Министерство науки и образования Российской Федерации

ФГБОУ ВО «Пермский Национальный Исследовательский Политехнический Университет»

Факультет прикладной математики и механики

Лабораторные работы   
по дисциплине

Интеллектуальные информационные системы и технологии

Выполнил: студент группы ИСТ-17-1бз

Лодейщикова Яна Ильдусовна

Проверил: старший преподаватель

Истомин Денис Андреевич

Пермь, 2023

**Оглавление**

[Лабораторная работа №4 3](#_Toc93589800)

[Постановка задачи 3](#_Toc93589801)

[Предметная область 3](#_Toc93589802)

[Структура программы 3](#_Toc93589803)

[Пример работы 4](#_Toc93589804)

[Код программы 5](#_Toc93589805)

[Файл Systems.cs 5](#_Toc93589806)

[Файл MainForm.cs 13](#_Toc93589807)

[Лабораторная работа №5 18](#_Toc93589808)

[Постановка задачи 18](#_Toc93589809)

[Предметная область 18](#_Toc93589810)

[Структура программы 18](#_Toc93589811)

[Пример работы 19](#_Toc93589812)

[Код программы 20](#_Toc93589813)

[Файл Network.cs 20](#_Toc93589814)

[Файл Program.cs 23](#_Toc93589815)

[Файл MainForm.cs 27](#_Toc93589816)

# Лабораторная работа №4

## Постановка задачи

В рамках данной работы необходимо реализовать систему, включающую следующие модели принятия коллективных решений:

1. относительного большинства;
2. модель Кондорсе:
   1. явный победитель,
   2. правило Копленда,
   3. правило Симпсона;
3. модель Борда;

## Предметная область

Тема, выбранная для системы – выбор наушников.

## Структура программы

Программа состоит из двух основных частей – интерфейса и логики реализации различных моделей принятия коллективного решения.

Код интерфейса содержит методы обработки операций изменения количества выборщиков (добавления и удаления голосов и списков предпочтений) и подсчета голосов (вывод результата коллективного решения при выбранной модели). Графические элементы интерфейса предоставляют возможность выбора кандидата из выпадающего списка и переключения используемой модели.

Все классы, реализующие разные модели принятия коллективного решения, содержат методы для определения победителя по предоставленному списку голосов и вывода объяснения полученного результата в текстовом виде, а также вспомогательные методы реализации конкретной системы (например, подсчет очков количества менее предпочтительных кандидатов в модели Борда).

## Пример работы

Пример работы системы для относительного большинства:

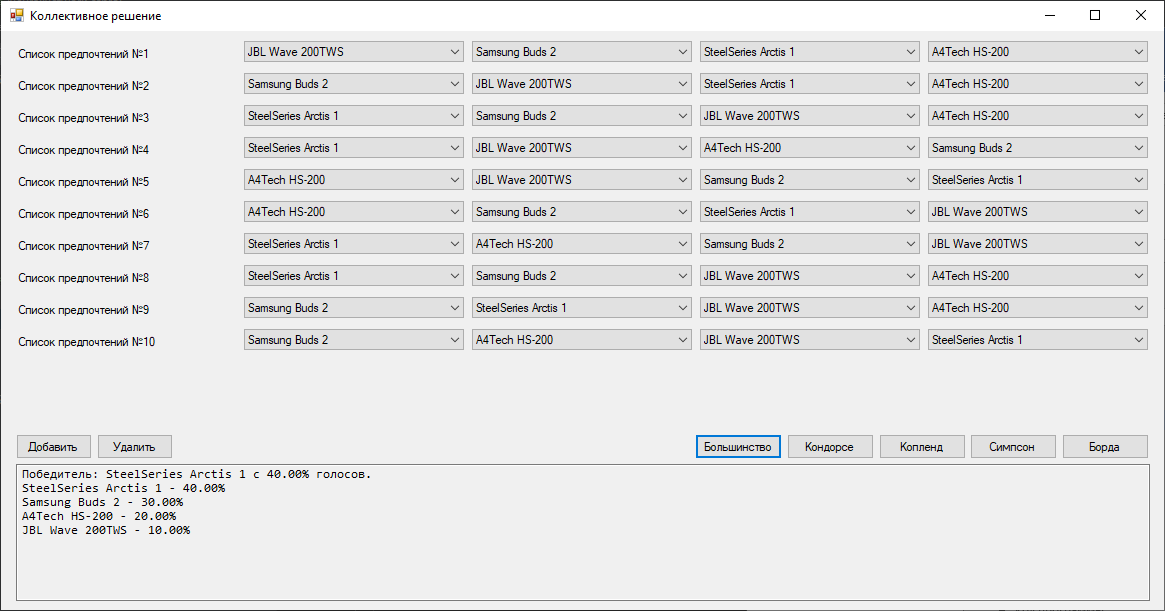


Рис. 1 Относительное большинство

Пример работы системы для модели Борда:

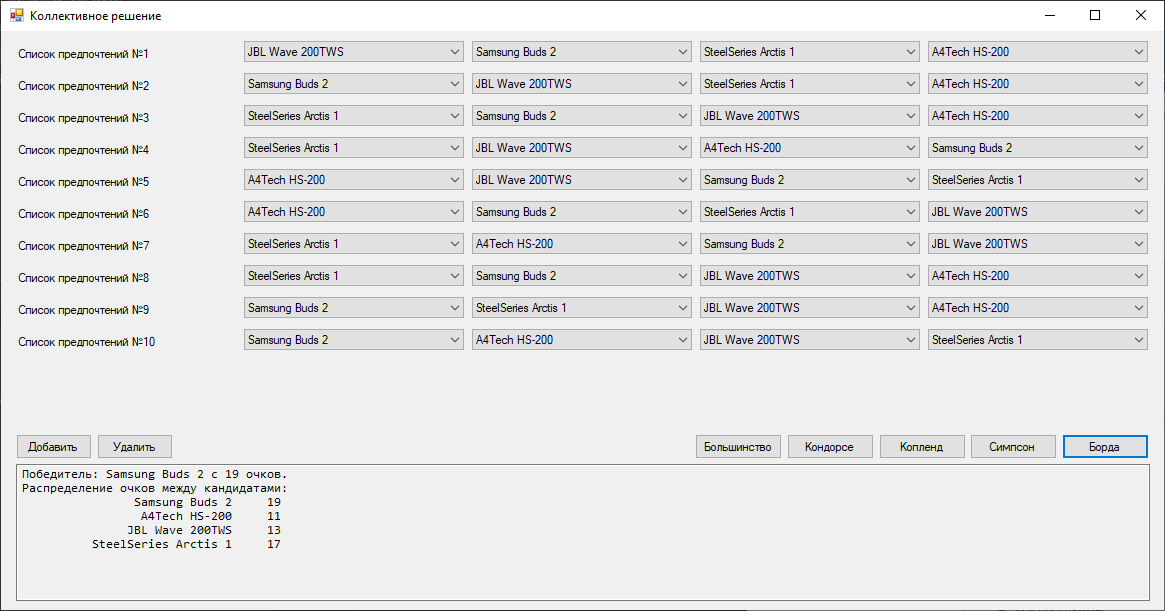


Рис. 2 Модель Борда

## Код программы

### Файл Form1.cs

using System;

using System.Collections.Generic;

using System.Drawing;

using System.Linq;

using System.Windows.Forms;

namespace CollectiveDecision

{

public partial class Form1 : Form

{

private static readonly int ControlsMarginPx = 8;

private static readonly (int Width, int Height) ControlsSizePx = (220, 24);

private static readonly string[] Candidates =

{

"JBL Wave 200TWS",

"Samsung Buds 2",

"SteelSeries Arctis 1",

"A4Tech HS-200",

};

private Stack<(Label Label, ComboBox[] ComboBoxes)> VoteOrderControls;

public Form1()

{

InitializeComponent();

VoteOrderControls = new Stack<(Label Label, ComboBox[] ComboBoxes)>();

}

// При нажатии на кнопку "Добавить" появляются элементы управления, позволяющие

// очередному выборщику заполнить список предпочтений

private void buttonAddVotes\_Click(object sender, EventArgs e)

{

var verticalLocation = (ControlsSizePx.Height + ControlsMarginPx) \* VoteOrderControls.Count;

var label = CreateLabel(

new Point(0, verticalLocation),

$"Список предпочтений №{VoteOrderControls.Count + 1}"

);

var comboBoxes = Candidates

.Select(

(candidate, i) => CreateComboBox(

new Point((ControlsMarginPx + ControlsSizePx.Width) \* (i + 1), verticalLocation),

candidate

)

)

.ToArray();

VoteOrderControls.Push((label, comboBoxes));

panel.SuspendLayout();

panel.Controls.Add(label);

panel.Controls.AddRange(comboBoxes);

panel.ResumeLayout();

}

private Label CreateLabel(Point location, string text)

{

var label = new Label();

label.Location = location;

label.Size = new Size(ControlsSizePx.Width, ControlsSizePx.Height);

label.Text = text;

label.TextAlign = ContentAlignment.MiddleLeft;

return label;

}

private ComboBox CreateComboBox(Point location, string candidate)

{

var comboBox = new ComboBox();

comboBox.DropDownStyle = ComboBoxStyle.DropDownList;

comboBox.Location = location;

comboBox.Size = new Size(ControlsSizePx.Width, ControlsSizePx.Height);

comboBox.Items.AddRange(Candidates);

comboBox.SelectedItem = candidate;

return comboBox;

}

// При нажатии кнопку "Удалить" уничтожаются элементы управления для последнего выборщика

private void buttonRemoveVotes\_Click(object sender, EventArgs e)

{

if (VoteOrderControls.Count > 0)

{

var lastVoteOrderControls = VoteOrderControls.Pop();

panel.Controls.Remove(lastVoteOrderControls.Label);

foreach (var comboBox in lastVoteOrderControls.ComboBoxes)

{

panel.Controls.Remove(comboBox);

}

}

}

// При нажатии на кнопку подсчета в текстовое поле выводится результат

// коллективного решения, полученный при помощи выбранной модели

// (либо ошибки, возникшие при ее работе)

private void buttonPlurality\_Click(object sender, EventArgs e)

{

string[] values = VoteOrderControls.Select(controls => (string) controls.ComboBoxes[0].SelectedItem).ToArray();

var system = new PluralitySystem(values);

textBox.Lines = system.GetDecision().Split('\n');

}

private void buttonCondorcet\_Click(object sender, EventArgs e)

{

var system = new CondorcetSystem(VoteOrders);

textBox.Lines = system.GetDecision().Split('\n');

}

private void buttonCopeland\_Click(object sender, EventArgs e)

{

var system = new CopelandSystem(VoteOrders);

textBox.Lines = system.GetDecision().Split('\n');

}

private void buttonSimpson\_Click(object sender, EventArgs e)

{

var system = new SimpsonSystem(VoteOrders);

textBox.Lines = system.GetDecision().Split('\n');

}

private void buttonBorda\_Click(object sender, EventArgs e)

{

var system = new BordaSystem(VoteOrders);

textBox.Lines = system.GetDecision().Split('\n');

}

private string[][] VoteOrders =>

VoteOrderControls.Select(

controls => controls.ComboBoxes.Select(

comboBox => (string)comboBox.SelectedItem

).ToArray()

).ToArray();

}

}

### Файл PluralitySystem.cs

using System.Collections.Generic;

using System.Linq;

namespace CollectiveDecision

{

// Система принятия коллективного решения по принципу относительного большинства

class PluralitySystem

{

private IEnumerable<string> Votes;

// Проценты всех кандидатов. Ключ - кандидат, значение - процент

private Dictionary<string, double> Percentages;

public PluralitySystem(IEnumerable<string> votes)

{

Votes = votes;

Percentages = GetPercentages();

}

// Подсчет процентов голосов для каждого кандидата

private Dictionary<string, double> GetPercentages()

{

return Votes

.GroupBy(\_ => \_)

.OrderByDescending(votesGroup => votesGroup.Count())

.ToDictionary(

votesGroup => votesGroup.Key,

votesGroup => (double) votesGroup.Count() / Votes.Count()

);

}

// Вывод победителя и процентов голосов для каждого кандидата

public string GetDecision()

{

var winnerPercentage = Percentages.First();

var winner = HasSingleWinner()

? $"Победитель: {winnerPercentage.Key} с {winnerPercentage.Value:P} голосов."

: "Невозможно определить победителя";

return $"{winner}\n" +

string.Join("\n", Percentages.Select(percentage => $"{percentage.Key} - {percentage.Value:P}"));

}

// Проверка наличия победителя (Нет двух кандидатов, имеющих процент голосов, равный максимальному)

private bool HasSingleWinner()

{

if (Percentages.Count <= 1) return true;

var first = Percentages.First();

var second = Percentages.Skip(1).First();

return first.Value != second.Value;

}

}

}

### Файл VoteOrderSystem.cs

using System.Collections.Generic;

using System.Linq;

namespace CollectiveDecision

{

// Абстрактная система коллективного решения на основе упорядоченного списка предпочтений

public abstract class VoteOrderSystem

{

protected IEnumerable<IEnumerable<string>> VoteOrders;

protected IEnumerable<string> Candidates;

protected VoteOrderSystem(IEnumerable<IEnumerable<string>> voteOrders)

{

VoteOrders = voteOrders;

Candidates = GetUniqueCandidates();

}

// Определение полного списка кандидатов по спискам предпочтений

private IEnumerable<string> GetUniqueCandidates() => VoteOrders.SelectMany(\_ => \_).Distinct();

}

}

### Файл CondorcetSystem.cs

using System.Collections.Generic;

using System.Linq;

namespace CollectiveDecision

{

// Система принятия коллективного решения по модели Кондорсе

public class CondorcetSystem : VoteOrderSystem

{

protected Dictionary<string, Dictionary<string, int>> CandidatePairValues;

public CondorcetSystem(IEnumerable<IEnumerable<string>> voteOrders) : base(voteOrders)

{

CandidatePairValues = GetPairValues();

}

// Вычисление попарных предпочтений: для каждой (упорядоченной) пары вычисляется,

// как часто первый элемент оказывается более предпочтительным, чем второй

private Dictionary<string, Dictionary<string, int>> GetPairValues()

{

return Candidates

.ToDictionary(

firstCandidate => firstCandidate,

firstCandidate => Candidates

.Where(secondCandidate => firstCandidate != secondCandidate)

.ToDictionary(

secondCandidate => secondCandidate,

secondCandidate => GetPairValue(firstCandidate, secondCandidate)

)

);

}

protected virtual int GetPairValue(string first, string second)

{

var (timesFirstHigher, timesSecondHigher) = GetPairDifference(first, second);

return timesFirstHigher - timesSecondHigher;

}

// Вычисление количества случаев, когда первый кандидат оказывается предпочтительнее второго, и наоборот

protected (int timesFirstHigher, int timesSecondHigher) GetPairDifference(string first, string second)

{

int timesFirstHigher = 0;

int timesSecondHigher = 0;

foreach (var voteOrder in VoteOrders)

{

var voteOrderList = voteOrder.ToList();

int indexFirst = voteOrderList.IndexOf(first);

int indexSecond = voteOrderList.IndexOf(second);

if (indexFirst < indexSecond)

{

timesFirstHigher++;

}

else if (indexFirst > indexSecond)

{

timesSecondHigher++;

}

}

return (timesFirstHigher, timesSecondHigher);

}

// Вывод результата коллективного решения

public string GetDecision()

{

var winners = Candidates.Where(Wins).ToArray();

var winner = winners.Length == 1

? $"Победитель: {winners.First()}, т.к. большинство выборщиков предпочитает именно этого кандидата при попарном сравнении с остальными."

: "Невозможно определить победителя";

return

$"{winner}\n" +

"Матрица попарного предпочтения:\n" +

string.Join("\n", Candidates.Select(GetMatrixLine));

}

protected virtual string GetMatrixLine(string firstCandidate)

{

return $"{firstCandidate,30} " +

string.Join(" ", Candidates.Select(secondCandidate => GetMatrixPairValue(firstCandidate, secondCandidate)));

}

private string GetMatrixPairValue(string firstCandidate, string secondCandidate)

{

int value = firstCandidate == secondCandidate

? 0

: CandidatePairValues[firstCandidate][secondCandidate];

return value.ToString().PadLeft(5);

}

// Условие победы: кандидат оказывается предпочтительнее любого другого кандидата в большинстве случаев

protected virtual bool Wins(string candidate) => CandidatePairValues[candidate].All(pairDifference => pairDifference.Value > 0);

}

}

### Файл CopelandSystem.cs

using System.Collections.Generic;

using System.Linq;

namespace CollectiveDecision

{

// Система принятия коллективного решения по модели Кондорсе с модификацией по правилу Копленда

class CopelandSystem : CondorcetSystem

{

private Dictionary<string, int> CandidatesPoints;

// Вычисление оценки Копленда: при сравнении в каждой паре кандидату начисляется 1 очко,

// если он предпочтительнее для большинства выборщиков, и вычитается 1 очко, если для меньшинства

public CopelandSystem(IEnumerable<IEnumerable<string>> voteOrders) : base(voteOrders)

{

CandidatesPoints = CandidatePairValues

.Select(candidate => new

{

Candidate = candidate.Key,

Points = candidate.Value.Aggregate(

0,

(i, pairDifference) =>

pairDifference.Value < 0 ? i - 1 :

pairDifference.Value > 0 ? i + 1 :

i

),

})

.ToDictionary(

candidatePoints => candidatePoints.Candidate,

candidatePoints => candidatePoints.Points

);

}

// Побеждает кандидат с максимальным значением оценки Копленда

protected override bool Wins(string candidate) => CandidatesPoints[candidate] == MaximumPoints;

private int MaximumPoints => CandidatesPoints

.Select(candidatePoints => candidatePoints.Value)

.Max();

protected override string GetMatrixLine(string firstCandidate)

{

return base.GetMatrixLine(firstCandidate) + $" Оценка Копленда: {CandidatesPoints[firstCandidate],5}";

}

}

}

### Файл SimpsonSystem.cs

using System.Collections.Generic;

using System.Linq;

namespace CollectiveDecision

{

// Система принятия коллективного решения по модели Кондорсе с модификацией по правилу Симпсона

class SimpsonSystem : CondorcetSystem

{

private Dictionary<string, int> CandidatesPoints;

// Вычисление оценки Симпсона для каждого кандидата: в каждой паре кандидатов подсчитывается количество выборщиков,

// считающих данного кандидата предпочтительнее, из всех таких чисел выбирается минимальное

public SimpsonSystem(IEnumerable<IEnumerable<string>> voteOrders) : base(voteOrders)

{

CandidatesPoints = CandidatePairValues

.Select(candidate => new

{

Candidate = candidate.Key,

Points = candidate.Value.Values.Min(),

})

.ToDictionary(

candidatePoints => candidatePoints.Candidate,

candidatePoints => candidatePoints.Points

);

}

protected override int GetPairValue(string first, string second) => GetPairDifference(first, second).timesFirstHigher;

// Побеждает кандидат с максимальным значением оценки Симпсона

protected override bool Wins(string candidate) => CandidatesPoints[candidate] == MaximumPoints;

private int MaximumPoints => CandidatesPoints

.Select(candidatePoints => candidatePoints.Value)

.Max();

protected override string GetMatrixLine(string firstCandidate)

{

return base.GetMatrixLine(firstCandidate) + $" Оценка Симпсона: {CandidatesPoints[firstCandidate],5}";

}

}

}

### Файл BordaSystem.cs

using System.Collections.Generic;

using System.Linq;

namespace CollectiveDecision

{

// Система принятия коллективного решения по модели Борда

class BordaSystem : VoteOrderSystem

{

private Dictionary<string, int> Points;

public BordaSystem(IEnumerable<IEnumerable<string>> voteOrders) : base(voteOrders)

{

Points = GetPoints();

}

// Вычисление очков для каждого кандидата: в каждом списке предпочтительности кандидату дается столько

// очков, сколько кандидатов оказываются менее предпочтительными для данного выборщика

private Dictionary<string,int> GetPoints()

{

return Candidates

.ToDictionary(

candidate => candidate,

GetPoints

);

}

private int GetPoints(string candidate)

{

int points = 0;

foreach (var voteOrder in VoteOrders)

{

int index = voteOrder.ToList().IndexOf(candidate);

points += Candidates.Count() - index - 1;

}

return points;

}

// Вывод результата коллективного решения

public string GetDecision()

{

var winners = Candidates.Where(Wins).ToArray();

var winner = winners.Length == 1

? $"Победитель: {winners.First()} с {Points[winners.First()]} очков."

: "Невозможно определить победителя";

return

$"{winner}\n" +

"Распределение очков между кандидатами:\n" +

string.Join("\n", Candidates.Select(candidate => $"{candidate,30} {Points[candidate],5}"));

}

// Побеждает кандидат с максимальным количеством очков

protected virtual bool Wins(string candidate) => Points[candidate] == Points.Values.Max();

}

}

# Лабораторная работа №5

## Постановка задачи

В рамках данной работы необходимо реализовать приложение, решающее задачу распознавания образов с использованием искусственных нейронных сетей. Возможный вариант реализации – многослойный персептрон с методом обратного распространения ошибки, где каждый входной сигнал соответствует пикселю образа.

## Предметная область

Тема, выбранная для системы – распознавание римских цифр от I до X.

## Структура программы

Программа состоит из двух основных частей – интерфейса и универсальной искусственной нейронной сети (с числовыми значениями на входе и выходе).

Нейросеть при создании определяется количеством нейронов в различных слоях и коэффициентом скорости обучения. Она предоставляет две функции – опрос сети по заданным входным значениям с получением результирующих выходных значений, и обучение нейросети на паре «входные значения – целевые выходные значения» с реализацией обратного распространения ошибки и обновления весов связей между нейронами при помощи метода градиентного спуска.

В интерфейсе реализован функционал загрузки тренировочных файлов-изображений для обучения сети, рисования символов в специальном поле с возможностью сохранить в файл, а также вызова функций обучения сети и распознавания нарисованного символа.

Логика основной части программы заключается в конвертации изображений в соответствующие сигналы для входных нейронов сети и реализации многократного обучения нейросети на полном наборе примеров.

## Пример работы

Пример распознавания цифры «X», нарисованной пользователем в специальном поле, после обучения системы:

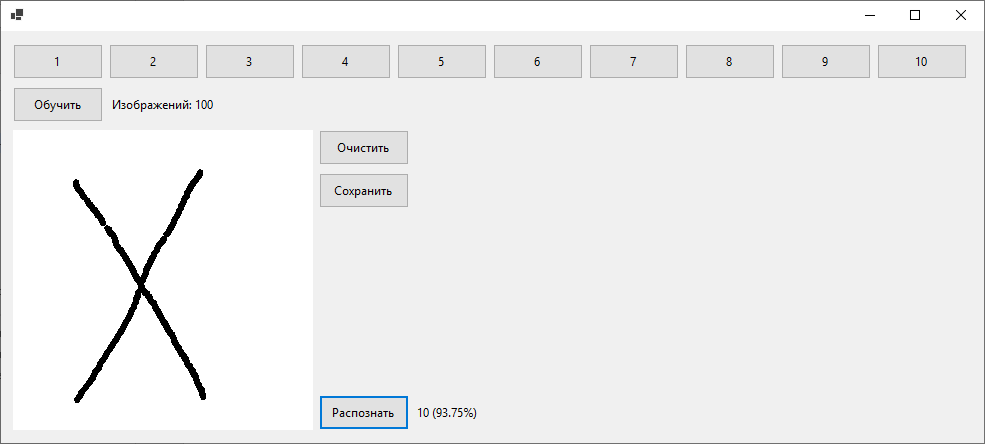


Рис. 3 Пример распознавания символа

## Код программы

### Файл Form1.cs

using System;

using System.Collections.Generic;

using System.Drawing;

using System.Drawing.Imaging;

using System.Linq;

using System.Windows.Forms;

namespace NeuralNetwork

{

public partial class Form1 : Form

{

// Количество входных нейронов в одной стороне квадратной сетки входного слоя

private const int BlocksOneSide = 10;

private Network Network;

private readonly Dictionary<string, IEnumerable<Image>> SymbolExampleImages = new Dictionary<string, IEnumerable<Image>>();

public Form1()

{

InitializeComponent();

ResetImage();

}

// Нажатие на кнопку добавления тренировочных изображений для символа - открывает окно выбора файлов-изображений

private void button1\_Click(object sender, EventArgs e) => AddExamples("1");

private void button2\_Click(object sender, EventArgs e) => AddExamples("2");

private void button3\_Click(object sender, EventArgs e) => AddExamples("3");

private void button4\_Click(object sender, EventArgs e) => AddExamples("4");

private void button5\_Click(object sender, EventArgs e) => AddExamples("5");

private void button6\_Click(object sender, EventArgs e) => AddExamples("6");

private void button7\_Click(object sender, EventArgs e) => AddExamples("7");

private void button8\_Click(object sender, EventArgs e) => AddExamples("8");

private void button9\_Click(object sender, EventArgs e) => AddExamples("9");

private void button10\_Click(object sender, EventArgs e) => AddExamples("10");

private void AddExamples(string symbol)

{

var dialog = new OpenFileDialog {Multiselect = true, Filter = "Bitmap Image|\*.bmp"};

if (dialog.ShowDialog() != DialogResult.OK) return;

var bitmapFilePaths = dialog.FileNames;

var bitmaps = bitmapFilePaths.Select(Image.FromFile).ToArray();

if (SymbolExampleImages.ContainsKey(symbol)) SymbolExampleImages.Remove(symbol);

SymbolExampleImages.Add(symbol, bitmaps);

labelImages.Text = $"Изображений: {SymbolExampleImages.SelectMany(example => example.Value).Count()}";

}

// Рисование символа мышкой

private void pictureBox\_Draw(object sender, MouseEventArgs mouseEvent)

{

if (mouseEvent.Button != MouseButtons.Left) return;

using (var image = Graphics.FromImage(pictureBox.Image))

{

image.DrawEllipse(new Pen(Color.Black, 3), mouseEvent.Location.X - 2, mouseEvent.Location.Y - 2, 3, 3);

pictureBox.Invalidate();

}

}

private void buttonClear\_Click(object sender, EventArgs e)

{

ResetImage();

labelResult.Text = null;

}

// Очистка поля рисования символа

private void ResetImage()

{

pictureBox.Image = new Bitmap(pictureBox.Width, pictureBox.Height);

using (var image = Graphics.FromImage(pictureBox.Image))

{

image.Clear(Color.White);

pictureBox.Invalidate();

}

}

// Сохранение нарисованного символа в файл

private void buttonSave\_Click(object sender, EventArgs e)

{

var dialog = new SaveFileDialog {OverwritePrompt = true, Filter = "Bitmap Image|\*.bmp"};

if (dialog.ShowDialog() != DialogResult.OK) return;

var file = dialog.OpenFile();

pictureBox.Image.Save(file, ImageFormat.Bmp);

file.Close();

}

// Распознавание нарисованного символа нейросетью

private void buttonTest\_Click(object sender, EventArgs e)

{

var symbols = SymbolExampleImages.Keys.ToArray();

var resultSymbolPercentages = Network.Query(ConvertImageToNetworkInputs(pictureBox.Image));

var resultSymbolIndex = Array.IndexOf(resultSymbolPercentages, resultSymbolPercentages.Max());

labelResult.Text = $"{symbols[resultSymbolIndex]} ({resultSymbolPercentages[resultSymbolIndex]:P})";

}

// Запуск обучения нейросети

private void buttonTrain\_Click(object sender, EventArgs e)

{

Network = new Network(BlocksOneSide \* BlocksOneSide, 50, SymbolExampleImages.Count, 0.3);

// 10 "эпох" повторения циклов обучения сети

for (int i = 0; i < 10; i++)

{

Console.WriteLine($"Обучение сети, эпоха №{i + 1}");

// Перебор всех изображений-примеров для всех распознаваемых символов

for (var j = 0; j < SymbolExampleImages.Count; j++)

{

var symbol = SymbolExampleImages.Keys.ElementAt(j);

Console.WriteLine($"Символ {symbol}");

foreach (var image in SymbolExampleImages[symbol])

{

var inputs = ConvertImageToNetworkInputs(image);

// Заполнение целевых значений: желаемое значение корректного символа - 0.99, остальные - 0.01

var targetOutputs = new double[SymbolExampleImages.Count]

.Select(\_ => 0.01)

.ToArray();

targetOutputs[j] = 0.99;

Network.Train(inputs, targetOutputs);

}

}

}

}

// Преобразование изображения (с большим разрешением в пикселях) в массив входных значений сети 10x10

private static double[] ConvertImageToNetworkInputs(Image image)

{

var bitmap = new Bitmap(image);

if (bitmap.Width % BlocksOneSide != 0 || bitmap.Height % BlocksOneSide != 0) throw new InvalidOperationException($"Изображение не раскладывается на сетку {BlocksOneSide}x{BlocksOneSide}");

// Изображение делится на равные блоки, каждый из которых соответствует одному входному нейрону

var inputBlocks = new double[BlocksOneSide, BlocksOneSide];

for (var blockX = 0; blockX < BlocksOneSide; blockX++)

{

for (var blockY = 0; blockY < BlocksOneSide; blockY++)

{

// Определение координат границ блока на изображении

int blockWidth = bitmap.Width / BlocksOneSide;

int blockHeight = bitmap.Height / BlocksOneSide;

int leftMargin = blockWidth \* blockX;

int topMargin = blockHeight \* blockY;

// Заполнение блока пикселями изображения

var blockPixels = new Color[blockWidth \* blockHeight];

for (var x = 0; x < blockWidth; x++)

{

for (var y = 0; y < blockHeight; y++)

{

blockPixels[y \* blockHeight + x] = bitmap.GetPixel(leftMargin + x, topMargin + y);

}

}

// Подсчет количества черных пикселей в блоке

var percentBlackPixelsInBlock =

(double)blockPixels.Count(pixel => pixel.GetBrightness() == 0)

/ blockPixels.Length;

var value = percentBlackPixelsInBlock \* 8;

if (value > 1) value = 1;

// Преобразование количества черных пикселей в результирующее входное значение нейрона

inputBlocks[blockX, blockY] = value \* 0.99 + 0.01;

}

}

// Преобразование двумерного массива блоков изображения в одномерный массив входных значений нейрона

var inputs = new double[BlocksOneSide \* BlocksOneSide];

for (var blockX = 0; blockX < BlocksOneSide; blockX++)

{

for (var blockY = 0; blockY < BlocksOneSide; blockY++)

{

inputs[blockY \* BlocksOneSide + blockX] = inputBlocks[blockX, blockY];

}

}

return inputs;

}

}

}

### Файл Network.cs

using System;

using MathNet.Numerics.LinearAlgebra;

namespace NeuralNetwork

{

// Нейронная сеть

public class Network

{

// Количество нейронов во входном, скрытом и выходном слоях сети

private readonly int InputNodes;

private readonly int HiddenNodes;

private readonly int OutputNodes;

// Коэффициент обучения

private readonly double LearningRate;

// Весовые коэффициенты связей между нейронами во входном и скрытом слое, скрытом и выходном

private Matrix<double> WeightsInputHidden;

private Matrix<double> WeightsHiddenOutput;

// Функция активации нейрона - сигмоида

private readonly Func<double, double> ActivationFunction = x => 1 / (1 + Math.Pow(Math.E, -x));

public Network(int inputNodes, int hiddenNodes, int outputNodes, double learningRate)

{

InputNodes = inputNodes;

HiddenNodes = hiddenNodes;

OutputNodes = outputNodes;

LearningRate = learningRate;

// Инициализация весов случайными значениями в диапазоне [-0,5; 0,5)

var random = new Random();

WeightsInputHidden = Matrix<double>.Build.Dense(HiddenNodes, InputNodes, (x, y) => random.NextDouble()) - 0.5;

WeightsHiddenOutput = Matrix<double>.Build.Dense(OutputNodes, HiddenNodes, (x, y) => random.NextDouble()) - 0.5;

}

// Обучение нейронной сети на одном примере эталонных выходных значений

public void Train(double[] networkInputs, double[] networkOutputTargets)

{

// Расчет фактических выходных значений сети, такой же как в методе Query

var inputLayerOutputs = Matrix<double>.Build.DenseOfColumnArrays(networkInputs);

var hiddenLayerInputs = WeightsInputHidden \* inputLayerOutputs;

var hiddenLayerOutputs = hiddenLayerInputs.Map(ActivationFunction);

var outputLayerInputs = WeightsHiddenOutput \* hiddenLayerOutputs;

var outputLayerOutputs = outputLayerInputs.Map(ActivationFunction);

var outputLayerOutputTargets = Matrix<double>.Build.DenseOfColumnArrays(networkOutputTargets);

// Ошибки выходного слоя - разница между целевым значением и действительным

var outputLayerErrors = outputLayerOutputTargets - outputLayerOutputs;

// Обратное распространение ошибки

// Ошибки скрытого слоя - ошибки выходного слоя, распределенные пропорционально весам связей между соответствующими нейронами

var hiddenLayerErrors = WeightsHiddenOutput.Transpose() \* outputLayerErrors;

// Обновление весов связей между нейронами скрытого и выходного слоя - минимизация ошибки методом градиентного спуска

WeightsHiddenOutput += LearningRate

\* outputLayerErrors.PointwiseMultiply(outputLayerOutputs).PointwiseMultiply(1 - outputLayerOutputs)

\* hiddenLayerOutputs.Transpose();

WeightsInputHidden += LearningRate

\* hiddenLayerErrors.PointwiseMultiply(hiddenLayerOutputs).PointwiseMultiply(1 - hiddenLayerOutputs)

\* inputLayerOutputs.Transpose();

}

// Опрос нейронной сети по заданным входным значениям

public double[] Query(double[] networkInputs)

{

// Преобразование массива входных значений в матрицу из одного столбца

var inputLayerOutputs = Matrix<double>.Build.DenseOfColumnArrays(networkInputs);

// Входные сигналы скрытых нейронов определяются выходными сигналами входных нейронов, домноженными на веса связей

var hiddenLayerInputs = WeightsInputHidden \* inputLayerOutputs;

// Выходные сигналы нейронов - это входные сигналы, преобразованные через функцию активации

var hiddenLayerOutputs = hiddenLayerInputs.Map(ActivationFunction);

// То же самое для выходного слоя

var outputLayerInputs = WeightsHiddenOutput \* hiddenLayerOutputs;

var outputLayerOutputs = outputLayerInputs.Map(ActivationFunction);

// Транспонирование столбца исходящих сигналов нейронов в массив выходных значений сети

return outputLayerOutputs.Column(0).ToArray();

}

}

}