Implementation of Image Binarizer

Software Engineering, WS19/20, Frankfurt University of Applied Sciences, Frankfurt, Germany

Daniel Okoyo, Uchechukwu John Emenike, Arinze Okpagu
danielokoyo1@gmail.com
jonemenike2000@gmail.com
bryancoins@yahoo.co.uk

Abstract: One of the key features of binarization is converting pixel image to binary image. It provides sharper and clearer contours of various objects present in the image. This feature extraction has proven helpful in the learning of AI models, optical character recognition, facial recognition, document analysis, quality inspection of materials and various image processing tasks. The use of binary images decreases computational load for these applications. Accuracy of binarization methods are affected by factors such as shadows, non-uniform illumination, low contrast, large signal-dependent noise etc This paper aims to present some of the more recent approaches in the field and compare their results with some of the classic algorithms.

Keywords: binarization, thresholding, histogram, pixel, bitmap

I. Introduction

This project has been carried out as part of the Software Engineering module of the Frankfurt University of Applied Sciences. Among other objectives, it serves to ensure that students can design and develop software projects and are familiar with software development cycle. Technically, this project is aimed to improve the current implementation of Image Binarization for the Learning API.

An image is a set of pixels (points) of different colours and each pixel can only be one colour per time. Pixels are so small that they blend seamlessly to produce a complete image. Pixels are the basic building blocks of a digital image. For instance, in colourful images, every pixel's colour is a portion of the primary colours mixed to achieve the original image and describes the respective grey level intensity, which represents how bright it is. This intensity value ranges from 0 to 255. 0 means there is no light at all, which consequently results to a black colour. The maximum value in the range, which is 255, means that the maximum possible amount of light is applied, producing white colour.

Image binarization converts a pixel image to a binary image. The binary image is produced by quantization of the image grey levels to only two values, usually 0 and 1. The process essentially reduces the information contained within the image from 256 shades of grey to two-pixel values. In the old days binarization was important for sending faxes [1]. These days, it is commonly employed in digitalizing text, shape recognition in artificial intelligence, scene matching, image segmentation etc. For instance, it is the most important step in pre-processing of scanned documents to save all or maximum subcomponents such us text, background and image.

II. Image Segmentation

Image binarization is intrinsically linked to the image histogram. An image histogram is the

graph plotting the frequency of occurrence of different pixel intensities in a digital image. The information contained in the graph is a graphical representation of pixel value distributions. Image histograms can be analysed for peaks and/or valleys [2].

Figure 1: Sample Image

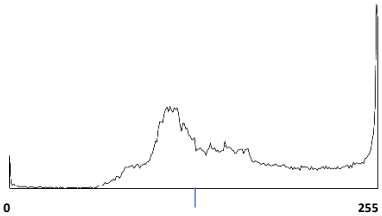


Figure 2: Sample Image Histogram [3]

The process of binarization works by identifying a threshold value in the histogram. This is a value that effectively divides the histogram into two parts, each representing one of two objects: background and foreground. Thresholding algorithms uses some type of information to determine the threshold. Sometimes the information is statistical and uses the mean value of the image pixel intensities. Other times information is in the form of shape characteristics of the histogram [1]. Given the sample image histogram, a suitable threshold for separating the background and foreground will be found somewhere in between the two peaks in the histogram. A threshold value of 120 can be chosen, having considered histogram. To binarize the image, pixels less than 120 are set to 0, while the pixels greater than 120 are set to 1.

Where the distribution is non-uniform, it is unlikely that a good segmentation can be produced by single threshold value. In such case it may still be possible to achieve a good segmentation by applying several individual thresholds. If two dominant peaks characterize the image histogram, it is called a bimodal histogram. Only one threshold is enough for partitioning the image. On the other hand, if an image is composed of multiple types of dark objects on a light background, three or more dominant peaks would characterize the image histogram. In such a case the histogram is called Multimodal histogram and must be segmented by multiple thresholds [1].

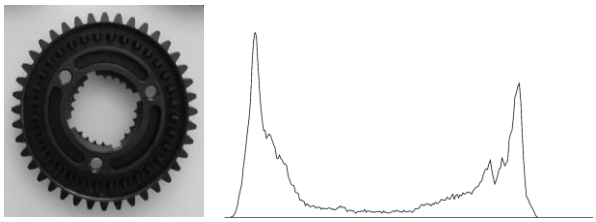


Figure 3: Bimodal Histogram

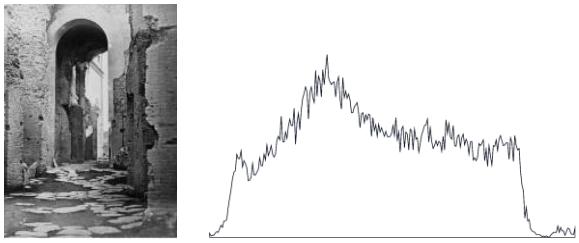


Figure 4: Multimodal Histogram

The most intuitive thresholding approach is Global thresholding. If the threshold depends on local properties of some image regions (e.g., the local average grey value), the thresholding is called local. If the local thresholds are selected independently for each pixel (or groups of pixels), the thresholding is called dynamic or adaptive [1].

The image segmentation process is based on the RGB model used in the display of digital images. The model involves the mixing of 3 primary colours; red, green and blue together in different proportions to make more different colours [4]. With this model, each pixel is represented by one 8-bit byte such that the colour intensity of each pixel ranges from 0 to 255. In this process, if the three colours (red, green, blue) are superimposed

with the least intensity (0,0,0), a black colour is formed and if they are added with the full intensity of light (255,255,255), then white colour is formed.

Choosing a global or localized thresholding algorithm depends on the content to be segmented. Most thresholding algorithms work well on images that contain objects with uniform intensity values on a contrasting background. However, they fail if there is a low contrast between the object and the background.

III. METHOD

This section deals with the methodological competencies which are necessary to implement the image binarizer algorithm.

A) Source Code

The procedures to achieve this objective was carefully implemented using C#, a type-safe object-oriented language that runs on the .NET framework. C# can be used to create Windows client applications, XML Web services, distributed components, client-server applications, database applications, and many robust applications [5].

Language interoperability is a key feature of the .NET Framework. Source code written in C# is compiled into an intermediate language (IL) that conforms to the common language infrastructure (CLI) specification.The IL code generated can interact with code that was generated from the .NET versions of Visual Basic, Visual C++ and any other Common Type Specification (CTS) compliant languages [5].

Figure 5: C# programming language on .NET platform

B) Image Binarization Algorithm

This defines the fundamental algorithm for the image binarization. The LearningApi platform, which comprises of different modules for machine leaning and image processing provided basis for the key Libraries utilized in this process. Here, the package version 1.3.0 was used.

```
<PropertyGroup Condition="'$(Configuration)|$(Platform)'=='Debug|AnyCPU'">
  <DocumentationFile>C:\Uni\Tutor\OtherBranches\se-dystsys-2018-2019-softt
</PropertyGroup>

<ItemGroup>
  <PackageReference Include="LearningApi" Version="1.3.0" />
  <PackageReference Include="System.Drawing.Common" Version="4.5.1" />
</ItemGroup>

</Project>
```

Figure 6: LearningApi Version 1.3.0

Because the following attributes: Width, Height and Size of any given image would be necessary to the algorithm, they were properly defined alongside the Bitmap object towards processing the given image and returning the binary equivalent. The GetPixel and SetPixel methods were used to get and set respectively the constituent RGB pixel values of the given image in a 3-Dimensional array.

```
Bitmap bitmap = new Bitmap(inputImagePath);

int imgWidth = bitmap.Width;
int imgHeight = bitmap.Height;
double[,,] inputData = new double[imgWidth, imgHeight, 3];

for (int i = 0; i < imgWidth; i++)
{
    for (int j = 0; j < imgHeight; j++)
    {
        Color color = bitmap.GetPixel(i, j);
        inputData[i, j, 0] = color.R;
        inputData[i, j, 1] = color.G;
        inputData[i, j, 2] = color.B;
    }
}
```

Figure 7: Image Binarizer Algorithm [6]

The image height, width and threshold are taken as the main parameters which are required to be passed as integer arguments.

```
{
    Dictionary<String, int> imageParams = new Dictionary<string, int>();
    imageParams.Add("imageWidth", imageWidth);
    imageParams.Add("imageHeight", imageHeight);
    imageParams.Add("redThreshold", redThreshold);
    imageParams.Add("greenThreshold", greenThreshold);
    imageParams.Add("blueThreshold", blueThreshold);

    Bitmap bitmap = new Bitmap(inputImagePath);

    int imgWidth = bitmap.Width;
    int imgHeight = bitmap.Height;
    double[,,] inputData = new double[imgWidth, imgHeight, 3];
```

Figure 8: Key Parameters [6]

The IPipelineModule, one of the key components of the ImageBinarizer Library undertakes the conversion of the image file to sequence of 1s and 0s.

```
/// <summary>
/// Method of Interface Ipipline
/// </summary>
/// <param name="data">this is the double data coming from unitest.</param>
/// <param name="ctx">this define the Interface IContext for Data descriptor</param>
/// <returns></returns>
0 references
public double[,,] Run(double[,,] data, IContext ctx)
{
    return GetBinary(data);
}

/// <summary>
/// Gets double array representation of the image.I.E.: 010000111000
/// </summary>
/// <params name="img">Image instance. Typically bitmap.</params>
/// <returns></returns>
2 references
public double[,,] GetBinary(double[,,] data)
{
    Bitmap img = new Bitmap(data.GetLength(0), data.GetLength(1));

    for (int i = 0; i < data.GetLength(0); i++)
    {
        for (int j = 0; j < data.GetLength(1); j++)
        {
            int r = (int)data[i, j, 0];
            int g = (int)data[i, j, 1];
            int b = (int)data[i, j, 2];

//set limits,bytes can hold values from 0 upto 255
            img.SetPixel(i, j, Color.FromArgb(255, r, g, b));
        }
    }

    if (this.m_TargetSize != null)
        img = new Bitmap(img, this.m_TargetSize.Value);
```

Figure 9: IPipelineModule Interface [6]

C) Console Application

The application is deployed as a console application. It is designed with useful prompts to help the user pass (valid) arguments relevant to the binarization functions. These prompts ensure a user-friendly interaction throughout the image binarization process.

```
Welcome to Image Binarizer Application [Version 1.0.3]
Copyright <c> 2020 daenet GmbH, Damir Dobric. All rights reserved.

Do you want to binarize an image?
        Y - Proceed
        N - Exit the application
Your option?
```

Figure 9: Console Application (Welcome Prompt)

Though the application has been built to handle exceptions, these prompts are in place to minimize wrong inputs and limit runtime errors.

```
Your option? Y
Use the following command for help: -help

-help

runtime options:

<path>                          Absolute path of the input-file|output-file. eg: c:\Images\flower.jpg
<int>                           Dimension of the image width|height. [use range : 100-500]
<value>                         Value of the RGB color thresholding. [Optimal value: 100-240]
--input-image <path>            Path containing the image file to be processed.
--output-image <path>           Path containing the binary image file.
-width <int>                    Specification of the image width dimension.
-height <int>                   Specification of the image height dimension.
-red <value>                    Specification of the redThreshold value.
-green <value>                  Specification of the greenThreshold value.
-blue <value>                   Specification of the blueThreshold value.
```

Figure 10: Console Application (Help Prompts)

The application provides two key functions: Automatic Thresholding and Explicit Thresholding. The explicit thresholding is a form of localized thresholding where the user can define specific threshold values for each of the RGB components. The binary image produced will depend on the parameter value selected for any of these components. The Help prompt guides the user to make choice between the two binarization procedures and pass the valid threshold arguments for the RGB components. The if-else construct statements defined under the Main method makes this possible.

```
Pass the arguments as following:

Example with automatic RGB:
--input-image c:\a.png --output-image c:\Images\out.txt -width 32 -height 32

Example with explicit RGB:
-input-image c:\a.png --output-image c:\Images\out.txt -width 32 -height 32 -red 100 -green 100 -blue 100
```

Figure 11: Console Application (Help Prompts II)

Unit Test

The UnitTest checks the efficiency of the source code. The unit tests algorithm was implemented against images of varying colour channel properties and at different thresholds. The unit tests verified the behaviour of the IPipelineModule, which helps to convert any given image from bitmap to 3D array.

```
public void TestMethod1()
{

    Dictionary<String, int> imageParams = new Dictionary<string, int>();
    imageParams.Add("imageWidth", 100);
    imageParams.Add("imageHeight", 200);
    imageParams.Add("redThreshold", 10);
    imageParams.Add("greenThreshold", 0);
    imageParams.Add("blueThreshold", 30);

    var api = new LearningApi();

    api.UseActionModule<double[,], double[,]>((input, ctx) =>
    {
        string path = Path.Combine(AppContext.BaseDirectory, "Images\\a.png");

        Bitmap bitmap = new Bitmap(path);

        int imgWidth = bitmap.Width;
        int imgHeight = bitmap.Height;
        double[,] data = new double[imgWidth, imgHeight, 3];

        for (int i = 0; i < imgWidth; i++)
        {
            for (int j = 0; j < imgHeight; j++)
            {
                Color color = bitmap.GetPixel(i, j);
                data[i, j, 0] = color.R;
                data[i, j, 1] = color.G;
                data[i, j, 2] = color.B;
            }
        }
        return data;
    });

    api.UseImageBinarizer(imageParams);
    var result = api.Run() as double[,];

    StringBuilder stringArray = new StringBuilder();
    for (int i = 0; i < result.GetLength(0); i++)
    {
        for (int j = 0; j < result.GetLength(1); j++)
        {
            stringArray.Append(result[i, j, 0]);
        }
        stringArray.AppendLine();
    }
    using (StreamWriter writer = File.CreateText(Path.Combine(AppContext.BaseDirectory, "Images\\aT0.txt")))
    {
        writer.Write(stringArray.ToString());
    }
}
```

Figure 12: UnitTest Code

The sample images were successfully converted to their respective binary equivalent. Each binary image comprises of matrix of 0s and 1s, typical of the values provided for the image Width, image Height and the selected threshold values. The degree of accuracy is a function of the threshold value and image noise as different threshold values are applicable per time depending on the colour channel properties of the given image.

Figure 13: Test Image A

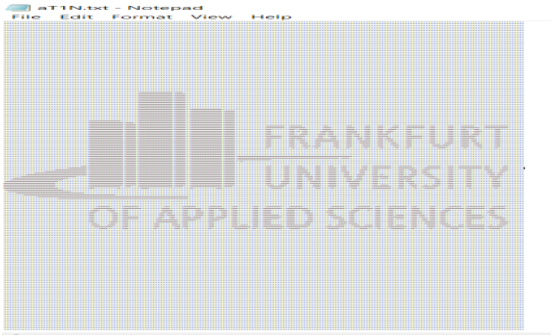


Figure 14: Binarized Image A. Threshold (R,G,B): (200,200,200)

Figure 15: Binarized Image A. Threshold (R,G,B): (70,120,150)

Figure 16: Test Image B

Figure 17: Binarized Image B. Threshold (R,G,B): (20,20,20)

Figure 18: Binarized Image B. Threshold (R,G,B): (100,120,120)

IV. Project

A) GitHub Reference

https://github.com/UniversityOfAppliedSciencesFrankfurt/LearningApi/tree/image-binarizer/SEIV-X/ImageBinarizer

B) Solution items

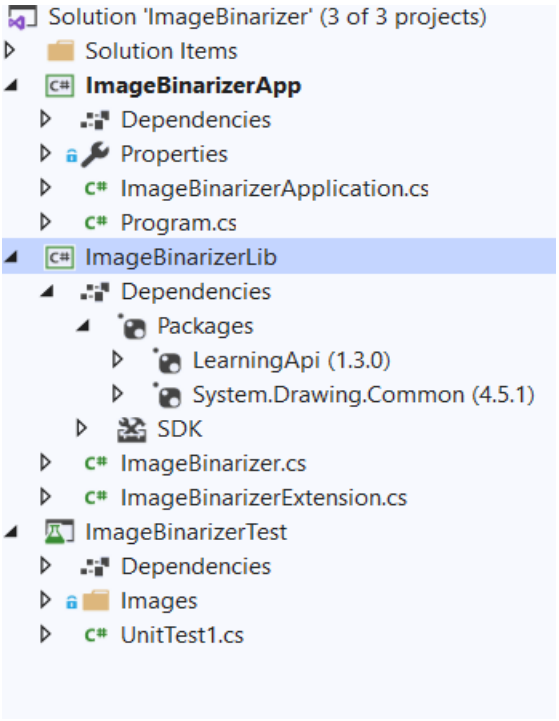
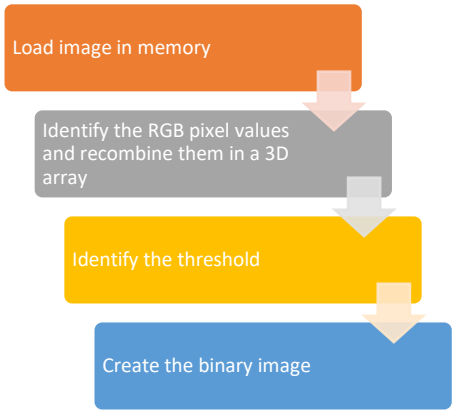


Figure 19: Project solution items

C) Program Flow



V. Conclusion

The quality of the output from image binarization is directly dependent on the quality of the data in the original image. Different samples of a given image with the same distribution could radically give different outcomes. This can happen due to data inconsistencies in the image. Factors such as random variation of brightness and low contrast otherwise categorized as image noise contributes to this. In most cases, the human eye would see a no difference between these samples, but the computer gets different set of pixels with each sample processed. The binarization of such samples would yield different results even with same threshold value.

Not all images can be binarized while some images are not optimal for binarization and would need some fine adjustments before the binarization process takes place. Images with non-uniform background and several overlapping objects could prove very difficult to binarize even with complex segmentation algorithm. Image binarization with dynamic thresholding techniques prove more efficient with complex images even with extremely fine details as they rely on the local property of the pixel as well as the immediate neighbourhood.

References

[1] M. Wirth, "Image Binarization," February 2017. [Online]. Available: https://craftofcoding.wordpress.com/2017/02/21/image-binarization-2-art-or-science/.

[2] Wikipedia, "Image Histogram," [Online]. Available: https://en.wikipedia.org/wiki/Image_histogram. [Accessed March 2020].

[3] R. Sisik, "Image Histogram Generator," 2020. [Online]. Available: https://www.sisik.eu/histo.

[4] Educba, "https://www.educba.com," [Online]. Available: https://www.educba.com/rgb-color-model/. [Accessed 22 March 2020].

[5] Microsoft, ".NET documentation," 2020. [Online]. Available: https://docs.microsoft.com/en-us/dotnet/.

[6] LearningAPI, "LearningAPI," 2020. [Online]. Available: https://github.com/UniversityOfAppliedSciencesFrankfurt/LearningApi.

[7] F. Niklas, "Binarization," 2020. [Online]. Available: https://felixniklas.com/imageprocessing/binarization.

[8] S. Imaging, "Thresholding," 2020. [Online]. Available: https://www.stemmer-imaging.com/en/knowledge-base/thresholding/.

[9] VeprIT, "what-is-image-histogram," 2020. [Online]. Available: https://veprit.com/photography-guide/using-histogram/what-is-image-histogram.

[10] A. K. Chaubey, "Comparison of The Local and Global Thresholding," *World Journal of Research and Review (WJRR)*, vol. 2, no. 1, pp. 01-04, January 2016.

[11] C. G. Inc., "https://www.charterglobal.com/," [Online]. Available: https://www.charterglobal.com/what-is-image-binarization-in-ai/. [Accessed 23 March 2020].