**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. 8383 |  | Мололкин К.А. |
| Преподаватель |  | Жангиров Т.Р. |

Санкт-Петербург

2021

**Цель работы**

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

**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**

В начале работы был написан код загрузки набора данных MNIST. Затем данные были подготовлены для обучения нейронной сети, создана модель нейронной сети, которая состоит из трех слоев первый слой – Flatten, второй слой состоит из 256 нейронов и имеет функцию активации relu, третий слой состоит из 10 нейронов и имеет активацию softmax. Так же были настроены параметры компиляции модели – выбрана функция потерь, оптимизатор и метрика мониторинга. Количество эпох обучения было выбрано равным 10. Полученная нейронная сеть имеет точность равную 97.8%.

Следующим шагом было проведено исследование влияния оптимизаторов.

Первая модель использует оптимизатор Adam с шагом сходимости 0.001. Затем шаг сходимости был уменьшен до 0.00001, точность уменьшилась до 88.5%. Увечив сходимость до 0.1, получим точность в 90%.

Изучим оптимизатор RMSprop с шагом сходимости 0.001 и rho = 0.9, точность равна 98%. Увеличим rho до 0.95, точность не сильно поменялась 97.8%. При шаге сходимости в 0.01, точность равна 97.5%. При шаге равном 0.00001, точность равна 0.95.

Оптимизатор Nadam при шаге сходимости равном 0.001 дает точность модели в 97.7%. При шаге в 0.1, точность равна 88.4%, при шаге в 0.0001, точность 95%

Таким образом лучший шаг сходимости для оптимизаторов равен 0.001. Так же можно сделать вывод, что на точность модели с оптимизатором RMSprop параметр rho, имеет малое влияние и также при изменении шага, точность модели меняется слабо.

Последним шагом была написана функция, которая загружает изображения и классифицирует их с помощью построенной модели нейронной сети. Код программы представлен в приложении А.

**Вывод**

Во время выполнения лабораторной работы была создана ИНС, которая классифицирует черно-белые рукописные цифры по 10 категориям от 0 до 9. Так же были изучены разные виды оптимизаторов и их влияние на точность модели. Еще был написана функция по вводу пользовательских изображений и классификации их.

**Приложение А**

**Листинг программы**

import numpy as np

from PIL import Image

import tensorflow as tf

import tensorflow.keras.optimizers as opts

from keras.utils import to\_categorical

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Activation, Flatten

def input\_img(path):

img = Image.open(path).convert('L')

matr\_img = np.array(img) / 255.0

matr\_img = np.array([matr\_img])

return matr\_img

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'))

#opt = opts.RMSprop(learning\_rate=0.01, rho=0.9)

#opt = opts.Nadam(learning\_rate=0.01, beta\_1=0.99, beta\_2=0.999)

#opt = opts.Adagrad(learning\_rate=0.01)

#opt = opts.Adam(learning\_rate=0.01, beta\_1=0.99, beta\_2=0.999)

#opt = opts.Adadelta(learning\_rate=0.01, rho=0.95)

opt = opts.Adamax(learning\_rate=0.01, beta\_1=0.99, beta\_2=0.999)

model.compile(opt, loss='categorical\_crossentropy', metrics=['accuracy'])

model.fit(train\_images, train\_labels, epochs=10, batch\_size=256, verbose=False)

test\_loss, test\_acc = model.evaluate(test\_images, test\_labels)

print('test\_acc:', test\_acc)

print('test\_loss:', test\_loss)

for i in range(0, 10):

img\_arr = input\_img('./images/' + str(i) + '.png')

result = model.predict(img\_arr)

print(np.argmax(result))