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 цифр

Студент гр. 8383 _____ Дейнега В. Е.

Преподаватель _____ Жангиров Т. Р.

Санкт-Петербург

2021

Цель работы

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

Задание.

- 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

Выполнение работы

1) Создадим модель ИНС.

Была реализована модель ИНС из методических указаний.

```
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'])
hist = model.fit(train_images, train_labels, epochs=5,
batch_size=128)
```

Точность этой модели составила 97.7%, такая архитектура подходит по условию, однако стоит рассмотреть другие оптимизаторы и их параметры.

2) Протестируем параметры оптимизаторов.

Используем AMSGrad-вариант алгоритма Adam, все остальные параметры оставлены по умолчанию.

```
opt = tf.keras.optimizers.Adam(lr=0.001, beta_1=0.9, beta_2=0.999,
epsilon=None, decay=0.0, amsgrad=True, clipnorm=1., clipvalue=0.5)
```

Точность варьируется от 97.8% до 97.9%

Увеличим экспоненциально убывающий коэффициент обновления смещенной оценки первого момента до 0.99

```
opt = tf.keras.optimizers.Adam(lr=0.001, beta_1=0.99, beta_2=0.999,
epsilon=None, decay=0.0, amsgrad=True, clipnorm=1., clipvalue=0.5)
```

Точность составила 96.8%, оставим этот параметр равным 0.9.

Экспериментально было выяснено, что экспоненциально убывающий коэффициент обновления смещенной оценки второго момента дает оптимальный результат, будучи равным 0.999 (по умолчанию).

Коэффициент убывания скорости обучения (decay) изменялся в диапазоне от 0.1 до 0.9, везде точность существенно упала, по сравнению с `decay = 0.0`, оставим этот параметр равным нулю.

Изменим параметр lr (скорость обучения)

```
opt = tf.keras.optimizers.Adam(lr=0.01, beta_1=0.99, beta_2=0.999,
epsilon=None, decay=0.0, amsgrad=True, clipnorm=1., clipvalue=0.5)
```

Точность составила 96.7%

```
opt = tf.keras.optimizers.Adam(lr=0.0001,beta_1=0.99, beta_2=0.999,
epsilon=None, decay=0.0, amsgrad=True, clipnorm=1., clipvalue=0.5)
```

Точность составила 94.5%

Для алгоритма adam оптимальными параметрами будет:

```
opt = tf.keras.optimizers.Adam(lr=0.001, beta_1=0.9, beta_2=0.999,
epsilon=None, decay=0.0, amsgrad=True, clipnorm=1., clipvalue=0.5)
```

Протестируем другие алгоритмы оптимизации:

Алгоритм SGD (Стохастический оптимизатор градиентного спуска).

```
opt = tf.keras.optimizers.SGD(lr=0.01, momentum=0.0, decay=0.0, nesterov=False, clipnorm=1.0, clipvalue=0.5)
```

Точность составила 91.1%

Изменим скорость обучения.

```
opt = tf.keras.optimizers.SGD(lr=0.9, momentum=0.0, decay=0.0, nesterov=False, clipnorm=1.0, clipvalue=0.5)
```

Точность составила 98.1%

Используем импульс Нестерова

```
opt = tf.keras.optimizers.SGD(lr=0.9, momentum=0.0, decay=0.0, nesterov=True, clipnorm=1.0, clipvalue=0.5)
```

Точность составила 97.8%

Алгоритм RMSprop

```
opt = tf.keras.optimizers.RMSprop(lr=0.001)
```

Точность составила 97.7%

При изменении скорости обучения, не удалось добиться повышения точности.

Алгоритм Adagrad

```
opt = tf.keras.optimizers.Adagrad(lr=0.01)
```

Точность составила 93.3%

Изменяя значение скорости обучения, при lr = 0.18 удалось добиться точности **98.0%**

Алгоритм Adadelta

```
opt = tf.keras.optimizers.Adadelta(lr=1.0, rho=0.95)
```

Точность составила 97.8%

Изменяя скорость обучения, существенного прибавления точности добиться не удалось, однако стоит отметить, что при отдельных запусках точность достигала 98.0%, однако эти результаты нестабильны.

Алгоритм Adamax

```
opt =tf.keras.optimizers.Adamax(lr=0.001, beta_1=0.9, beta_2=0.999)
```

Точность составила 96.8%

При изменении скорости, не удалось добиться повышения точности.

Алгоритм Nadam

```
opt = tf.keras.optimizers.Nadam(lr=0.002, beta_1=0.9, beta_2=0.999)
```

Точность составила 97.7%

При изменении скорости, не удалось добиться повышения точности.

3) Была написана функция `get_img(path)`, позволяющая загружать пользовательское изображение

```
def get_img(path):
    image_file = Image.open(path)
    image_file = image_file.convert('L')
    resized_image = image_file.resize((28, 28))
    resized_image.save("result.png", "PNG")
    return np.array(resized_image) / 255.0
```

Также были реализованы несколько вспомогательных функций, таких как

```
def load_model():
```

Которая позволяет загрузить модель из файлов, а не обучать каждый раз заново.

```
def predict(model, img)
```

Которая на основе обученной модели и изображению, представленном в виде массива делает препдоположение какая цифра изображена.

```
def interface()
```

Внутри которой идет общение с пользователем и вызов нужных функций. Полный листинг представлен в приложении А.

Выводы.

В ходе лабораторной работы была реализована классификация черно-белых изображений рукописных цифр (28x28) по 10 категориям (от 0 до 9). В ходе испытаний было выяснено, что наилучшие результаты показали следующие оптимизаторы с приведенными параметрами:

```
tf.keras.optimizers.Adam(lr=0.001, beta_1=0.9, beta_2=0.999,
epsilon=None, decay=0.0, amsgrad=True, clipnorm=1., clipvalue=0.5)

tf.keras.optimizers.SGD(lr=0.9,momentum=0.0,decay=0.0,nesterov=Fal
se, clipnorm=1.0, clipvalue=0.5)

tf.keras.optimizers.Adagrad(lr=0.01)
```

Приложение А

```
import pandas
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.models import Sequential
from tensorflow.keras.utils import to_categorical
from keras.models import model_from_json
import tensorflow as tf
from PIL import Image
import numpy as np


def get_img(path):
    image_file = Image.open(path)
    image_file = image_file.convert('L')
    resized_image = image_file.resize((28, 28))
    resized_image.save("result.png", "PNG")
    return np.array(resized_image) / 255.0


def interface():
    print("Type path to 28x28 (recomended) .png file:")
    path = input()
    predict(load_model(), get_img(path))

def create_model():
    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)
    #       opt       =       tf.keras.optimizers.Adam(lr=0.001,       beta_1=0.9,
beta_2=0.999,epsilon=None, decay=0.0, amsgrad=True, clipnorm=1., clipvalue=0.5)
    opt = tf.keras.optimizers.SGD(lr=0.9, momentum=0.0, decay=0.0, nesterov=False,
clipnorm=1.0, clipvalue=0.5)
    # opt = tf.keras.optimizers.Adagrad(lr=0.18)
    #    opt    =    tf.keras.optimizers.Nadam(learning_rate=0.003,    beta_1=0.9,
beta_2=0.999)
    model = Sequential()
    model.add(Flatten())
    model.add(Dense(256, activation='relu'))
    model.add(Dense(10, activation='softmax'))
    model.compile(optimizer=opt,                 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)

    model_json = model.to_json()
    with open("model.json", "w") as json_file:
        json_file.write(model_json)
    model.save_weights("model.h5")



def predict(model, img):
    # mnist = tf.keras.datasets.mnist
```

```
            # (train_images, train_labels), (test_images, test_labels) = mnist.load_data()
            #
            # test_images = test_images / 255.0
            # test_labels = to_categorical(test_labels)
            #
            # test_loss, test_acc = model.evaluate(test_images, test_labels)
            # print('test_acc:', test_acc)
            pred = model.predict(np.array([img]))
            print(np.argmax(pred))

        def load_model():
            json_file = open('model.json', 'r')
            loaded_model_json = json_file.read()
            json_file.close()
            loaded_model = model_from_json(loaded_model_json)
            loaded_model.load_weights("model.h5")
            optim    =    tf.keras.optimizers.SGD(lr=0.9,    momentum=0.0,    decay=0.0,
nesterov=False, clipnorm=1.0, clipvalue=0.5)
            loaded_model.compile(optimizer=optim,        loss='categorical_crossentropy',
metrics=['accuracy'])
            return loaded_model


        interface()
```