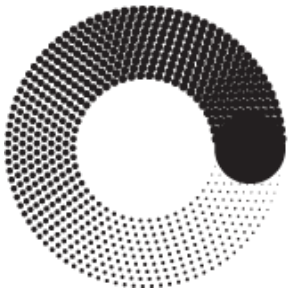
МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ



МОСКОВСКИЙ ПОЛИТЕХ

Кафедра СМАРТ-технологии

Лабораторная работа № 1:
«Алгоритмические методы сегментации изображений»

По дисциплине: «Нейронные сети глубокого обучения в обработке изображений»

Группа	221-327 № группы
Студент	Мезенцев И.В. Подпись студента
Дата	27.05.2025 Дата сдачи
Преподаватель	Идиатуллов Т.Т Подпись преподавателя

2025

Цель работы

Разработать приложение для автоматического распознавания символов на автомобильных номерных знаках, представленных на изображениях. Система должна включать этапы обнаружения номерных знаков и последующего оптического распознавания символов (OCR) на них.

Задачи

1. Разработать графический интерфейс пользователя (GUI) для загрузки изображений, запуска процесса распознавания и отображения результатов.
2. Реализовать модуль обнаружения номерных знаков на изображении с использованием скользящего окна и классификатора на основе многослойного персептрона (MLP).
3. Разработать модуль оптического распознавания символов (OCR) на выделенных областях номерных знаков, используя метод сопоставления с шаблонами.
4. Реализовать функционал обучения MLP для задачи классификации "номерной знак / не номерной знак".
5. Обеспечить возможность загрузки и сохранения весовых коэффициентов обученного MLP.
6. Реализовать загрузку библиотеки шаблонов символов для OCR и предусмотреть возможность использования нескольких вариантов шаблонов для одного символа.
7. Организовать вывод информативных логов о ходе работы программы и полученных результатах.

Ход работы

1. Создание проекта и интерфейса пользователя

Разработка была начата с создания проекта Windows Forms на языке C#. Графический интерфейс пользователя (рис. 1) был спроектирован для обеспечения следующих функций:

- Загрузка исходного изображения (`btnLoadImage` и `pictureBoxOriginal`).
- Запуск процесса обработки и распознавания (`btnProcess`).
- Отображение обработанного изображения с выделенными номерами и распознанным текстом (`pictureBoxProcessed`).

- Вывод логов операций и результатов (txtLog).
- Индикация прогресса выполнения длительных операций (progressBar и statusStrip1).
- Кнопка для запуска обучения MLP (btnTrainMLP).
- Флажок HideNoCharCheckBox для скрытия результатов, где не удалось надежно сегментировать символы.

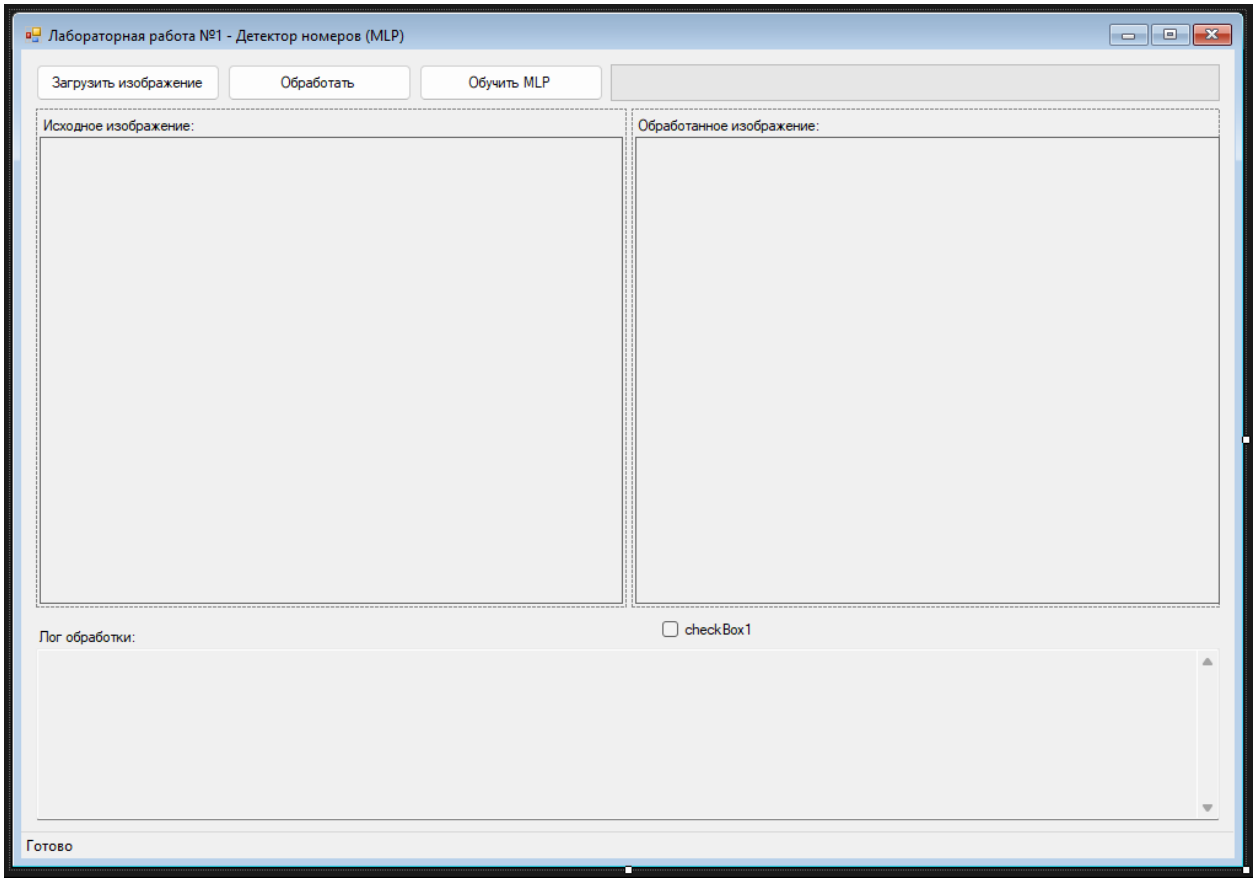


Рисунок 1 — Интерфейс программы.

2. Загрузка и предварительная обработка изображения

При нажатии кнопки "Загрузить изображение" (btnLoadImage_Click) пользователю предлагается выбрать файл изображения. Загруженное изображение отображается в pictureBoxOriginal. Для последующей обработки создается рабочая копия изображения (_scaledBaseImage), которая масштабируется до стандартного размера 640x480 пикселей для унификации обработки.

3. Обучение многослойного персептрона (MLP) для детекции номеров

Функционал обучения MLP реализован в методе btnTrainMLP_Click и классе PlateMLPClassifier.

- **Подготовка данных:** Обучающая выборка формируется из двух наборов изображений: положительные (фрагменты номерных знаков) и отрицательные (фрагменты фона). Изображения из этих наборов загружаются, конвертируются в оттенки серого, масштабируются до размера 24x24 пикселя (`MLP_TRAIN_TARGET_PATCH_SIZE`). Пиксельные значения нормализуются и преобразуются в векторы признаков.
- **Архитектура MLP:** Используется MLP со следующей архитектурой: 576 входов (24x24 пикселя), два скрытых слоя (128 и 64 нейрона) и 1 выходной нейрон. В качестве функции активации для всех слоев используется сигмоидальная функция (`SigmoidFunction` из Accord.NET).
- **Обучение:** Обучение сети производится с помощью алгоритма обратного распространения ошибки (`BackPropagationLearning` из Accord.NET). Параметры обучения, такие как количество эпох и скорость обучения, заданы в коде.
- **Сохранение весов:** После обучения весовые коэффициенты MLP сохраняются в бинарный файл (`plate_mlp_weights.bin`) для последующего использования при детекции. При запуске приложения веса загружаются автоматически.

4. Обнаружение номерных знаков

Процесс обнаружения номеров запускается кнопкой "Обработать" (`btnProcess_Click`):

- **Скользящее окно:** Изображение (`_scaledBaseImage` в оттенках серого) сканируется скользящим окном различных размеров (`windowScales`), пропорциональных ширине изображения. Шаг окна (`stepSizeFactor`) также адаптивен.
- **Классификация патчей:** Каждый фрагмент (патч), полученный из скользящего окна, преобразуется к размеру 24x24 пикселя и подается на вход обученного MLP. MLP выдает оценку вероятности того, что патч является номерным знаком.
- **Отбор кандидатов:** Патчи с оценкой MLP выше порога (0.70) добавляются в список потенциальных номерных знаков.
- **Немаксимальное подавление (NMS):** Для устранения множественных перекрывающихся детекций одного и того же номера применяется упрощенный алгоритм NMS (`ApplySimplifiedNMS`). Он сортирует кандидатов по убыванию оценки MLP и итеративно выбирает лучшего, удаляя сильно перекрывающиеся с ним (IoU > 0.2).

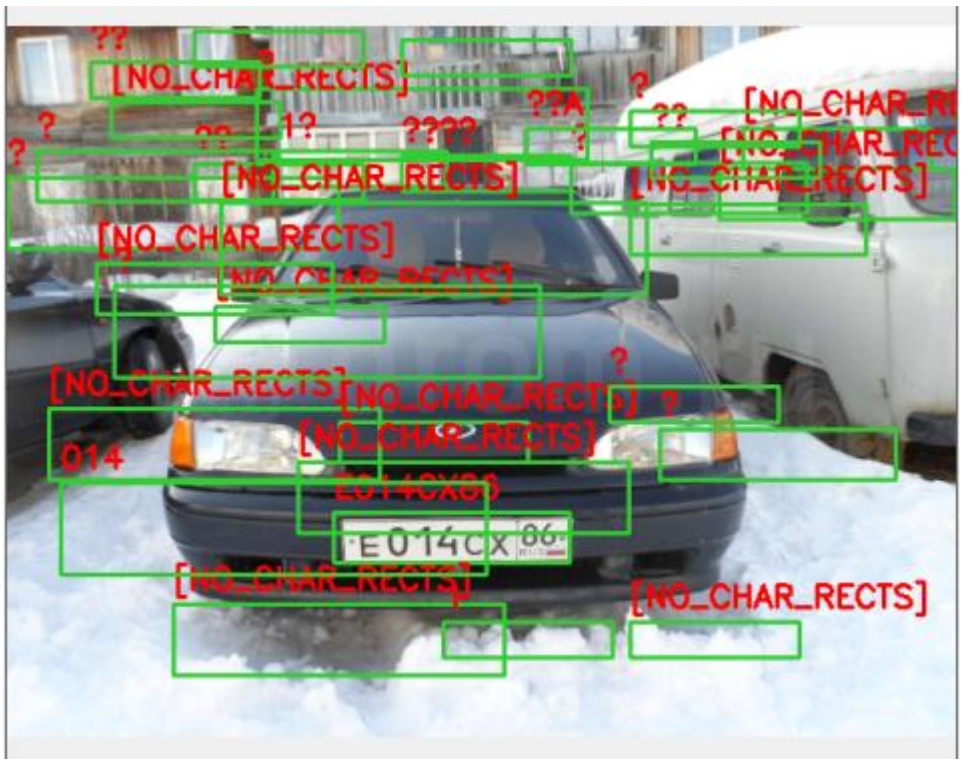


Рисунок 2 — Пример обнаружения номерного знака.

5. Распознавание символов на номерном знаке (OCR)

Для каждого обнаруженного и отфильтрованного NMS прямоугольника номера (`Rect`) выполняется распознавание символов методом `RecognizeCharsOnPlate`:

- **Предобработка области номера:**
  - Выделенная область конвертируется в оттенки серого.
  - Применяется адаптивное выравнивание гистограммы с ограничением контраста (CLAHE) для улучшения видимости символов.
  - Изображение бинаризуется с использованием метода Оцу.
- **Сегментация символов:**
  - На бинаризованном изображении номера выполняется поиск контуров (`Cv2.FindContours`).
  - Найденные контуры фильтруются по геометрическим признакам (высота, ширина, соотношение сторон) для отсеивания шума и выделения только тех, которые похожи на символы.
  - Отфильтрованные контуры (прямоугольники, описывающие символы) сортируются по их X-координате (слева направо).
- **Сопоставление с шаблонами:**
  - Каждый выделенный кандидат в символы масштабируется до стандартного размера (20x40 пикселей).

- Загружается библиотека шаблонов символов (_charTemplatesMulti). Шаблоны хранятся в виде PNG-файлов в папках, названных по имени символа (например, templates/A/). Поддерживается несколько вариантов шаблонов для каждого символа.
- Масштабированный символ сопоставляется с каждым шаблоном из библиотеки с помощью функции Cv2.MatchTemplate (метод CCoeffNormed).
- Символ, шаблон которого дал максимальный коэффициент корреляции (выше порога 0.50), считается распознанным. Если совпадение недостаточно уверенное, символ помечается как '?'.

- **Формирование результата:** Распознанные символы объединяются в строку. Если сегментация символов не удалась, возвращается специальная метка [NO_CHAR_RECTS].

Рисунок 3 — Результат сегментации символов на номере

6. Отображение результатов и логирование

- На обработанном изображении (_processedImage) вокруг обнаруженных номеров рисуются зеленые прямоугольники. Над каждым прямоугольником выводится распознанный текст красным цветом.
- Пользователь может использовать флажок HideNoCharCheckBox, чтобы скрыть рамки для тех номеров, где OCR вернул [NO_CHAR_RECTS], что обычно означает неудачную сегментацию символов.
- Все ключевые этапы работы, ошибки и результаты распознавания логируются в текстовое поле txtLog.

Вывод

В ходе выполнения курсовой работы было разработано приложение на языке C# с использованием Windows Forms, предназначенное для обнаружения и распознавания символов на автомобильных номерных знаках.

Основные достигнутые результаты:

1. Создан графический интерфейс пользователя, позволяющий загружать изображения, инициировать процесс распознавания, просматривать результаты и логи работы.
2. Реализован модуль обнаружения номерных знаков, использующий многомасштабное скользящее окно и классификатор на основе многослойного персептрона (MLP), обученный на положительных и отрицательных примерах. Применено немаксимальное подавление для уточнения детекций.
3. Разработан модуль оптического распознавания символов (OCR), который выполняет предобработку области номера, сегментацию символов на основе анализа контуров и их последующее распознавание методом сопоставления с эталонными шаблонами. Поддерживается использование нескольких вариантов шаблонов для повышения робастности.
4. Реализован функционал для обучения MLP, включая загрузку и предобработку обучающих данных, обучение сети и сохранение/загрузку ее весовых коэффициентов.
5. Для обработки изображений и реализации нейросетевых алгоритмов использовались библиотеки OpenCvSharp и Accord.NET соответственно.

Разработанное приложение демонстрирует работоспособность выбранных подходов для решения задачи распознавания автомобильных номеров и может служить основой для дальнейшего усовершенствования.

Приложение А

Листинг кода Form1.cs

```
    using System;
    using System.Collections.Generic;
    using System.ComponentModel;
    using System.Data;
    // System.Drawing убрали, чтобы избежать конфликтов, если
он не нужен явно для другого
    using System.IO;
    using System.Linq;
    using System.Text;
    using System.Threading.Tasks;
    using System.Windows.Forms;
    using OpenCvSharp;
    using OpenCvSharp.Extensions; // Для BitmapConverter
    // using Accord.Neuro; // Accord.Neuro используется в
PlateMLPClassifier
    // using OpenCvSharp.ML; // Если не используется напрямую
здесь, можно убрать

    namespace L1_oneN
    {
        public partial class Form1 : Form
        {
            private Mat _originalImage;
            private Mat _processedImage;
            private Mat _scaledBaseImage; // Для перерисовки
с учетом состояния HideNoCharCheckBox

            // Структура для хранения информации об элементах
для отрисовки
            private struct DrawingInfo
            {
                public Rect Rectangle;
                public Scalar RectColor;
                public string Text;
                public Point TextPosition;
                public HersheyFonts FontFace;
                public double FontScale;
                public Scalar TextColor;
                public int Thickness;
                public bool IsNoCharResult; // Флаг для рамок
с "NO_CHAR_RECTS"
```

```
            }
            private List<DrawingInfo> _plateDrawingInfos = new
List<DrawingInfo>();


            // Обновленные константы для MLP
            private const int MLP_INPUT_SIZE = 24;
            private const double PLATE_ASPECT_RATIO = 520.0 /
115.0;
            private    static    readonly    OpenCvSharp.Size
TEMPLATE_CHAR_SIZE = new OpenCvSharp.Size(20, 40);

            private        Dictionary<char,        List<Mat>>
_charTemplatesMulti = new Dictionary<char, List<Mat>>();
            private      string      _templateChars      =
"0123456789ABEKMHOPCTYX";

            private PlateMLPClassifier _plateMlp;
            private    const    string    MlpWeightsFile    =
"plate_mlp_weights.bin";

            private  const  int  MLP_TRAIN_TARGET_PATCH_SIZE  =
24;


            public Form1()
            {
                InitializeComponent();

                _plateMlp = new PlateMLPClassifier();
                UpdateMlpStatusLabel();

                LoadCharTemplates("templates");

                // Предполагается, что CheckBox с именем
HideNoChar уже добавлен на форму через дизайнер.
                // Если его имя другое, замените "HideNoChar"
ниже.
                // Пример, если бы его нужно было создать
программно (но он должен быть из дизайнера):
                // this.HideNoChar = new CheckBox { Name =
"HideNoChar", Text = "Скрыть 'нет символов'", Location = new
System.Drawing.Point( /*...*/ ) };
                // this.Controls.Add(this.HideNoChar);
```

```
                // Убедитесь, что у вашего CheckBox в дизайнере установлено имя "HideNoChar"
                // или измените имя в следующей строке на правильное.
                if (this.Controls.ContainsKey("HideNoChar"))
                {
                    (this.Controls["HideNoChar"] as CheckBox).CheckedChanged += HideNoChar_CheckedChanged;
                }
                else
                {
                    // Можно вывести предупреждение или создать чекбокс программно, если он критичен
                    AppendLogUiThread("ПРЕДУПРЕЖДЕНИЕ: CheckBox 'HideNoChar' не найден на форме.");
                }
            }

            private void UpdateMlpStatusLabel()
            {
                if (_plateMlp.LoadWeights(MlpWeightsFile))
                {
                    AppendLogUiThread("MLP: Веса успешно загружены.");
                    if (this.statusStrip1 != null && this.statusStrip1.Items["toolStripStatusLabel"] != null)
                        ((ToolStripStatusLabel)this.statusStrip1.Items["toolStripStatusLabel"]).Text = "MLP: Готов (веса загружены)";
                }
                else
                {
                    AppendLogUiThread("ПРЕДУПРЕЖДЕНИЕ: Веса MLP не загружены. Детектор номеров будет работать некорректно.");
                    if (this.statusStrip1 != null && this.statusStrip1.Items["toolStripStatusLabel"] != null)
                        ((ToolStripStatusLabel)this.statusStrip1.Items["toolStripStatusLabel"]).Text = "MLP: Не обучен (веса не найдены)";
                }
            }
```

```
            private void btnLoadImage_Click(object sender,
EventArgs e)
            {
                using (OpenFileDialog openFileDialog = new
OpenFileDialog())
                {
                    openFileDialog.Filter      =      "Image
Files(*.BMP;*.JPG;*.JPEG;*.PNG)|*.BMP;*.JPG;*.JPEG;*.PNG|All
files (*.*)|*.*";
                    if    (openFileDialog.ShowDialog()    ==
DialogResult.OK)
                    {
                        try
                        {
                            _originalImage?.Dispose();
                            _scaledBaseImage?.Dispose();   //
Очищаем также и базовое изображение для перерисовки
                            _processedImage?.Dispose();    //
И обработанное
                            _plateDrawingInfos.Clear();    //
Очищаем инструкции рисования

                            _originalImage                 =
Cv2.ImRead(openFileDialog.FileName, ImreadModes.Color);
                            if (_originalImage.Empty())
                            {
                                MessageBox.Show("Не   удалось
загрузить   изображение.",   "Ошибка",   MessageBoxButtons.OK,
MessageBoxIcon.Error);
                                return;
                            }
                            pictureBoxOriginal.Image       =
OpenCvSharp.Extensions.BitmapConverter.ToBitmap(_originalImag
e);
                            AppendLogUiThread("---      Новое
изображение загружено ---");
                            pictureBoxProcessed.Image = null;
// Очищаем предыдущий результат

                            if (_plateMlp.IsTrained)
                            {
                                AppendLogUiThread("Состояние
MLP: Веса загружены.");
                            }
```

```
                            else
                            {
                                AppendLogUiThread("Состояние
MLP: Веса НЕ загружены или MLP не обучен.");
                            }
                        }
                        catch (Exception ex)
                        {
                            MessageBox.Show($"Ошибка      при
загрузке      изображения:      {ex.Message}",      "Ошибка",
MessageBoxButtons.OK, MessageBoxIcon.Error);
                        }
                    }
                }
            }

            private       void       LoadCharTemplates(string
baseTemplatesFolderPath)
            {
                _charTemplatesMulti.Clear();
                if
(!Directory.Exists(baseTemplatesFolderPath))
                {
                    AppendLogUiThread($"Базовая    папка    с
шаблонами '{baseTemplatesFolderPath}' не найдена.");
                    return;
                }

                int totalTemplatesLoaded = 0;
                foreach (char c in _templateChars)
                {
                    string          charFolderPath          =
Path.Combine(baseTemplatesFolderPath,
c.ToString().ToUpperInvariant());

                    if (Directory.Exists(charFolderPath))
                    {
                        List<Mat>   templatesForChar   =   new
List<Mat>();
                        string[]       templateFiles       =
Directory.GetFiles(charFolderPath, "*.png");

                        if (templateFiles.Length == 0)
                        {
```

```
                            //        AppendLogUiThread($"Для
символа '{c}' в папке '{charFolderPath}' не найдено файлов
шаблонов .png.");
                            continue;
                        }

                        foreach (string templatePath in
templateFiles)
                        {
                            using (Mat tpl =
Cv2.ImRead(templatePath, ImreadModes.Grayscale))
                            {
                                if (!tpl.Empty())
                                {
                                    Mat processedTpl = new
Mat();
                                    Cv2.Threshold(tpl,
processedTpl, 128, 255, ThresholdTypes.Binary |
ThresholdTypes.Otsu);
                                    Cv2.Resize(processedTpl,
processedTpl, TEMPLATE_CHAR_SIZE, interpolation:
InterpolationFlags.Linear);

templatesForChar.Add(processedTpl);
                                    totalTemplatesLoaded++;
                                }
                                else
                                {
                                    AppendLogUiThread($"Не
удалось загрузить шаблон из файла: {templatePath}");
                                }
                            }
                        }

                        if (templatesForChar.Count > 0)
                        {
                            _charTemplatesMulti[c] =
templatesForChar;
                            //        AppendLogUiThread($"Для
символа '{c}' загружено {templatesForChar.Count} шаблонов.");
                        }
                    }
                    else
                    {
```

```
                        AppendLogUiThread($"Папка для символа
'{c}' не найдена: {charFolderPath}");
                    }
                }
                AppendLogUiThread($"Всего загружено
{totalTemplatesLoaded} шаблонов для
{_charTemplatesMulti.Count} уникальных символов.");
            }


            private async void btnProcess_Click(object sender,
EventArgs e)
            {
                if (_originalImage == null ||
_originalImage.Empty())
                {
                    MessageBox.Show("Сначала загрузите
изображение.", "Информация", MessageBoxButtons.OK,
MessageBoxIcon.Information);
                    return;
                }
                if (_charTemplatesMulti.Count == 0)
                {
                    MessageBox.Show("Шаблоны символов не
загружены. Распознавание символов невозможно. Проверьте папку
'templates' и лог загрузки шаблонов.", "Ошибка",
MessageBoxButtons.OK, MessageBoxIcon.Error);
                    return;
                }

                btnProcess.Enabled = false;
                btnLoadImage.Enabled = false;
                if (this.Controls.ContainsKey("btnTrainMLP"))
((Button)this.Controls["btnTrainMLP"]).Enabled = false;
                if (this.Controls.ContainsKey("HideNoChar"))
(this.Controls["HideNoChar"] as CheckBox).Enabled = false;


                AppendLogUiThread("Начало обработки...");
                if (this.statusStrip1 != null &&
this.statusStrip1.Items["toolStripStatusLabel"] != null)

((ToolStripStatusLabel)this.statusStrip1.Items["toolStripStat
usLabel"]).Text = "Обработка...";
```

```
                progressBar.Style = ProgressBarStyle.Marquee;
                progressBar.Value = 0;

                _plateDrawingInfos.Clear();     //     Очищаем
предыдущие инструкции рисования

                // Создаем _scaledBaseImage - это будет наша
основа для рисования
                _scaledBaseImage?.Dispose();
                _scaledBaseImage = _originalImage.Clone();
                if (_scaledBaseImage.Empty())
                {
                    AppendLogUiThread("Критическая ошибка: не
удалось клонировать _originalImage для _scaledBaseImage.");
                    // Восстановление UI
                    btnProcess.Enabled        =        true;
btnLoadImage.Enabled = true;
                    if
(this.Controls.ContainsKey("btnTrainMLP"))
((Button)this.Controls["btnTrainMLP"]).Enabled = true;
                    if
(this.Controls.ContainsKey("HideNoChar"))
(this.Controls["HideNoChar"] as CheckBox).Enabled = true;
                    UpdateMlpStatusLabel();
                    return;
                }
                Cv2.Resize(_scaledBaseImage,
_scaledBaseImage, new OpenCvSharp.Size(640, 480));

                // Изображение в оттенках серого для детекции
                Mat grayImage = new Mat();
                Cv2.CvtColor(_scaledBaseImage,     grayImage,
ColorConversionCodes.BGR2GRAY); // Используем _scaledBaseImage
для получения grayImage

                List<Tuple<Rect,  double>>  potentialPlates  =
new List<Tuple<Rect, double>>();
                int[] windowScales = {
                    (int)(grayImage.Width      *      0.18),
(int)(grayImage.Width * 0.20),
                    (int)(grayImage.Width      *      0.25),
(int)(grayImage.Width * 0.35),
                    (int)(grayImage.Width * 0.45),
                };
```

```
                int stepSizeFactor = 8;

                await Task.Run(() =>
                {
                    foreach (int baseWidth in windowScales)
                    {
                        int windowW = baseWidth;
                        int   windowH   =   (int)(windowW   /
PLATE_ASPECT_RATIO);
                        if   (windowW   <   MLP_INPUT_SIZE   ||
windowH < MLP_INPUT_SIZE || windowW <= 0 || windowH <= 0)
continue;
                        int  stepX  =  Math.Max(1,  windowW  /
stepSizeFactor);
                        int  stepY  =  Math.Max(1,  windowH  /
stepSizeFactor);

                        for (int y = 0; y <= grayImage.Rows -
windowH; y += stepY)
                        {
                            for   (int   x   =   0;   x   <=
grayImage.Cols - windowW; x += stepX)
                            {
                                Rect roiRect = new Rect(x, y,
windowW, windowH);
                                using   (Mat   patch   =   new
Mat(grayImage, roiRect))
                                using (Mat mlpInputPatch = new
Mat())
                                {
                                    if        (patch.Empty())
continue;
                                    Cv2.Resize(patch,
mlpInputPatch,      new      OpenCvSharp.Size(MLP_INPUT_SIZE,
MLP_INPUT_SIZE), 0, 0, InterpolationFlags.Area);
                                    double      score      =
_plateMlp.Predict(mlpInputPatch);
                                    if (score > 0.70)
                                    {
                                        lock
(potentialPlates)
                                        {

potentialPlates.Add(new Tuple<Rect, double>(roiRect, score));
```

```
                                }
                            }
                        }
                    }
                }
            });

            grayImage.Dispose();

            List<Rect> detectedPlates = ApplySimplifiedNMS(potentialPlates, 0.2);

            int plateCounter = 0;
            foreach (Rect plateRectOriginal in detectedPlates)
            {
                plateCounter++;
                Rect plateRectForOCR = plateRectOriginal;

                if (plateRectForOCR.X < 0 || plateRectForOCR.Y < 0 ||
                    plateRectForOCR.Width <= 0 || plateRectForOCR.Height <= 0 ||
                    plateRectForOCR.Right > _scaledBaseImage.Cols ||
                    plateRectForOCR.Bottom > _scaledBaseImage.Rows)
                {
                    AppendLogUiThread($"ПРЕДУПРЕЖДЕНИЕ: Обнаружен некорректный Rect для номера: {plateRectForOCR}. Пропуск.");
                    continue;
                }

                // Кандидат для OCR берем из _scaledBaseImage (чистое, цветное, масштабированное)
                using (Mat plateCandidate = new Mat(_scaledBaseImage, plateRectForOCR))
                {
                    if (plateCandidate.Empty())
                    {
```

```
AppendLogUiThread($"ПРЕДУПРЕЖДЕНИЕ: Не удалось создать plateCandidate для Rect: {plateRectForOCR}. Пропуск.");
                            continue;
                        }
                        string recognizedText = RecognizeCharsOnPlate(plateCandidate.Clone());

                        if (recognizedText == null)
                        {

AppendLogUiThread($"ПРЕДУПРЕЖДЕНИЕ: RecognizeCharsOnPlate вернула null для области {plateRectForOCR}. Используется \"[OCR_ERR]\".");
                            recognizedText = "[OCR_ERR]";
                        }

                        bool isNoCharRect = recognizedText == "[NO_CHAR_RECTS]";

                        _plateDrawingInfos.Add(new DrawingInfo
                        {
                            Rectangle = plateRectOriginal,
                            RectColor = Scalar.LimeGreen,
                            Text = recognizedText,
                            TextPosition = new OpenCvSharp.Point(plateRectOriginal.X, plateRectOriginal.Y - 10),
                            FontFace = HersheyFonts.HersheySimplex,
                            FontScale = 0.7,
                            TextColor = Scalar.Red,
                            Thickness = 2,
                            IsNoCharResult = isNoCharRect
                        });

                        AppendLogUiThread($"Номер {plateCounter}: {recognizedText} (область: {plateRectOriginal})");
                    }
                }

                if (detectedPlates.Count == 0)
```

```
                {
                    AppendLogUiThread("Номерные знаки не
найдены.");
                }
                if (!_plateMlp.IsTrained &&
detectedPlates.Count > 0)
                {
                    AppendLogUiThread("ПРЕДУПРЕЖДЕНИЕ:
Найдены кандидаты, но MLP не обучен. Результаты могут быть
случайными!");
                }

                RefreshProcessedImageDisplay();           //
Первоначальная отрисовка

                AppendLogUiThread("Обработка завершена.");
                progressBar.Style = ProgressBarStyle.Blocks;
                progressBar.Value = progressBar.Maximum;

                btnProcess.Enabled = true;
                btnLoadImage.Enabled = true;
                if (this.Controls.ContainsKey("btnTrainMLP"))
((Button)this.Controls["btnTrainMLP"]).Enabled = true;
                if  (this.Controls.ContainsKey("HideNoChar"))
(this.Controls["HideNoChar"] as CheckBox).Enabled = true;
                UpdateMlpStatusLabel();
            }

            private void RefreshProcessedImageDisplay()
            {
                if    (_scaledBaseImage    ==    null    ||
_scaledBaseImage.Empty())
                {
                    _processedImage?.Dispose();
                    _processedImage = null;
                    pictureBoxProcessed.Image = null;
                    return;
                }

                _processedImage?.Dispose();
                _processedImage = _scaledBaseImage.Clone();

                bool hideNoCharResults = false;
```

```
                // Убедитесь, что имя "HideNoChar" соответствует имени вашего чекбокса в дизайнере
                if (this.Controls.ContainsKey("HideNoChar") && this.Controls["HideNoChar"] is CheckBox hideNoCharCb)
                {
                    hideNoCharResults = hideNoCharCb.Checked;
                }

                foreach (var drawingInfo in _plateDrawingInfos)
                {
                    if (drawingInfo.IsNoCharResult && hideNoCharResults)
                    {
                        continue; // Пропускаем рисование, если это "пустой" результат и чекбокс отмечен
                    }

                    Cv2.Rectangle(_processedImage, drawingInfo.Rectangle, drawingInfo.RectColor, drawingInfo.Thickness);

                    if (!string.IsNullOrEmpty(drawingInfo.Text))
                    {
                        Cv2.PutText(_processedImage, drawingInfo.Text, drawingInfo.TextPosition,
                                    drawingInfo.FontFace, drawingInfo.FontScale, drawingInfo.TextColor, drawingInfo.Thickness);
                    }
                }

                if (_processedImage != null && !_processedImage.Empty())
                {
                    pictureBoxProcessed.Image = OpenCvSharp.Extensions.BitmapConverter.ToBitmap(_processedImage);
                }
                else
                {
                    pictureBoxProcessed.Image = null;
```

```
                    AppendLogUiThread("Ошибка: Обработанное
изображение (_processedImage) пустое или null после попытки
перерисовки.");
                }
            }


            private void HideNoChar_CheckedChanged(object
sender, EventArgs e)
            {
                // Перерисовываем, только если есть базовое
изображение
                //  _plateDrawingInfos может быть пустым,
RefreshProcessedImageDisplay это обработает.
                if (_scaledBaseImage != null &&
!_scaledBaseImage.Empty())
                {
                    RefreshProcessedImageDisplay();
                }
            }

            private List<Rect>
ApplySimplifiedNMS(List<Tuple<Rect, double>> proposals, double
overlapThreshold)
            {
                List<Rect> finalDetections = new List<Rect>();
                if (proposals.Count == 0) return
finalDetections;

                proposals.Sort((a, b) =>
b.Item2.CompareTo(a.Item2));
                List<Tuple<Rect, double>> picked = new
List<Tuple<Rect, double>>();

                while (proposals.Count > 0)
                {
                    var current = proposals[0];
                    picked.Add(current);
                    proposals.RemoveAt(0);
                    proposals.RemoveAll(proposal =>
CalculateIoU(current.Item1, proposal.Item1) >
overlapThreshold);
                }
```

```
                finalDetections.AddRange(picked.Select(p   =>
p.Item1));
                return finalDetections;
            }

            private double CalculateIoU(Rect r1, Rect r2)
            {
                int xA = Math.Max(r1.Left, r2.Left);
                int yA = Math.Max(r1.Top, r2.Top);
                int xB = Math.Min(r1.Right, r2.Right);
                int yB = Math.Min(r1.Bottom, r2.Bottom);
                int  interArea  =  Math.Max(0,  xB  -  xA)  *
Math.Max(0, yB - yA);
                if (interArea == 0) return 0;
                int boxAArea = r1.Width * r1.Height;
                int boxBArea = r2.Width * r2.Height;
                double iou = (double)interArea / (boxAArea +
boxBArea - interArea);
                return iou;
            }

            private     string     RecognizeCharsOnPlate(Mat
plateImage)
            {
                if (plateImage.Empty())
                {

AppendLogUiThread("[RecognizeCharsOnPlate]            Входное
изображение номера пустое.");
                    return "[PLATE_EMPTY]";
                }

                StringBuilder    recognizedText    =    new
StringBuilder();
                Mat grayPlate = new Mat();
                Mat preprocessedPlate = new Mat();
                List<Rect>    charBoundingRects    =    new
List<Rect>();

                try
                {
                    if (plateImage.Channels() > 1)
                        Cv2.CvtColor(plateImage,   grayPlate,
ColorConversionCodes.BGR2GRAY);
```

```
                    else
                        plateImage.CopyTo(grayPlate);

                    using (var clahe = Cv2.CreateCLAHE(clipLimit: 2.0, tileGridSize: new OpenCvSharp.Size(8, 8)))
                    {
                        clahe.Apply(grayPlate, preprocessedPlate);
                    }
                    Cv2.Threshold(preprocessedPlate, preprocessedPlate, 0, 255, ThresholdTypes.BinaryInv | ThresholdTypes.Otsu);

                    OpenCvSharp.Point[][] contours;
                    HierarchyIndex[] hierarchy;

                    Cv2.FindContours(preprocessedPlate.Clone(), out contours, out hierarchy, RetrievalModes.External, ContourApproximationModes.ApproxSimple);

                    if (contours == null || contours.Length == 0)
                    {
                        // AppendLogUiThread("[RecognizeCharsOnPlate] Контуры символов не найдены после FindContours.");
                        grayPlate.Dispose(); preprocessedPlate.Dispose();
                        return "[NO_CONTOURS]";
                    }

                    double plateH = preprocessedPlate.Height;
                    double plateW = preprocessedPlate.Width;

                    foreach (OpenCvSharp.Point[] contour in contours)
                    {
                        if (contour == null || contour.Length < 3) continue;
                        Rect boundingRect = Cv2.BoundingRect(contour);
                        bool hCond = boundingRect.Height > plateH * 0.30 && boundingRect.Height < plateH * 0.95;
```

```
                        bool wCond = boundingRect.Width > plateW * 0.03 && boundingRect.Width < plateW * 0.30;
                        double ar = (boundingRect.Height == 0) ? 1000 : (double)boundingRect.Width / boundingRect.Height;
                        bool arCond = ar > 0.1 && ar < 1.2;
                        bool sizeCond = boundingRect.Width > 3 && boundingRect.Height > 7;
                        if (hCond && wCond && arCond && sizeCond)
                        {
                            charBoundingRects.Add(boundingRect);
                        }
                    }

                    if (charBoundingRects.Count == 0)
                    {
                        // AppendLogUiThread("[RecognizeCharsOnPlate] Прямоугольники символов не найдены после фильтрации.");
                        grayPlate.Dispose(); preprocessedPlate.Dispose();
                        return "[NO_CHAR_RECTS]"; // Это ключевая строка для скрытия
                    }

                    charBoundingRects = charBoundingRects.OrderBy(r => r.X).ToList();

                    foreach (Rect charRect in charBoundingRects)
                    {
                        Rect safeCharRect = new Rect(
                            Math.Max(0, charRect.X), Math.Max(0, charRect.Y),
                            Math.Min(charRect.Width, preprocessedPlate.Cols - Math.Max(0, charRect.X)),
                            Math.Min(charRect.Height, preprocessedPlate.Rows - Math.Max(0, charRect.Y))
                        );
                        if (safeCharRect.Width <= 0 || safeCharRect.Height <= 0) continue;
```

```
                        using (Mat charCandidate = new
Mat(preprocessedPlate, safeCharRect))
                        using (Mat resizedChar = new Mat())
                        {
                            if (charCandidate.Empty())
continue;
                            Cv2.Resize(charCandidate,
resizedChar, TEMPLATE_CHAR_SIZE, interpolation:
InterpolationFlags.Linear);
                            char bestMatchCharOverall = '?';
                            double maxMatchScoreOverall = -
1.0;
                            foreach (var charEntry in
_charTemplatesMulti)
                            {
                                char currentSymbolKey =
charEntry.Key;
                                List<Mat>
listOfTemplatesForSymbol = charEntry.Value;
                                if (listOfTemplatesForSymbol
== null) continue;
                                double
bestScoreForThisSymbolKeyFromVariants = -1.0;
                                foreach (Mat templateVariant
in listOfTemplatesForSymbol)
                                {
                                    if (templateVariant ==
null || templateVariant.Empty()) continue;
                                    using (Mat result = new
Mat())
                                    {

Cv2.MatchTemplate(resizedChar, templateVariant, result,
TemplateMatchModes.CCoeffNormed);
                                        Cv2.MinMaxLoc(result,
out _, out double currentVariantScore, out _, out _);
                                        if
(currentVariantScore > bestScoreForThisSymbolKeyFromVariants)
                                        {

bestScoreForThisSymbolKeyFromVariants = currentVariantScore;
                                        }
                                    }
                                }
```

```
                                if
(bestScoreForThisSymbolKeyFromVariants                          >
maxMatchScoreOverall)
                                {
                                    maxMatchScoreOverall    =
bestScoreForThisSymbolKeyFromVariants;
                                    bestMatchCharOverall    =
currentSymbolKey;
                                }
                            }
                            if (maxMatchScoreOverall > 0.50)
                            {

recognizedText.Append(bestMatchCharOverall);
                            }
                            else
                            {
                                recognizedText.Append('?');
                            }
                        }
                    }
                }
                catch (Exception ex)
                {

AppendLogUiThread($"[RecognizeCharsOnPlate]       Исключение:
{ex.Message}\n{ex.StackTrace}");
                    return "[OCR_EXCEPTION]";
                }
                finally
                {
                    grayPlate.Dispose();
                    preprocessedPlate.Dispose();
                }

                if    (charBoundingRects.Count    >    0    &&
(recognizedText.Length == 0 || recognizedText.ToString().All(c
=> c == '?')))
                {
                    //
AppendLogUiThread("[RecognizeCharsOnPlate]    Кандидаты    на
символы были, но не распознаны или все с низким баллом.");
```

```
                    return new string('?', Math.Min(charBoundingRects.Count, 8)); // Возвращаем '???' вместо "[NO_CHAR_RECTS]"
                }

                return recognizedText.ToString();
            }

            protected override void OnFormClosing(FormClosingEventArgs e)
            {
                _originalImage?.Dispose();
                _processedImage?.Dispose();
                _scaledBaseImage?.Dispose(); // Освобождаем _scaledBaseImage
                foreach (var charTemplatesList in _charTemplatesMulti.Values)
                {
                    if (charTemplatesList != null)
                    {
                        foreach (var tpl in charTemplatesList)
                        {
                            tpl?.Dispose();
                        }
                    }
                }
                _charTemplatesMulti.Clear();
                base.OnFormClosing(e);
            }

            private async void btnTrainMLP_Click(object sender, EventArgs e)
            {
                btnLoadImage.Enabled = false;
                btnProcess.Enabled = false;
                if (this.Controls.ContainsKey("btnTrainMLP")) ((Button)this.Controls["btnTrainMLP"]).Enabled = false;
                if (this.Controls.ContainsKey("HideNoChar")) (this.Controls["HideNoChar"] as CheckBox).Enabled = false;


                AppendLogUiThread("\n--- Начало обучения MLP ---");
```

```
                if    (this.statusStrip1    !=    null    &&
this.statusStrip1.Items["toolStripStatusLabel"] != null)

((ToolStripStatusLabel)this.statusStrip1.Items["toolStripStat
usLabel"]).Text = "MLP: Обучение...";
                progressBar.Style = ProgressBarStyle.Marquee;
                progressBar.Value = 0;

                string          positiveSamplesDir          =
@"C:\trasher\app\6sem\Neuro\L1_Nomera\test";
                string          negativeSamplesDir          =
@"C:\trasher\app\6sem\Neuro\L1_Nomera\Negative";

                int epochs = 1500;
                double learningRate = 0.001;

                List<double[]>    trainingInputs    =    new
List<double[]>();
                List<double[]>    trainingOutputs    =    new
List<double[]>();

                bool trainingSuccess = await Task.Run(() =>
                {
                    AppendLogUiThread("Загрузка положительных
примеров...");

LoadAndProcessSamplesForTraining(positiveSamplesDir,     1.0,
trainingInputs, trainingOutputs);
                    AppendLogUiThread("Загрузка отрицательных
примеров...");

LoadAndProcessSamplesForTraining(negativeSamplesDir,     0.0,
trainingInputs, trainingOutputs);

                    if (trainingInputs.Count == 0)
                    {
                        AppendLogUiThread("Ошибка: Нет данных
для обучения! Проверьте пути и наличие изображений.");
                        return false;
                    }
                    if (trainingInputs.Count < 50)
                    {
```

```
                    AppendLogUiThread($"Предупреждение:
Очень мало обучающих данных ({trainingInputs.Count}). Качество
модели может быть низким.");
                }

                AppendLogUiThread("Перемешивание
данных...");
                Random rng = new Random();
                int n = trainingInputs.Count;
                for (int i = 0; i < n - 1; i++)
                {
                    int k = rng.Next(i, n);
                    var tempInput = trainingInputs[k];
trainingInputs[k] = trainingInputs[i]; trainingInputs[i] =
tempInput;
                    var tempOutput = trainingOutputs[k];
trainingOutputs[k] = trainingOutputs[i]; trainingOutputs[i] =
tempOutput;
                }
                AppendLogUiThread("Данные перемешаны.");
                AppendLogUiThread($"Всего подготовлено
{trainingInputs.Count} примеров для обучения (размер патча:
{MLP_TRAIN_TARGET_PATCH_SIZE}x{MLP_TRAIN_TARGET_PATCH_SIZE}).
");

                _plateMlp = new PlateMLPClassifier();
                _plateMlp.Train(trainingInputs,
trainingOutputs, epochs: epochs, learningRate: learningRate,
logCallback: AppendLogUiThread);

                if
(_plateMlp.SaveWeights(MlpWeightsFile))
                {
                    AppendLogUiThread($"Веса MLP успешно
сохранены в файл: {Path.GetFullPath(MlpWeightsFile)}");
                    return true;
                }
                else
                {
                    AppendLogUiThread("Ошибка при
сохранении весов MLP.");
                    return false;
                }
            });
```

```
                if (trainingSuccess)
                {
                    AppendLogUiThread("---    Обучение    MLP
успешно завершено ---");
                }
                else
                {
                    AppendLogUiThread("---  Обучение  MLP  не
удалось или было прервано ---");
                }

                progressBar.Style = ProgressBarStyle.Blocks;
                progressBar.Value = progressBar.Maximum;

                btnLoadImage.Enabled = true;
                btnProcess.Enabled = true;
                if (this.Controls.ContainsKey("btnTrainMLP"))
((Button)this.Controls["btnTrainMLP"]).Enabled = true;
                if  (this.Controls.ContainsKey("HideNoChar"))
(this.Controls["HideNoChar"] as CheckBox).Enabled = true;
                UpdateMlpStatusLabel();
            }

            private                                        void
LoadAndProcessSamplesForTraining(string directoryPath, double
label, List<double[]> inputs, List<double[]> outputs)
            {
                if (!Directory.Exists(directoryPath))
                {
                    AppendLogUiThread($"Папка   не   найдена:
{directoryPath}");
                    return;
                }
                string[]             imageFiles             =
Directory.GetFiles(directoryPath,                          "*.*",
SearchOption.TopDirectoryOnly)
                    .Where(s      =>      s.EndsWith(".png",
StringComparison.OrdinalIgnoreCase) ||
                                s.EndsWith(".jpg",
StringComparison.OrdinalIgnoreCase) ||
                                s.EndsWith(".jpeg",
StringComparison.OrdinalIgnoreCase) ||
```

```
                                s.EndsWith(".bmp",
StringComparison.OrdinalIgnoreCase)).ToArray();
                AppendLogUiThread($"Найдено
{imageFiles.Length}                 файлов                 в
{Path.GetFileName(directoryPath)}");
                int processedCount = 0;
                foreach (string filePath in imageFiles)
                {
                    try
                    {
                        using     (Mat     originalMat     =
Cv2.ImRead(filePath, ImreadModes.Color))
                        {
                            if          (originalMat.Empty())
continue;
                            using (Mat grayMat = new Mat())
                            using (Mat resizedMat = new Mat())
                            {
                                Cv2.CvtColor(originalMat,
grayMat, ColorConversionCodes.BGR2GRAY);
                                Cv2.Resize(grayMat,
resizedMat, new OpenCvSharp.Size(MLP_TRAIN_TARGET_PATCH_SIZE,
MLP_TRAIN_TARGET_PATCH_SIZE), 0, 0, InterpolationFlags.Area);
                                double[]  inputVector  =  new
double[MLP_TRAIN_TARGET_PATCH_SIZE                          *
MLP_TRAIN_TARGET_PATCH_SIZE];
                                int k = 0;
                                for  (int  i  =  0;  i  <
resizedMat.Rows; i++)
                                {
                                    for  (int  j  =  0;  j  <
resizedMat.Cols; j++)
                                    {
                                        inputVector[k++]    =
resizedMat.Get<byte>(i, j) / 255.0;
                                    }
                                }
                                inputs.Add(inputVector);
                                outputs.Add(new  double[]  {
label });
                                processedCount++;
                            }
                        }
                    }
```

```
                    catch (Exception ex)
                    {
                        AppendLogUiThread($"Ошибка при
обработке файла {filePath} для обучения: {ex.Message}");
                    }
                }
                AppendLogUiThread($"Успешно обработано
{processedCount} файлов из {Path.GetFileName(directoryPath)}
для обучения.");
            }

            private void AppendLogUiThread(string message)
            {
                if (txtLog.InvokeRequired)
                {
                    txtLog.Invoke(new Action(() =>
AppendLogUiThread(message)));
                }
                else
                {
                    if (txtLog.Text.Length > 30000)
                    {
                        txtLog.Text =
txtLog.Text.Substring(txtLog.Text.Length - 15000);
                    }
                    txtLog.AppendText(message +
Environment.NewLine);
                    txtLog.SelectionStart =
txtLog.Text.Length;
                    txtLog.ScrollToCaret();
                }
            }
        }
    }
```

Листинг кода Form1.cs

```
    // PlateMLPClassifier.cs
    using Accord.Neuro;
    using Accord.Neuro.Learning;
    using Accord.Neuro.ActivationFunctions; // Убедитесь, что
    это пространство имен добавлено
    using OpenCvSharp;
```

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq; // Добавлено для ToArray() если используется, и для других LINQ операций

namespace L1_oneN
{
    public class PlateMLPClassifier
    {
        private ActivationNetwork _network;
        private const int PatchSize = 24;
        private const int InputSize = PatchSize * PatchSize;
        private const int HiddenLayer1Size = 128;
        private const int HiddenLayer2Size = 64;
        private const int OutputSize = 1;

        private bool _isTrained = false;
        public bool IsTrained => _isTrained;

        public PlateMLPClassifier()
        {
            _network = new ActivationNetwork(
                new SigmoidFunction(2), // Альфа-параметр для SigmoidFunction
                InputSize,
                HiddenLayer1Size,
                HiddenLayer2Size,
                OutputSize
            );
            new NguyenWidrow(_network).Randomize();
        }

        public double Predict(Mat patch)
        {
            if (patch.Rows != PatchSize || patch.Cols != PatchSize || patch.Channels() != 1)
            {
                throw new ArgumentException($"Патч для MLP должен быть размером {PatchSize}x{PatchSize} и одноканальным (grayscale).");
            }
```

```
            if (!_isTrained) return 0.0; // Или выбросить
исключение, если предсказание без обучения недопустимо

            double[] inputVector = new double[InputSize];
            int k = 0;
            for (int i = 0; i < patch.Rows; i++)
            {
                for (int j = 0; j < patch.Cols; j++)
                {
                    inputVector[k++] = patch.Get<byte>(i,
j) / 255.0;
                }
            }
            double[]                output                =
_network.Compute(inputVector);
            return output[0];
        }

        public    void    Train(List<double[]>    inputs,
List<double[]> expectedOutputs, int epochs = 1000, double
learningRate = 0.1, Action<string> logCallback = null)
        {
            if (inputs == null || expectedOutputs == null
||    inputs.Count    ==    0    ||    inputs.Count    !=
expectedOutputs.Count)
            {
                logCallback?.Invoke("Ошибка:  Данные  для
обучения некорректны или отсутствуют.");
                return;
            }

            // Убедимся, что данные в правильном формате
для BackPropagationLearning
            double[][] inputArray = inputs.ToArray();
            double[][]            outputArray           =
expectedOutputs.ToArray();

            var           teacher           =           new
BackPropagationLearning(_network)
            {
                LearningRate = learningRate,
                Momentum = 0.5 // Стандартное значение,
можно настроить
            };
```

```
            logCallback?.Invoke($"Начало обучения MLP: {epochs} эпох, скорость обучения {learningRate:F4}, архитектура: {InputSize}-{HiddenLayer1Size}-{HiddenLayer2Size}-{OutputSize} (все слои Sigmoid)");

            for (int i = 0; i < epochs; i++)
            {
                // RunEpoch ожидает массивы массивов
                double error = teacher.RunEpoch(inputArray, outputArray) / inputs.Count; // Делим на количество выборок для получения средней ошибки
                if (((i + 1) % 50 == 0) || (i == 0)) // Логируем каждые 50 эпох и первую эпоху
                {
                    logCallback?.Invoke($"Эпоха {i + 1}/{epochs}, Средняя ошибка: {error:F6}");
                }
                if (error < 0.005) // Условие ранней остановки
                {
                    logCallback?.Invoke($"Обучение остановлено на эпохе {i + 1} из-за достижения низкой ошибки ({error:F6}).");
                    break;
                }
            }
            _isTrained = true;
            logCallback?.Invoke("Обучение MLP завершено.");
        }

        public bool SaveWeights(string filePath)
        {
            try
            {
                _network.Save(filePath); // Метод Save доступен для ActivationNetwork
                return true;
            }
            catch (Exception ex)
            {
                Console.WriteLine($"[PlateMLPClassifier] Ошибка сохранения весов: {ex.Message}");
```

```
                return false;
            }
        }

        public bool LoadWeights(string filePath)
        {
            try
            {
                if (File.Exists(filePath))
                {
                    // Network.Load является статическим
методом
                    var        baseLoadedNetwork        =
Accord.Neuro.Network.Load(filePath);

                    if       (baseLoadedNetwork       is
ActivationNetwork loadedActivationNetwork)
                    {
                        // Проверяем общую структуру сети
и количество выходов каждого слоя
                        // Приводим слои к ActivationLayer
для доступа к OutputsCount
                        bool    architectureMatches    =
loadedActivationNetwork.InputsCount == InputSize &&

loadedActivationNetwork.Layers.Length == 3;

                        if (architectureMatches)
                        {
                            //    Проверка    количества
выходов для каждого слоя
                            var          layer1         =
loadedActivationNetwork.Layers[0] as ActivationLayer;
                            var          layer2         =
loadedActivationNetwork.Layers[1] as ActivationLayer;
                            var          layer3         =
loadedActivationNetwork.Layers[2] as ActivationLayer; //
Это выходной слой

                            if   (layer1   !=   null   &&
layer1.Neurons.Length   ==   HiddenLayer1Size   &&   //
Neurons.Length     эквивалентно     OutputsCount     для
ActivationLayer
```

```
                                layer2    !=    null    &&
layer2.Neurons.Length == HiddenLayer2Size &&
                                layer3    !=    null    &&
layer3.Neurons.Length == OutputSize)
                            {
                                _network                =
loadedActivationNetwork;
                                _isTrained = true;

Console.WriteLine("[PlateMLPClassifier]   Веса   успешно
загружены и архитектура совместима.");
                                return true;
                            }
                            else
                            {

Console.WriteLine("[PlateMLPClassifier] Ошибка загрузки
весов: архитектура слоев несовместима.");
                                if (layer1 == null ||
layer1.Neurons.Length        !=        HiddenLayer1Size)
Console.WriteLine($"  -  Layer  0  Outputs:  ожидалось
{HiddenLayer1Size}, найдено {(layer1?.Neurons.Length ?? -
1)} (или слой не ActivationLayer)");
                                if (layer2 == null ||
layer2.Neurons.Length        !=        HiddenLayer2Size)
Console.WriteLine($"  -  Layer  1  Outputs:  ожидалось
{HiddenLayer2Size}, найдено {(layer2?.Neurons.Length ?? -
1)} (или слой не ActivationLayer)");
                                if (layer3 == null ||
layer3.Neurons.Length != OutputSize) Console.WriteLine($"
-  Layer  2  Outputs:  ожидалось  {OutputSize},  найдено
{(layer3?.Neurons.Length   ??   -1)}   (или   слой   не
ActivationLayer)");
                                _isTrained = false;
                                return false;
                            }
                        }
                        else
                        {

Console.WriteLine("[PlateMLPClassifier] Ошибка загрузки
весов:  базовая  архитектура  сети  (InputsCount  или
Layers.Length) несовместима.");
```

```
                            if
(loadedActivationNetwork.InputsCount    !=    InputSize)
Console.WriteLine($" - InputsCount: ожидалось {InputSize},
найдено {loadedActivationNetwork.InputsCount}");
                            if
(loadedActivationNetwork.Layers.Length       !=       3)
Console.WriteLine($" - Layers.Length: ожидалось 3, найдено
{loadedActivationNetwork.Layers.Length}");
                            _isTrained = false;
                            return false;
                        }
                    }
                    else
                    {

Console.WriteLine($"[PlateMLPClassifier] Ошибка загрузки
весов: загруженный файл не является ActivationNetwork. Тип
загруженной                                          сети:
{baseLoadedNetwork?.GetType().FullName}");
                        _isTrained = false;
                        return false;
                    }
                }
                else
                {

Console.WriteLine($"[PlateMLPClassifier]  Файл  весов  не
найден: {filePath}");
                    _isTrained = false;
                    return false;
                }
            }
            catch (Exception ex)
            {
                Console.WriteLine($"[PlateMLPClassifier]
Исключение при загрузке весов: {ex.Message}");
                _isTrained = false;
                return false;
            }
        }
    }
}
```