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

ЛАБОРАТОРНАЯ РАБОТА №4

по дисциплине «Искусственные нейронные сети»

Тема: «Распознавание рукописных символов»

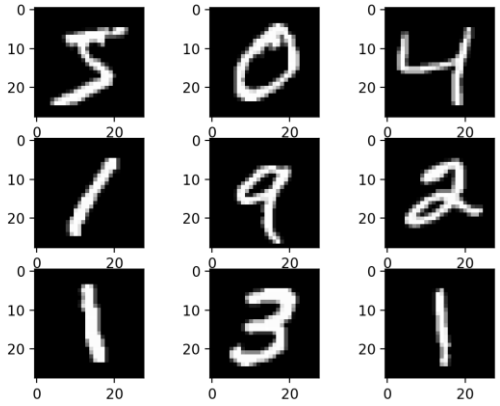
Студентка гр. 7383 _____ Прокопенко Н.

Преподаватель _____ Жукова Н. А.

Санкт-Петербург
2020

Цели.

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.

Задачи.

- 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
- 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.

Задаем архитектуру сети, настраиваем параметры для этапа компиляции:

• функцию потерь, которая определяет, как сеть должна оценивать качество своей работы на обучающих данных и, соответственно, как корректировать ее в правильном направлении

• оптимизатор — механизм, с помощью которого сеть будет обновлять себя, опираясь на наблюдаемые данные и функцию потерь

• метрики для мониторинга на этапах обучения и тестирования — здесь нас будет интересовать только точность (доля правильно классифицированных изображений).

Сравним работу сети под влиянием различных оптимизаторов и их параметров. На рис. 1-8 представлены графики ошибки и точности для оптимизаторов, а в табл.1 представлены значения точности:

Таблица 1 – значения точности

	Adam	Adagrad	SGD	RMSprop
0.001	0.9780	0.8727	0.9819	0.9782
0.01	0.9729	0.9832	0.9795	0.9747

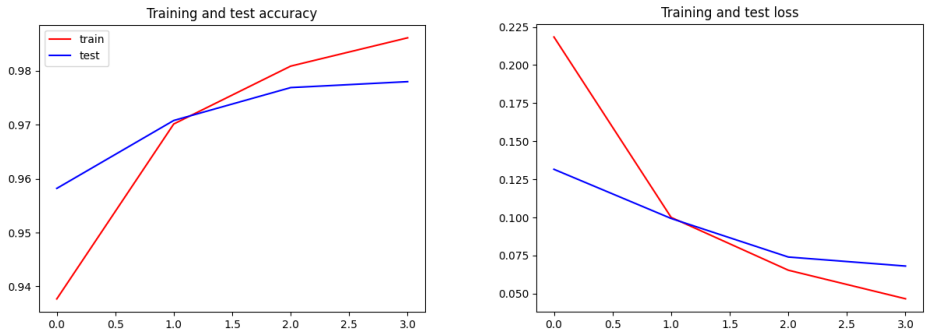


Рисунок 1 – Adam, learning_rate = 0.001

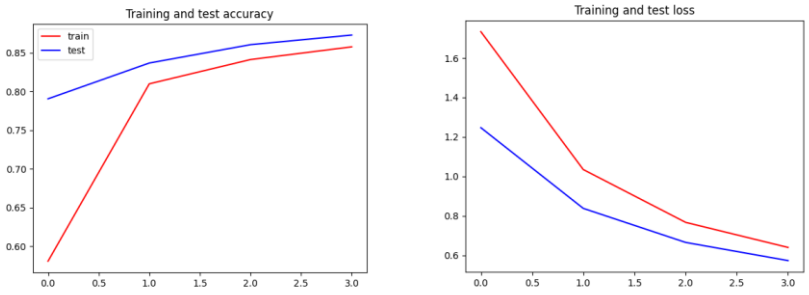


Рисунок 2– Adagrad, learning_rate = 0.001

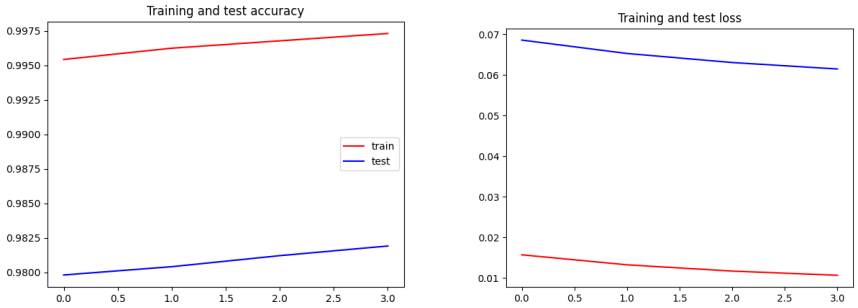


Рисунок 3 – SDG, learning_rate = 0.001

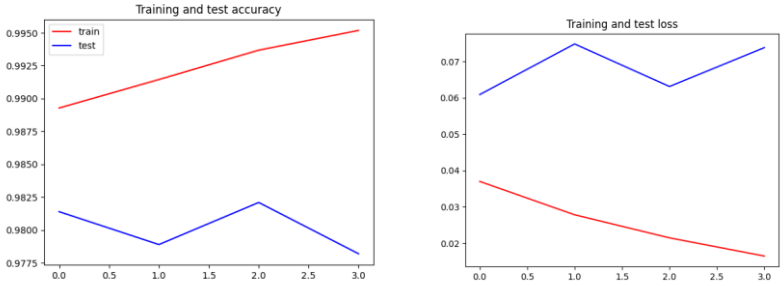


Рисунок 4 – RMSprop, learning_rate = 0.001

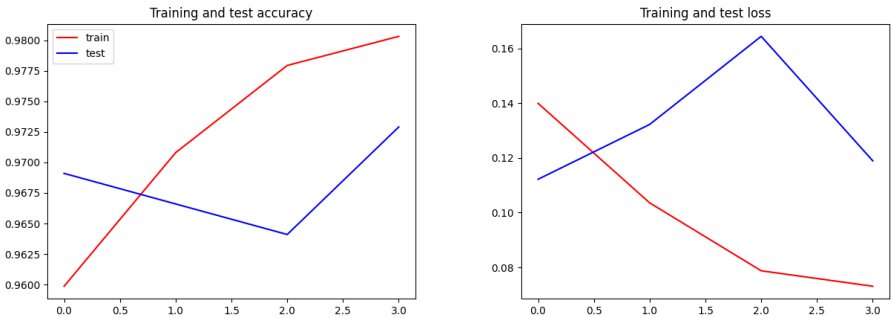


Рисунок 5 – Adam, learning_rate = 0.01

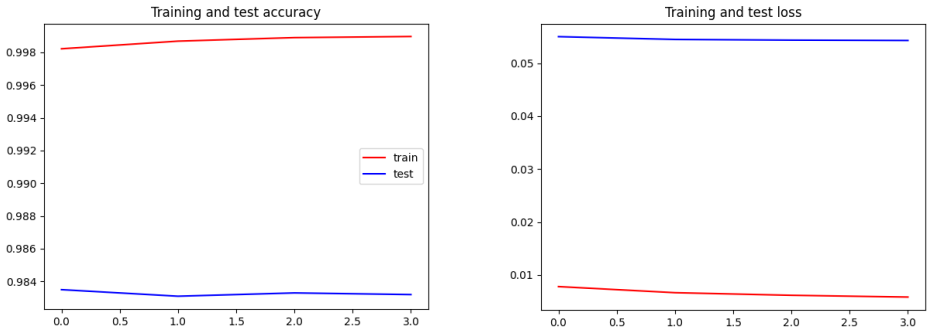


Рисунок 6 – Adagrad, learning_rate = 0.01

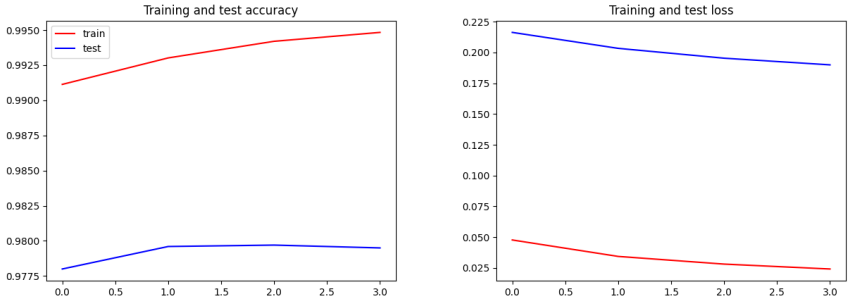


Рисунок 7 – SDG, learning_rate = 0.01

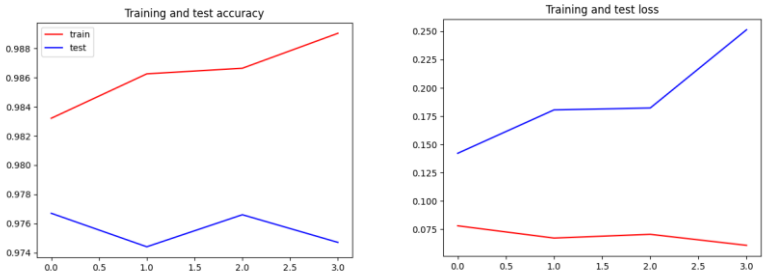


Рисунок 8 – RMSprop, learning_rate = 0.01

Можно заметить, что показатели улучшились, при увелечении скорости обучения для Adagrad. Для оптимизатора Adam, SGD и RMSprop, при увеличении скорости обучения, точность уменьшилась, а ошибка увеличилась.

Из графиков и значений следует, что наилучшая работа сети наблюдается при заданном оптимизаторе Adagrad с параметром learning_rate = 0.01

Вывод.

В ходе выполнения лабораторной работы была определена архитектура сети, при которой точность классификации будет не менее 95%. Было исследовано влияние различных оптимизаторов, а также их параметров, на процесс обучения. Написана функция, которая позволит загружать пользовательское изображение не из датасета. Код программы представлен в приложении А.

Приложения

Приложение А

```
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.preprocessing.image import load_img, img_to_array
from tensorflow.keras import optimizers


def load_img(path):
    img = load_img(path=path, target_size=(28, 28))
    return img_to_array(img)


mnist = tf.keras.datasets.mnist
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
train_images = train_images.reshape((60000, 28 * 28))
train_images = train_images / 255.0
test_images = test_images.reshape((10000, 28 * 28))
test_images = test_images / 255.0
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(28 * 28,)))
model.add(Dense(10, activation='softmax'))


def run_research(optimizer):
    optimizerConf = optimizer.get_config()
    model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['accuracy'])
    history = model.fit(train_images, train_labels, epochs=4, batch_size=128, validation_data=(test_images, test_labels))
    test_loss, test_acc = model.evaluate(test_images, test_labels)
    print('test_acc:', test_acc)
    print('optimizer', optimizerConf)
    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("%s_%s_%s_acc.png"   %   (optimizerConf["name"],
optimizerConf["learning_rate"], test_acc), 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.savefig("%s_%s_%s_loss.png"  %  (optimizerConf["name"],
optimizerConf["learning_rate"], test_acc), format='png')
    plt.legend()
    plt.clf()
    return model

for learning_rate in [0.001, 0.01]:

  run_research(optimizers.Adagrad(learning_rate=learning_rate))
    run_research(optimizers.Adam(learning_rate=learning_rate))

run_research(optimizers.RMSprop(learning_rate=learning_rate))
    run_research(optimizers.SGD(learning_rate=learning_rate))
ctions_single("five.png", model))
```