МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Искусственные нейронные сети»

Тема: «Распознавание рукописных символов»

Студент гр. 7381 _____ Вологдин М.Д.

Преподаватель _____ Жукова Н.А.

Санкт-Петербург

2020

Цель работы.

Реализовать классификацию черно-белых изображений рукописных цифр (28x28) по 10 категориям (от 0 до 9). Набор данных содержит 60,000 изображений для обучения и 10,000 изображений для тестирования.

Порядок выполнения работы.

- 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

Ход работы.

Набор данных MNIST — большой (порядка 60 000 тренировочных и 10 000 проверочных объектов, помеченных на принадлежность одному из десяти классов — какая цифра изображена на картинке) набор картинок с рукописными цифрами, часто используемый для тестирования различных алгоритмов распознавания образов. Он содержит черно-белые картинки размера 28x28 пикселей, исходно взятые из набора образцов из бюро переписи населения США, к которым были добавлены тестовые образцы, написанные студентами американских университетов.

1. Найдем архитектуру сети с точностью выше 95%.

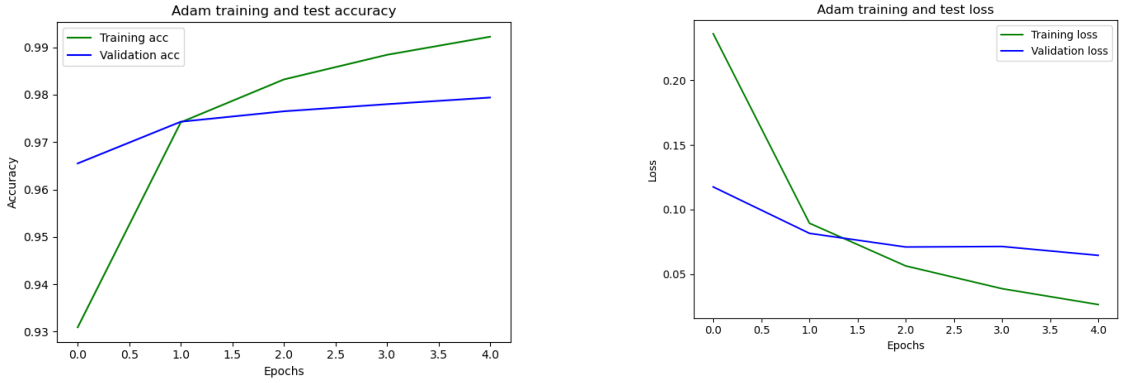
В результате тестов пришли к следующей архитектуре:

- 3 слоя:

```
model.add(Flatten(input_shape=(28, 28)))
model.add(Dense(1024, activation='relu'))
model.add(Dense(10, activation='softmax'))
```

- Оптимизатор – adam
- `batch_size=128`
- `loss='categorical_crossentropy'`
- `epochs=5`

  Точность ~98%

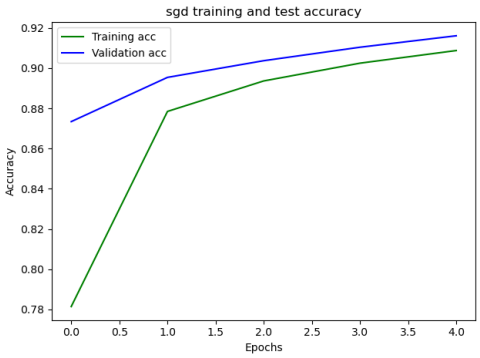


а б

Рисунок 1 – Графики точности и потерь данной архитектуры

Рассмотрим различные оптимизаторы:

- SGD

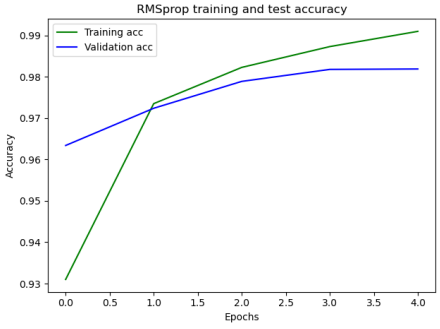


а

б

Рисунок 2 – Графики точности и потерь SGD

- RMSprop

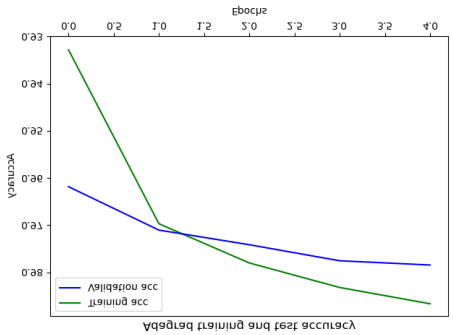


а

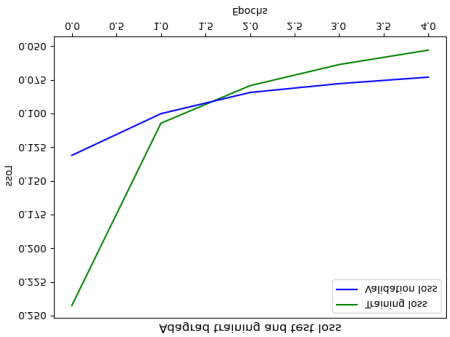
б

Рисунок 3 – Графики точности и потерь RMSprop

- Adagrad



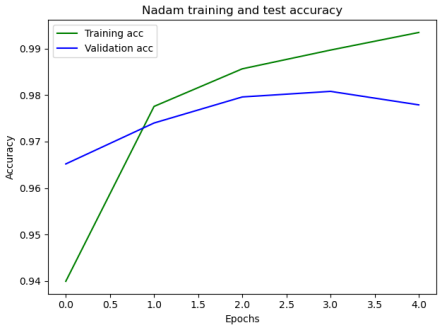
а



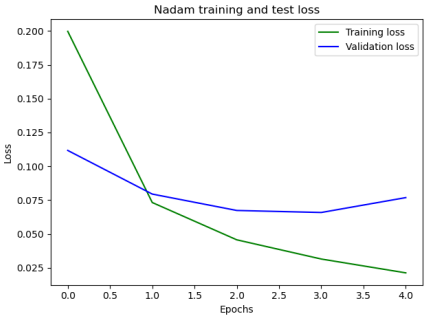
б

Рисунок 4 – Графики точности и потерь Adagrad

- Nadam



а



б

Рисунок 5 – Графики точности и потерь Nadam

Как видим, все сети показывают примерно одинаковый результат, и только SGD значительно отстает. Остановимся на adam’е.

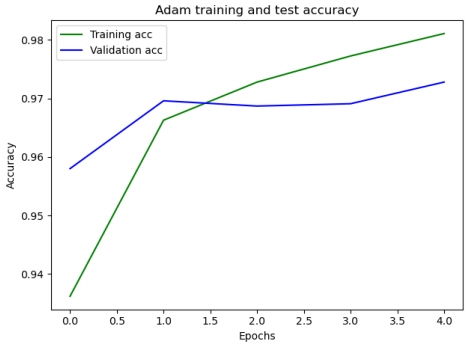
Рассмотрим различные значения параметра скорости обучения оптимизатора adam (Стандартное значение 0.001, графики для него выше)

- learning_rate=0.1

а

б

- Рисунок 6 – Графики точности и потерь для learning_rate=0.1
- learning_rate=0.01

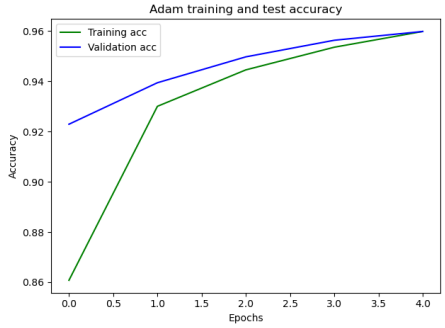


а

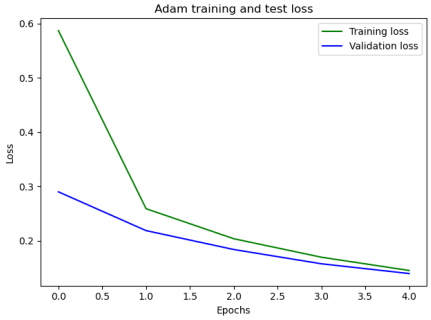
б

- Рисунок 7 – Графики точности и потерь для learning_rate=0.01

- learning_rate=0.0001



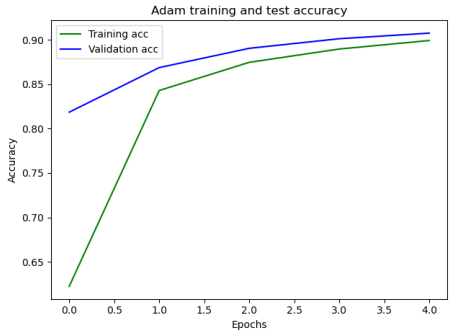
а



б

Рисунок 8 – Графики точности и потерь для learning_rate=0.0001

- learning_rate=0.00001



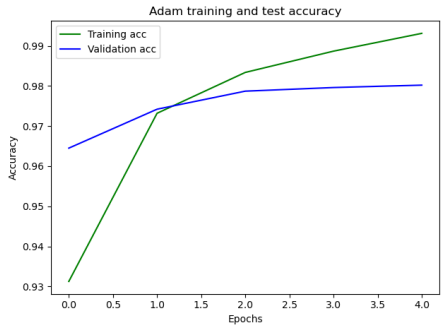
а

б

Рисунок 9 – Графики точности и потерь для learning_rate=0.00001

Как видим, лучше оставить значение по умолчанию 0.001.

Попробуем поставить параметр amsgrad=True.



а

б

Рисунок 10 – Графики точности и потерь для amsgrad=True

Как видим, графики стали немного плавнее, но особых результатов в точности это не принесло.

Напишем функцию для загрузки пользовательского изображения

```
def getimage(path):
    return (np.asarray(Image.open(path).convert("L")) / 255.0)[newaxis, :, :]
```

Протестируем нашу сеть на пользовательских изображениях, для этого нарисуем в paint'е цифры и уменьшим их до размера 28x28.

Функция для вывода предсказаний:

```
print(mainModel.predict(img))
print(np.argmax(mainModel.predict(img)))
```

В первой строчке – вероятность быть каждой из цифр, во второй – номер с наибольшей вероятностью – предсказание сети.

Тест 1:

Изображение:

Результат:

```
[[7.3293038e-03 7.2464123e-03 9.2430627e-01 5.9442483e-03 3.8432814e-02
  1.7432420e-04 1.0950598e-02 4.4098729e-03 1.0227599e-03 1.8341975e-04]]
2
```

Тест 2:

Изображение:

Результат:

```
[[1.0207805e-04 2.4687180e-03 9.3111154e-03 9.3821126e-01 3.3727449e-03
  3.6313370e-02 8.6796668e-04 1.5977441e-03 4.3328758e-03 3.4219918e-03]]
3
```

Тест 3:

Изображение:

Результат:

```
[[1.1617091e-03 6.0024031e-04 3.1683335e-04 5.7655852e-04 8.8687909e-01
  1.2074018e-02 8.1500201e-04 2.6168380e-02 1.8172398e-02 5.3235698e-02]]
4
```

Как видим, сеть довольно точно определила пользовательские рисунки

Выводы.

В ходе выполнения данной работы было изучено представление графических данных. Была построена и протестирована на пользовательских изображениях сеть с точностью ~98%.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД

```
import numpy as np
from numpy import newaxis
from keras.layers import Dense, Flatten
from keras.models import Sequential
from keras.utils import to_categorical
from keras.datasets import mnist
from keras import optimizers
from PIL import Image
import matplotlib.pyplot as plt


def getimage(path):
    return (np.asarray(Image.open(path).convert("L")) /
255.0)[newaxis, :, :]


def createModel(opt, name):
    (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)

    model = Sequential()
    model.add(Flatten(input_shape=(28, 28)))
    model.add(Dense(1024, activation='relu'))
    model.add(Dense(10, activation='softmax'))
    model.compile(optimizer=opt, loss='categorical_crossentropy',
metrics=['accuracy'])
    h = model.fit(train_images, train_labels, epochs=5,
batch_size=128, verbose=0,
                  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)
```

```
        plt.title("{} training and test accuracy".format(name))
        plt.plot(h.history['accuracy'], 'g', label='Training acc')
        plt.plot(h.history['val_accuracy'], 'b', label='Validation
acc')
        plt.xlabel('Epochs')
        plt.ylabel('Accuracy')
        plt.legend()
        plt.show()

        plt.clf()
        plt.title("{} training and test loss".format(name))
        plt.plot(h.history['loss'], 'g', label='Training loss')
        plt.plot(h.history['val_loss'], 'b', label='Validation loss')
        plt.xlabel('Epochs')
        plt.ylabel('Loss')
        plt.legend()
        plt.show()
        plt.clf()
        return model


    mainModel = createModel(optimizers.Adam(), 'Adam')
    # createModel(optimizers.sgd(), 'sgd')
    # createModel(optimizers.RMSprop(), 'RMSprop')
    # createModel(optimizers.Adagrad(), 'Adagrad')
    # createModel(optimizers.Nadam(), 'Nadam')

    img = getimage('2.bmp')
    print(mainModel.predict(img))
    print(np.argmax(mainModel.predict(img)))

    img = getimage('3.bmp')
    print(mainModel.predict(img))
    print(np.argmax(mainModel.predict(img)))

    img = getimage('4.bmp')
    print(mainModel.predict(img))
    print(np.argmax(mainModel.predict(img)))
```