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

Студент гр. 8383 _____ Костарев К.В.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2021

Цель работы.

Реализовать классификацию черно-белых изображений рукописных цифр (28x28) по 10 категориям (от 0 до 9).

Постановка 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

Выполнение работы.

Для выполнения лабораторной работы был написан код программы на языке Python, которая сначала загружает готовые изображения. Набор данных MNIST уже входит в состав Keras в форме набора из четырех массивов Numpy:

```
mnist = tf.keras.datasets.mnist
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
```

Набор данных содержит 60,000 изображений для обучения и 10,000 изображений для тестирования.

Исходные изображения представлены в виде массивов чисел в интервале [0, 255]. Перед обучением их необходимо преобразовать так, чтобы все значения оказались в интервале [0, 1]. Также необходимо закодировать метки категорий. В данном случае прямое кодирование меток заключается в конструировании вектора с нулевыми элементами со значением 1 в элементе, индекс которого соответствует индексу метки.

```
train_images = train_images / 255.0
test_images = test_images / 255.0
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
```

Далее уже была построена архитектура сети с тремя слоями, из которых два скрытых, включая выходной, содержат 256 и 10 нейронов соответственно, с функциями активаций Relu и Softmax соответственно. Обучение было проведено в течение 5 эпох, размер выборки – 128, оптимизатор Adam, функция потерь – категориальная кросс-энтропия, метрика – точность.

```
model = Sequential()
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dense(10, activation='softmax'))
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.fit(train_images, train_labels, epochs=5, batch_size=128)
```

В конце работы будем выводить точность сети на обучающих данных.

```
test_loss, test_acc = model.evaluate(test_images, test_labels)
print('test_acc:', test_acc)
```

При вышеприведенной конфигурации сети ее точность составила 0.9789000153541565, что удовлетворяет условию (точность должна быть выше 95%).

Исследуем работу сети при различных оптимизаторах и их параметрах. Для этого будем создавать экземпляр оптимизатора с конкретными параметрами из библиотеки Keras:

```
op_1 = tf.keras.optimizers.Adam(learning_rate = 0.001, beta_1 = 0.9, beta_2 = 0.999)
op_2 = tf.keras.optimizers.RMSprop(learning_rate = 0.001, rho = 0.9)
op_3 = tf.keras.optimizers.Adagrad(learning_rate = 0.001)
```

Для оптимизатора Adam:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t + \epsilon}}\hat{m}_t$$

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t},\ \hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

Будем изменять значения η (скорость обучения), β_1, β_2 и исследовать зависимость точности от этих значений. Результаты приведены в табл. 1 (красным выделены значения по умолчанию).

Таблица 1 – Зависимость точности обучения от параметров оптимизатора Adam

η	β_1	β_2	Точность
0.001	0.9	0.999	0.9789000153541565
0.0001	0.9	0.999	0.9445000290870667
0.1	0.9	0.999	0.8707000017166138
0.001	0.6	0.999	0.977400004863739
0.001	0.9	0.8	0.9786999821662903

Изменение любого из параметров по-отдельности не улучшило точность обучения при значении параметров по умолчанию.

Для оптимизатора RMSprop:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t$$

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1 - \gamma) g_t^2$$

Будем изменять значения η (скорость обучения) и γ. Результаты приведены в табл. 2 (красным выделены значения по умолчанию).

Таблица 2 – Зависимость точности обучения от параметров оптимизатора RMSprop

η	γ	Точность
0.001	0.9	0.9776999950408936
0.0001	0.9	0.9444000124931335
0.01	0.9	0.9611999988555908
0.001	0.8	0.9750999808311462

0.001	0.99	0.9761999845504761

Изменение любого из параметров по-отдельности не улучшило точность обучения при значении параметров по умолчанию. Оптимизатор RMSprop практически не уступает оптимизатору Adam при выборе параметров по умолчанию.

Для оптимизатора Adagrad:

$$G_t = G_t + g_t^2$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} g_t$$

Исследуем зависимость точности обучения от параметра скорости η. Результаты приведены в табл. 3.

Таблица 3 – Зависимость точности обучения от параметров оптимизатора Adagrad

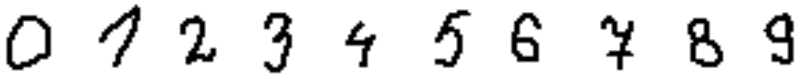
η	Точность
0.001	0.871999979019165
0.005	0.9199000000953674
0.01	0.9315999746322632
0.05	0.9671000242233276
0.1	0.9760000109672546
0.3	0.9794999957084656
0.5	0.9671000242233276

Можно заметить, что при увеличении параметра скорости обучения оптимизатора Adagrad повышается и точность модели. Максимальная точность был определена при η = 0.3. Также эта конфигурация дает максимальную точность при сравнении с другими оптимизаторами, хотя разница достаточно невелика.

Далее была реализована функция, которая загружает пользовательское изображение по файловому пути:

```
def load_image(path):
    image = load_img(path, color_mode='grayscale', target_size=(28, 28))
    image_array = img_to_array(image)
    image_array -= 255
    image_array = image_array / -255.0
    plt.imshow(image_array, cmap=plt.cm.binary)
    plt.show()
    image_array = np.asarray([image_array])
    return image_array
```

Далее были созданы собственные ч/б изображения с рукописными цифрами размером 28 на 28 пикселей:



Каждое изображение было загружено с помощью реализованной функции и передано в обученную модель:

```
for i in range(10):
    arr = load_image('images/' + str(i) + '.png')
    predict = model.predict(arr)
    print(np.argmax(predict, 1)[0])
```

В результате нейросетью были правильно определены 7 из 10 цифр (все, кроме 1, 6 и 7). Это возможно связано с тем, что изображения не являются слепками настоящих рукописных цифр, а были нарисованы на ноутбуке с помощью инструмента Карандаш в программе Paint без сглаживания).

Выводы.

В данной лабораторной работе была исследована зависимость точности ИНС от ее метода оптимизации и его параметров.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.models import Sequential
from tensorflow.keras.preprocessing.image import load_img, img_to_array
import matplotlib.pyplot as plt


def load_image(path):
    image = load_img(path, color_mode='grayscale', target_size=(28, 28))
    image_array = img_to_array(image)
    image_array -= 255
    image_array = image_array / -255.0
    plt.imshow(image_array, cmap=plt.cm.binary)
    plt.show()
    image_array = np.asarray([image_array])
    return image_array


mnist = tf.keras.datasets.mnist
(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())
model.add(Dense(256, activation='relu'))
model.add(Dense(10, activation='softmax'))
op_1 = tf.keras.optimizers.Adam(beta_2=0.8)
op_2 = tf.keras.optimizers.RMSprop(rho=0.99)
```

```
op_3 = tf.keras.optimizers.Adagrad(learning_rate=0.3)
model.compile(optimizer=op_3, loss='categorical_crossentropy', metrics=['accuracy'])
model.fit(train_images, train_labels, epochs=5, batch_size=128)

test_loss, test_acc = model.evaluate(test_images, test_labels)
print('test_acc:', test_acc)

for i in range(10):
    arr = load_image('images/' + str(i) + '.png')
    predict = model.predict(arr)
    print(np.argmax(predict, 1)[0])
```