ДОДАТОК А

Вихідний код програми

Program.cs

```
class Program
{
   public void Main(string[] args)
   {
      if (args.Length < 1)
      {
         Console.WriteLine("Usage: VisualInspector <image_path> [<output_dir>] [<yolo_path>] [<unet_path>]");
         return;
      }
      string image = args[0];
      string outputDir = args.Length > 1 ? args[1] : @"C:\Dev\we-vcsel-subassembly-defect-analysis\VisualInspection_Dir\Dir_DotNet\VisualInspection_output";
      string yoloPath = args.Length > 2 ? args[2] : @"C:\Dev\we-vcsel-subassembly-defect-analysis\VisualInspection_Dir\Dir_DotNet\best_v4_3.onnx";
      string unetPath = args.Length > 3 ? args[3] : @"C:\Dev\we-vcsel-subassembly-defect-analysis\VisualInspection_Dir\Dir_DotNet\unet_v5.onnx";
      Inspect(image, outputDir, yoloPath, unetPath);
   }
   static void Inspect(string image, string OutputDir, string YoloPath = "", string UnetPath = "")
   {
      if (!Directory.Exists(OutputDir))
```

```
            {
                Directory.CreateDirectory(OutputDir);
            }
            DateTime startTime = DateTime.Now;
            Console.WriteLine($"Start time: {startTime}");
            YoloClassifier.ClassifyObjects(YoloPath, image, OutputDir);
            DateTime endTime = DateTime.Now;
            Console.WriteLine($"YOLO processing took: {endTime - startTime}");
            foreach (var file in Directory.GetFiles(Path.Combine(OutputDir, "Yolo_output", "Yolo_output_wires"), "*.jpg"))
            {
                DateTime startTimeUnet = DateTime.Now;
                UnetSegmentator.SegmentImage(UnetPath, file, OutputDir);
                DateTime endTimeUnet = DateTime.Now;
                Console.WriteLine($"Unet processing took: {endTimeUnet - startTimeUnet}");
            }
            Console.WriteLine("Total processing time: " + (DateTime.Now - startTime));
        }
    }
```

YoloClassifier.cs

```
    class YoloClassifier
    {
        public static void ClassifyObjects(string modelPath, string imagePath, string OutputDir)
        {
```

```
            OutputDir = Path.Combine(OutputDir, "Yolo_output");
            string imageName = imagePath.Split(Path.DirectorySeparatorChar).Last();
            imageName = imageName.Split('.').First() + "_result";
            Console.WriteLine(imageName);
            using var yolo = new Yolo(new YoloDotNet.Models.YoloOptions
            {
               OnnxModel = modelPath,
               ModelType = YoloDotNet.Enums.ModelType.ObjectDetection,
               Cuda = false,
               GpuId = 0,
               PrimeGpu = false,
            });
            using var image = SKImage.FromEncodedData(imagePath);
            var results = yolo.RunObjectDetection(image, confidence: 0.2, iou: 0.3);
            var unwantedLabels = new HashSet<string> { "Text" };
            results = results.Where(r => !unwantedLabels.Contains(r.Label.Name)).ToList();
            using var resultImage = image.Draw(results);

            if (!Directory.Exists(OutputDir))
            {
               Directory.CreateDirectory(OutputDir);
            }
            resultImage.Save(Path.Combine(OutputDir, $"{imageName}.jpg"), SKEncodedImageFormat.Jpeg, 80);
            SaveJson(results, Path.Combine(OutputDir, $"{imageName}.json"));
            CropWires(results, image, Path.Combine(OutputDir, "Yolo_output_wires"));
```

```
        }
        public static void SaveJson(List<YoloDotNet.Models.ObjectDetection>
results, string filePath)
        {
            var simpleResults = results.Select(r => new YoloDto
            {
                Label = new { r.Label.Index, r.Label.Name, r.Label.Color },
                Confidence = r.Confidence,
                X1 = r.BoundingBox.Left,
                Y1 = r.BoundingBox.Top,
                X2 = r.BoundingBox.Right,
                Y2 = r.BoundingBox.Bottom
            }).ToList();
            var settings = new JsonSerializerSettings
            {
                ReferenceLoopHandling = ReferenceLoopHandling.Ignore,
                Formatting = Formatting.Indented
            };
            var json = JsonConvert.SerializeObject(simpleResults, settings);
            File.WriteAllText(filePath, json);
        }
        public                    static                    void
CropWires(List<YoloDotNet.Models.ObjectDetection> results, SKImage image,
string outputDirectory)
        {
            if (!Directory.Exists(outputDirectory))
            {
                Directory.CreateDirectory(outputDirectory);
            }
            else
```

```
            {
                foreach (var file in Directory.GetFiles(outputDirectory).Where(f
=> f.EndsWith(".jpg") || f.EndsWith(".png")))
                {
                    File.Delete(file);
                }
            }
            int i = 0;
            foreach (var result in results.Where(r => r.Label.Index == 1))
            {
                var bbox = new SKRectI(
                    result.BoundingBox.Left - 5,
                    result.BoundingBox.Top - 5,
                    result.BoundingBox.Right + 5,
                    result.BoundingBox.Bottom + 5
                );
                using var croppedImage = image.Subset(bbox);
                string fileName = $"crop_{i++}.jpg";
                croppedImage.Save(Path.Combine(outputDirectory,      fileName),
SKEncodedImageFormat.Jpeg, 80);
            }
        }
    }
```

UnetSegmentation.cs

```
    class UnetSegmentator
    {
        public static void SegmentImage(string modelPath, string imagePath,
string OutputDir)
```

```
        {
            float threshold = 0.7f;
            string imageName = Path.GetFileNameWithoutExtension(imagePath);
            OutputDir = Path.Combine(OutputDir, "U-net_output", imageName);

            if (!Directory.Exists(OutputDir))
            {
                Directory.CreateDirectory(OutputDir);
            }
            else
            {
                foreach (var file in Directory.GetFiles(OutputDir).Where(f => f.EndsWith(".png")))
                {
                    File.Delete(file);
                }
            }
            var session = new InferenceSession(modelPath);
            Bitmap origImage = new Bitmap(imagePath);
            Bitmap resized = new Bitmap(origImage, new System.Drawing.Size(256, 256));
            var inputTensor = PreprocessImage(resized);
            var inputName = session.InputMetadata.Keys.First();
            var inputs = new List<NamedOnnxValue>
            {
                NamedOnnxValue.CreateFromTensor(inputName, inputTensor)
            };
            using var results = session.Run(inputs);
            var output = results.First().AsTensor<float>();
```

```
            var mask = PostprocessOutput(output, threshold);
            Bitmap maskBitmap = MaskToBitmap(mask, resized.Width, resized.Height);
            Bitmap maskResized = new Bitmap(maskBitmap, origImage.Size);
            Bitmap overlay = OverlayMask(origImage, maskResized);
            overlay.Save(Path.Combine(OutputDir, "U-net_rez_binary_mask.png"));
            Mat maskMat = BitmapConverter.ToMat(maskBitmap);
            Mat filtered = FilterLargestComponents(maskMat, maxComponents: 2, minArea: 300);
            Cv2.ImWrite(Path.Combine(OutputDir, "U-net_rez_clean_mask.png"), filtered);
            Mat cleaned = new Mat();
            Mat kernel = Cv2.GetStructuringElement(MorphShapes.Rect, new OpenCvSharp.Size(3, 3));
            Cv2.MorphologyEx(filtered, cleaned, MorphTypes.Open, kernel);
            Cv2.MorphologyEx(cleaned, cleaned, MorphTypes.Close, kernel);
            Mat pr = cleaned.Clone();
            PreprocessForSkeleton(pr);
            Mat skeleton = ZhangSuenThinning(pr);
            Mat pp = RemoveSmallBranches(skeleton, minLength: 10);
            Cv2.ImWrite(Path.Combine(OutputDir, "U-net_rez_skeleton.png"), pp);
            var endpoints = FindEndpoints(pp);
            Mat endpointsMat = new Mat(filtered.Size(), MatType.CV_8UC3, Scalar.All(0));
            Cv2.CvtColor(filtered, endpointsMat, ColorConversionCodes.GRAY2BGR);
            foreach (OpenCvSharp.Point pt in endpoints)
                Cv2.Circle(endpointsMat, center: pt, 3, new Scalar(0, 0, 200), -1);
```

```
            Cv2.ImWrite(Path.Combine(OutputDir, "U-
net_rez_mask_with_endpoints.png"), endpointsMat);
            if (endpoints.Count != 2)
                Console.WriteLine($"{endpoints.Count} Possible broken wire
detected!");
            else
                Console.WriteLine($"{endpoints.Count} Wire appears intact.");
        }
        private static DenseTensor<float> PreprocessImage(Bitmap bmp)
        {
            var tensor = new DenseTensor<float>(new[] { 1, 3, bmp.Height,
bmp.Width });
            for (int y = 0; y < bmp.Height; y++)
            {
                for (int x = 0; x < bmp.Width; x++)
                {
                    Color pixel = bmp.GetPixel(x, y);
                    tensor[0, 0, y, x] = ((pixel.R / 255f) - 0.5f) / 0.5f;
                    tensor[0, 1, y, x] = ((pixel.G / 255f) - 0.5f) / 0.5f;
                    tensor[0, 2, y, x] = ((pixel.B / 255f) - 0.5f) / 0.5f;
                }
            }
            return tensor;
        }
        private static byte[,] PostprocessOutput(Tensor<float> output, float
threshold)
        {
            int h = output.Dimensions[2];
            int w = output.Dimensions[3];
            byte[,] mask = new byte[h, w];
```

```
            for (int y = 0; y < h; y++)
                for (int x = 0; x < w; x++)
                    mask[y, x] = output[0, 0, y, x] > threshold ? (byte)255 : (byte)0;
            return mask;
        }
        private static Bitmap MaskToBitmap(byte[,] mask, int width, int height)
        {
            Bitmap bmp = new Bitmap(width, height, PixelFormat.Format8bppIndexed);
            ColorPalette palette = bmp.Palette;
            for (int i = 0; i < 256; i++) palette.Entries[i] = Color.FromArgb(i, i, i);
            bmp.Palette = palette;
            BitmapData data = bmp.LockBits(new Rectangle(0, 0, width, height), ImageLockMode.WriteOnly, bmp.PixelFormat);
            unsafe
            {
                for (int y = 0; y < height; y++)
                {
                    byte* row = (byte*)data.Scan0 + y * data.Stride;
                    for (int x = 0; x < width; x++)
                        row[x] = mask[y, x];
                }
            }
            bmp.UnlockBits(data);
            return bmp;
        }
        private static Bitmap OverlayMask(Bitmap image, Bitmap mask)
        {
            Bitmap overlay = new Bitmap(image.Width, image.Height, PixelFormat.Format24bppRgb);
```

```
            using (Graphics g = Graphics.FromImage(overlay))
            {
                g.DrawImage(image, 0, 0);
                for (int y = 0; y < image.Height; y++)
                {
                    for (int x = 0; x < image.Width; x++)
                    {
                        Color maskPixel = mask.GetPixel(x, y);
                        if (maskPixel.R > 0)
                        {
                            Color orig = image.GetPixel(x, y);
                            Color blended = Color.FromArgb(
                                (int)(orig.R * 0.7 + 255 * 0.3),
                                (int)(orig.G * 0.7),
                                (int)(orig.B * 0.7)
                            );
                            overlay.SetPixel(x, y, blended);
                        }
                    }
                }
            }
            return overlay;
        }
        private static Mat FilterLargestComponents(Mat mask, int maxComponents, int minArea)
        {
            Mat labels = new Mat();
            Mat stats = new Mat();
            Mat centroids = new Mat();
```

```
            int nLabels = Cv2.ConnectedComponentsWithStats(mask, labels,
stats, centroids);

            List<(int idx, int area)> areas = new();
            for (int i = 1; i < nLabels; i++)
               areas.Add((i,                                    stats.At<int>(i,
(int)ConnectedComponentsTypes.Area)));
            var largest = areas.OrderByDescending(a => a.area)
                        .Where(a => a.area >= minArea)
                        .Take(maxComponents)
                        .Select(a => a.idx)
                        .ToList();

            Console.WriteLine($"Largest components found: {largest.Count}");
            Mat filtered = Mat.Zeros(mask.Size(), MatType.CV_8UC1);
            for (int y = 0; y < labels.Rows; y++)
            {
               for (int x = 0; x < labels.Cols; x++)
               {
                  int label = labels.At<int>(y, x);
                  if (largest.Contains(label))
                     filtered.Set<byte>(y, x, 255);
               }
            }
            return filtered;
         }
         private static Mat PreprocessForSkeleton(Mat mask)
         {
            Mat processed = mask.Clone();
```

```
            Mat kernel = Cv2.GetStructuringElement(MorphShapes.Rect, new
OpenCvSharp.Size(5, 5));
            Cv2.MorphologyEx(processed, processed, MorphTypes.Close,
kernel, iterations: 2);
            Mat labels = new Mat();
            Mat stats = new Mat();
            Mat centroids = new Mat();
            int nLabels = Cv2.ConnectedComponentsWithStats(processed, labels,
stats, centroids);
            int maxArea = 0, maxLabel = 0;
            for (int i = 1; i < nLabels; i++)
            {
               int area = stats.At<int>(i, (int)ConnectedComponentsTypes.Area);
               if (area > maxArea)
               {
                  maxArea = area;
                  maxLabel = i;
               }
            }
            Mat largest = Mat.Zeros(processed.Size(), MatType.CV_8UC1);
            for (int y = 0; y < labels.Rows; y++)
               for (int x = 0; x < labels.Cols; x++)
                  if (labels.At<int>(y, x) == maxLabel)
                     largest.Set<byte>(y, x, 255);
            return largest;
         }
         private static Mat RemoveSmallBranches(Mat skeleton, int minLength
= 10)
         {
            Mat labels = new Mat();
```

```
            int nLabels = Cv2.ConnectedComponents(skeleton, labels);
            Mat cleaned = Mat.Zeros(skeleton.Size(), MatType.CV_8UC1);
            for (int i = 1; i < nLabels; i++)
            {
                Mat componentMask = new Mat();
                Cv2.Compare(labels, i, componentMask, CmpType.EQ);
                int area = Cv2.CountNonZero(componentMask);
                if (area >= minLength)
                    cleaned.SetTo(255, componentMask);
            }
            return cleaned;
        }
        private static List<OpenCvSharp.Point> FindEndpoints(Mat skeleton)
        {
            List<OpenCvSharp.Point> endpoints = new();
            int[,] kernel = { { 1, 1, 1 }, { 1, 10, 1 }, { 1, 1, 1 } };
            for (int y = 1; y < skeleton.Rows - 1; y++)
            {
                for (int x = 1; x < skeleton.Cols - 1; x++)
                {
                    if (skeleton.At<byte>(y, x) == 0) continue;
                    int sum = 0;
                    for (int ky = -1; ky <= 1; ky++)
                        for (int kx = -1; kx <= 1; kx++)
                            sum += skeleton.At<byte>(y + ky, x + kx) > 0 ? kernel[ky
+ 1, kx + 1] : 0;
                    if (sum == 11) endpoints.Add(new OpenCvSharp.Point(x, y));
                }
            }
            return endpoints;
```

```
}
private static Mat ZhangSuenThinning(Mat src)
{
    Mat img = src.Clone();
    if (img.Type() != MatType.CV_8UC1)
        Cv2.CvtColor(img, img, ColorConversionCodes.BGR2GRAY);
    Cv2.Threshold(img, img, 127, 1, ThresholdTypes.Binary);
    bool changed;
    Mat prev = new Mat();
    do
    {
        prev = img.Clone();
        for (int iter = 0; iter < 2; iter++)
        {
            Mat marker = Mat.Zeros(img.Size(), MatType.CV_8UC1);
            for (int y = 1; y < img.Rows - 1; y++)
            {
                for (int x = 1; x < img.Cols - 1; x++)
                {
                    byte p2 = img.At<byte>(y - 1, x);
                    byte p3 = img.At<byte>(y - 1, x + 1);
                    byte p4 = img.At<byte>(y, x + 1);
                    byte p5 = img.At<byte>(y + 1, x + 1);
                    byte p6 = img.At<byte>(y + 1, x);
                    byte p7 = img.At<byte>(y + 1, x - 1);
                    byte p8 = img.At<byte>(y, x - 1);
                    byte p9 = img.At<byte>(y - 1, x - 1);

                    int A = (p2 == 0 && p3 == 1 ? 1 : 0) +
                            (p3 == 0 && p4 == 1 ? 1 : 0) +
```

```
                        (p4 == 0 && p5 == 1 ? 1 : 0) +
                        (p5 == 0 && p6 == 1 ? 1 : 0) +
                        (p6 == 0 && p7 == 1 ? 1 : 0) +
                        (p7 == 0 && p8 == 1 ? 1 : 0) +
                        (p8 == 0 && p9 == 1 ? 1 : 0) +
                        (p9 == 0 && p2 == 1 ? 1 : 0);
                    int B = p2 + p3 + p4 + p5 + p6 + p7 + p8 + p9;
                    byte m = img.At<byte>(y, x);
                    if (m == 1 &&
                        B >= 2 && B <= 6 &&
                        A == 1 &&
                        ((iter == 0 && (p2 * p4 * p6 == 0) && (p4 * p6 * p8 ==
0)) ||
                        (iter == 1 && (p2 * p4 * p8 == 0) && (p2 * p6 * p8 ==
0))))
                    {
                        marker.Set<byte>(y, x, 1);
                    }
                }
            }
            img -= marker;
        }
        Mat diff = new Mat();
        Cv2.Compare(img, prev, diff, CmpType.NE);
        changed = Cv2.CountNonZero(diff) > 0;
    } while (changed);
    img *= 255;
    return img;
  }
}
```

ДОДАТОК Б

Відомість кваліфікаційної роботи

Позначення	Найменування	Дод. відомості
	Текстові документи	
1.	Пояснювальна записка	88 с.
	Інші документи	
2.	Презентаційні матеріали	__ плакатів

.	.	Прізвище та ініціали.	Підп.	Дата			
Розробив		Сагалович Т.О.			Інтелектуальний аналіз зображень кристалів мікроелектронних компонентів для виявлення дефектів із використанням глибоких згорткових мереж і YOLO	Шифр групи	Код напр./спец.
Перевірив		Шевченко О.Ю.				ІТШІ-21-3	122
Н.контр.		Малєєва І.А.				ХНУРЕ кафедра ШІ	
Затв.		Золотухін О.В.					