ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ

НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук

Департамент программной инженерии

СОГЛАСОВАНО Старший преподаватель департамента программной инженерии факультета компьютерных наук _____ Д.В. Пантюхин «___» _____ 2020 г.	УТВЕРЖДАЮ Академический руководитель образовательной программы «Программная инженерия» _____ В.В. Шилов «___» _____2020 г.

Подп. и дата	
Инв. № дубл.	
Взам. Инв. №	
Подп. и дата	
Инв. № подл.	RU.17701729.04.09-01 12 01-1-ЛУ

Программа определения лесных пожаров по спутниковым фотографиям с помощью нейронных сетей

Клиент

Текст программы

ЛИСТ УТВЕРЖДЕНИЯ

RU.17701729.04.09-01 12 01-1-ЛУ

Исполнитель

Студент группы БПИ 199

_____/Мостачев А.О./

«___»_____2020 г.

2020

Утверждено

RU.17701729.04.09-01 12 01-1

Программа определения лесных пожаров по спутниковым фотографиям с помощью нейронных сетей

Клиент

Текст программы

RU.17701729.04.09-01 12 01-1

Листов 58

2020

Содержание

1. Текст программы

1.1.Firewatch.cs

```
using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Net;

using System.Text;

using System.Threading;

using System.Windows.Forms;

using System.IO;

using System.Runtime.CompilerServices;


namespace Kursach

{

    public partial class Firewatch : Form

    {

        // Путь к настройкам.

        const string settingspath = "settings.ini";

        // Путь к файлу скрипта сервера.

        const string pathtoserver = "../../Server/server.py";

        // Количество попыток подключения.

        const int maxattempt = 50;
```

```
        // Счетчик текущего изображения.
        static int currentimage = 0;
        // Пути к принятым на обработку файлам.
        static List<string> filenames = new List<string>();
        // Процесс, отвечающий за сервер.
        static System.Diagnostics.Process server;
        // Список предсказаний нейросети.
        static List<string> predictions = new List<string>();
        // Словарь для хранения путей к изображениям с соответствующими
предсказаниями нейросети.
        static Dictionary<string, string> imagepredictions = new
Dictionary<string, string>();
        // Попытка подключения к серверу.
        static int attempt = 0;

        public Firewatch()
        {
            try
            {
                Tools.StartServer(out server, pathtoserver);

                Icon = new Icon("../../Pics/Logo.ico");
            }
            catch (Win32Exception)
            {
                // Обработка ошибок при запуске сервера.
```

```
                MessageBox.Show("Unable to start the server!", "Error",
MessageBoxButtons.OK);
                Close();
            }
            catch (IOException)
            {
                // Обработка ошибки при загрузке иконки.
                Icon = default;
            }

            InitializeComponent();

            // Запуск таймера для подключения к серверу.
            Thread.Sleep(50);
            ConnectTimer.Start();

            // Просто текст сообщения.
            PleaseWait.Text = ".." + Environment.NewLine +
PleaseWait.Text + Environment.NewLine + "..";
        }

        /// <summary>
        /// Выбор изображений для обработки.
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
```

```
        private void SelectImages_Click(object sender, EventArgs e)
        {
            // Проверка на режим загрузки: индивидуальные файлы или
папки.
            if (Settings.IfSingle)
            {
                MessageBox.Show("Select an image for analysis from a
dialog box.",
                    "File Selection", MessageBoxButtons.OK);

                // Вызов метода для выбора файлов.
                if (ChooseFile())
                {
                    // Форматирование файлов для отправки.
                    filenames = Sender.Replace(filenames);

                    // Удаление неподдерживаемых файлов.
                    if(!Tools.RemoveUnsupported(ref filenames))
                    {
                        MessageBox.Show("Some of the selected files
cannot be processed. They will be ignored.",
                            "Format Error",
                            MessageBoxButtons.OK);
                    }

                    // Запрос подтверждения обработки.
```

```
                if (MessageBox.Show(
                    $"Would you like to proceed to classifying?
{filenames.Count} images selected.",
                    "",
                    MessageBoxButtons.YesNo) == DialogResult.Yes)
                {
                    // Запрос к серверу на обработку изображений.
                    SendPred();
                }
            }
        }
        else
        {
            MessageBox.Show("Select a folder for analysis from a
dialog box.",
                "Folder Selection", MessageBoxButtons.OK);

            string directory;

            // Вызов метода для выбора папки.
            if (ChooseFolder(out directory))
            {
                // Получение файлов, удаление неподдерживаемых.
                string result = Tools.GetAndRemove(out filenames,
directory);
```

```
                    if(result == "unreachable")
                    {
                        // Показ сообщения об ошибке.
                        MessageBox.Show("Unable to reach to
directory!", "Error", MessageBoxButtons.OK);
                    }
                    else
                    {
                        if(result == "incorrect")
                        {
                            MessageBox.Show("Some of the selected files
cannot be processed. They will be ignored.",
                                "Format Error",
                                MessageBoxButtons.OK);
                        }

                        // Запрос подтверждения на обработку
изображений.
                        if (MessageBox.Show(
                            $"Would you like to proceed to classifying?
{filenames.Count} images selected.",
                            "", MessageBoxButtons.YesNo) ==
DialogResult.Yes)
                        {
                            // Запрос к серверу на ообработку
изображений.
                            SendPred();
```

```
                }
            }
        }
    }
}

/// <summary>
/// Выбор папки для анализа из диалогового окна.
/// </summary>
/// <returns>Выбрана ли какая-либо папка</returns>
bool ChooseFolder(out string directory)
{
    directory = string.Empty;

    // Открытие диалогового окна.
    folderBrowserDialog1.Description = default;
    if (folderBrowserDialog1.ShowDialog() == DialogResult.OK)
    {
        // Присвоение пути к папке соответствующей переменной.
        directory = folderBrowserDialog1.SelectedPath;
        return true;
    }
    else
    {
        return false;
```

```
            }
        }

        /// <summary>
        /// Выбор файлов для обработки.
        /// </summary>
        /// <returns></returns>
        bool ChooseFile()
        {
            // Разрешение выбора нескольких файлов, так как этот диалог используется еще в одном месте.
            openFileDialog1.Multiselect = true;

            // Открытие диалогового окна.
            if (openFileDialog1.ShowDialog() == DialogResult.OK)
            {
                // Очистка переменной путей к файлам.
                filenames = new List<string>();

                // Заполнение путей.
                foreach (var file in openFileDialog1.FileNames)
                {
                    filenames.Add(file);
                }
```

```
                return true;
            }
            else
            {
                return false;
            }
        }

        /// <summary>
        /// Обработка закрытия приложения.
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void Form1_FormClosing(object sender,
FormClosingEventArgs e)
        {
            try
            {
                // Закрытие сервера.
                server.Kill();
            }
            catch (Exception ex) when (ex is Win32Exception || ex is
InvalidOperationException)
            {
                // Вывод сообщения об ошибке.
```

```
                MessageBox.Show("Unable to shut down the server!",
"Error", MessageBoxButtons.OK);
            }
        }

        /// <summary>
        /// Скрытие и показ элементов в состоянии, когда приложение
подключено к серверу.
        /// </summary>
        void Connected()
        {
            PleaseWait.Hide();
            SelectImages.Show();
            IfMulticlass.Show();
            IfSingle.Show();
            Frame1.Show();
            UploadWeights.Show();
            Frame2.Show();
            SettingsLabel.Show();
            SaveLogs.Show();
            SaveSettings.Show();

            // Попытка загрузки настроек.
            if (!Settings.LoadSettings(settingspath))
            {
```

```
                MessageBox.Show("Error while loading settings",
"Error", MessageBoxButtons.OK);
            }

            // Обновление чекбоксов.
            IfMulticlass.Checked = Settings.IfMulticlass;
            IfSingle.Checked = Settings.IfSingle;
            SaveLogs.Checked = Settings.SaveLogs;

            // Загрузка изображений к кнопкам.
            try
            {
                Next.BackgroundImage = new
Bitmap("../../Pics/ArrowRight.png");
                Prev.BackgroundImage = new
Bitmap("../../Pics/ArrowLeft.png");
            }
            catch
            {
                MessageBox.Show("Some files have not been loaded
properly.", "Error", MessageBoxButtons.OK);
            }
        }

        /// <summary>
```

```
        /// Тик таймера подключения к серверу. Новая попытка или
программа сдается и умирает (но не совсем)
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void timer1_Tick(object sender, EventArgs e)
        {
            // Проверка на превышение планки максимальной попытки
подключения.
            if (attempt >= maxattempt)
            {
                // Остановка таймера.
                ConnectTimer.Stop();

                // Вывод сообщения об ошибке.
                MessageBox.Show("Unable to connect to server! The app
will be closed.");

                // Закрытие приложения.
                Close();
            }
            else
            {
                // Повторная попытка подключения.
                attempt++;
                string response = Sender.TryToConnect();
```

```
                // При успехе остановка таймера, вызов соответствующего
делегата.
                if (response == "connected")
                {
                    ConnectTimer.Stop();
                    Connected();
                }
                else
                {
                    if(response == "error")
                    {
                        ConnectTimer.Stop();
                        MessageBox.Show("Error while loading a neural
network model! The app will be closed.", "Error",
MessageBoxButtons.OK);
                        Close();
                    }
                }
            }

            // Изменение строки сообщения, потому что мне так хочется.
            if
(PleaseWait.Text.Contains("....................................."))
            {
```

```
                PleaseWait.Text = PleaseWait.Text.Substring(36,
PleaseWait.Text.Length - 72);
            }
            else
            {
                PleaseWait.Text = "...." + PleaseWait.Text + "....";
            }
        }



        /// <summary>
        /// Метод для отправки запроса на загрузку весов к серверу.
        /// </summary>
        void SendPred()
        {
            // Обновление интерфейса.
            HideAll();
            WaitForPred.Text = "Please wait: the images are being
processed";
            WaitForPred.Show();
            Refresh();

            // Флаг получения ответа.
            bool gotit = true;
```

```
            // Попытка отправки запроса и получения ответа.
            try
            {
                predictions = Sender.PredictRequest(filenames,
Settings.IfMulticlass);
            }
            catch (Exception)
            {
                // Вывод сообщения об ошибке.
                MessageBox.Show("Error while classifying images",
"Error", MessageBoxButtons.OK);

                // Возвращение на главный экран.
                WaitForPred.Hide();
                Connected();
                gotit = false;
            }

            if (gotit)
            {
                // Активация делегата полученных предсказаний.
                AnalyzePredictions();
            }
        }
```

```
        /// <summary>
        /// Обработка полученных предсказаний.
        /// </summary>
        void AnalyzePredictions()
        {
            // Очищение словаря путей с предсказаниями.
            imagepredictions = new Dictionary<string, string>();

            // Заполнение словаря путями и предсказаниями.
            for (int i = 0; i < Math.Min(filenames.Count,
predictions.Count); i++)
            {
                imagepredictions.Add(filenames[i], predictions[i]);
            }

            // Проверка на необходимость сохранять журналы
классификации.
            if (Settings.SaveLogs)
            {
                if (!Tools.SaveLogs(imagepredictions))
                {
                    MessageBox.Show("Error while saving logs!",
"Error", MessageBoxButtons.OK);
                }
            }
```

```
            // Количество файлов, классифицированных, как дым.
            int wildfires = 0;

            // Подсчет количества изображений, классифицированных, как
дым.
            foreach (var pred in predictions)
            {
                if (pred == "Wildfire")
                {
                    wildfires++;
                }
            }

            // Вывод сообщения о количестве изображений,
классифицированных, как дым. Отныне сокращаю ИККД.
            WaitForPred.Text = $"{wildfires} images have been
classified as wildfires.";

            // Показ сообщения.
            WaitForPred.Show();

            // Показ кнопки просмотра изображений.
            ViewImages.Show();

            // Показ кнопки "Назад"
            BackBut.Show();
```

```
            if (wildfires > 0)
            {
                // Проверка необходимости сохранения ИККД в отдельную
папку.
                if (MessageBox.Show("Would you like to save images
classified as wildfires to separate folder?",
                    "Choose", MessageBoxButtons.YesNo) ==
DialogResult.Yes)
                {
                    string savedir;
                    // Выбор папки для сохранения.
                    if (ChooseSaveFolder(out savedir))
                    {
                        if (!Tools.SaveImages(savedir,
imagepredictions))
                        {
                            // Вывод сообщения об ошибке сохранения.
                            MessageBox.Show("Some images could not be
saved!", "Error", MessageBoxButtons.OK);
                        }
                        else
                        {
                            // Вывод сообщения об успешности
сохранения.
                            MessageBox.Show("Images have been saved to
selected folder.", "Success", MessageBoxButtons.OK);
```

```
                }
            }
        }
    }
}

/// <summary>
/// Нажатие кнопки"Назад".
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void BackBut_Click(object sender, EventArgs e)
{
    // Скрытие и показ соответствующих элементов.
    WaitForPred.Hide();
    BackBut.Hide();
    ViewImages.Hide();
    LoadImage.Hide();
    Pics.Hide();
    PredClass.Hide();
    Next.Hide();
    Prev.Hide();
    currentimage = 0;
    // BackColor = Color.FromArgb(33, 34, 38);
    Connected();
```

```
        }

        /// <summary>
        /// Загрузка весов для нейросети.
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void UploadWeights_Click(object sender, EventArgs e)
        {
            // Показ сообщения о выборе весов.
            MessageBox.Show(
                "Select weights for the neural network to use from the dialog. Weights will be automatically assigned to the suitable network.",
                "Selection",
                MessageBoxButtons.OK);

            string path;

            // Выбор весов.
            if (ChooseWeights(out path))
            {
                // Проверка на соответствие типу файла.
                if (path.EndsWith(".index"))
                {
                    // Запрос одобрения на загрузку весов.
```

```
                    if (MessageBox.Show("Proceed with uploading
weights?", "Choose", MessageBoxButtons.YesNo) == DialogResult.Yes)
                    {
                        // Отправка запроса на загрузку весов к
серверу.
                        SendWeights(path);

                        // Возвращение на главный экран.
                        Connected();
                        WaitForPred.Hide();
                    }
                }
                else
                {
                    // Вывод сообщния о несоответствии типа файла.
                    MessageBox.Show("Please choose a file containing
weights.", "Error", MessageBoxButtons.OK);
                }
            }
        }

        /// <summary>
        /// Метод для отправки запроса на обработку изображений к
серверу.
        /// </summary>
        void SendWeights(string path)
```

```
        {
            // Обновление интерфейса.
            HideAll();
            WaitForPred.Text = "Please wait: the weights are being
uploaded...";
            WaitForPred.Show();
            Refresh();

            // Вывод сообщения об успехе или провале загрузки весов.
            if (Sender.WeightsRequest(path))
            {
                MessageBox.Show("Weights have been successfully
uploaded", "Success", MessageBoxButtons.OK);
            }
            else
            {
                MessageBox.Show("Error while uploading weights!",
"Error", MessageBoxButtons.OK);
            }
        }

        /// <summary>
        /// Выбор весов.
        /// </summary>
        /// <returns>Выбран ли файл.</returns>
        bool ChooseWeights(out string weightspath)
```

```
        {
            weightspath = string.Empty;

            // Показ соответствующего окна, выбор файла.
            if (weightUpload.ShowDialog() == DialogResult.OK)
            {
                weightspath = weightUpload.FileName;

                return true;
            }
            else
            {
                return false;
            }
        }


        /// <summary>
        /// Скрытие всех элементов управления (ну, всех, которые нужно скрывать).
        /// </summary>
        void HideAll()
        {
            SelectImages.Hide();
            IfMulticlass.Hide();
```

```
            IfSingle.Hide();
            Frame1.Hide();
            Frame2.Hide();
            SettingsLabel.Hide();
            UploadWeights.Hide();
            SaveLogs.Hide();
            SaveSettings.Hide();
        }

        /// <summary>
        /// Сохранение настроек.
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void SaveSettings_Click(object sender, EventArgs e)
        {
            if (!Settings.SaveSettings(settingspath))
            {
                // Вывод сообщения об ошибке.
                MessageBox.Show("Error while saving settings", "Error",
MessageBoxButtons.OK);
            }
        }

        /// <summary>
```

```
        /// Выбор папки для сохранения изображений.
        /// </summary>
        /// <returns>Выбрана ли папка.</returns>
        bool ChooseSaveFolder(out string savedir)
        {
            savedir = string.Empty;

            // Ну тут все очевидно, ну правда.
            folderBrowserDialog1.Description = "Select a folder to save
images.";
            if (folderBrowserDialog1.ShowDialog() == DialogResult.OK)
            {
                savedir = folderBrowserDialog1.SelectedPath;
                return true;
            }
            else
            {
                return false;
            }
        }

        /// <summary>
        /// Просмотр изображений.
        /// </summary>
        /// <param name="sender"></param>
```

```
/// <param name="e"></param>
private void ViewImages_Click(object sender, EventArgs e)
{
    // Скрытие и показ соответствующих элементов интерфейса.
    ViewImages.Hide();
    WaitForPred.Hide();

    Pics.Show();
    PredClass.Show();
    Prev.Show();
    Next.Show();
    LoadImage.Show();

    // Показ начального изображения.
    if (imagepredictions.Count > 0)
    {
        ShowImage(currentimage);
    }
    else
    {
        // Обработка пустого словаря предсказаний.
        PredClass.Text = "Error";
        Pics.Image = Pics.ErrorImage;
    }
}
```

```
/// <summary>
/// Следующее изображение.
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void Next_Click(object sender, EventArgs e)
{
    // Проверка на наличие следующего.
    if (currentimage + 1 < imagepredictions.Keys.Count)
    {
        currentimage++;

        ShowImage(currentimage);
    }
}

/// <summary>
/// Предыдущее изображение.
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void Prev_Click(object sender, EventArgs e)
{
    // Проверка на выход за пределы словаря.
```

```
            if (currentimage - 1 < imagepredictions.Keys.Count &&
currentimage - 1 >= 0)
            {
                currentimage--;

                ShowImage(currentimage);
            }
        }

        /// <summary>
        /// Выбор изображения для просмотра.
        /// </summary>
        /// <returns></returns>
        bool ChooseImage(out string loadedimage)
        {
            loadedimage = string.Empty;

            // Вот и понадобилось изменение мультиселекта, потому что
тут он запрещен.
            openFileDialog1.Multiselect = false;

            // Выбор файла в диалоговом окне.
            if (openFileDialog1.ShowDialog() == DialogResult.OK)
            {
                // Проверка на соответствие разрешенным форматам.
```

```
                if
(Tools.AllowedFormats.Contains(openFileDialog1.FileName.Split('.').Last
()))
                {
                    loadedimage = openFileDialog1.FileName;
                    return true;
                }
                else
                {
                    return false;
                }
            }
            else
            {
                return false;
            }
        }

        /// <summary>
        /// Загрузка изображения для просмотра.
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void LoadImage_Click(object sender, EventArgs e)
        {
            // Просто сообщение пользователю.
```

```
            MessageBox.Show("Choose an image to view", "",
MessageBoxButtons.OK);


            string loadedimage;


            // Выбор файла.
            if (ChooseImage(out loadedimage))
            {
                // Попытка загрузки из файла.
                try
                {
                    Pics.Image = Image.FromFile(loadedimage);


                    PredClass.Text = Tools.GetClass(loadedimage,
imagepredictions);
                }
                catch (Exception ex) when (ex is IOException || ex is
System.Security.SecurityException || ex is UnauthorizedAccessException)
                {
                    Pics.Image = Pics.ErrorImage;
                        PredClass.Text = "Error";
                }
            }
        }
```

```
        // Обновление класса настроек при изменении статуса
соответствующих чекбоксов.
        private void IfSingle_CheckedChanged(object sender, EventArgs
e)
        {
            Settings.IfSingle = IfSingle.Checked;
        }
        private void IfMulticlass_CheckedChanged(object sender,
EventArgs e)
        {
            Settings.IfMulticlass = IfMulticlass.Checked;
        }
        private void SaveLogs_CheckedChanged(object sender, EventArgs
e)
        {
            Settings.SaveLogs = SaveLogs.Checked;
        }


        void ShowImage(int index)
        {
            // Попытка чтения из файла.
            try
            {
                Pics.Image =
Image.FromFile(imagepredictions.Keys.ToList()[index]);
                PredClass.Text =
imagepredictions[imagepredictions.Keys.ToList()[index]];
```

```
            }
            catch (Exception e) when (e is IOException || e is
System.Security.SecurityException || e is UnauthorizedAccessException)
            {
                Pics.Image = Pics.ErrorImage;
                PredClass.Text = "Error";
            }
        }
    }
}
```

1.2.Sender.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Net;
using System.Threading;
using System.IO;

namespace Kursach
{
    public static class Sender
    {
        // Сервер, на котором запускается обработка нейросетью.
        const string uri = "http://127.0.0.1:5000/";
```

```
/// <summary>
/// Попытка подключения к серверу.
/// </summary>
/// <returns>Строка состояния подключения</returns>
public static string TryToConnect()
{
    // Создание GET-запроса по адресу сервера.
    WebRequest request = WebRequest.Create($"{uri}index");
    request.Method = "GET";

    // Попытка получить ответ от сервера.
    // Возврат соответствующего результата.
    try
    {
        WebResponse response = request.GetResponse();

        Stream dataStream;
        string responsefromserver;

        using (dataStream = response.GetResponseStream())
        {
            StreamReader reader = new StreamReader(dataStream);
            responsefromserver = reader.ReadToEnd();
        }
```

```
            if(responsefromserver == "000")
            {
                return "connected";
            }
            if(responsefromserver == "200")
            {
                return "error";
            }
            return "unable";
        }
        catch (Exception)
        {
            return "unable";
        }
    }

    /// <summary>
    /// Отправка запроса к серверу на обработку изображений.
    /// </summary>
    public static List<string> PredictRequest(List<string> paths,
bool ifmulticlass)
    {
        // Создание строки из путей к файлам.
```

```
            /*
             * Тут стоит пояснить, что можно было использовать какой-
нибудь широкодоступный формат, типа JSON,
             * но используемые запросы содержат только информацию
одного типа, которую легко запаковать.
             * Тем более, подлый C# все равно все кодирует в один
большой массив байт,
             * с раскодированием которого на сервере мне возиться не
хочется. So here:
             * строка из имен файлов.
             *
             * В качестве разделителя взят символ *, так как его не
может быть в именах файлов.
             */

            string datastr = string.Join("*", paths);

            // Кодирование строки в массив байтов.
            byte[] data = Encoding.UTF8.GetBytes(datastr);

            // Создание запроса.
            WebRequest request;

            // Определение модели нейросети, к которой подается запрос.
            if (ifmulticlass)
            {
```

```
        request = WebRequest.Create(uri + "predictmanymulti");
    }
    else
    {
        request = WebRequest.Create(uri + "predictmanybinary");
    }

    request.Method = "POST";
    request.ContentLength = data.Length;
    request.ContentType = "application/json";
    Stream dataStream = request.GetRequestStream();
    dataStream.Write(data, 0, data.Length);
    dataStream.Close();

    WebResponse response = request.GetResponse();

    List<string> predictions = new List<string>();

    // Расшифровка ответа.
    using (dataStream = response.GetResponseStream())
    {
        StreamReader reader = new StreamReader(dataStream);
        string responseFromServer = reader.ReadToEnd();
        predictions = responseFromServer.Split('*').ToList();
    }
```

```
            return predictions;
        }

        /// <summary>
        ///
        /// </summary>
        /// <param name="paths"></param>
        /// <returns></returns>
        public static List<string> Replace(List<string> paths)
        {
            // Форматирование строк путей к файлам для избежания
проблем с кодировкой.
            for (int i = 0; i < paths.Count; i++)
            {
                paths[i] = paths[i].Replace('\\', '/');
            }

            return paths;
        }

        /// <summary>
        /// Запрос на загрузку весов.
        /// </summary>
        public static bool WeightsRequest(string weightspath)
```

```
        {
            // Создание массива байтов для отправки.
            byte[] data = Encoding.UTF8.GetBytes(weightspath);

            // Создание POST-запроса
            WebRequest request;

            // Путь к запросу. Сначала для бинарной модели.
            request = WebRequest.Create(uri + "uploadweightsbinary");

            // Флаг успешности запроса.
            bool gotresponse = true;

            // Попытка загрузки весов.

            /*
             * Сначала происходит попытка загрузить веса в бинарную
модель.
             * Если по какой-то причине сервер выдал ошибку
(несоответствие размеров тензоров, отсутствие необходимых файлов),
             * запрос посылается для многоклассовой модели.
             * Если опять что-то не получается, то не судьба - больше
не пытаемся.
             */

            try
```

```
            {
                request.Method = "POST";
                request.ContentLength = data.Length;
                request.ContentType = "application/json";
                Stream dataStream = request.GetRequestStream();
                dataStream.Write(data, 0, data.Length);
                dataStream.Close();

                WebResponse response = request.GetResponse();
                gotresponse = true;
            }
            catch
            {
                // Изменение флага успешности.
                gotresponse = false;
            }

            // Попытка загрузить веса для другой модели, если первая
попытка безуспешна.
            if (!gotresponse)
            {
                // Все то же самое, но на другую страницу.
                request = WebRequest.Create(uri +
"uploadweightsmulti");

                try
```

```
                {
                    request.Method = "POST";
                    request.ContentLength = data.Length;
                    request.ContentType = "application/json";
                    request.ContentType = "application/json";
                    Stream dataStream = request.GetRequestStream();
                    dataStream.Write(data, 0, data.Length);
                    dataStream.Close();

                    WebResponse response = request.GetResponse();
                    gotresponse = true;
                }
                catch
                {
                    gotresponse = false;
                }
            }

            return gotresponse;
        }
    }
}
```

1.3.Settings.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
```

```
using System.Text;
using System.Threading.Tasks;
using System.IO;

namespace Kursach
{
    public static class Settings
    {
        static Settings()
        {
            IfMulticlass = false;
            IfSingle = false;
            SaveLogs = false;
        }

        public static bool IfSingle { get; set; }

        public static bool SaveLogs { get; set; }

        public static bool IfMulticlass { get; set; }

        /// <summary>
        /// Загрузка настроек.
        /// </summary>
        /// <param name="path">Путь к файлу настроек</param>
```

```
        /// <returns>Успешность загрузки</returns>
        public static bool LoadSettings(string path)
        {
            // Массив строк настроек.
            string[] settings = new string[0];

            // Попытка чтения настроек из файла.
            try
            {
                settings = File.ReadAllLines(path);
            }
            catch (Exception e) when (e is IOException || e is
System.Security.SecurityException || e is UnauthorizedAccessException)
            {
                return false;
            }

            // Флаг успеха обработки.
            bool parsesuccess = true;

            // Парс файла в словарь.
            Dictionary<string, string> settingsdict = new
Dictionary<string, string>();

            for (int i = 0; i < settings.Length; i++)
            {
```

```
                try
                {
                    settingsdict.Add(settings[i].Split('=')[0],
settings[i].Split('=')[1]);
                }
                catch(ArgumentOutOfRangeException)
                {
                    parsesuccess = false;
                }

            }

            // Возвращение информации о безуспещшной загрузке.
            if (!parsesuccess)
            {
                return false;
            }
            else
            {
                try
                {
                    // Интерпретация прочтенных настроек.
                    if (settingsdict["IfMulticlass"] == "True")
                    {
                        IfMulticlass = true;
```

```
            }

            if (settingsdict["IfSingle"] == "True")
            {
                IfSingle = true;
            }

            if (settingsdict["SaveLogs"] == "True")
            {
                SaveLogs = true;
            }

            return true;
        }
        catch (KeyNotFoundException)
        {
            return false;
        }
    }
}

/// <summary>
/// Сохранение настроек.
/// </summary>
/// <param name="path">Путь к файлу настроеке</param>
```

```
        /// <returns>Успешность сохранения</returns>
        public static bool SaveSettings(string path)
        {
            // Создание и заполнение массива строк настроек.
            string[] settings = new string[3];

            settings[0] = "IfMulticlass=" + IfMulticlass;
            settings[1] = "IfSingle=" + IfSingle;
            settings[2] = "SaveLogs=" + SaveLogs;

            // Попытка записи настроек в файл.
            try
            {
                File.WriteAllLines(path, settings);
                return true;
            }
            catch (Exception e) when (e is IOException || e is
System.Security.SecurityException || e is UnauthorizedAccessException)
            {
                return false;
            }
        }
    }
}
```

1.4.Tools.cs

```
using System;
```

```
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.IO;

namespace Kursach
{
    public static class Tools
    {

        // Разрешенные форматы изображений.
        public static string[] AllowedFormats { get
            {
                return new string[] { "png", "jpg", "jpeg", "PNG",
"JPG", "JPEG" };
            }
        }

        /// <summary>
        /// Удаление неподдерживаемых файлов.
        /// </summary>
        /// <param name="filenames">Список путей к файлам</param>
        /// <returns>Все ли файлы подлежат обработке</returns>
        public static bool RemoveUnsupported(ref List<string>
filenames)
```

```
{
    // Список файлов на удаление.
    List<string> toremove = new List<string>();

    // Флаг возможности обработки всех файлов.
    bool isallowed = true;

    // Определение возможности обработки каждого файла.
    foreach (var filename in filenames)
    {
        // Флаг отдельного файла.
        bool oneallowed = false;

        // Проверка соответствия файла поддерживаемым форматам.
        foreach (var format in AllowedFormats)
        {
            if (filename.EndsWith(format))
            {
                oneallowed = true;
                break;
            }
        }

        // Если файл не поддерживается, отрежение этого в
переменной для всех файлов.
```

```
            if (!oneallowed)
            {
                isallowed = false;
                // Добавление файла в список на удаление.
                toremove.Add(filename);
            }
        }

        // Удаление неподдерживаемых файлов.
        if (!isallowed)
        {
            // Удаление неподдерживаемых файлов.
            foreach (var filename in toremove)
            {
                filenames.Remove(filename);
            }
        }

        return isallowed;
    }

    /// <summary>
    /// Сохранение журнала классификации.
    /// </summary>
```

```
        /// <param name="imagepredictions">Словарь путей к файлам с
соответствующими предсказаниями</param>
        /// <returns>Успешно ли сохранение</returns>
        public static bool SaveLogs(Dictionary<string, string>
imagepredictions)
        {

            // Создание массива строк для записи в журнал.
            string[] linestowrite = new string[imagepredictions.Count];
            int counter = 0;

            // Заполнение массива-журнала.
            foreach (var key in imagepredictions.Keys)
            {
                linestowrite[counter] = key + "\t:\t" +
imagepredictions[key];
                counter++;
            }

            // Создание имени журнала.
            string path = DateTime.Now.Year + "-" + DateTime.Now.Month
+ "-" +
                DateTime.Now.Day + "--" + DateTime.Now.Hour + "-" +
                DateTime.Now.Minute + "-" + DateTime.Now.Second +
".txt";
```

```
            // Попытка записи журнала в файл.
            try
            {
                File.WriteAllLines(path, linestowrite);
                return true;
            }
            catch (Exception e) when (e is IOException || e is
System.Security.SecurityException || e is UnauthorizedAccessException)
            {
                return false;
            }
        }

        /// <summary>
        /// Сохранение изображений, классифицированных, как дым (ИККД)
        /// </summary>
        /// <param name="savedir">Директория сохранения</param>
        /// <param name="imagepredictions">Словарь путей к файлам с
соответствующими предсказаниями</param>
        /// <returns>Успешность сохранения</returns>
        public static bool SaveImages(string savedir,
Dictionary<string, string> imagepredictions)
        {

            // Флаг успешности сохранения.
            bool savessuccessful = true;
```

```
            // Сохранение ИККД.
            foreach (var key in imagepredictions.Keys)
            {
                if (imagepredictions[key] == "Wildfire")
                {
                    try
                    {
                        // Сохранение названия файла.
                        string filename = key.Split('/').Last();

                        // Копирование файла в выбранную папку.
                        File.Copy(key, savedir + "/" + filename);
                    }
                    catch (Exception e) when (e is IOException || e is
System.Security.SecurityException || e is UnauthorizedAccessException)
                    {
                        savessuccessful = false;
                    }
                }
            }

            // Возвращение успешности сохранения.
            if (!savessuccessful)
            {
```

```
                return false;
            }
            else
            {
                return true;
            }
        }

        /// <summary>
        /// Получить класс изображения по пути к нему.
        /// </summary>
        /// <param name="loadedimage">Путь к данному
изображению</param>
        /// <param name="imagepredictions">Словарь путей с
предсказаниями</param>
        /// <returns>Класс изображения</returns>
        public static string GetClass(string loadedimage,
Dictionary<string, string> imagepredictions)
        {
            // Если этот файл был классифицирован, изображние будет
подписано данным классом.
            if (imagepredictions.ContainsKey(loadedimage.Replace('\\',
'/')))
            {
                return imagepredictions[loadedimage.Replace('\\',
'/')];
            }
```

```
            else
            {
                // Иначе подпись "Неизвестно".
                return "Unknown";
            }
        }

        /// <summary>
        /// Запуск сервера
        /// </summary>
        /// <param name="server">Объект-процесс, отвечающий за
сервер</param>
        /// <param name="pathtoserver">Путь к скрипту сервера</param>
        public static void StartServer(out System.Diagnostics.Process
server, string pathtoserver)
        {

            // Создание процесса, отвечающего за сервер.
            server = new System.Diagnostics.Process
            {
                StartInfo = new System.Diagnostics.ProcessStartInfo
                {
                    FileName = "python.exe",
                    Arguments = "\"" + pathtoserver + "\"",

                    // Аргументы для запуска в "тихом режиме".
```

```
                CreateNoWindow = true,
                UseShellExecute = false
            }
        };

        // Запуск сервера.
        server.Start();
    }

    /// <summary>
    /// Получение файлов из директории, удаление неподдерживаемых, подготовка к отправке на сервер.
    /// </summary>
    /// <param name="filenames">Список путей к файлам</param>
    /// <param name="directory">Директория для обработки</param>
    /// <returns>Статус операции</returns>
    public static string GetAndRemove(out List<string> filenames, string directory)
    {
        filenames = new List<string>();

        // Попытка чтения файлов из директории.
        try
        {
            filenames = Directory.GetFiles(directory).ToList<string>();
```

```
            }
            catch (Exception e) when (e is IOException || e is
System.Security.SecurityException || e is UnauthorizedAccessException)
            {
                return "unreachable";
            }

            // Форматирование путей.
            filenames = Sender.Replace(filenames);



            // Удаление неподдерживаемых файлов.
            if (RemoveUnsupported(ref filenames))
            {
                return "correct";
            }
            else
            {
                return "incorrect";
            }
        }
    }
}
```

Лист регистрации 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.17701729.04.09-01 12 01-1				
Инв. № подл.	Подп. и дата	Взам. Инв. №	Инв. № дубл.	Подп. и дата