**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’e.

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’e цифры и уменьшим их до размера 28х28.

Функция для вывода предсказаний:

print(mainModel.predict(img))

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

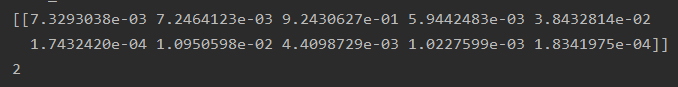
В первой строчке – вероятность быть каждой из цифр, во второй – номер с наибольшей вероятностью – предсказание сети.

Тест 1:

Изображение:

C:\Users\mvolo\PycharmProjects\lab4\2.bmp

Результат:

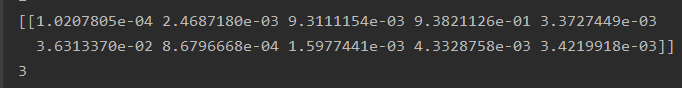


Тест 2:

Изображение:

C:\Users\mvolo\PycharmProjects\lab4\3.bmp

Результат:

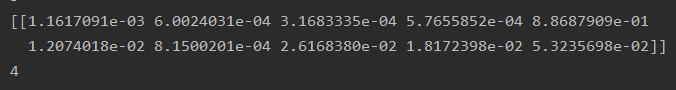


Тест 3:

Изображение:

C:\Users\mvolo\PycharmProjects\lab4\4.bmp

Результат:



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)))