

**KAUNO TECHNOLOGIJOS UNIVERSITETAS**

**INFORMATIKOS FAKULTETAS**

INTELEKTIKOS PAGRINDAI (P176B101)

**PROJEKTO ATASKAITA**

**„Devanagari simbolių atpažinimas“**

Atliko:

Martynas Kaunas IFF-6/14

Deividas Pečiulionis IFB-6

Gediminas Milašius IFF-6/2

Priėmė:

Doc. Germanas Budnikas

**KAUNAS 2019**

**Turinys**

[**1.** **Projekto užduotis** 3](#_Toc8921118)

[**2.** **Duomenys** 3](#_Toc8921119)

[**2.1.** **Duomenų pavyzdžiai** 3](#_Toc8921120)

[**2.2.** **Duomenų valymas** 5](#_Toc8921121)

[**2.3.** **Dimensijų sumažinimas** 6](#_Toc8921122)

[**3.** **Metodas su mokytoju Nr.1 (Simbolio kontūro intensyvumo žemėlapis)** 7](#_Toc8921123)

[**3.2.** **Eksperimentai** 7](#_Toc8921124)

[**4.** **Metodas su mokytoju Nr.2 (Neuroninis tinklas)** 8](#_Toc8921125)

[**4.2.** **Eksperimentai** 8](#_Toc8921126)

[**5.** **Metodų palyginimas** 9](#_Toc8921127)

[**7.** **Programos kodas** 11](#_Toc8921128)

[**7.1.** **Program.cs** 11](#_Toc8921129)

[**7.2.** **NeuralNetwork.cs** 19](#_Toc8921130)

[**7.3.** **Image.cs** 21](#_Toc8921131)

# **Projekto užduotis**

Devanagari – alfabetas, kuris daugiausia naudojamas Indijos subkontinento bei Himalajų kalnų tautų kalbose.

Šio darbo tikslas – sukurti programą, kuri gebėtų atpažinti ir klasifikuoti .png formatu pateiktus ranka parašytų Devanagari simbolių paveikslėlius.

Tikslui pasiekti naudojami mašininio mokymosi metodai.

Programoje siekiama panaudoti keletą duomenų apdorojimo bei daugiau nei vieną skirtingą mašininiu mokymusi paremtą simbolių atpažinimo metodą.

Mašininio mokymosi metodų tikslumo įvertinimui naudojama kelių duomenų sluoksnių (k-fold) kryžminė patikra.

# **Duomenys**

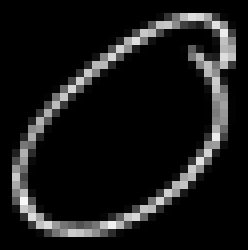
Projekte naudojami .png formato, 32x32 pikselių dydžio, ranka parašytų Devanagari simbolių paveikslėliai. Duomenys gauti iš mašininio mokymosi užduočių duomenų archyvo:

<https://archive.ics.uci.edu/ml/datasets/Devanagari+Handwritten+Character+Dataset>

Devanagari alfabetą viso sudaro 47 skirtingi simboliai. Atsižvelgiant į skaičiavimų trukmę ir galimus skaičiavimo resursų poreikius, projekte apsiribojama 10-mi simbolių, kuriais išreiškiami alfabeto skaitmenys 0-9.

Duomenų rinkinyje yra po 2000 paveikslėlių kiekvienam Devanagari skaitmeniui. Taigi viso – 20000 paveikslėlių.

# **Duomenų pavyzdžiai**

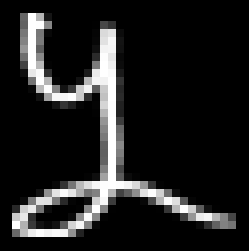
Šiame skyriuje pateikiami pavyzdiniai paveikslėliai kiekvienam iš 10 projekte naudojamų skaitmenų.

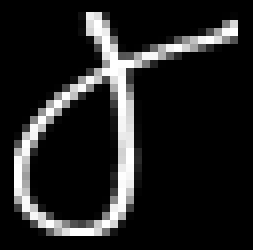
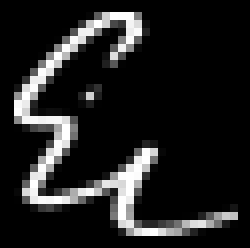
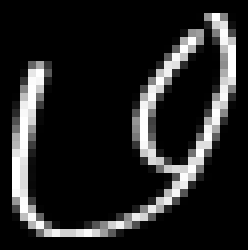
pav. 4 Devanagari "3"

pav. 3 Devanagari "2"

pav. 2 Devanagari "1"

pav. 1 Devanagari "0"



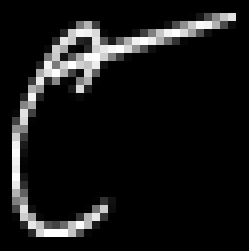
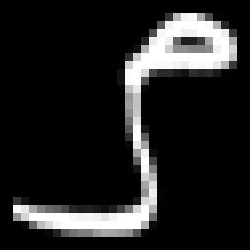


pav. 8 Devanagari "7"

pav. 7Devanagari "6"

pav. 6 Devanagari "5"

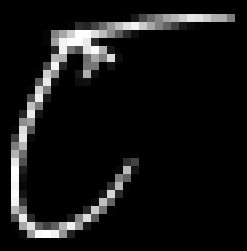
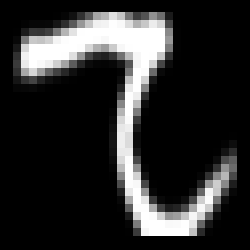
pav. 5 Devanagari "4"



pav. 10 Devanagari "9"

pav. 9 Devanagari "8"

Taip pat verta pastebėti, kad kiekvieno simbolio išvaizda 2000 paveikslėlių rinkiniuose stipriai varijuoja. Pavyzdžiui, nors visi trys pav. 11-13 pavaizduoti simboliai atrodo labai skirtingai, bet reiškia tą patį – „8“.



pav. 13 Devanagari "8" 3

pav. 11 Devanagari "8" 1

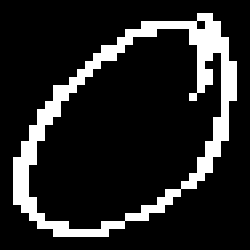
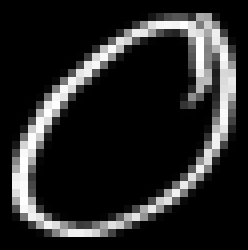
pav. 12 Devanagari "8" 2

# **Duomenų valymas**

Programoje duomenų valymą apima du pagrindiniai metodai:

* **ImageToBlackWhite()** – Gautas duomenų failo paveikslėlis yra apdorojamas taip, kad visi pilkų atspalvių pikseliai būtų paversti juodais arba baltais pagal programoje pasirinktą šviesumo slenkstį.
* **ImageToMatrix()** – Gautas juodai/baltas paveikslėlis konvertuojamas į dvejetainę matricą, kuri naudojama vėlesniuose skaičiavimuose.

Pateikiama šių metodų veikimo schema su pavyzdiniu Devanagari nulį vaizduojančiu paveisklėliu:



**ImageToBlackWhite()**

**ImageToMatrix()**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

# **Dimensijų sumažinimas**

Dimensijų sumažinimas programoje gali būti atliekamas dviem skirtingais būdais:

* **Paveikslėlio sutraukimas** – paveikslėlis performuojamas į pasirinktą dydį. Pavyzdžiui 32x32 -> 16x16
* Naudojant metodą **cutMatrix()** – į matricą konvertuotas paveikslėlis apkarpomas taip, kad jo kraštuose nebeliktų tuščių juodų eilučių ir stulpelių.

# **Metodas su mokytoju Nr.1 (Simbolio kontūro intensyvumo žemėlapis)**

text

* 1. **Metodo veikimas**

text

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 3 | 10 | 41 | 114 | 221 | 316 | 366 | 338 | 286 | 202 | 126 | 46 | 8 | 0 |
| 0 | 3 | 65 | 266 | 576 | 842 | 1058 | 1192 | 1234 | 1114 | 990 | 732 | 452 | 233 | 65 | 0 |
| 0 | 27 | 274 | 583 | 774 | 891 | 955 | 930 | 878 | 867 | 853 | 875 | 790 | 466 | 169 | 0 |
| 0 | 157 | 553 | 775 | 822 | 758 | 566 | 465 | 457 | 492 | 569 | 681 | 861 | 763 | 357 | 0 |
| 0 | 328 | 786 | 853 | 657 | 372 | 207 | 137 | 141 | 184 | 273 | 404 | 669 | 920 | 555 | 0 |
| 0 | 628 | 954 | 740 | 303 | 115 | 57 | 35 | 42 | 64 | 97 | 204 | 448 | 927 | 808 | 0 |
| 0 | 943 | 1035 | 425 | 97 | 33 | 18 | 14 | 20 | 19 | 31 | 70 | 290 | 874 | 1021 | 0 |
| 0 | 1180 | 1058 | 155 | 29 | 13 | 7 | 8 | 7 | 8 | 13 | 28 | 166 | 876 | 1118 | 0 |
| 0 | 1317 | 942 | 77 | 19 | 9 | 4 | 2 | 3 | 5 | 8 | 36 | 146 | 911 | 1023 | 0 |
| 0 | 1269 | 879 | 167 | 19 | 5 | 1 | 1 | 1 | 2 | 22 | 71 | 294 | 915 | 803 | 0 |
| 0 | 1000 | 975 | 429 | 92 | 22 | 3 | 1 | 4 | 21 | 68 | 245 | 600 | 821 | 479 | 0 |
| 0 | 422 | 1052 | 760 | 429 | 224 | 112 | 82 | 112 | 211 | 436 | 706 | 781 | 530 | 168 | 0 |
| 0 | 56 | 607 | 1018 | 895 | 735 | 639 | 644 | 733 | 876 | 923 | 742 | 449 | 147 | 17 | 0 |
| 0 | 1 | 43 | 359 | 793 | 1073 | 1175 | 1204 | 1081 | 821 | 536 | 289 | 80 | 12 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

# **Eksperimentai**

BwThreshold

Paveikslėlio dydis po sutraukimo

Testing() Riba

# **Metodas su mokytoju Nr.2 (Neuroninis tinklas)**

* 1. **Metodo veikimas**

# **Eksperimentai**

# **Metodų palyginimas**

1. **Išvados**

# **Programos kodas**

# **Program.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Drawing;

using System.IO;

using System.Drawing.Drawing2D;

using System.Threading;

namespace IntelektikaPR

{

class Program

{

private static readonly string TrainigData = Path.Combine(Environment.CurrentDirectory, @"Data\Data");

private static readonly string TestData = Path.Combine(Environment.CurrentDirectory, @"Data\Test");

//metodo 1 simbolių "heatmap" matricos

static double[,] Digit0;

static double[,] Digit1;

static double[,] Digit2;

static double[,] Digit3;

static double[,] Digit4;

static double[,] Digit5;

static double[,] Digit6;

static double[,] Digit7;

static double[,] Digit8;

static double[,] Digit9;

//į kokį dydį performuojami paveiksliukai

private const int ConvWidth = 16;

private const int ConvHeigth = 16;

static readonly Color ColorBlack = Color.FromArgb(255, 0, 0, 0);

static readonly Color ColorWhite = Color.FromArgb(255, 255, 255, 255);

private const double BwThreshold = 0.5;

private static readonly List<Vertex> Vertices = new List<Vertex>();

static void Main(string[] args)

{

new Thread(NeuralNetworkTests).Start();

List<double> result = XValidation(5, TrainigData);

Console.WriteLine("Kryžminės patikros rezultatai:");

int i = 0;

foreach (double value in result)

{

Console.WriteLine((i + 1) + " kryžminės patikros bendras tikslumas: " + $"{value:F1}" + " %");

i++;

}

Console.ReadLine();

}

/// <summary>

/// Performs the Neural Netowrk tests.

/// </summary>

public static void NeuralNetworkTests()

{

// Read the trainig data.

Console.WriteLine("Reading input data.");

var inputData = ReadAllFolders(TrainigData)

.OrderBy(o => Guid.NewGuid()) // Scramble the list

.ToArray(); // Execute the request

// Prepare data for the neural network.

var outputVector = inputData.Select(d => d.Key)

.Select(iValue => // Create a key array, with index of the key having value 1, rest = 0.

{

var a = new double[10];

a[iValue] = 1;

return a;

}).ToArray();

var inputVector = inputData.Select(d => d.Value.ToDoubleArray())

.ToArray(); // Create a value array, with having appropriate indexes.

// Create a neural network

var network = new NeuralNetwork(inputVector, outputVector);

// Execute the learning process

network.Teach();

// Read the test data.

Console.WriteLine("Reading output data.");

var outputData = ReadAllFolders(TestData);

// Print out the result.

var correctCount = outputData.GroupBy(k => k.Key)

.Select(g => new

{

Index = g.Key,

TotalElements = g.Count(),

SuccessCount = g.Count(testData => testData.Key == network.Compute(testData.Value))

}).ToArray();

foreach (var el in correctCount)

{

Console.WriteLine($"Neural network accuracy for number {el.Index}: {(double)el.SuccessCount / el.TotalElements:P}");

}

var overallSuccess = correctCount.Sum(e => e.SuccessCount);

Console.WriteLine($"Overall neural network accuracy: [{overallSuccess} / {outputData.Length}] - {(double)overallSuccess / outputData.Length:P}");

}

/// <summary>

/// Reads all of the contents of a given folder and creates an array of Key - Image pairings

/// </summary>

/// <returns>An array of Key-Image pairings. Key is int - image's true number</returns>

public static KeyValuePair<int, Image>[] ReadAllFolders(string folderPath)

{

var index = 0;

var inputData = Enumerable.Empty<KeyValuePair<int, Image>>();

foreach (var directory in Directory.EnumerateDirectories(folderPath))

{

inputData = inputData.Concat(ReadFolder(directory) // Read the contents and concatinate the image data.

.Select(f => new KeyValuePair<int, Image>(index, f))) // Attach an index to a given image, i.e. 2 = file in directory no.2

.ToArray(); // Execute the request immedietly.

index++;

Console.WriteLine($"{index \* 10}%");

}

return (KeyValuePair<int, Image>[])inputData;

}

/// <summary>

/// Reads folder data for images

/// </summary>

/// <param name="folderPath">Path to search for images</param>

/// <returns>List of Images in the given folder.</returns>

public static IEnumerable<Image> ReadFolder(string folderPath)

{

var rect = new Rectangle(2, 2, 28, 28);

#if DEBUG

var index = 0;

#endif

foreach (var filePath in Directory.EnumerateFiles(folderPath))

{

#if DEBUG

//if (index++ >= 200) continue;

#endif

var bitmap = new Bitmap(filePath);

yield return new Image(bitmap.Clone(rect, bitmap.PixelFormat));

}

}

//-------------------------------------------------------------------------------

//METODAS SU MOKYTOJU 1 BEGIN

//-------------------------------------------------------------------------------

// Vykdoma kryžminė patikra

public static List<double> XValidation(int dataSetCount, string data)

{

List<double> result = new List<double>();

int start = 0; // testuojamų duomenų pradžios indeksas

int step = (int)(2000.0 / dataSetCount); // testuojamų duomenų indekso žingsnis tarp iteracijų

int end = (int)(2000.0 / dataSetCount); // testuojamų duomenų pabaigos indeksas

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

// Surašomi failų pavadinimai į sąrašą. Sąrašas saugo kiekvieno skaitmens paveikslėlių sąrašą (kelius (path) iki paveikslėlių).

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

{

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

foreach (string fpath in Directory.GetFiles(Path.Combine(data, "digit\_" + i), "\*.png"))

{

list.Add(fpath);

}

listList.Add(list);

}

//Pradedamas apmokymas

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

{

Console.WriteLine("Vykdoma " + (i + 1) + " kryžminės patikros iteracija.");

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

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

{

List<string> t = listList[j].ToList();

temp.Add(t);

}

if (i < dataSetCount - 1)

{

//Pašalinami testuojami duomenys iš apmokymo sąrašo, kai i < kryž. patikros iteracijų sk. - 1

foreach (List<string> list in temp)

{

list.RemoveRange(start, step);

}

}

else

{

//Pašalinami testuojami duomenys iš apmokymo sąrašo, kai i == kryž. patikros iteracijų sk. - 1

foreach (List<string> list in temp)

{

list.RemoveRange(start, list.Count - start);

}

}

// Pradedamas apmokymas

Digit0 = Digit1 = Digit2 = Digit3 = Digit4 = Digit5 = Digit6 = Digit7 = Digit8 = Digit9 = null;

Digit0 = TrainSymbolMatrix(Digit0, temp[0]);

Digit1 = TrainSymbolMatrix(Digit1, temp[1]);

Digit2 = TrainSymbolMatrix(Digit2, temp[2]);

Digit3 = TrainSymbolMatrix(Digit3, temp[3]);

Digit4 = TrainSymbolMatrix(Digit4, temp[4]);

Digit5 = TrainSymbolMatrix(Digit5, temp[5]);

Digit6 = TrainSymbolMatrix(Digit6, temp[6]);

Digit7 = TrainSymbolMatrix(Digit7, temp[7]);

Digit8 = TrainSymbolMatrix(Digit8, temp[8]);

Digit9 = TrainSymbolMatrix(Digit9, temp[9]);

int corr = 0; // teisingų spėjimų kiekis

int count = 0; // bendras spėjimų kiekis

double[] foldresults = new double[10];

// Vykdoma kryžminė patikra. Ciklas keliauja per sąrašo skaičius (0-9)

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

{

corr = 0; // teisingų spėjimų kiekis

count = 0; // bendras spėjimų kiekis

// Ciklas keliauja per j - tojo skaitmens paveikslėlių sąrašą

int step2 = step;

if (i >= dataSetCount - 1) step2 = listList[j].ToList().Count - start;

foreach (string fpath in listList[j].GetRange(start, step2))

{

count++;

double[,] img = processImage(fpath);

//Atlieka skaitmens testavimą

string rez = Testing(img, Digit0, Digit1, Digit2, Digit3, Digit4, Digit5, Digit6, Digit7, Digit8, Digit9);

if (rez == "" + j) corr++;

}

Console.WriteLine("Skaitmens " + j + " atpažinimo tikslumas: " + $"{(double)corr / count \* 100:F1} %");

foldresults[j] = (double)corr / count \* 100;

Console.WriteLine(corr + " " + count);

}

double sum = 0;

for (int z = 0; z < foldresults.Length; z++)

{

sum += foldresults[z];

}

double acc = sum / foldresults.Length;

result.Add(acc);

Console.WriteLine("Tikslumas: " + string.Format("{0:F1} %", acc));

Console.WriteLine("------------------------------------------------------------------------");

Console.WriteLine();

// Paslenkama testuojamų duomenų pradžios bei pabaigos indeksas

start = start + step;

end = end + step;

}

return result;

}

//

public static double[,] TrainSymbolMatrix(double[,] symbolmatrix, List<string> trainpath)

{

symbolmatrix = processImage(trainpath[0]);

foreach (string fpath in trainpath)

{

double[,] img = processImage(fpath);

symbolmatrix = Training(img, symbolmatrix);

}

return symbolmatrix;

}

//Apmokymas. Gaunam svorinę matricą. "Kaip atrodo simbolis".Kuo daugiau kartų tame pikselyje buvo balta - tuo didesnis skaičius

public static double[,] Training(double[,] trainmatrix, double[,] finalMatrix)

{

double[,] newFinalmatrix = new double[ConvWidth, ConvHeigth];

for (int i = 0; i < trainmatrix.GetLength(0); i++)

{

for (int j = 0; j < trainmatrix.GetLength(1); j++)

{

newFinalmatrix[i, j] = Math.Round((trainmatrix[i, j] + finalMatrix[i, j]));

}

}

return newFinalmatrix;

}

//Testavimas, tikrinama, kurią apmokymo metu gautą svorių matricą geriausiai atitinka testuojamas paveiksliukas

public static string Testing(double[,] testmatrix, double[,] SymbolMatrix0, double[,] SymbolMatrix1, double[,] SymbolMatrix2, double[,] SymbolMatrix3, double[,] SymbolMatrix4,

double[,] SymbolMatrix5, double[,] SymbolMatrix6, double[,] SymbolMatrix7, double[,] SymbolMatrix8, double[,] SymbolMatrix9)

{

int[] scores = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };

string[] results = { "0", "1", "2", "3", "4", "5", "6", "7", "8", "9" };

int riba = 300; //kiek baltos spalvos pasikartojimų heatmap'e skaitosi kaip "atitikimas"

for (int i = 0; i < testmatrix.GetLength(0); i++)

{

for (int j = 0; j < testmatrix.GetLength(1); j++)

{

if ((SymbolMatrix0[i, j] >= riba) && (testmatrix[i, j] == 1))

{

scores[0] += 1;

}

if ((SymbolMatrix1[i, j] >= riba) && (testmatrix[i, j] == 1))

{

scores[1] += 1;

}

if ((SymbolMatrix2[i, j] >= riba) && (testmatrix[i, j] == 1))

{

scores[2] += 1;

}

if ((SymbolMatrix3[i, j] >= riba) && (testmatrix[i, j] == 1))

{

scores[3] += 1;

}

if ((SymbolMatrix4[i, j] >= riba) && (testmatrix[i, j] == 1))

{

scores[4] += 1;

}

if ((SymbolMatrix5[i, j] >= riba) && (testmatrix[i, j] == 1))

{

scores[5] += 1;

}

if ((SymbolMatrix6[i, j] >= riba) && (testmatrix[i, j] == 1))

{

scores[6] += 1;

}

if ((SymbolMatrix7[i, j] >= riba) && (testmatrix[i, j] == 1))

{

scores[7] += 1;

}

if ((SymbolMatrix8[i, j] >= riba) && (testmatrix[i, j] == 1))

{

scores[8] += 1;

}

if ((SymbolMatrix9[i, j] >= riba) && (testmatrix[i, j] == 1))

{

scores[9] += 1;

}

}

}

int maxValue = scores.Max();

int maxIndex = scores.ToList().IndexOf(maxValue);

return results[maxIndex];

}

//-------------------------------------------------------------------------------

//METODAS SU MOKYTOJU 1 END

//-------------------------------------------------------------------------------

//-------------------------------------------------------------------------------

//DUOMENŲ TVARKYMAS BEGIN

//-------------------------------------------------------------------------------

// Paima paveikslėlį iš failo, sumažina, pilkus pixelius padaro juodais/baltais, konvertuoaj į 0/1 matricą, yra galimybė apkarpyti

public static double[,] processImage(string fpath)

{

Bitmap og = new Bitmap(fpath); //Paima iš failo

Bitmap conv = new Bitmap(ConvWidth, ConvHeigth); //Sumažina

using (Graphics gr = Graphics.FromImage(conv))

{

gr.SmoothingMode = SmoothingMode.HighQuality;

gr.InterpolationMode = InterpolationMode.HighQualityBicubic;

gr.PixelOffsetMode = PixelOffsetMode.HighQuality;

gr.DrawImage(og, new Rectangle(0, 0, ConvWidth, ConvHeigth));

}

conv = ImageToBlackWhite(conv, BwThreshold); //Pilkus pixelius -> juodus/baltus

conv.RotateFlip(RotateFlipType.Rotate270FlipY);

double[,] img = ImageToMatrix(conv); //konvertuoja į 0/1 matricą

//img = CutMatrix(img); //apkarpo -2px nuo kiekvieno krašto

return img;

}

//Paverčia visus pilkus pixelius juodais arba baltais pagal slenksčio reikšmę

public static Bitmap ImageToBlackWhite(Bitmap imgSrc, double threshold)

{

int width = imgSrc.Width;

int height = imgSrc.Height;

Color pixel;

Bitmap imgOut = new Bitmap(imgSrc);

for (int row = 0; row < height - 1; row++)

{

for (int col = 0; col < width - 1; col++)

{

pixel = imgSrc.GetPixel(col, row);

if (pixel.GetBrightness() < threshold){

Vertices.Add(new Vertex(col, row));

imgOut.SetPixel(col, row, ColorBlack);

}

else{

imgOut.SetPixel(col, row, ColorWhite);

}

}

}

return imgOut;

}

//Konvertuoja juodą/baltą paveiksliuką į 0/1 matricą

public static double[,] ImageToMatrix(Bitmap img)

{

int height = img.Height;

int width = img.Width;

double[,] Matrix = new double[ConvWidth, ConvHeigth];

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

{

for (int j = 0; j < height; j++)

{

if (img.GetPixel(i, j).Equals(ColorBlack))

{

Matrix[i, j] = 0;

}

else if (img.GetPixel(i, j).Equals(ColorWhite))

{

Matrix[i, j] = 1;

}

}

}

return Matrix;

}

//Nukerpa tuščius 2px nuo kiekvieno krašto

public static double[,] CutMatrix(double[,] matrix)

{

double[,] newmatrix = new double[ConvWidth - 2, ConvHeigth - 2];

for (int i = 2; i < matrix.GetLength(0) - 2; i++)

{

for (int j = 2; j < matrix.GetLength(1) - 2; j++)

{

newmatrix[i - 2, j - 2] = matrix[i, j];

}

}

return newmatrix;

}

//spausdina 0/1 matricą

public static void PrintMatrix(double[,] matrix)

{

for (int i = 0; i < matrix.GetLength(0); i++){

for (int j = 0; j < matrix.GetLength(1); j++){

Console.Write($"{matrix[i, j]}");

}

Console.Write(Environment.NewLine + Environment.NewLine);

}

}

public class Vertex

{

public Vertex(int i, int j)

{

X = i;

Y = j;

}

public int X { get; set; }

public int Y { get; set; }

public override string ToString()

{

return $"({X}/{Y})";

}

}

//-------------------------------------------------------------------------------

//DUOMENŲ TVARKYMAS END

//-------------------------------------------------------------------------------

}

}

# **NeuralNetwork.cs**

using System;

using Accord.Math;

using Accord.Neuro;

using Accord.Neuro.Learning;

namespace IntelektikaPR

{

class NeuralNetwork

{

private readonly double[][] \_inputData;

private readonly double[][] \_outputData;

private const int EpochsMax = 2000;

private const int OverfittedErrorTreshold = 5;

private ActivationNetwork Network { get; }

public NeuralNetwork(double[][] inputData, double[][] outputData)

{

\_inputData = inputData;

\_outputData = outputData;

Network = new ActivationNetwork(new SigmoidFunction(),

inputData[0].Length, // Input neuron count

outputData[0].Length); // Output neuron count

Network.Randomize();

}

public void Teach()

{

// Teacher method.

// Perceptron was experimentally found to give best results.

var teacher = new PerceptronLearning(Network);

// Value to track network's effectivness.

// If value increases over time, network is overfitted - terminate learning process.

var previousError = double.MaxValue;

// Value tracks the amount of times error value increased.

// Once it reaches a treshold, stop trainig - network is overfitted.

var errorIncreaseCount = 0;

// Epochs tracks the amount of cycles neural network took.

// Used to prevent infinite training.

var epochs = 0;

Console.WriteLine("Teaching network.");

// Train the neural network.

// Training lasts until EITHER network starts to get overfitted with data,

// or trainig has reached maximum epoch count.

while (errorIncreaseCount < OverfittedErrorTreshold && epochs < EpochsMax)

{

var error = teacher.RunEpoch(\_inputData, \_outputData);

if (error > previousError)

errorIncreaseCount++;

else

errorIncreaseCount = 0;

previousError = error;

epochs += 1;

if (epochs % 20 == 0)

Console.WriteLine($"{(double)epochs / EpochsMax \* 100}%");

}

}

public int Compute(Image img)

{

return Compute(img.ToDoubleArray());

}

private int Compute(byte[,] img)

{

var inputVector = new double[img.Length];

var index = 0;

foreach (var pixel in img)

{

inputVector[index++] = (double)pixel / 255;

}

return Compute(inputVector);

}

private int Compute(double[] inputVector)

{

var output = Network.Compute(inputVector);

output.Max(out var index);

return index;

}

}

}

# **Image.cs**

using System;

using System.Drawing;

namespace IntelektikaPR

{

public class Image

{

public Bitmap Bitmap { get; }

public Image(Bitmap bitmap)

{

Bitmap = bitmap;

}

/// <summary>

/// Converts the given Image object to a 2D [width, height] byte array.

/// Each byte consists of grayscale 0-255 value for the image.

/// </summary>

/// <returns>Byte array consisting of values 0-255 for each element.</returns>

public byte[,] ToByteMatrix()

{

var matrix = new byte[Bitmap.Width, Bitmap.Height];

for (int i = 0; i < Bitmap.Width; i++)

{

for (int j = 0; j < Bitmap.Height; j++)

{

// Take the blue value, as image is monochrome, B, G, and R are identical.

matrix[i, j] = Bitmap.GetPixel(i, j).B;

}

}

return matrix;

}

/// <summary>

/// Converts the given Image object to a width-first double array.

/// Each number consists of grayscale 0-1 value for the image.

/// 0 = Black, 1 = White, 0.5 = Gray

/// </summary>

/// <returns>Double array consisting of values 0-255 for each element.</returns>

public double[] ToDoubleArray()

{

var vector = new double[Bitmap.Width \* Bitmap.Height];

for (int i = 0; i < Bitmap.Width; i++)

{

for (int j = 0; j < Bitmap.Height; j++)

{

// Take the blue value, as image is monochrome, B, G, and R are identical.

vector[i \* Bitmap.Width + j] = (double)Bitmap.GetPixel(i, j).B / 255;

}

}

return vector;

}

}

}

1. **Paveikslėlių sąrašas**
2. **Šaltiniai**