**Приложение Б**

Класс CNN

class CNN

{

Net<double> net;

int stepCount;

string loadedJson;

string JsonToSave;

int Aim;

double Acc;

int classes;

bool isNetLearned;

private SgdTrainer<double> trainer;

string path;

private ConvLayer convLayer;

private readonly CircularBuffer<double> testAccWindow = new CircularBuffer<double>(100);

private readonly CircularBuffer<double> trainAccWindow = new CircularBuffer<double>(100);

public CNN(string path)

{

net = new Net<double>();

Aim = 0;

Acc = 0;

isNetLearned = false;

path = "";

}

public int CreateCNN(int inpx = 32, int inpy = 32, int inpd = 1, int classesCount = 11)

{

if (GetLayersCount() == 0)

{

net.AddLayer(new InputLayer(inpx, inpy, 1));

convLayer = new ConvLayer (5, 5, 12) { };

net.AddLayer(this.convLayer);

net.AddLayer(new LeakyReluLayer(0.3));

net.AddLayer(new PoolLayer(2, 2) { });//субдискр

net.AddLayer(new ConvLayer(5, 5, 24) { });

net.AddLayer(new LeakyReluLayer(0.3));

net.AddLayer(new PoolLayer(2, 2) { });

net.AddLayer(new FullyConnLayer(150));

net.AddLayer(new DropoutLayer<double>(0.5));

net.AddLayer(new FullyConnLayer(100));

net.AddLayer(new DropoutLayer<double>(0.5));

net.AddLayer(new FullyConnLayer(classesCount));

net.AddLayer(new SoftmaxLayer(classesCount));

classes = classesCount;

isNetLearned = false;

}

return net.Layers.Count;

}

public double TeachCNN(string pathL, string pathT, int acc, double learnRate, int size, double mom)

{

if (net.Layers.Count == 0)

{

CreateCNN();

}

var datasets = new DataSets(pathL, pathT);

Console.WriteLine("DataSets Created");

if (!datasets.Load())

{

return -2;

}

Console.WriteLine("DataSets Loaded");

Aim = acc;

trainer = new SgdTrainer<double>(net)

{

LearningRate = learnRate,

BatchSize = size,

Momentum = mom

};

if (net.Layers.Count != 0)

{

do

{

var trainSample = datasets.Train.NextBatch(trainer.BatchSize, classes);

Train(trainSample.Item1, trainSample.Item2, trainSample.Item3);

var testSample = datasets.Test.NextBatch(trainer.BatchSize, classes);

Test(testSample.Item1, testSample.Item3, testAccWindow);

Console.WriteLine("Loss: {0} Train accuracy: {1}% Test accuracy: {2}%", trainer.Loss,

Math.Round(trainAccWindow.Items.Average() \* 100.0, 2),

Math.Round(testAccWindow.Items.Average() \* 100.0, 2));

Console.WriteLine("Example seen: {0} Fwd: {1}ms Bckw: {2}ms", stepCount,

Math.Round(trainer.ForwardTimeMs, 2),

Math.Round(trainer.BackwardTimeMs, 2));

Acc = Math.Round(testAccWindow.Items.Average() \* 100.0, 2);

} while (Acc < Aim);

Console.WriteLine($"{stepCount}");

isNetLearned = true;

return Acc;

}

else return -1;

}

public string SaveCNN(string tmppath)

{

if (net.Layers.Count == 0)

return "";

if (!isNetLearned)

{

return "";

}

JsonToSave = net.ToJson();

var name = "CNN"+ GetClassesCount().ToString()+"cl"+GetLayersCount().ToString()+"l\_" + GetAccuracy().ToString() + ".txt";

var newPath = Path.Combine(Path.GetDirectoryName(tmppath), name);

if (File.Exists(newPath))

{

File.Delete(newPath);

}

File.WriteAllText(newPath, JsonToSave);

return newPath;

}

public int LoadCNN(string path)

{

if (path!= "")

{

loadedJson = File.ReadAllText(path);

Net<double> deserialized = new Net<double>();

try

{

deserialized = SerializationExtensions.FromJson<double>(loadedJson);

}

catch (Exception)

{

return -1;

}

this.net = deserialized;

isNetLearned = true;

this.path = path;

classes = deserialized.Layers[GetLayersCount() - 1].OutputDepth;

return net.Layers.Count;

}

else return -1;

}

public string Recognize(byte[] image)

{

if (net.Layers.Count < 0)

return "Network is not Created";

if (!isNetLearned)

{

return "Network is not learned";

}

if (image.Length != net.Layers[0].InputWidth \* net.Layers[0].InputHeight)

{

return "Wrong Image Size";

}

if (net.Layers.Count == 0)

{

return "Network is not created";

}

if (!isNetLearned)

{

return "Network is not learned or not loaded";

}

var dataShape = new Shape(net.Layers[0].InputWidth, net.Layers[0].InputHeight, 1, 1);

var data = new double[dataShape.TotalLength];

var dataVolume = BuilderInstance.Volume.From(data, dataShape);

var j = 0;

for (var y = 0; y < net.Layers[0].InputHeight; y++)

{

for (var x = 0; x < net.Layers[0].InputWidth; x++)

{

dataVolume.Set(x, y, 0, 0, image[j++]);

}

}

net.Forward(dataVolume);

var prediction = net.GetPrediction();

return GetClassNameFromNumber(prediction[0]);

}

public string[] RecognizeAll( ImageFolder imf)

{

int count = imf.GetCount();

string[] result = new string[count];

for(int i=0;i<count;i++){

var image = new Image<Gray, byte>(imf.GetImg(i)).Bytes;

if (image.Length != 32 \* 32)

{

result[0] = "wrong image size";

return result;

}

if (net.Layers.Count == 0)

{

this.CreateCNN();

LoadCNN(path);

}

var dataShape = new Shape(32, 32, 1, 1);

var data = new double[dataShape.TotalLength];

var dataVolume = BuilderInstance.Volume.From(data, dataShape);

var j = 0;

for (var y = 0; y < net.Layers[0].InputHeight; y++)

{

for (var x = 0; x < net.Layers[0].InputWidth; x++)

{

dataVolume.Set(x, y, 0, 0, image[j++]);

}

}

net.Forward(dataVolume);

var prediction = net.GetPrediction();

result[i] = GetClassNameFromNumber(prediction[0]);

}

return result;

}

private void Train(Volume<double> x, Volume<double> y, int[] labels)

{

this.trainer.Train(x, y);

Test(x, labels, this.trainAccWindow, false);

stepCount += labels.Length;

}

private void Test(Volume<double> x, int[] labels, CircularBuffer<double> accuracy, bool forward = true)

{

if (forward)

{

net.Forward(x);

}

var prediction = net.GetPrediction();

for (var i = 0; i < labels.Length; i++)

{

accuracy.Add(labels[i] == prediction[i] ? 1.0 : 0.0);

}

}

public int GetLayersCount()

{

return net.Layers.Count;

}

public int GetClassesCount()

{

return this.classes;

}

public double GetAccuracy()

{

return Acc;

}

public bool Reset()

{

return false;

}

public bool IsLearned()

{

return this.isNetLearned;

}

public string GetClassNameFromNumber(int n)

{

switch (n)

{

case 1:

return "60 km/h";

case 2:

return "Main road";

case 3:

return "Secondary road";

case 4:

return "Stop sign";

case 5:

return "Road up";

case 6:

return "Kirpich";

case 7:

return "Warning Sign";

case 8:

return "Sleeping policeman";

case 9:

return "Road Works";

case 10:

return "Only forward";

case 11:

return "Pesh Perehod";

default:

break;

}

return "";

}

}

Класс DataSet

internal class DataSet

{

private readonly List<ImageEntry> ImagesList;

private readonly Random random = new Random(RandomUtilities.Seed);

private int start;

private int epochCompleted;

public DataSet(List<ImageEntry> trainImages)

{

this.ImagesList = trainImages;

}

public Tuple<Volume<double>, Volume<double>, int[]> NextBatch(int batchSize, int numClasses = 10)

{

const int w = 32;

const int h = 32;

var dataShape = new Shape(w, h, 1, batchSize);

var labelShape = new Shape(1, 1, numClasses, batchSize);

var data = new double[dataShape.TotalLength];

var label = new double[labelShape.TotalLength];

var labels = new int[batchSize];

// Shuffle for the first epoch

if (this.start == 0 && this.epochCompleted == 0)

{

for (var i = this.ImagesList.Count - 1; i >= 0; i--)

{

var j = random.Next(i);

var temp = ImagesList[j];

this.ImagesList[j] = this.ImagesList[i];

this.ImagesList[i] = temp;

}

}

var dataVolume = BuilderInstance.Volume.From(data, dataShape);

for (var i = 0; i < batchSize; i++)

{

var entry = ImagesList[this.start];

labels[i] = entry.Label;

var j = 0;

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

{

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

{

dataVolume.Set(x, y, 0, i, entry.Image[j++] / 255.0);

}

}

if(i \* numClasses + entry.Label<label.Length)

label[i \* numClasses + entry.Label] = 1.0;

start++;

if (start == ImagesList.Count)

{

start = 0;

epochCompleted++;

Console.WriteLine($"Epoch #{this.epochCompleted}");

}

}

var labelVolume = BuilderInstance.Volume.From(label, labelShape);

return new Tuple<Volume<double>, Volume<double>, int[]>(dataVolume, labelVolume, labels);

}

Класс DataSets

internal class DataSets

{

string TrainPath;

string TestPath;

public DataSets(string pathL, string pathT)

{

TrainPath = pathL;

TestPath = pathT;

}

public DataSet Train { get; set; }

public DataSet Test { get; set; }

public bool Load()

{

Console.WriteLine("Loading the datasets...");

Console.WriteLine("Loading train dataset");

var train\_images = ImageReader.Load(TrainPath);

Console.WriteLine("Loaded the train dataset...");

Console.WriteLine("Loading the test dataset...");

var testing\_images = ImageReader.Load(TestPath);

Console.WriteLine("Loaded the test dataset...");

if (train\_images.Count == 0 || testing\_images.Count == 0)

{

Console.WriteLine("Missing training/testing files.");

return false;

}

Train = new DataSet(train\_images);

Test = new DataSet(testing\_images);

return true;

}

}

Класс DropoutLayer<T>

public class DropoutLayer<T> : LayerBase<T> where T : struct, IEquatable<T>, IFormattable

{

public DropoutLayer(Dictionary<string, object> data) : base(data)

{

this.DropProbability = (T)Convert.ChangeType(data["DropProbability"], typeof(T));

}

public DropoutLayer(T dropProbability)

{

this.DropProbability = dropProbability;

}

public T DropProbability { get; set; }

public override void Backward(Volume<T> outputGradient)

{

this.OutputActivationGradients = outputGradient;

this.InputActivationGradients.Clear();

this.OutputActivation.DropoutGradient(this.InputActivation, this.OutputActivationGradients, this.DropProbability, this.InputActivationGradients);

}

protected override Volume<T> Forward(Volume<T> input, bool isTraining = false)

{

input.Dropout(isTraining ? this.DropProbability : Ops<T>.Zero, this.OutputActivation);

return this.OutputActivation;

}

public override Dictionary<string, object> GetData()

{

var dico = base.GetData();

dico["DropProbability"] = this.DropProbability;

return dico;

}

public override void Init(int inputWidth, int inputHeight, int inputDepth)

{

base.Init(inputWidth, inputHeight, inputDepth);

this.OutputDepth = inputDepth;

this.OutputWidth = inputWidth;

this.OutputHeight = inputHeight;

}

}

Класс ImageReader

public static class ImageReader

{

public static List<ImageEntry> Load(string Path, int maxItem = -1)

{

var images = LoadImages(Path, out ImageFolder fold, maxItem);

var label = LoadLabels(Path, fold, maxItem);

if (label.Count == 0 || images.Count == 0)

{

return new List<ImageEntry>();

}

return label.Select((t, i) => new ImageEntry { Label = t, Image = images[i] }).ToList();

}

private static List<byte[]> LoadImages(string path,out ImageFolder fold, int maxItem = -1)

{

var result = new List<byte[]>();

var f = new ImageFolder();

f.Load(path);

foreach (string imgpath in f.GetAll())

{

result.Add(new Image<Gray, byte>(imgpath).Bytes);

}

fold = f;

return result;

}

private static List<int> LoadLabels(string path, ImageFolder f, int maxItem = -1 )

{

var result = new List<int>();

var name = "labels.txt";

var filePath = Path.Combine(path, name);

string labels = "";

if(File.Exists(filePath))

{

File.Delete(filePath);

}

if (!File.Exists(filePath))

{

foreach (string imgpath in f.GetAll())

{

string n = Path.GetFileName(imgpath);

n = n.Substring(n.IndexOf('\_') + 1, n.IndexOf('.') - n.IndexOf('\_') - 1);

labels += " " + n;

}

File.WriteAllText(filePath, labels);

}

labels = File.ReadAllText(filePath, Encoding.UTF8) + " ";

int c = 0;

int i = 0;

string tmp = "";

do

{

if ((labels[i] != ' '))

tmp += labels[i];

else

{

if (tmp.Length > 0)

{

result.Add(Int32.Parse(tmp));

tmp = "";

c++;

}

}

i++;

} while (i < labels.Length);

return result;

}

}

Класс DetectFolder

static class DetectFolder

{

public static void DetectAll(string folderPath, string[] cascadesPath, string savePath)

{

List<SignsHaarCascade> cascadesList = new List<SignsHaarCascade>();

List<ImageFolder> foldersList = new List<ImageFolder>();

foreach(string casPath in cascadesPath)

{

cascadesList.Add(new SignsHaarCascade(casPath));

}

ImageFolder f = new ImageFolder();

f.Load(folderPath);

int c = 0;

foreach (string image\_path in f.GetAll())

{

Mat img = new Image<Bgr, byte>(image\_path).Mat;

foreach(SignsHaarCascade cascade in cascadesList)

{

Console.WriteLine(f.GetPath());

Console.WriteLine(c++);

List<Mat> list = cascade.detectAll(img, 1.1, 4);

var n = new DirectoryInfo(f.GetPath()).Name;

var newFolderPath = Path.Combine(savePath, n.ToString());

var imageName = Path.GetFileNameWithoutExtension(image\_path);

for (int i = 0; i < list.Count; i++)

list[i].Save(newFolderPath +"\_" + imageName+ "\_" + i + ".jpg");

}

}

}

}

Класс SignsHaarCascade

class SignsHaarCascade

{

public CascadeClassifier cascadeClassifier;

private string path;

public SignsHaarCascade(string path)

{

this.path = path;

if (this.path != "")

{

cascadeClassifier = new CascadeClassifier(this.path);

}

}

public List<Mat> detectAll(Mat img, double sFactor = 1.1, int minNeigh = 4)

{

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

foreach (Rectangle rect in cascadeClassifier.DetectMultiScale(img, sFactor, minNeigh))

{

list.Add(new Mat(img, rect));

}

return list;

}

public string getPath()

{

return this.path;

}

}

Класс FFMPEGConverter

public class FFMPEGConverter

{

private readonly string sourcePath;

private readonly string savePath;

int timeStepMilliseconds = 5000;

List<MovementPoint> pointsList = new List<MovementPoint>();

public FFMPEGConverter(string spath, string svpath)

{

sourcePath = spath;

savePath = svpath;

}

public bool ConvertAll( bool rewrite)

{

bool isOk = true;

var nPath = System.IO.Path.GetFileName(sourcePath);

string destinationFolderPath = savePath + @"\convertedFrom" + nPath.Substring(0, nPath.LastIndexOf("."));

string subsPath = "";

if ((File.Exists(destinationFolderPath)) && (rewrite = true))

{

Directory.Delete(destinationFolderPath,true);

Directory.CreateDirectory(destinationFolderPath);

}

ConvertVidToImages(destinationFolderPath);

subsPath = ConvertVidToSubs(destinationFolderPath);

pointsList = ParseSubtitleFile(subsPath);

ImageFolder f = new ImageFolder();

f.Load(destinationFolderPath);

f.Sort();

List<PhotoData> dataList = PhCoord\_Connect.Connect(f, pointsList, timeStepMilliseconds).ToList();

for (int i = 0; i < dataList.Count; i++)

{

Console.WriteLine(

dataList[i].Photo.Number.ToString() + " "+

dataList[i].Photo.FileName +" "+

dataList[i].MovementPoint.Lon.ToString() + " " +

dataList[i].MovementPoint.Lat.ToString() + " " +

dataList[i].MovementPoint.Date.ToString() + " " +

dataList[i].MovementPoint.Azimuth);

}

return isOk;

}

public string ConvertVidToSubs(string dFolderPath)

{

string destinationPath = dFolderPath + @"\subs.txt";

try

{

using (Process myProcess = new Process())

{

if (File.Exists(destinationPath))

File.Delete(destinationPath);

string args = " -i " + sourcePath + " -an -vn -map 0:s -c:s copy -f rawvideo " + destinationPath;

System.Diagnostics.Debug.WriteLine(args);

myProcess.StartInfo.UseShellExecute = false;

myProcess.StartInfo.FileName = "ffmpeg";

myProcess.StartInfo.Arguments = args;

myProcess.StartInfo.CreateNoWindow = true;

myProcess.Start();

myProcess.WaitForExit();

}

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

}

return destinationPath;

}

public bool ConvertVidToImages(string dFolderPath)

{

bool isStarted = false;

try

{

using (Process myProcess = new Process())

{

if (!System.IO.Directory.Exists(dFolderPath))

{

System.IO.Directory.CreateDirectory(dFolderPath);

}

var fps = $"1000/{timeStepMilliseconds}";

string args = " -i " + sourcePath + " -qscale:v 5 -r " + fps + " " + dFolderPath + @"\image%d.jpg";

System.Diagnostics.Debug.WriteLine(args);

myProcess.StartInfo.UseShellExecute = false;

myProcess.StartInfo.FileName = "ffmpeg";

myProcess.StartInfo.Arguments = args;

myProcess.StartInfo.CreateNoWindow = false;

isStarted = myProcess.Start();

myProcess.WaitForExit();

}

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

}

return isStarted;

}

private List<MovementPoint> ParseSubtitleFile(string fileName)

{

if (!File.Exists(fileName))

{

return null;

}

var subtitles = File.ReadAllText(fileName);

var movementPointRegExp = @"\d{6}\.\d+,A,\d{4}(\.\d\*)?,N,\d{5}(\.\d\*)?,E,\d\*(\.\d\*)?,\d\*(\.\d\*)?,\d{6},,,.{4}";

var matches = Regex.Matches(subtitles, movementPointRegExp);

List<MovementPoint> en = new List<MovementPoint>();

foreach (Match m in matches){

string g = m.Value;

MovementPoint mov = ParseMovementPoint(g);

pointsList.Add(mov);

en.Add(mov);

}

return en;

}

private MovementPoint ParseMovementPoint(string movementPoint)

{

var parts = movementPoint.Split(new char[] { ',' });

decimal? azimuth = null;

if (parts[7].Length > 0)

{

azimuth = Convert.ToDecimal(parts[7].Replace(',', '.'), CultureInfo.InvariantCulture);

}

var dateString = parts[8] + parts[0].Substring(0, parts[0].IndexOf('.'));

var date = DateTime.ParseExact(dateString, "ddMMyyHHmmss",

CultureInfo.CurrentCulture, DateTimeStyles.AssumeLocal);

decimal lat = ConvertDegreesAndDecimalMinutesStringToDecimalDegrees(parts[2]);

decimal lon = ConvertDegreesAndDecimalMinutesStringToDecimalDegrees(parts[4]);

return new MovementPoint

{

Date = date,

Lat = lat,

Lon = lon,

Azimuth = azimuth

};

}

private decimal ConvertDegreesAndDecimalMinutesStringToDecimalDegrees(string degreesAndDecimalMinutesString)

{

decimal degreesAndDecimalMinutes = Convert.ToDecimal(

degreesAndDecimalMinutesString.Replace('.', ','),

new NumberFormatInfo() { NumberDecimalSeparator = "," });

decimal degrees = Math.Truncate(degreesAndDecimalMinutes / 100);

decimal minutes = (degreesAndDecimalMinutes / 100 - degrees) \* 100;

decimal decimalDegrees = degrees + minutes / 60;

return decimalDegrees;

}

}

Класс ImgOps

class ImgOps

{

public static Mat RGBtoHSV(Mat img)

{

Mat tmp = new Mat();

try

{

Emgu.CV.Image<Bgr, Byte> image = img.ToImage<Bgr, Byte>();

Bitmap b = image.ToBitmap();

Emgu.CV.Image<Hsv, Byte> imgHsv = new Image<Hsv, Byte>(b);

tmp = imgHsv.Mat;

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

return tmp;

}

public static Image<Gray, Byte> RGBtoGrey(Mat img)

{

Image<Gray, Byte> grayImage = new Image<Gray, byte>(new Size(0,0));

try

{

Emgu.CV.Image<Bgr, Byte> image = img.ToImage<Bgr, Byte>();

grayImage = image.Convert<Gray, Byte>();

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

return grayImage;

}

public static Mat toBinary(Mat img, int threshold, int maxValue)

{

Mat tmp = new Mat();

try

{

tmp = ImgOps.RGBtoGrey(img).Mat;

CvInvoke.Threshold(tmp, tmp, threshold, maxValue,0);

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

return tmp;

}

public static Mat InterpolationResize(Mat img, int w, int h)

{

try

{

CvInvoke.Resize(img, img, new Size(w, h), (double)Inter.Cubic);

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

return img;

}

public static Mat ContrastAlignment(Image<Gray, Byte> img)

{

Mat result = new Mat();

CvInvoke.CLAHE(img, 4, new Size(16, 16), result);

return result;

}

public static Image<Gray, Byte> RGBFilter(Image<Hsv, Byte> input, double Hmin, double Hmax, double Smin, double Smax, double Vmin, double Vmax)

{

Bitmap b = input.Bitmap;

Hsv lowerLimit = new Hsv(Hmin, Smin, Vmin);

Hsv upperLimit = new Hsv(Hmax, Smax, Vmax);

Image<Gray, byte> result = input.InRange(lowerLimit, upperLimit);

return result;

}

public static Image<Bgr, byte> cannydetect(Image<Bgr, byte> img)

{

Mat src = img.Mat;

Mat gray = new Mat();

CvInvoke.CvtColor(src, gray, Emgu.CV.CvEnum.ColorConversion.Bgr2Gray);

Mat edges = new Mat();

CvInvoke.Canny(gray, edges, 60, 60 \* 3);

Image<Bgr, byte> res = new Image<Bgr, byte>(edges.Bitmap);

return res;

}

}

MainForm

public partial class MainForm : Form

{

VideoFolder vidFolder;

ImageFolder trainFolder;

ImageFolder testFolder;

ImageFolder imageFolder;

XmlFolder xmlFolder;

List<string> folders\_to\_detect = new List<string>();

string path\_to\_cutted\_images;

string path\_to\_save\_detected\_images;

string path\_to\_detect\_images;

int images\_count\_to\_detect;

string path\_to\_cascades;

string path\_to\_results\_save;

int aim;

CNN network;

ResultExport exp;

public MainForm()

{

InitializeComponent();

if (Properties.Settings.Default.last\_path\_to\_videos != "")

{

vidFolder = new VideoFolder();

vidFolder.Load(Properties.Settings.Default.last\_path\_to\_videos);

lVideos\_count.Text = "Found " + vidFolder.getCount();

tb\_videos\_path.Text = vidFolder.getPath();

}

if (Properties.Settings.Default.last\_path\_for\_images\_to\_save != "")

{

var path = Properties.Settings.Default.last\_path\_for\_images\_to\_save;

tb\_images\_to\_save\_path.Text = path;

}

if ((Properties.Settings.Default.last\_path\_for\_images\_to\_detect != "") && (Directory.Exists(Properties.Settings.Default.last\_path\_for\_images\_to\_detect)))

{

path\_to\_detect\_images = Properties.Settings.Default.last\_path\_for\_images\_to\_detect;

tb\_images\_to\_detect\_path.Text = path\_to\_detect\_images;

folders\_to\_detect.Add(path\_to\_detect\_images);

foreach (string s in Directory.GetDirectories(path\_to\_detect\_images))

folders\_to\_detect.Add(s);

lFoldersToDetectCount.Text = folders\_to\_detect.Count.ToString();

images\_count\_to\_detect = 0;

foreach (string p in folders\_to\_detect)

{

string[] dirs = Directory.GetFiles(p, "\*.jpg");

images\_count\_to\_detect += dirs.Length;

}

lImagesToDetectCount.Text = images\_count\_to\_detect.ToString();

}

if (Properties.Settings.Default.last\_path\_to\_cascades != "")

{

path\_to\_cascades = Properties.Settings.Default.last\_path\_to\_cascades;

if (path\_to\_cascades.Length != 0)

{

xmlFolder = new XmlFolder();

xmlFolder.Load(path\_to\_cascades);

tb\_cascade\_path.Text = path\_to\_cascades;

if (xmlFolder.GetCount() > 0)

{

lCascadeLoaded.Text = "Loaded " + xmlFolder.GetCount().ToString();

}

else

{

lCascadeLoaded.Text = "Not Exists";

}

tb\_cascade\_path.Text = Properties.Settings.Default.last\_path\_to\_cascades;

}

else

{

lCascadeLoaded.Text = "Not loaded";

}

}

if (Properties.Settings.Default.last\_path\_for\_detected\_images\_to\_save != "")

{

var path = Properties.Settings.Default.last\_path\_for\_detected\_images\_to\_save;

tb\_detected\_images\_to\_save\_path.Text = path;

}

if (Properties.Settings.Default.last\_path\_to\_learn\_pictures != "")

{

trainFolder = new ImageFolder();

trainFolder.Load(Properties.Settings.Default.last\_path\_to\_learn\_pictures);

lLearn\_count.Text = "Found " + trainFolder.GetCount();

tb\_train\_imgs\_path.Text = trainFolder.GetPath();

}

if (Properties.Settings.Default.last\_path\_to\_test\_pictures != "")

{

testFolder = new ImageFolder();

testFolder.Load(Properties.Settings.Default.last\_path\_to\_test\_pictures);

lTest\_count.Text = "Found " + testFolder.GetCount();

tb\_test\_imgs\_path.Text = testFolder.GetPath();

}

if (Properties.Settings.Default.last\_path\_to\_test\_pictures != "")

{

imageFolder = new ImageFolder();

imageFolder.Load(Properties.Settings.Default.last\_path\_to\_pictures);

lImages.Text = "Found " + imageFolder.GetCount();

tb\_imgs\_path.Text = imageFolder.GetPath();

}

if (Properties.Settings.Default.last\_path\_to\_results\_save != "")

{

path\_to\_results\_save = Properties.Settings.Default.last\_path\_to\_results\_save;

if (Directory.Exists(path\_to\_results\_save))

{

tb\_result\_save\_path.Text = path\_to\_results\_save;

}

}

if (Properties.Settings.Default.last\_path\_to\_network != "")

{

lNetwork.Text = "exists";

var path = Properties.Settings.Default.last\_path\_to\_network;

tb\_network\_path.Text = path;

network = new CNN(path);

}

else

{

network = new CNN("");

}

aim = Int32.Parse(tb\_network\_acc.Text);

//Properties.Settings.Default.is\_opened\_first\_time = true;

/\*if (Properties.Settings.Default.is\_opened\_first\_time)

{

MessageBox.Show("Hello new User");

Properties.Settings.Default.is\_opened\_first\_time = false;

Properties.Settings.Default.Save();

}\*/

}

private void btn\_videos\_open\_Click(object sender, EventArgs e)

{

var folder = OpenVideoFolderFileDialog.openFolder();

if (folder.getPath() != "")

{

vidFolder = folder;

if (vidFolder.getCount() > 0)

{

lVideos\_count.Text = "Found " + vidFolder.getCount() + " videos";

tb\_videos\_path.Text = vidFolder.getPath();

Properties.Settings.Default.last\_path\_to\_videos = vidFolder.getPath();

Properties.Settings.Default.Save();

Properties.Settings.Default.Upgrade();

}

else

lVideos\_count.Text = "Videos not found";

}

}

private void btn\_images\_to\_save\_open\_Click(object sender, EventArgs e)

{

var dialog = new FolderBrowserDialog

{

SelectedPath = tb\_images\_to\_save\_path.Text

};

if (dialog.ShowDialog() == DialogResult.OK)

{

path\_to\_cutted\_images = dialog.SelectedPath;

tb\_images\_to\_save\_path.Text = path\_to\_cutted\_images;

}

Properties.Settings.Default.last\_path\_for\_images\_to\_save = path\_to\_cutted\_images;

Properties.Settings.Default.Save();

Properties.Settings.Default.Upgrade();

}

private void btn\_convert\_videos\_Click(object sender, EventArgs e)

{

if (tb\_videos\_path.Text == "")

{

MessageBox.Show("Path to videos is empty");

return;

}

if (vidFolder.getCount() == 0)

{

MessageBox.Show("Video Folder is empty");

return;

}

if (tb\_images\_to\_save\_path.Text == "")

{

MessageBox.Show("Please choose folder for image save");

return;

}

MessageBox.Show("Please wait till the end of conversion");

foreach (string videopath in vidFolder.videoArray)

{

//int fps = 5;

FFMPEGConverter conv = new FFMPEGConverter(videopath, tb\_images\_to\_save\_path.Text);

bool rewrite = false;

if (conv.ConvertAll(rewrite) == false)

{

MessageBox.Show("Something went wrong with" + videopath);

}

}

lFoldersToDetectCount.Text = folders\_to\_detect.Count.ToString();

images\_count\_to\_detect = 0;

foreach (string p in folders\_to\_detect)

{

string[] dirs = Directory.GetFiles(p, "\*.jpg");

images\_count\_to\_detect += dirs.Length;

}

lImagesToDetectCount.Text = images\_count\_to\_detect.ToString();

}

private void btn\_cascade\_open\_Click(object sender, EventArgs e)

{

xmlFolder = OpenXmlFolderFileDialog.openFolder();

path\_to\_cascades = xmlFolder.GetFolderPath();

tb\_cascade\_path.Text = path\_to\_cascades;

if (xmlFolder.GetCount() > 0)

{

lCascadeLoaded.Text = "Loaded " + xmlFolder.GetCount().ToString();

}

else

{

lCascadeLoaded.Text = "Not Exists";

}

Properties.Settings.Default.last\_path\_to\_cascades = path\_to\_cascades;

Properties.Settings.Default.Save();

Properties.Settings.Default.Upgrade();

}

private void btn\_images\_to\_detect\_open\_Click(object sender, EventArgs e)

{

var dialog = new FolderBrowserDialog

{

SelectedPath = tb\_images\_to\_detect\_path.Text

};

if (dialog.ShowDialog() == DialogResult.OK)

{

path\_to\_detect\_images = dialog.SelectedPath;

tb\_images\_to\_detect\_path.Text = path\_to\_detect\_images;

folders\_to\_detect.Clear();

folders\_to\_detect.Add(path\_to\_detect\_images);

foreach (string s in Directory.GetDirectories(path\_to\_detect\_images))

folders\_to\_detect.Add(s);

lFoldersToDetectCount.Text = folders\_to\_detect.Count.ToString();

int count = 0;

foreach (string p in folders\_to\_detect)

{

string[] dirs = Directory.GetFiles(p, "\*.jpg");

count += dirs.Length;

}

lImagesToDetectCount.Text = count.ToString();

}

Properties.Settings.Default.last\_path\_for\_images\_to\_detect = path\_to\_detect\_images;

Properties.Settings.Default.Save();

Properties.Settings.Default.Upgrade();

}

private void btn\_Haar\_Detect\_Click(object sender, EventArgs e)

{

foreach (string path in folders\_to\_detect)

{

DetectFolder.DetectAll(path, xmlFolder.GetAll(), tb\_detected\_images\_to\_save\_path.Text);

}

}

private void btn\_detected\_images\_to\_save\_open\_Click(object sender, EventArgs e)

{

var dialog = new FolderBrowserDialog

{

SelectedPath = tb\_detected\_images\_to\_save\_path.Text

};

if (dialog.ShowDialog() == DialogResult.OK)

{

path\_to\_save\_detected\_images = dialog.SelectedPath;

tb\_detected\_images\_to\_save\_path.Text = path\_to\_save\_detected\_images;

}

Properties.Settings.Default.last\_path\_for\_detected\_images\_to\_save = path\_to\_save\_detected\_images;

Properties.Settings.Default.Save();

Properties.Settings.Default.Upgrade();

}

private void btn\_img1\_resize\_Click(object sender, EventArgs e)

{

// img = ImgOps.InterpolationResize(img, 50, 50);

// {

// int w = Convert.ToInt32(tb\_resize\_x.Text);

// int h = Convert.ToInt32(tb\_resize\_y.Text);

// imgbox3.Image = ImgOps.InterpolationResize(new Image<Bgr, byte>(imgbox1.Image.Bitmap).Mat, w, h);

// }

// else

// {

MessageBox.Show("Plz load img");

// }

}

private void btn\_img1\_clahe\_Click(object sender, EventArgs e)

{

MessageBox.Show("Plz load img");

//imgbox1.Image = ImgOps.ContrastAlignment(new Image<Gray, byte>(imgbox1.Image.Bitmap));

}

private void btn\_CNN\_create\_Click(object sender, EventArgs e)

{

int c = network.CreateCNN(32, 32, 1, 11);

lLayers\_count.Text = c.ToString();

lClasses\_count.Text = network.GetClassesCount().ToString();

}

private void tb\_network\_acc\_TextChanged(object sender, EventArgs e)

{

Int32.TryParse(tb\_network\_acc.Text, out aim);

}

private void btn\_CNN\_learn\_Click(object sender, EventArgs e)

{

if (network.IsLearned())

{

lLearned.Text = "Learned";

return;

}

if ((trainFolder != null) && (testFolder != null))

{

double d = network.TeachCNN(trainFolder.GetPath(), testFolder.GetPath(), aim, 0.02, 100, 0.9);

if (d == -1)

MessageBox.Show("Please add layers and try again");

else

lAccuracy.Text = d.ToString();

if (network.IsLearned())

{

lLearned.Text = "Learned";

MessageBox.Show("Network is learned acc is " + lAccuracy.Text);

}

}

}

private void btn\_CNN\_recognize\_Click(object sender, EventArgs e)

{

if (tb\_imgs\_path.Text != "")

{

if (!File.Exists(tb\_imgs\_path.Text))

{

MessageBox.Show("File not exists");

return;

}

Mat im = new Mat(tb\_imgs\_path.Text);

if (im.NumberOfChannels != 1)

{

im = ImgOps.RGBtoGrey(im).Mat;

}

if (im.Size != new Size(32, 32))

{

im = new Image<Gray, byte>(ImgOps.InterpolationResize(im, 32, 32).Bitmap).Mat;

}

Image<Gray, byte> img = new Image<Gray, byte>(im.Bitmap);

string sign = network.Recognize(img.Bytes);

lLayers\_count.Text = network.GetLayersCount().ToString();

MessageBox.Show(sign);

}

else

{

MessageBox.Show("Path to image is empty");

}

}

private void btn\_CNN\_load\_Click(object sender, EventArgs e)

{

int n = network.LoadCNN(tb\_network\_path.Text);

if (n == -1)

{

MessageBox.Show("Input network file is not correct");

return;

}

lLayers\_count.Text = n.ToString();

lClasses\_count.Text = network.GetClassesCount().ToString();

if (network.IsLearned())

lLearned.Text = "Learned";

var p = Path.GetFileName(tb\_network\_path.Text);

p = p.Substring(p.IndexOf('\_') + 1, p.IndexOf('.') - p.IndexOf('\_') - 1);

lAccuracy.Text = p;

}

private void btn\_CNN\_save\_Click(object sender, EventArgs e)

{

var path = network.SaveCNN(tb\_network\_path.Text);

if (path != "")

{

Console.WriteLine(path);

tb\_network\_path.Text = path;

Properties.Settings.Default.last\_path\_to\_network = path;

Properties.Settings.Default.Save();

Properties.Settings.Default.Upgrade();

}

else

MessageBox.Show("Error");

}

private void btn\_autoCompleteAll\_Click(object sender, EventArgs e)

{

btn\_convert\_videos\_Click(sender, e);

btn\_Haar\_Detect\_Click(sender, e);

}

private void btn\_learn\_resize\_Click(object sender, EventArgs e)

{

foreach (string imgpath in trainFolder.GetAll())

{

Mat tmp = new Mat(imgpath);

tmp = ImgOps.InterpolationResize(tmp, 32, 32);

tmp = ImgOps.RGBtoGrey(tmp).Mat;

}

}

private void btn\_train\_imgs\_open\_Click(object sender, EventArgs e)

{

var folder = OpenPictureFolderFileDialog.openFolder();

if (folder.GetPath() != "")

{

trainFolder = folder;

if (trainFolder.GetCount() > 0)

{

lLearn\_count.Text = "Found " + trainFolder.GetCount();

tb\_train\_imgs\_path.Text = trainFolder.GetPath();

Properties.Settings.Default.last\_path\_to\_learn\_pictures = trainFolder.GetPath();

Properties.Settings.Default.Save();

Properties.Settings.Default.Upgrade();

}

else

lLearn\_count.Text = "Not found";

}

}

private void btn\_test\_imgs\_open\_Click(object sender, EventArgs e)

{

var folder = OpenPictureFolderFileDialog.openFolder();

if (folder.GetPath() != "")

{

testFolder = folder;

if (testFolder.GetCount() > 0)

{

lTest\_count.Text = "Found " + testFolder.GetCount();

tb\_test\_imgs\_path.Text = testFolder.GetPath();

Properties.Settings.Default.last\_path\_to\_test\_pictures = testFolder.GetPath();

Properties.Settings.Default.Save();

Properties.Settings.Default.Upgrade();

}

else

lTest\_count.Text = "Not found";

}

}

private void tb\_train\_imgs\_path\_TextChanged(object sender, EventArgs e)

{

trainFolder.SetPath(tb\_train\_imgs\_path.Text);

trainFolder.Load(tb\_train\_imgs\_path.Text);

lLearn\_count.Text = "Found " + trainFolder.GetCount();

Properties.Settings.Default.last\_path\_to\_learn\_pictures = trainFolder.GetPath();

Properties.Settings.Default.Save();

Properties.Settings.Default.Upgrade();

}

private void tb\_test\_imgs\_path\_TextChanged(object sender, EventArgs e)

{

testFolder.SetPath(tb\_test\_imgs\_path.Text);

testFolder.Load(tb\_test\_imgs\_path.Text);

lTest\_count.Text = "Found " + testFolder.GetCount();

Properties.Settings.Default.last\_path\_to\_test\_pictures = testFolder.GetPath();

Properties.Settings.Default.Save();

Properties.Settings.Default.Upgrade();

}

private void btn\_toGrey\_Click(object sender, EventArgs e)

{

}

private void btn\_imgs\_open\_Click(object sender, EventArgs e)

{

var path = OpenPictureFileDialog.openFile();

tb\_imgs\_path.Text = path;

Properties.Settings.Default.last\_path\_to\_pictures = path;

Properties.Settings.Default.Save();

Properties.Settings.Default.Upgrade();

/\*imageFolder = OpenPictureFolderFileDialog.openFolder();

if (imageFolder.getCount() > 0)

{

lImages.Text = "Found " + imageFolder.getCount();

tb\_imgs\_path.Text = imageFolder.getPath();

Properties.Settings.Default.last\_path\_to\_pictures = imageFolder.getPath();

Properties.Settings.Default.Save();

Properties.Settings.Default.Upgrade();

}

else

lImages.Text = "Not found";\*/

}

private void btn\_network\_open\_Click(object sender, EventArgs e)

{

OpenFileDialog dlg = new OpenFileDialog();

string path;

dlg.Title = "Open Network";

dlg.Filter = "txt files (\*.txt)|\*.txt";

if (dlg.ShowDialog() == DialogResult.OK)

{

path = dlg.FileName;

tb\_network\_path.Text = path;

Properties.Settings.Default.last\_path\_to\_network = path;

Properties.Settings.Default.Save();

Properties.Settings.Default.Upgrade();

}

}

private void tb\_network\_path\_TextChanged(object sender, EventArgs e)

{

}

private void btn\_results\_save\_Click(object sender, EventArgs e)

{

Result[] arr = new Result[10];

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

{

Result r = new Result(i, 0, "123", "456", new DateTime(2019, 5, 27), 1, 100);

arr[i] = r;

}

exp = new ResultExport("l", "p", "s", tb\_result\_save\_path.Text);

exp.ExportToFile(arr);

MessageBox.Show("Done");

}

private void btn\_results\_to\_save\_open\_Click(object sender, EventArgs e)

{

var dialog = new FolderBrowserDialog

{

SelectedPath = tb\_result\_save\_path.Text

};

if (dialog.ShowDialog() == DialogResult.OK)

{

path\_to\_results\_save = dialog.SelectedPath;

tb\_result\_save\_path.Text = path\_to\_results\_save;

Properties.Settings.Default.last\_path\_to\_results\_save = path\_to\_results\_save;

Properties.Settings.Default.Save();

Properties.Settings.Default.Upgrade();

}

}

private void btn\_results\_upload\_Click(object sender, EventArgs e)

{

exp = new ResultExport("l", "p", "s", tb\_result\_save\_path.Text);

exp.ExportFolder(tb\_result\_save\_path.Text);

}

private void btn\_ROI\_makeAll\_Click(object sender, EventArgs e)

{

}

}