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

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Искусственные нейронные сети»

Тема: Распознавание рукописных символов

Студент гр. 8382 _____ Кобенко В.П.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2021

Цель работы.

Реализовать классификацию черно-белых изображений рукописных цифр (28x28) 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, позволяющая загружать изображение пользователи и классифицировать его.

Требования к выполнению задания.

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.

Основные теоретические положения.

MINIST – база данных рукописных цифр, имеющая подготовленный набор обучающих значений в размере 60000 примеров и тестовых значений из 10000 примеров. Это подмножество более широкого набора, доступное из NIST. Наш набор состоит из изображений размером 28x28, каждый пиксель которого представляет собой оттенок серого, цифры нормализированы по размеру и имеют фиксированный размер изображения. Таким образом, есть 10 цифр (от 0 до 9) или 10 классов для прогнозирования.

Результаты сообщаются с использованием ошибки прогнозирования, которая является не чем иным, как инвертированной точностью классификации.

Ход работы.

Была создана и обучена модель искусственной нейронной сети. Код предоставлен в приложении А.

1. Оптимизатор adam.

Архитектура:

- Скорость обучения = 0.001.
- Инициализация весов – normal.
- Epochs = 7, batch_size = 100, loss = categorical_crossentropy
- Слои:

```
model.add(Dense(128, activation='relu', input_shape=(784,)))
model.add(Dense(100, activation='relu'))
model.add(Dense(10, activation='softmax'))
```

Данная архитектура дает точность ~ 97%. Графики точности и ошибки предоставлены на рис. 1 и рис. 2 соответственно.

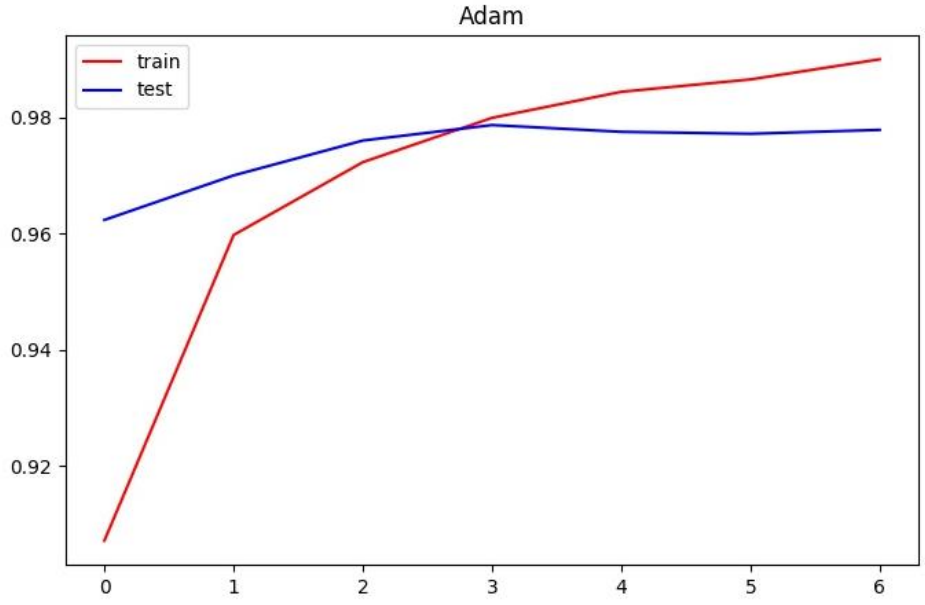

Рисунок 1 – График точности для оптимизатора adam

Рисунок 2 – График потерь для оптимизатора adam

2. Оптимизатор SGD.

Архитектура:

- Скорость обучения = 0.001, momentum = 0.2.
- Инициализация весов – normal.
- Epochs = 7, batch_size = 100, loss = categorical_crossentropy
- Слои:

```
model.add(Dense(128, activation='relu', input_shape=(784,)))
model.add(Dense(100, activation='relu'))
model.add(Dense(10, activation='softmax'))
```

Данная архитектура дает точность ~ 84%. Графики точности и ошибки предоставлены на рис. 3 и рис. 4 соответственно.

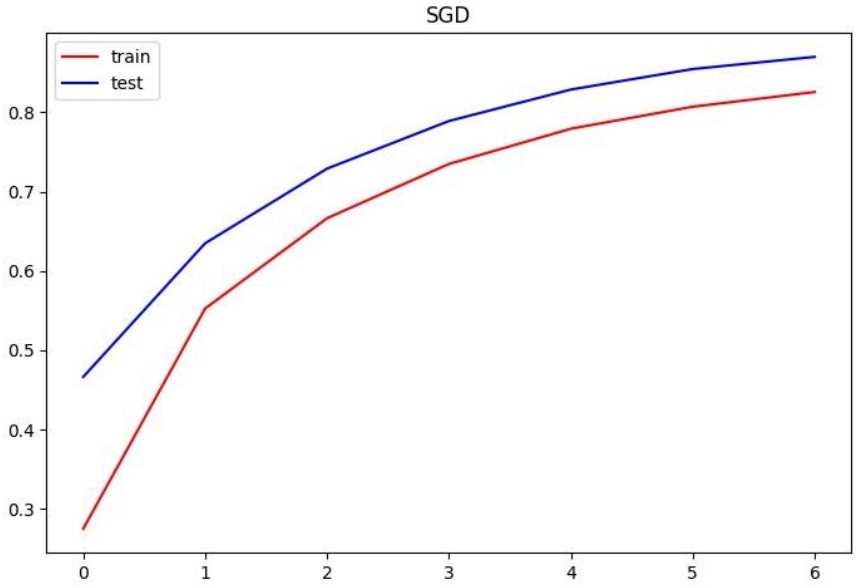

Рисунок 3 – График точности для оптимизатора SGD

Рисунок 4 – График потерь для оптимизатора SGD

3. Оптимизатор RMSprop.

Архитектура:

- Скорость обучения = 0.001.
- Инициализация весов – normal.
- Epochs = 7, batch_size = 100, loss = categorical_crossentropy
- Слои:

```
model.add(Dense(128, activation='relu', input_shape=(784,)))
model.add(Dense(100, activation='relu'))
model.add(Dense(10, activation='softmax'))
```

Данная архитектура дает точность ~ 97%. Графики точности и ошибки предоставлены на рис. 5 и рис. 6 соответственно.

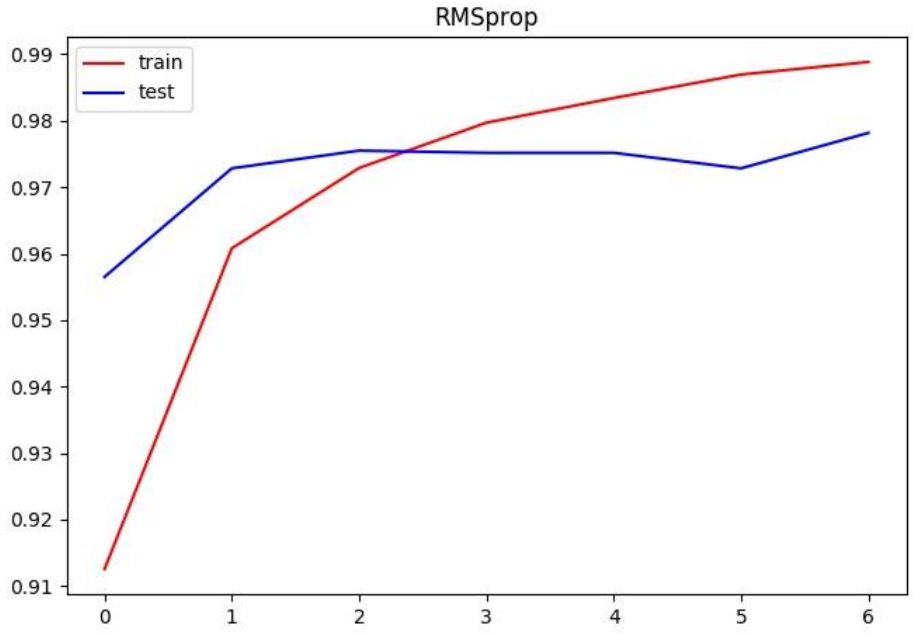

Рисунок 5 – График точности для оптимизатора RMSprop

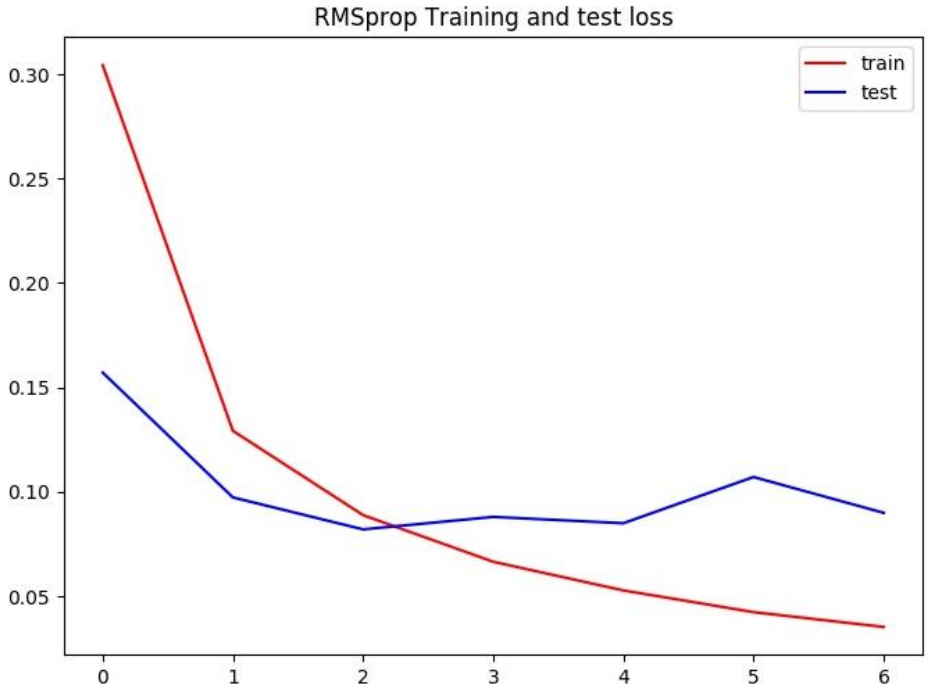

Рисунок 6 – График потерь для оптимизатора RMSprop

Напишем функцию, которая позволит загружать пользовательское изображение не из датасета:

```
def upload_image(path):
    img = image.load_img(path=path, grayscale=True, target_size=(28, 28, 1))
    img = image.img_to_array(img)
    return img.reshape((1, 784))
```

Полный код программы предоставлен в приложении А.

Выводы.

В ходе работы была изучена задача классификации рукописных цифр с помощью азы данных MINIST. Подобраны архитектуры, дающие точность свыше 95%, таковыми оказали adam и RMSprop. Также была написана функция загрузки изображения в память программы.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

```
import tensorflow as tf
from tensorflow import keras
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.preprocessing import image
from tensorflow.keras.utils import to_categorical


mnist = keras.datasets.mnist
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
train_images = train_images.reshape(60000, 784)
test_images = test_images.reshape(10000, 784)
train_images = train_images / 255.0
test_images = test_images / 255.0
train_labels = to_categorical(train_labels, 10)
test_labels = to_categorical(test_labels, 10)


def upload_image(path):
    img = image.load_img(path=path, grayscale=True, target_size=(28, 28, 1))
    img = image.img_to_array(img)
    return img.reshape((1, 784))


def build_model():
    model = Sequential()
    model.add(Dense(128, activation='relu', input_shape=(784,)))
    model.add(Dense(100, activation='relu'))
    model.add(Dense(10, activation='softmax'))
    return model



def draw(H, title_name):
    plt.figure(1, figsize=(8, 5))
    plt.title(title_name)
    plt.plot(H.history['acc'], 'r', label='train')
    plt.plot(H.history['val_acc'], 'b', label='test')
    plt.legend()
    plt.show()
    plt.clf()
    plt.figure(1, figsize=(8, 5))


    plt.title("{} Training and test loss".format(title_name))
    plt.plot(H.history['loss'], 'r', label='train')
```

```
    plt.plot(H.history['val_loss'], 'b', label='test')
    plt.legend()
    plt.show()
    plt.clf()


path = 'C:/Users/vladk/INS2021/lab4/odin.png'
upload_image(path)


optimizer = tf.keras.optimizers.Adam(lr=0.001)
model = build_model()
model.compile(optimizer=optimizer, loss='categorical_crossentropy',
metrics=['acc'])
H = model.fit(train_images, train_labels, epochs=7, batch_size=100,
validation_split=0.1)
test_loss, test_acc = model.evaluate(test_images, test_labels)
draw(H, "Adam")
model = build_model()


optimizer = tf.keras.optimizers.SGD(lr=0.001,momentum=.2)
model.compile(optimizer=optimizer, loss='categorical_crossentropy',
metrics=['acc'])
H = model.fit(train_images, train_labels, epochs=7, batch_size=100,
validation_split=0.1)
test_loss, test_acc = model.evaluate(test_images, test_labels)
draw(H, "SGD")
model = build_model()


optimizer = tf.keras.optimizers.RMSprop(lr=0.001)
model.compile(optimizer=optimizer, loss='categorical_crossentropy',
metrics=['acc'])
H = model.fit(train_images, train_labels, epochs=7, batch_size=100,
validation_split=0.1)
test_loss, test_acc = model.evaluate(test_images, test_labels)
draw(H, "RMSprop")
```