ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ

НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИ УНИВЕРСИТЕТ

«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук

Департамент программной инженерии

СОГЛАСОВАНО	УТВЕРЖДАЮ
Доцент департамента программной инженерии факультета компьютерных наук	Академический руководитель образовательной программы «Программная инженерия», канд. техн. наук, профессор ДПИ ФКН
_____ Р.А. Родригес Залепинос	_____ В.В. Шилов
«____» _____ 2018 г.	«____» _____ 2018 г.

Подп. и дата	
Инв. № дубл.	
Взам. инв. №	
Подп. и дата	
Инв. № подл	

Программа обработки растровых данных с помощью TensorFlow

Текст программы

ЛИСТ УТВЕРЖДЕНИЯ

RU.17701729.04.16-01 12 01-1-ЛУ

Исполнитель

Студент группы БПИ177

_____ / Д. А. Потапенков /

«____» _____ 2018 г.

Москва 2018

УТВЕРЖДЕН

RU.17701729. 04.16-01 12 01-1-ЛУ

Программа обработки растровых данных с помощью TensorFlow

Текст программы

RU.17701729.04.16-01 12 01-1

Листов 15

Подп. и дата	
Инв. № дубл.	
Взам. инв. №	
Подп. и дата	
Инв. № подл	

Москва 2018

ОГЛАВЛЕНИЕ

1. Server.py

```
import  io
from socket import *
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt
#import Code.SupFunc as sp
import SupFunc as sp
#import Code.CNN as CNN
import CNN as CNN



def GetAns(data, model):
    ''' Функция получения предсказания объекта на фотографии'''

    stream = io.BytesIO(data)
    img = Image.open(stream)
    img = np.array(img, dtype=np.float32)[:, :, 0:3]
    img = sp.Kontrast(sp.RGBtogrey(img))[:, :, 0]
    img = np.array(img, dtype=np.float32)
    # plt.imshow(img)
    # plt.show()
    ans = CNN.Predict(img, model)
    print(ans)
    return ans


def InitServer():
    '''Функция задания сервера'''
    try:
        server = socket(AF_INET, SOCK_STREAM)
        server.bind(('127.0.0.1', 10001))
        server.listen(1)
    except:
        server = None
        print("Сервер на этом порту уже запущенн")
    return server

def InitTrainedModel(path):
    '''Функция загрузки модели и проверки на что что она натренирована'''
    model = CNN.InitModel(path)
    try:
        img = Image.open("../Resources/TestImg.png")
        img = np.array(img, dtype=np.float32)[:, :, 0:3]
        img = sp.Kontrast(sp.RGBtogrey(img))[:, :, 0]
        img = np.array(img, dtype=np.float32)
        # plt.imshow(img)
        # plt.show()
        ans = CNN.Predict(img, model)
        return model
    except:
        print("Модель не обучена")
        return None
```

```
def Main():
    '''Логика работы сервера'''

    server = InitServer() #Задаем параметры сервера
    model = InitTrainedModel('../Resources/CNN_model')#Подгружаем модель
    if (server == None or model == None):
        return
    while(True):
        print("Смотрим подключения")
        try:
            # Принимаем подключение
            conn, addr = server.accept()
        except:
            print("Ошибка во время приема")
        else:
            try:
                # Получаем данные
                data = conn.recv(2048)
                print(data)
                try:
                    # Предсказываем класс и отправляем ответ
                    ans = GetAns(data, model)
                    conn.sendall(bytes([ans]))
                except:
                    print("Ошибка возникшая во вребя обработки")
                    conn.sendall(bytes([3]))
            except:
                print("Ошибка при получении данных")
            finally:
                # Закрываем подключение
                conn.shutdown(SHUT_WR)
                conn.close()

if __name__ == "__main__":
  Main()
```

2. CNN.py

```
import json
import numpy as np
import tensorflow as tf

tf.logging.set_verbosity(tf.logging.INFO)

def cnn_model_fn(features, labels, mode):
    '''Функция описывающая модель'''

    input_layer = tf.reshape(features, [-1, 20, 20, 1])
    #1.    Свёрточный слой#1: Применяет 32 фильтра 5x5, с функцией активации
ReLU
    conv1 = tf.layers.conv2d(
        inputs=input_layer,
        filters=32,
        kernel_size=[5, 5],
        padding="same",
        activation=tf.nn.relu)
    #2.    Слой подвыборки#1: Объединяет фрагменты 2x2 в одно значение
выбирая максимальный
    pool1 = tf.layers.max_pooling2d(inputs=conv1, pool_size=[2, 2],
strides=2)

    #3.    Свёрточный слой#2: Применяет 64 фильтра 5x5, с функцией активации
ReLU
    conv2 = tf.layers.conv2d(
        inputs=pool1,
        filters=64,
        kernel_size=[5, 5],
        padding="same",
        activation=tf.nn.relu)
    #4.    Слой подвыборки#2: Опять объединяет фрагменты 2x2 в одно значение
выбирая максимальный
    pool2 = tf.layers.max_pooling2d(inputs=conv2, pool_size=[2, 2],
strides=2)
    pool2_flat = tf.reshape(pool2, [-1, 5 * 5 * 64])

    # Плотный слой
    dense = tf.layers.dense(inputs=pool2_flat, units=1024,
activation=tf.nn.relu)

    dropout = tf.layers.dropout(
        inputs=dense, rate=0.4, training=mode == tf.estimator.ModeKeys.TRAIN)

    # Уровень логирования
    logits = tf.layers.dense(inputs=dropout, units=2)

    predictions = {
        "classes": tf.argmax(input=logits, axis=1),
        "probabilities": tf.nn.softmax(logits, name="softmax_tensor")
    }
```

```
    if mode == tf.estimator.ModeKeys.PREDICT:
        return tf.estimator.EstimatorSpec(mode=mode, predictions=predictions)

    #Расчет функции потерь по Softmax
    loss = tf.losses.sparse_softmax_cross_entropy(labels=labels,
logits=logits)

    if mode == tf.estimator.ModeKeys.TRAIN:
        optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.001)
        train_op = optimizer.minimize(
            loss=loss,
            global_step=tf.train.get_global_step())
        return tf.estimator.EstimatorSpec(mode=mode, loss=loss,
train_op=train_op)

    eval_metric_ops = {
        "accuracy": tf.metrics.accuracy(
            labels=labels, predictions=predictions["classes"])}
    return tf.estimator.EstimatorSpec(
        mode=mode, loss=loss, eval_metric_ops=eval_metric_ops)



def InitModel(path_to_model):
    '''Функция загрузки модели'''
    planes_classifier = tf.estimator.Estimator(
        model_fn=cnn_model_fn, model_dir=path_to_model)
    return planes_classifier



def TrainModel(TrainData, EvalData):
    # Загружаем тренировочные и проверочные данные
    train_data = np.array(TrainData["data"], dtype=np.float32)
    train_labels = np.array(TrainData["labels"], dtype=np.int32)
    eval_data = np.array(EvalData["data"], dtype=np.float32)
    eval_labels = np.array(EvalData["labels"], dtype=np.int32)

    # Создаем модель и видимые результаты
    path_to_model = '../Resources/CNN_model'

    planes_classifier = InitModel(path_to_model)
    tensors_to_log = {"probabilities": "softmax_tensor"}
    logging_hook = tf.train.LoggingTensorHook(
        tensors=tensors_to_log, every_n_iter=50)

    # Тренируе модель
    train_input_fn = tf.estimator.inputs.numpy_input_fn(
        x=train_data,
        y=train_labels,
        batch_size=100,
        num_epochs=None,
        shuffle=True)
```

```
    planes_classifier.train(
        input_fn=train_input_fn,
        steps=10000,
        hooks=[logging_hook])

    print("Training end")
    # Проверяем модель
    eval_input_fn = tf.estimator.inputs.numpy_input_fn(
        x=eval_data,
        y=eval_labels,
        num_epochs=1,
        shuffle=False)
    eval_results = planes_classifier.evaluate(input_fn=eval_input_fn)

    # Сохраняем результат проверки
    with open("../Resources/EvalResults.txt", "w", encoding="utf-8") as file:
        file.write(str(eval_results))


def Predict(predict_data, model):
    ''' Функция получегния предсказания от модели'''
    predict_input_fn = tf.estimator.inputs.numpy_input_fn(
        x=predict_data, shuffle=False)
    predictions = model.predict(input_fn=predict_input_fn)
    return list(predictions)[0]['classes']
```

3. SupFunc.py

```
import json
import numpy as np


def Kontrast(bands, lower_percent=2, higher_percent=98):
    ''' Функция контрасного фильтра'''
    out = np.zeros_like(bands)
    for i in range(3):
        a = 0 #np.min(band)
        b = 255  #np.max(band)
        c = np.percentile(bands[:,:,i], lower_percent)
        d = np.percentile(bands[:,:,i], higher_percent)
        t = a + (bands[:,:,i] - c) * (b - a) / (d - c)
        t[t<a] = a
        t[t>b] = b
        out[:,:,i] =t
    return out.astype(np.uint8)


def RGBtogrey(img):
    ''' Функция черно-белого фильтра'''
    imgn = np.zeros_like(img)
    for i in range(20):
        for j in range(20):
            a = img[i][j][0]
            b = img[i][j][1]
            c = img[i][j][2]
            S = (a + b + c) // 3
            imgn[i][j][0] = S
            imgn[i][j][1] = S
            imgn[i][j][2] = S
    return imgn


def UnisonShuffle(a, b):
    ''' Функция совместного перемешивания'''
    assert len(a) == len(b)
    p = np.random.permutation(len(a))
    return a[p], b[p]


def GetGreyData(path_to_data, file):
    ''' Функция переводящая сырые данные в удобные для работы'''
    # Загружаем данные
    planes = json.load(open(path_to_data + '/' + file))
    Data = np.array(planes['data'], dtype=np.float32)
    Labels = planes['labels']
    # Приводим к виду (n, m, 3)
    n = Data.shape[0]
    Data.shape = (n, 3, 20, 20)
```

```
    Data = np.transpose(Data, (0, 2, 3, 1))
    NewData = np.zeros((n, 20, 20, 3))
    # Делаем картинки черно-белыми и более контрастными
    for i in range(n):
        NewData[i] = Kontrast(RGBtogrey(Data[i]))
    NewData = NewData[:,:,:,0]
    # Сохраняем
    planesNew = {"data": NewData.tolist(), "labels": Labels}
    with open(path_to_data + "/planesnetgrey.json", "w", encoding="utf-8")
as file:
        json.dump(planesNew, file)


def MakeTrainEvalData(path_to_data, file):
    ''' Функция разделяющая на данные для тренировки и проверки'''
    # Загружаем данные
    planes = json.load(open(path_to_data + '/' + file))
    Data = np.array(planes['data'])
    Labels = np.array(planes['labels'])
    # Перемешиваем(внутри одного типа)
    np.random.shuffle(Data[0:8000])
    np.random.shuffle(Data[8000:32000])
    # Разделяем данные
    TrainData = np.concatenate((Data[0:7000], Data[8000:29000]), axis=0)
    TrainLabels = np.concatenate((Labels[0:7000], Labels[8000:29000]),
axis=0)
    EvalData = np.concatenate((Data[7000:8000], Data[29000:32000]),
axis=0)
    EvalLabels = np.concatenate((Labels[7000:8000], Labels[29000:32000]),
axis=0)
    # Перемешиваем
    TrainData, TrainLabels = UnisonShuffle(TrainData, TrainLabels)
    EvalData, EvalLabels = UnisonShuffle(EvalData, EvalLabels)

    # Сохраняем в json файлы
    planesTrain = {"data": TrainData.tolist(),
                   "labels": TrainLabels.tolist()}
    with open(path_to_data + "/planestrain.json", "w", encoding="utf-8")
as file:
        json.dump(planesTrain, file)

    planesEval = {"data": EvalData.tolist(),
                  "labels": EvalLabels.tolist()}
    with open(path_to_data + "/planeseval.json", "w", encoding="utf-8") as
file:
        json.dump(planesEval, file)
```

4. PredProc.py

```
import json
import Code.CNN as CNN
import Code.SupFunc as sp

''' Скрипт обрабатывающий сыпые данные и запускающий тренировку подели'''

def main():
    print("Start")
    path_to_data = '../Resources'
    try:
        sp.GetGreyData(path_to_data, 'planesnet.json')
        print("Making grey data has been ended")
    except:
        print("Ошибка при открытии или чтении 'planesnet.json'")
        return
    sp.MakeTrainEvalData(path_to_data, 'planesnetgrey.json')
    print("Making train and eval data has been ended")

    #Загрузка данных в нужном формате
    planesTrain = json.load(open(path_to_data + '/planestrain.json'))
    planesEval = json.load(open(path_to_data + '/planeseval.json'))
    print("End loading")
    print("Start traning")
    #Тренировка
    CNN.TrainModel(planesTrain, planesEval)
    print("Done")

if __name__ == "__main__":
  main()
```

5. MainForm.cs

```
using System;
using System.Drawing;
using System.Drawing.Imaging;
using System.IO;
using System.Net;
using System.Net.Sockets;
using System.Windows.Forms;

namespace MainForm
{
    public partial class MainForm : Form
    {
        OpenFileDialog openFileDialog = new OpenFileDialog()
        {
            Filter = "Images|*.png;*.jpg"
        };
        /// <summary>
        /// Картинка в байтах
        /// </summary>
        byte[] Bitmapbytes;
        /// <summary>
        /// Количество байт в Bitmapbytes
        /// </summary>
        int BitmapbytesLen;
        /// <summary>
        /// Указание куда подключатся
        /// </summary>
        IPAddress ipAddress;
        IPEndPoint remoteEP;
        /// <summary>
        /// Сокет
        /// </summary>
        Socket client;

        /// <summary>
        /// Конструктор
        /// </summary>
        public MainForm()
        {
            InitializeComponent();

            ipAddress = IPAddress.Parse("127.0.0.1"); ;
            remoteEP = new IPEndPoint(ipAddress, 10001);
        }
        /// <summary>
        /// Метод запускабющийся при нажатии на кнопку "Загрузить"
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void buttonLoadImg_Click(object sender, EventArgs e)
        {
            if (openFileDialog.ShowDialog() == DialogResult.OK)
            {
                Bitmap img = LoadBitmap(openFileDialog.FileName);
                if (img.Height != 20 && img.Width != 20)
```

```
                {
                    MessageBox.Show("Вы выбрали картинку с разрешением не равным 20x20.
Результаты могут быть не точными.");
                }

                pictureBox1.Image = img;
                this.buttonSend.Enabled = true;
            }
            else return;
        }

        /// <summary>
        /// Метод вызывающийся при нажатии на кнопку "Узнать"
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void buttonSend_Click(object sender, EventArgs e)
        {
            if (BitmapbytesLen > 2048)
            {
                MessageBox.Show("Пакет больше принимаемого", "Warning");
            }
            int ansver = GetInfoAboutImg();
        }

        /// <summary>
        /// Метод загрузки выбранного изображения и перевода его в байтовый формат
        /// </summary>
        /// <param name="fileName"></param>
        /// <returns></returns>
        private Bitmap LoadBitmap(string fileName)
        {
            Bitmap Img;
            using (FileStream fs = new FileStream(fileName, FileMode.Open,
FileAccess.Read, FileShare.Read))
            {
                Img = new Bitmap(fs);
            }
            // Изменение размера нужно для того чтобы передавать меньше, так как сервер
обрабатывает изображения 20на20
            Img = resizeImage(Img, new Size(20, 20));
            // Этот немного костыльный код написанн из-за того что я не знаю как сделать
иначе
            using (FileStream saveStream = new FileStream("SendingImage.png",
FileMode.Create, FileAccess.Write))
            {
                Img.Save(saveStream, ImageFormat.Png);
            }
            using (FileStream loadStream = new FileStream("SendingImage.png",
FileMode.Open, FileAccess.Read))
            {
                Bitmapbytes = new byte[loadStream.Length];
                BitmapbytesLen = (int)loadStream.Length;
                int numBytesRead = 0;
                while (BitmapbytesLen > 0)
                {
                    // Read может вернуть любое значение между 0 и BitmapbytesLen
```

```
                int n = loadStream.Read(Bitmapbytes, numBytesRead, BitmapbytesLen);
                if (n == 0)
                    break;

                numBytesRead += n;
                BitmapbytesLen -= n;
            }
        }
        return Img;
    }

    /// <summary>
    /// Метод изменяющий размер изображения
    /// </summary>
    /// <param name="imgToResize"></param>
    /// <param name="size"></param>
    /// <returns></returns>
    public static Bitmap resizeImage(Bitmap imgToResize, Size size)
    {
        return new Bitmap(imgToResize, size);
    }

    /// <summary>
    /// Метод для отправки изображения и получения ответа от сервера
    /// </summary>
    /// <returns></returns>
    private int GetInfoAboutImg()
    {
        byte[] bytes = new byte[4];

        client = new Socket(ipAddress.AddressFamily,
                SocketType.Stream, ProtocolType.Tcp);
        try
        {
            client.Connect(remoteEP);

            int BytesToSend = Bitmapbytes.Length;

            int bytesSent = client.Send(Bitmapbytes);
            // MessageBox.Show(string.Format("Отправлено {0}", bytesSent));

            int bytesRec = client.Receive(bytes);
            int ans = BitConverter.ToInt32(bytes, 0);
            this.Activate();
            string text = "";
            switch (ans)
            {
                case 0:
                    text = "Не самолет";
                    break;
                case 1:
                    text = "Самолет";
                    break;
                default:
                    text = "Ошибка";
                    break;
            }
```

```
                textBox1.Text = text;
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
            finally
            {
                if (client.Connected)
                {
                    client.Shutdown(SocketShutdown.Both);
                    client.Close();

                }
            }
            return 0;
        }
    }
}
```

ЛИСТ РЕГИСТРАЦИЙ ИЗМЕНЕНИЙ

Изм.	Номера листов (страниц)				Всего листов (страниц) в документе	№ документа	Входящий № сопроводительного документа и дата	Подпись	Дата
	измененных	замененных	новых	аннулированных					

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729. 04.16-01 12 01-1				
Инв. № подл.	Подп. и дата	Взам. Инв. №	Инв. № дубл.	Подп. и дата