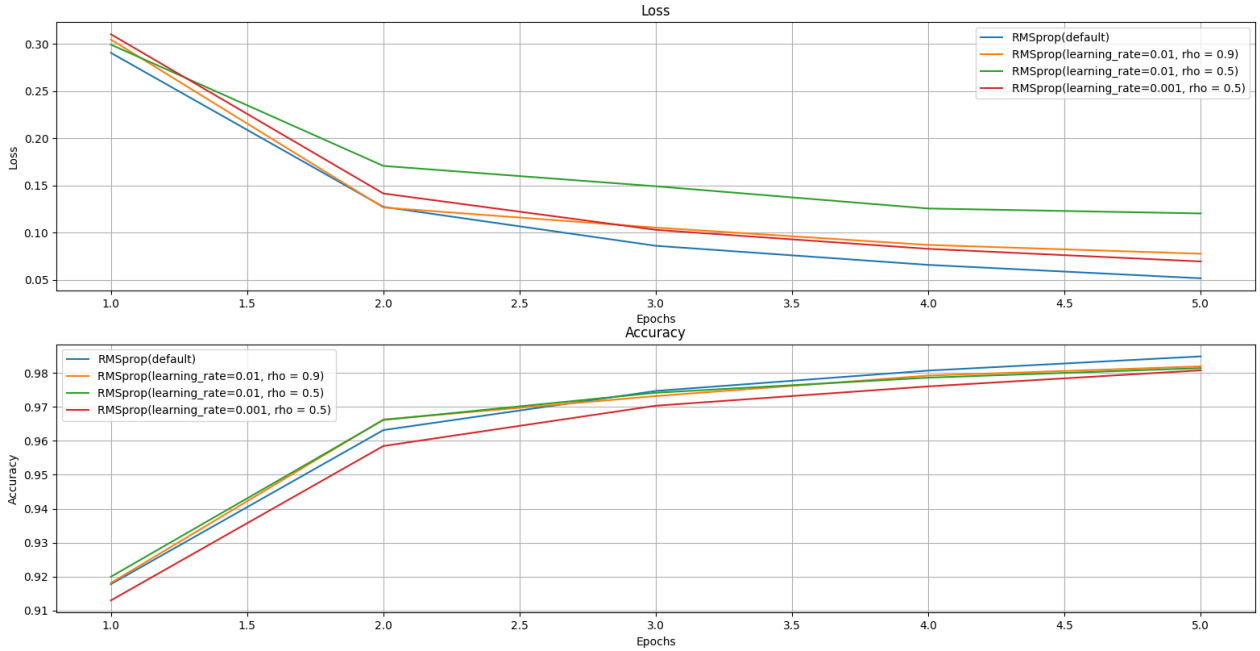
- По умолчанию - точность 91%
- Скорость обучения: 0.1 – точность 96%
- Скорость обучения: 0.1, momentum = 0.8 – точность 98%
- Скорость обучения: 0.01, momentum = 0.8 – точность 95%

**Анализ оптимизатора RMSProp.**

Параметры при запусках:

- По умолчанию
- Скорость обучения: 0.01, rho = 0.9
- Скорость обучения: 0.01, rho = 0.5
- Скорость обучения: 0.001, rho = 0.5

Результат обучения модели представлен на рисунках ниже (графики точности и потерь):




Точность на проверочных данных:

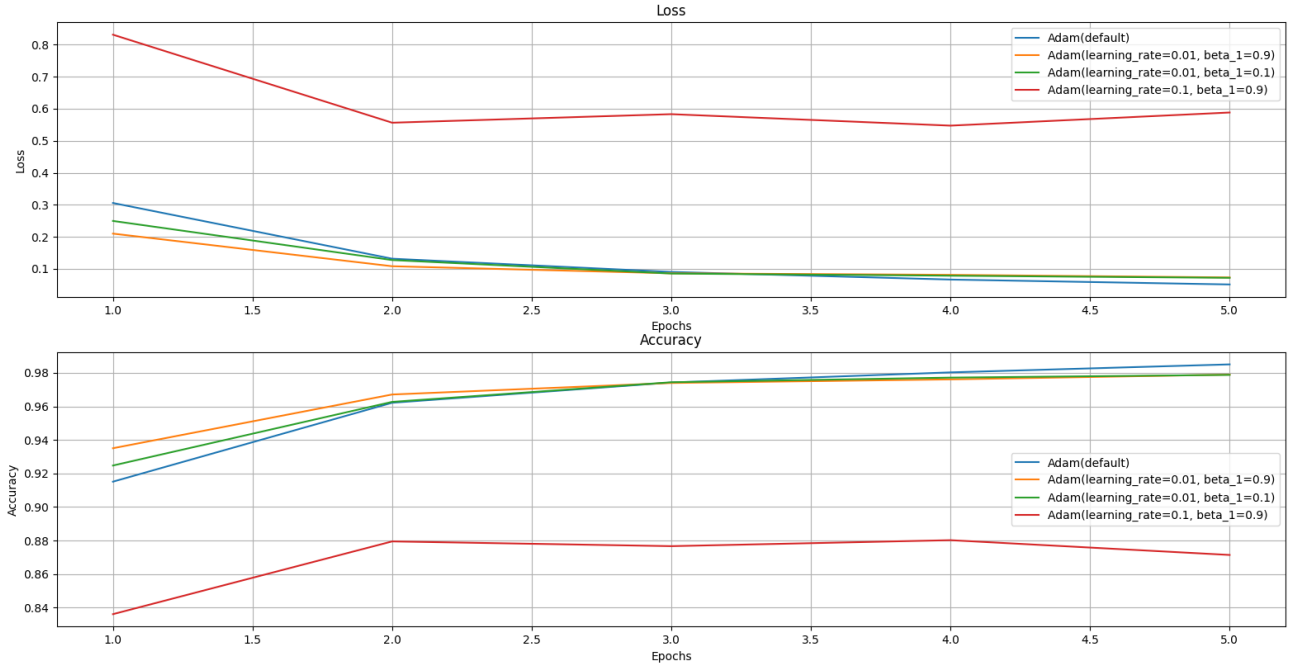
- По умолчанию – 98%
- Скорость обучения: 0.01, rho = 0.9 – 98%
- Скорость обучения: 0.01, rho = 0.5 – 98%
- Скорость обучения: 0.001, rho = 0.5 – 98%

**Анализ оптимизатора Adam.**

Параметры при запусках:

- По умолчанию
- Скорость обучения: 0.01, beta1 = 0.9
- Скорость обучения: 0.01, beta1 = 0.1
- Скорость обучения: 0.1, beta1 = 0.9

Результат обучения модели представлен на рисунках ниже (графики точности и потерь):



Результаты на проверочных данных:

- По умолчанию – 98%
- Скорость обучения: 0.01, beta1 = 0.9 – 97%
- Скорость обучения: 0.01, beta1 = 0.1 – 97%
- Скорость обучения: 0.1, beta1 = 0.9 – 98%

Все три оптимизатора достигли точности 98% на данных для проверки.

При этом параметры:

- SGD (3-й вариант):
  - коэффициент скорости обучения – 0.1
  - momentum – 0.8
- RMSProp (по умолчанию):
  - коэффициент скорости обучения – 0.001
  - rho – 0.9 (по умолчанию)
- Adam (по умолчанию):
  - коэффициент скорости обучения – 0.001
  - beta_1=0.9
  - beta_2=0.999

Оптимизатору SGD требуется большая скорость обучения, а также большую эффективность оптимизатор показал с показателем momentum = 0.8. Данный показатель ускоряет градиентный спуск в соответствующем направлении и гасит колебания, которые возможны при стандартном стохастическом градиентном спуске.

Остальные оптимизаторы наилучший результат показывают при параметрах, которые установлены в Keras по умолчанию.

Были взяты изображения из датасета и сохранены в файлы в формате png. Результаты запуска model.predict для файлов:

```
prediction for file n_2.png -> [ 2 ]
prediction for file n_3.png -> [ 3 ]
prediction for file n_4.png -> [ 4 ]
```

```
prediction for file n_8.png -> [ 8 ]
```

Модель верно распознала цифры на изображениях.

**Выводы.**

В ходе выполнения лабораторной работы было изучено решение задачи классификации небольших черно-белых изображений. Была создана модель и проведено сравнение обучения модели при разных оптимизаторах и их параметрах. Была создана функция, позволяющая подготавливать для нейронной сети пользовательские изображения.

## ПРИЛОЖЕНИЕ А

### Исходный код программы. Файл main.py

```
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

from tensorflow.keras.layers import Dense, Flatten

(X_train, y_train), (X_test, y_test) = mnist.load_data()

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()

X_train = X_train / 255.0
X_test = X_test / 255.0

y_train = to_categorical(y_train)
y_test = to_categorical(y_test)


def baseline_model(optimizers_list, labels):
    acc_list = []
    history_loss_list = []
    history_acc_list = []

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

        model.compile(optimizer=opt,           loss='categorical_crossentropy',
metrics=['accuracy'])
```

```
        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['accuracy'])
        test_loss, test_acc = model.evaluate(X_test, y_test)
        acc_list.append(test_acc)

    print("-------------------------------")
    print("Точность = " + str(np.round(acc_list, 2)))
    x = range(1, 6)

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

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

    return model


optimizers_list = []

optimizers_list.append(optimizers.SGD())
optimizers_list.append(optimizers.SGD(learning_rate=0.1, momentum=0.0))
optimizers_list.append(optimizers.SGD(learning_rate=0.1, momentum=0.8))
optimizers_list.append(optimizers.SGD(learning_rate=0.01, momentum=0.8))
baseline_model(optimizers_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)"))

optimizers_list.append(optimizers.RMSprop())    #  default  learning_rate=0.001,
rho=0.9
optimizers_list.append(optimizers.RMSprop(learning_rate=0.01, rho=0.9))
optimizers_list.append(optimizers.RMSprop(learning_rate=0.01, rho=0.5))
```

```
optimizers_list.append(optimizers.RMSprop(learning_rate=0.001, rho=0.5))
baseline_model(optimizers_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)"))

optimizers_list.append(optimizers.Adam())    #  default  (lr=0.001,  beta_1=0.9,
beta_2=0.999)
optimizers_list.append(optimizers.Adam(learning_rate=0.01, beta_1=0.9))
optimizers_list.append(optimizers.Adam(learning_rate=0.01, beta_1=0.1))
optimizers_list.append(optimizers.Adam(learning_rate=0.1, beta_1=0.9))
model         =         baseline_model(optimizers_list,         ("Adam(default)",
"Adam(learning_rate=0.01, beta_1=0.9)",
                                  "Adam(learning_rate=0.01, beta_1=0.1)",
                                  "Adam(learning_rate=0.1, beta_1=0.9)",
                                  ))


def read_and_predict(path, model):
    image = Image.open(path).convert('L')
    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

read_and_predict("n_2.png", model)
read_and_predict("n_3.png", model)
read_and_predict("n_4.png", model)
read_and_predict("n_8.png", model)
```