**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 _____ Федоров И.А.

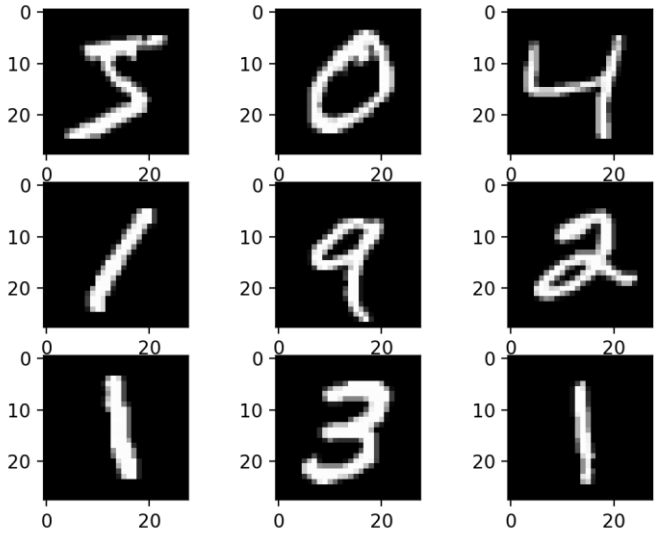
Преподаватель _____ Жангиров Т.Р.

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.**

Набор данных MNIST уже входит в состав Keras в форме набора из четырех массивов Numpy, поэтому он просто импортируется из модуля:

```
from tensorflow.keras.datasets import mnist
```

Здесь train_images и train_labels — это тренировочный набор, то есть данные, необходимые для обучения. После обучения модель будет проверяться тестовым (или контрольным) набором, test_images и test_labels.

Изображения хранятся в массивах Numpy, а метки — в массиве цифр от 0 до 9. Изображения и метки находятся в прямом соответствии, один к одному.

Перед обучение данные преобразовываютсяв форму удобную для нейронной сети. Выполняется масштабирование значение так, чтобы они оказались в интервале [0,1]. Изначально данные хранятся в трехмерном массиве (60000, 28, 28) и значения находятся в интервале [0, 255], в результате будет получен массив с формой (60000, 28 * 28) (слой сети Flatten(), использующийся в методическом пособие делает тоже самое). Также необходимо закодировать метки категорий. В данном случае используется прямое кодирование.

```
train_images = train_images.reshape((60000, 28 * 28))
train_images = train_images.astype('float32') / 255
test_images = test_images.reshape((10000, 28 * 28))
test_images = test_images.astype('float32') / 255
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
```

После этого создается сама сеть:

```
model = Sequential()
model.add(Dense(256, activation='relu', input_shape=(28 * 28,)))
model.add(Dense(10, activation='softmax'))
model.compile(optimizer='rmsprop',
loss='categorical_crossentropy', metrics=['accuracy'])
```

В результате данная сеть показала точность на контрольном наборе равную **0.977**.

Были проведены эксперименты запуска с различными оптимизаторами и их параметрами, результаты точности на контрольных данных которых показаны в таблице 1.

- *learning_rate* - Скорость обучения.
- *amsgrad* - Применять ли AMSGrad вариант этого алгоритма

Таблица 1

| Оптимизатор и его параметры | Точность |
|---|---|
| SGD(по умолчанию) | 0.9119 |

| | |
|---|---|
| SGD(learning_rate=0.001) | 0.8086 |
| SGD(learning_rate=0.1) | 0.9499 |
| RMSprop(по умолчанию) | 0.9771 |
| RMSprop(learning_rate=0.1) | 0.8827 |
| Adam(по умолчанию) | 0.9772 |
| Adam(learning_rate=0.01) | 0.9560 |
| Adam(learning_rate=0.01, amsgrad=**True**) | 0.9731 |
| Adamax(по умолчанию) | 0.9744 |
| Adamax(learning_rate=0.2) | 0.9282 |

Видно, что выбор параметра, отвечающего за скорость обучения сильно влияет на конечный результат. Можно сделать вывод, что для данного случая самыми оптимальными являются оптимизаторы **RMSprop** с установками по умолчанию, и **Adam**.

Была реализована функция `image_2_tensor()`, которая позволяет загружать пользовательское изображение, и переводить его в тензор необходимого формата. Полный программы представлен в приложении. Для загрузки и работы с изображение была импортирована библиотека ***PIL***.

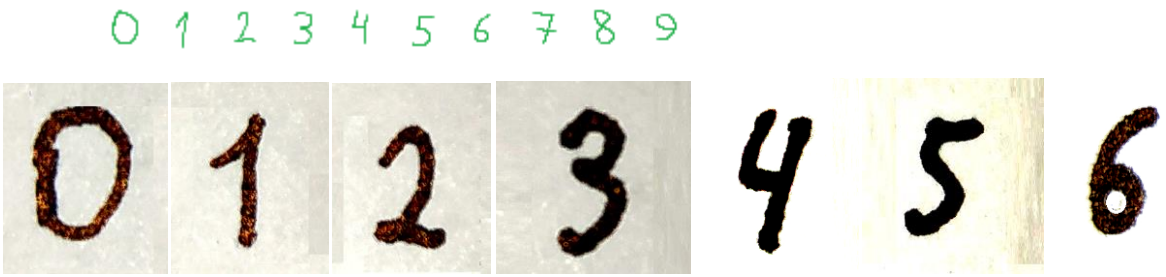
```
from PIL import Image
def image_2_tensor(image_name, save_name=None, resize_=False,
brightness=0.8, newsize=(28, 28), format='JPEG',is_bw=False,
norm=False, plot_num=False)
```

Функция загружает изображение, при необходимость изменяет его размер (например до (28, 28) как в mnist), после чего преобразует изображение в черно-белый формат исходя из формулы:

$$separator = 255 / brightness / 2 * 3$$

где *brightness* - яркость, передаваемая в функцию. После обработки изображения оно с помощью функций numpy переводится в тензор нужной формы (RGB форма отбрасывается, т.к. в данном случае сеть работает с трехмерными тензорами).

Были проведены эксперименты с двумя наборами изображений, отличных от библиотеки mnist:



Первый набор имеет размер 28*28, второй набор - фотографии написанных от руки цифр, имеют другой размер.

В первом наборе ошибается в 1-2 цифрах:

```
ilya@ilya-Aspire-A315-51: ~/NN_Keras_ETU_3_COURSE/Laboratory/LR_
Файл  Правка  Вид  Поиск  Терминал  Справка
 1.7624445e-02 5.0697131e-03 3.1733483e-02 3.5241561e-04 3.5771796e-01]
I think number:  0
I think number:  5
I think number:  2
I think number:  3
I think number:  4
I think number:  5
I think number:  6
I think number:  7
I think number:  8
I think number:  9
[[9.03096437e-01 7.70800455e-07 1.68777979e-03 7.94434600e-05
  2.98722793e-04 6.78971130e-03 5.34563733e-04 2.83016283e-02
  1.02404025e-04 5.91084957e-02]
```

Во втором наборе ошибается в 1 цифре:

```
ilya@ilya-Aspire-A315-51: ~/NN_Keras_ETU_3_COURSE/Laboratory/LR_4
Файл  Правка  Вид  Поиск  Терминал  Справка
8383_Fedorov_Lr4.py:28: UserWarning: The image need to resize!
  warnings.warn('The image need to resize!')
[9.9230075e-01 1.2305432e-12 6.6423728e-03 2.8773475e-06 2.4144728e-07
 4.1112989e-07 1.2779487e-05 1.0277886e-03 9.7111042e-06 3.0082938e-06]
I think number:  0
I think number:  1
I think number:  2
I think number:  3
I think number:  4
I think number:  8
I think number:  6
[[9.92300749e-01 1.23054323e-12 6.64237281e-03 2.87734747e-06
  2.41447282e-07 4.11129889e-07 1.27794874e-05 1.02778862e-03
  9.71110421e-06 3.00829379e-06]
```

**Выводы.**

В ходе выполнения данной работы была реализована ИНС для классификации черно-белых изображений рукописных цифр (28x28) по 10 категориям (от 0 до 9). Была найдена конфигурация, дающая 95% и более точности на контрольных данных, были проведены эксперименты с разными оптимизаторами и их параметрами. Была реализована функция, позволяющая загружать пользовательское изображения и преобразовывать его в тензор.

# Приложение

```
import pandas
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.layers import Dense, Activation, Flatten
from tensorflow.keras.models import Sequential
from tensorflow.keras.utils import to_categorical
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.utils import plot_model

from tensorflow.keras import optimizers

from PIL import Image
import warnings

from tensorflow.keras.datasets import mnist


def image_2_tensor(image_name, save_name=None, resize_=False,
                                    brightness=0.8, newsize=(28, 28),
format='JPEG',
                                    is_bw=False, norm=False,
plot_num=False):
    try:
        img = Image.open(image_name)
    except FileNotFoundError:
        print('Error name file')
        return

    if resize_:
        warnings.warn('The image need to resize!')
        img = img.resize(newsize)

    if is_bw == False:
        separator = 255 / brightness / 2 * 3
        for x in range(img.size[0]):
            for y in range(img.size[1]):
                r, g, b = img.getpixel((x, y))
                total = r + g + b
                if total > separator:
                    img.putpixel((x, y), (0, 0, 0))
                else:
                    img.putpixel((x, y), (250, 250, 250))

    res_tensor = np.asarray(img, dtype='uint8')

    if len(res_tensor.shape) > 2:
        height, width, _ = res_tensor.shape
        res_tensor = np.round(np.sum(res_tensor/3,
axis=2)).astype(np.uint8).reshape((height, width))

    if save_name:
        img2 = Image.fromarray(res_tensor)
        img2.save(save_name, format)

    if norm:
        res_tensor = res_tensor.astype('float32') / 255
```

```
    if plot_num:
        plt.imshow(res_tensor, cmap=plt.cm.binary)
        plt.show()
    return res_tensor


def get_test_set(file_names, is_bw_=False, plot_num=False,
resize_=True):
    test_set = []
    for i in range(len(file_names)):
        test_set.append(image_2_tensor(file_names[i], resize_=resize_,
norm=True, is_bw=is_bw_, plot_num = plot_num))

    return np.asarray(test_set)



# загрузка набора данных mnist 60000 & 10000
(train_images, train_labels), (test_images, test_labels) =
mnist.load_data()


# подготовка данных [0, 1]
train_images = train_images.reshape((60000, 28 * 28))
train_images = train_images.astype('float32') / 255

test_images = test_images.reshape((10000, 28 * 28))
test_images = test_images.astype('float32') / 255



# к категориальному вектору (5 == [0, 0, 0, 0, 0, 1, 0, ... 0])
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)


# создание модели сети
model = Sequential()
#model.add(Flatten())
model.add(Dense(256, activation='relu', input_shape=(28 * 28,)))
#model.add(Dense(256, activation='relu'))
model.add(Dense(10, activation='softmax'))

#optimizer='rmsprop'
#model.compile(optimizer= optimizers.Adam(learning_rate=0.001),
loss='categorical_crossentropy', metrics=['accuracy'])
model.compile(optimizer= optimizers.Adamax(learning_rate=0.2,
beta_1=0.9, beta_2=0.999),  loss='categorical_crossentropy',
metrics=['accuracy'])

# обучение сети
epochs_num = 5
history = model.fit(train_images, train_labels, epochs=epochs_num,
batch_size=128, validation_data=(test_images, test_labels))
```

```
# проверка на контрольном наборе
test_loss, test_acc = model.evaluate(test_images, test_labels)
print('test_acc:', test_acc)


# схема модели
plot_model(model, to_file='model.png', show_shapes=True)
print(model.summary())

# Провека модели на изображениях
images_names_1 = ['images/1/0.png',
                                  'images/1/1.png',
                                  'images/1/2.png',
                                  'images/1/3.png',
                                  'images/1/4.png',
                                  'images/1/5.png',
                                  'images/1/6.png',
                                  'images/1/7.png',
                                  'images/1/8.png',
                                  'images/1/9.png']

images_names_2 = ['images/2/0.jpg',
                                  'images/2/1.jpg',
                                  'images/2/2.jpg',
                                  'images/2/3.jpg',
                                  'images/2/4.jpg',
                                  'images/2/5.jpg',
                                  'images/2/6.jpg']


def test_on_images(model, images_names):
    test_set = get_test_set(images_names, is_bw_=False, resize_=True,
plot_num=False)
    test_set = test_set.reshape((len(images_names), 28 * 28))

    result_img = model.predict(test_set)
    print(result_img[0])
    for i in range(len(result_img)):
        print("I think number: ",
list(result_img[i]).index(max(result_img[i])))
    print(result_img)


test_on_images(model, images_names_2)
```