

Technische Hochschule Deggendorf
Fakultät Angewandte Informatik
Studiengang Master Angewandte Informatik

VERGLEICHENDE EVALUIERUNG VON
BILDVERARBEITUNGSBIBLIOTHEKEN FÜR
INDUSTRIELLE ANWENDUNGEN BEI DASSAULT
SYSTEMS

COMPARATIVE EVALUATION OF IMAGE
PROCESSING LIBRARIES FOR INDUSTRIAL
APPLICATIONS AT DASSAULT SYSTEMS

Masterarbeit zur Erlangung des akademischen Grades:

Master of Engineering (M.Eng.)

an der Technischen Hochschule Deggendorf

Vorgelegt von:
Sepehr Fazeli Shahroudi
Matrikelnummer: 12200627

Prüfungsleitung:
Prof. Dr. Schober

Am: 01. Sep 2024

Ergänzende Prüfende:
Martin Steglich

Erklärung

Name des Studierenden: Sepehr Fazeli Shahroudi

Name des Betreuenden: Prof. Dr. Schober

Thema der Abschlussarbeit:

Vergleichende Evaluierung von Bildverarbeitungsbibliotheken für industrielle Anwendungen bei Dassault Systems

.....
.....

1. Ich erkläre hiermit, dass ich die Abschlussarbeit gemäß § 35 Abs. 7 RaPO (Rahmenprüfungsordnung für die Fachhochschulen in Bayern, BayRS 2210-4-1-4-1-WFK) selbständig verfasst, noch nicht anderweitig für Prüfungszwecke vorgelegt, keine anderen als die angegebenen Quellen oder Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

Deggendorf,
Datum

.....
Unterschrift des Studierenden

2. Ich bin damit einverstanden, dass die von mir angefertigte Abschlussarbeit über die Bibliothek der Hochschule einer breiteren Öffentlichkeit zugänglich gemacht wird:

- Nein
 Ja, nach Abschluss des Prüfungsverfahrens
 Ja, nach Ablauf einer Sperrfrist von ...Jahren.

Deggendorf,
Datum

.....
Unterschrift des Studierenden

Bei Einverständnis des Verfassenden vom Betreuenden auszufüllen:

Eine Aufnahme eines Exemplars der Abschlussarbeit in den Bestand der Bibliothek und die Ausleihe des Exemplars wird:

- Befürwortet
 Nicht befürwortet

Deggendorf,
Datum

.....
Unterschrift des Betreuenden

Abstract

This thesis presents a comprehensive evaluation of alternatives to ImageSharp for image processing in software applications. ImageSharp, though powerful and widely used, comes with an annual licensing cost of 5,000\$, significantly affecting project budgets. The primary goal of this research is to explore more cost-effective, performance-oriented alternatives that can either replace or complement ImageSharp while meeting the application's image processing requirements.

The study begins by identifying the core functionalities currently supported by ImageSharp, such as image loading, creation, manipulation, pixel access, resizing, format conversion, and image composition. These functions are essential in tasks like image transformation, cropping, resampling, and metadata management. Performance metrics were established, focusing on key operations like image conversion and pixel iteration. Several alternative libraries were then evaluated based on their ability to meet these functional and performance criteria.

The alternatives investigated include Emgu CV, SkiaSharp, Magick.NET, OpenCvSharp, and others, all of which were assessed for their support of advanced image processing features, licensing costs, integration effort, and community support. Each library was tested for specific capabilities such as pixel manipulation, image format support, encoding efficiency, and rendering performance. Benchmarks were conducted to measure execution times for image conversion and pixel iteration across these libraries, providing insight into their real-world performance.

Among the evaluated libraries, the combination of Emgu CV and SkiaSharp was identified as the most suitable alternative. Emgu CV, based on the powerful OpenCV library, excels in high-performance image processing tasks, including pixel-level manipulation, resizing, and format conversion. SkiaSharp, on the other hand, complements Emgu CV by providing efficient 2D graphics rendering, image creation, and layer composition. Together, these libraries offer a cost-effective solution that maintains high performance while supporting the full range of image processing functionalities required by the application.

Benchmarking results showed that the Emgu CV and SkiaSharp combination significantly reduced processing times for common tasks compared to other alternatives. For example, image conversion times decreased to 490 ms, compared to 2754 ms for ImageSharp. Additionally, pixel iteration tasks were completed more efficiently, making this combination an optimal choice for scenarios requiring both image processing and rendering.

In conclusion, the Emgu CV and SkiaSharp combination was selected as the best alternative to ImageSharp, based on its balance of performance, functionality, ease of integration, and cost. This decision ensures that the project can maintain its image processing capabilities without incurring high licensing fees, while also benefiting from enhanced performance and flexibility.

Contents

Abstract	v
1. Introduction	1
1.1. Background	1
1.2. Problem Statement	1
1.3. Research Objectives	1
1.4. Thesis Structure	1
2. Literature Review	3
2.1. Overview of Image Processing Libraries	3
2.2. ImageSharp	3
2.3. Other Popular Libraries	3
2.3.1. OpenImageIO	3
2.3.2. SkiaSharp	3
2.3.3. Magick.NET	3
2.3.4. Emgu CV	3
3. Methodology	5
3.1. Functional Requirements and Tasks	5
3.1.1. Image Loading and Creation	5
3.1.2. Image Processing and Manipulation	5
3.1.3. Pixel Formats	5
3.1.4. Pixel Access and Manipulation	5
3.1.5. Image Metadata and Conversion	6
3.1.6. Creating and Disposing Image Instances	6
3.1.7. Cropping and Resizing	6
3.1.8. Encoding Images in Various Formats	6
3.1.9. Composing Image Layers	6
3.1.10. Resampling Methods	7
3.1.11. Saving the image	7
3.2. Performance Metrics	7
3.2.1. Performance Metrics	7
3.3. Benchmarking Setup and Tools	8
3.4. Alternatives Evaluation Criteria	8
4. Evaluation of Alternatives	9
4.1. Evaluation of Alternatives	9
4.1.1. 1. OpenImageIO (OIIO)	9

4.1.2.	2. SkiaSharp	12
4.1.3.	3. Magick.NET	15
4.1.4.	4. Emgu CV	19
4.1.5.	5. MagicScaler	22
4.1.6.	6. SimpleITK	24
4.1.7.	7. Structure.Sketching	26
4.1.8.	8. OpenCvSharp	29
4.1.9.	9. Microsoft.Maui.Graphics	31
4.1.10.	10. LeadTools	33
4.2.	Evaluation of Alternatives in GO	35
4.2.1.	11. Golang Image Package	35
4.2.2.	12. Bild	38
4.2.3.	13. GoCV	40
4.2.4.	14. Gift	42
4.2.5.	15. ImageMagick (via Go bindings)	44
4.3.	Summary of Evaluations	47
4.3.1.	Suggestion 1: OpenCvSharp + SkiaSharp	47
4.3.2.	Suggestion 2: Magick.NET + MagicScaler	47
4.3.3.	Suggestion 3: LEADTOOLS (Single Library Solution)	48
4.3.4.	Suggestion 4: Emgu CV + Structure.Sketching	48
5.	Analysis and Discussion	49
5.1.	Performance Benchmarking Results	49
5.1.1.	Benchmark Overview	49
5.1.2.	Benchmark Results:	50
5.2.	Memory Benchmarking	52
5.2.1.	Benchmarking Overview:	52
5.2.2.	Benchmark Results:	53
5.3.	Development Effort Estimation	55
5.3.1.	Overview	55
5.3.2.	Custom Development of Image Processing Library	55
5.3.3.	Use alternative Image Processing Library	55
5.4.	Overall Comparison and Key Insights	55
5.4.1.	Overall Comparison:	55
5.4.2.	Key Insights:	56
5.4.3.	Meeting Outcome and Final Decision:	56
6.	Conclusion and Recommendations	57
6.1.	Summary of Findings	57
6.2.	Final Recommendation	57
6.3.	Future Work	57
A.	Appendices	59
A.1.	Appendix A: Detailed Benchmarking Results	59

Contents

A.2. Appendix B: Implementation Details	59
A.3. Appendix C: Resource Links and Additional Documentation	59

1. Introduction

1.1. Background

The purpose of this investigation is to identify and evaluate potential alternatives to ImageSharp for image processing. Currently, ImageSharp costs \$5,000 per year, which impacts our pricing structure. This review explores cost-effective and efficient alternatives.

1.2. Problem Statement

ImageSharp has limitations regarding cost and performance. These limitations motivate the search for a viable alternative that balances cost, functionality, and performance.

1.3. Research Objectives

The objectives are:

- Identify cost-effective alternatives.
- Evaluate alternatives based on functionality and performance.

1.4. Thesis Structure

This thesis is organized as follows:

- Chapter 2 provides a literature review of image processing libraries.
- Chapter 3 describes the methodology.
- Chapter 4 evaluates the alternatives.
- Chapter 5 discusses the analysis and insights.
- Chapter 6 concludes with recommendations.

2. Literature Review

2.1. Overview of Image Processing Libraries

Image processing libraries provide essential tools for manipulating and processing images in various formats.

2.2. ImageSharp

ImageSharp is a .NET library offering basic image processing capabilities. However, its high cost impacts feasibility for certain projects.

2.3. Other Popular Libraries

2.3.1. OpenImageIO

An open-source library widely used in professional pipelines for handling multiple image formats.

2.3.2. SkiaSharp

A high-performance 2D graphics library for various image processing tasks.

2.3.3. Magick.NET

Provides advanced image manipulation with support for a wide range of formats.

2.3.4. Emgu CV

A .NET wrapper for OpenCV, offering robust image processing capabilities.

3. Methodology

3.1. Functional Requirements and Tasks

3.1.1. Image Loading and Creation

- – **Image.Load**: Loads an image from a byte array with the RGBA32 pixel format.
- – **Image.LoadPixelData**: Loads pixel data into an image with L8 (grayscale) format.
- – **Image.LoadAsync**: Loads an image asynchronously.
- – **new Image**: Creates a new image with Byte4 pixel format.

3.1.2. Image Processing and Manipulation

- – **Clone**: Clones the image for manipulation.
- – **Mutate**: Applies various image processing operations.
- – **Resize**: Resizes the image.
- – **Grayscale**: Converts the image to grayscale.
- – **ColorSpaceConverter**: Provides methods to allow the conversion of color values between different color spaces.

3.1.3. Pixel Formats

- – **Rgba32**: Represents a pixel format with red, green, blue, and alpha channels.
- – **Rgb24**: Represents a pixel format with red, green, and blue channels.
- – **L8**: Represents a grayscale pixel format.
- – **Byte4**: Represents a pixel format with four byte values.
- – **YCbCr**: Represents a YCbCr (luminance, blue chroma, red chroma) color as defined in the ITU-T T.871 specification for the JFIF use with Jpeg.

3.1.4. Pixel Access and Manipulation

- – Access individual pixels and convert them to Rgba32.
- – **ProcessPixelRows**: Processes image pixel rows for manipulation.
- – **ToRgba32**: Converts pixel to RGBA32 format.

3.1.5. Image Metadata and Conversion

- – **ImageFrame.SavePixelData**: Saves pixel data of the specified format.
- **PixelOperations.PackFromRgbPlanes**: Packs pixel data from RGB planes.
- **ImageFrame.GetPixelSpan**: Retrieves pixel span of the specified format.

3.1.6. Creating and Disposing Image Instances

- – **Creating an Empty Canvas**:
 - * Create an empty canvas using `Image<Rgba32>` with specified width, height, and RGBA color values.
- **Disposing of the Canvas**:
 - * Call the dispose method on the canvas object to free resources when it is no longer needed.
- **new Rectangle**:
 - * Creates a new structure that represents a rectangular region defined by its location (x, y, width, height).

3.1.7. Cropping and Resizing

- – **Mutate Method**:
 - * Perform image transformations such as cropping and resizing.
 - * Specify cropping rectangle and resizing options, including mode, size, and resampler.

3.1.8. Encoding Images in Various Formats

- – **ConvertImage Method**:
 - * Handle image encoding into different formats.
 - * Use encoders like `GifEncoder`, `JpegEncoder`, `PbmEncoder`, `PngEncoder`, `TgaEncoder`, `TiffEncoder`, and `WebpEncoder` based on the specified output format.

3.1.9. Composing Image Layers

- – **Stitch Method**:
 - * Combine image tiles into the canvas.
 - * Assemble the final image from smaller pieces.

3.2. Performance Metrics

3.1.10. Resampling Methods

- – **GetResampler Method:**
 - * Return a resampler based on the specified resampler type.
 - * Common resampling techniques include Bicubic and Box, among others.
- **public interface IResampler:**
 - * Encapsulates an interpolation algorithm for resampling images.

3.1.11. Saving the image

- – **SaveAsBmpAsync:** Saves the image to the given stream with the Bmp format.

3.2. Performance Metrics

This section outlines the performance metrics used to evaluate the image processing libraries. The metrics focus on two primary tests: Image Conversion and Pixel Iteration. Each test measures the time taken and memory usage for specific image processing tasks, providing a comprehensive assessment of each library's performance.

3.2.1. Performance Metrics

Image Conversion Test

Measure the time taken to load an image and convert its format using each library, as well as the memory usage during the process.

Steps:

- a. Load the image into memory.
- b. Convert the image to another format (e.g., JPG to PNG).
- c. Save the converted image to disk.

Metrics:

- Time taken to load the image and to convert the image format.
- Memory usage will be measured.

Pixel Iteration Test

Measure the time taken and memory usage to iterate through all the pixels of an image, which is often necessary for tasks like filtering, color adjustments, or complex image processing operations.

Steps:

- a. Load the image into memory.
- b. Iterate through every pixel in the image, applying a simple operation (converting to grayscale).

Metrics:

- Time taken to complete the pixel iteration process.
- Memory usage during the pixel iteration process.

3.3. Benchmarking Setup and Tools

This evaluation uses BenchmarkDotNet to measure and compare library performance.

3.4. Alternatives Evaluation Criteria

Libraries are evaluated based on:

- Functionality
- Licensing
- Integration effort
- Performance

4. Evaluation of Alternatives

4.1. Evaluation of Alternatives

4.1.1. 1. OpenImageIO (OIIO)

- **Type:** Open-source
- **Key Features:** Supports numerous image formats, extensive image processing functionalities
- **Licensing:** Free (BSD license)
- **Performance:** Known for high performance in professional pipelines
- **Integration Effort:** Moderate, requires familiarity with C++ or Python bindings
- **Community and Support:** Active community, well-documented

There might be a need to use P/Invoke or shell out to OIIO command-line utilities if there's no direct C# wrapper.

Feature Category	Supported by OIIO	Not Natively Supported / Requires Custom Implementation
Image Loading and Creation	<ul style="list-style-type: none">- ImageInput:open: Opens an image file for reading.- ImageInput:read_image: Reads image data.- ImageOutput:create: Creates a new image file for writing.- ImageOutput:write_image: Writes the image data.- ImageBuf: Can create an empty image buffer.	<ul style="list-style-type: none">- Asynchronous image loading (no equivalent to Image.LoadAsync).

Feature Category	Supported by OIIO	Not Natively Supported / Requires Custom Implementation
Image Processing and Manipulation	<ul style="list-style-type: none"> - Basic pixel manipulation through scanlines/tiles. - ImageBufAlgo provides some algorithms for manipulation. 	<ul style="list-style-type: none"> - No built-in functions for Clone, Mutate, Resize, Grayscale. - Requires external libraries (e.g., OpenCV) or custom code for advanced processing.
Pixel Formats	<ul style="list-style-type: none"> - Supports various pixel formats including RGBA, RGB, L (grayscale), YUV, etc. - ImageBufAlgo:colorconvert allows conversion between formats. 	<ul style="list-style-type: none"> - Specific formats like Byte4, YCbCr may require manual handling.
Pixel Access and Manipulation	<ul style="list-style-type: none"> - Provides pixel access through ImageInput:read_scanline and ImageInput:read_tile. - Can manipulate pixels by reading and writing scanlines/tiles. 	<ul style="list-style-type: none"> - No direct method like ProcessPixelRows; manual processing required.
Image Metadata and Conversion	<ul style="list-style-type: none"> - ImageSpec: Handles image metadata. - ImageBuf: Manages pixel data and metadata. - ImageBufAlgo: Offers conversion algorithms. 	<ul style="list-style-type: none"> - Handling complex metadata and conversions might require custom implementation depending on needs.
Creating and Disposing Instances	<ul style="list-style-type: none"> - ImageBuf: Creates and manages image instances. - Resources automatically managed in C++ (via destructors). 	<ul style="list-style-type: none"> - Explicit disposal may be needed for resource-intensive operations, especially in languages without automatic garbage collection.
Cropping and Resizing	<ul style="list-style-type: none"> - ImageBufAlgo:crop: Crops images. - ImageBufAlgo:resize: Resizes images with various techniques. 	<ul style="list-style-type: none"> - No direct equivalent to Mutate for fluent transformations.
Encoding Images in Various Formats	<ul style="list-style-type: none"> - Supports encoding in multiple formats (BMP, JPEG, PNG, TIFF, WebP, etc.) via ImageOutput. 	<ul style="list-style-type: none"> - none.
Composing Image Layers	<ul style="list-style-type: none"> - ImageBufAlgo:paste: Combines image layers or tiles. 	<ul style="list-style-type: none"> - No built-in methods equivalent to ImageSharp's Stitch method.

4.1. Evaluation of Alternatives

Feature Category	Supported by OIIO	Not Natively Supported / Requires Custom Implementation
Resampling Methods	- ImageBufAlgo:resample: Provides resampling techniques.	- Lacks a direct equivalent to the IResampler interface; resampling management is manual.
Saving the Image	- ImageOutput:write_image: Saves images to a file.	- No asynchronous saving; requires standard async techniques for implementation.

4.1.2. 2. SkiaSharp

- **Type:** Open-source
- **Key Features:** High-performance 2D graphics library, various image processing tasks
- **Licensing:** Free (MIT License)
- **Performance:** High performance, optimized for cross-platform use
- **Integration Effort:** Easy, seamless integration with .NET Core
- **Community and Support:** Active community, extensive documentation and examples
- **Restricted Countries contributors:** Need a check!

Feature Category	Supported by SkiaSharp	Not Natively Supported / Requires Custom Implementation
Image Loading and Creation	<ul style="list-style-type: none"> - SKBitmap.Decode: Loads images from byte arrays or streams. - SKBitmap: Creates new bitmaps with specified dimensions and color types. - SKImage.FromBitmap: Creates an image from a bitmap. - SKImage.FromEncodedData: Loads images from encoded data. 	<ul style="list-style-type: none"> - No built-in asynchronous image loading (no equivalent to Image.LoadAsync).
Image Processing and Manipulation	<ul style="list-style-type: none"> - SKBitmap.Copy: Clones the bitmap. - SKBitmap.Resize: Resizes bitmaps with various filtering modes. - SKBitmap.ExtractSubset: Crops bitmaps. - SKImage.FilterImage: Applies filters like grayscale or other color transformations. - SKColorFilter.CreateColorMatrix: Provides additional color transformations. 	<ul style="list-style-type: none"> - No high-level fluent API like ImageSharp's Mutate for chaining multiple operations.

4.1. Evaluation of Alternatives

Feature Category	Supported by SkiaSharp	Not Natively Supported / Requires Custom Implementation
Pixel Formats	<ul style="list-style-type: none"> - Supports various pixel formats like SKColorType.Rgba8888, SKColorType.Bgra8888, SKColorType.Gray8, etc. - SKColorSpace: Manages color space conversions. 	<ul style="list-style-type: none"> - Specific formats like Byte4, YCbCr may require custom conversion.
Pixel Access and Manipulation	<ul style="list-style-type: none"> - SKBitmap.GetPixel / SetPixel: Accesses and sets individual pixels. - SKBitmap.Pixels: Provides access to the pixel data for bulk manipulation. 	<ul style="list-style-type: none"> - No direct method like ProcessPixelRows; manual processing of pixel rows is required.
Image Metadata and Conversion	<ul style="list-style-type: none"> - SKImageInfo: Manages basic image properties such as dimensions and color type. - SKImage.Encode: Converts images into various formats like PNG, JPEG, WebP, etc. 	<ul style="list-style-type: none"> - Limited support for complex metadata handling compared to ImageSharp and OIIO.
Creating and Disposing Instances	<ul style="list-style-type: none"> - SKBitmap and SKImage: Create and manage image instances. - Proper resource management using the Dispose method is necessary to free resources. 	<ul style="list-style-type: none"> - No direct equivalent to Image<Rgba32>; pixel format and dimensions must be managed manually.
Cropping and Resizing	<ul style="list-style-type: none"> - SKBitmap.ExtractSubset: Crops images. - SKBitmap.Resize: Resizes images with different resampling techniques. 	<ul style="list-style-type: none"> - No built-in equivalent to ImageSharp's Mutate method for complex transformations.
Encoding Images in Various Formats	<ul style="list-style-type: none"> - SKImage.Encode: Encodes images in multiple formats (e.g., PNG, JPEG, BMP, WebP). 	<ul style="list-style-type: none"> - Custom handling might be needed for less common formats.
Composing Image Layers	<ul style="list-style-type: none"> - SKCanvas.DrawBitmap: Composes images by drawing one bitmap onto another. - SKPicture: Records a sequence of drawing commands for later playback and compositing. 	<ul style="list-style-type: none"> - No built-in method equivalent to ImageSharp's Stitch for seamless image stitching.

Feature Category	Supported by SkiaSharp	Not Natively Supported / Requires Custom Implementation
Resampling Methods	- SKBitmap.Resize : Provides resampling techniques during resizing operations.	- No direct equivalent to IResampler interface; resampling techniques are more basic.
Saving the Image	- SKImage.Encode : Saves images to a stream or byte array in the desired format.	- No built-in asynchronous saving method; async saving requires custom implementation.

4.1. Evaluation of Alternatives

4.1.3. 3. Magick.NET

- **Type:** Open-source
- **Key Features:** .NET wrapper for ImageMagick, extensive image manipulation capabilities
- **Licensing:** Free (Apache 2.0 License)
- **Performance:** Excellent for complex image processing
- **Integration Effort:** Moderate, straightforward API
- **Community and Support:** Large user base, comprehensive documentation
- **Restricted Countries contributors:** No

Feature Category	Supported by Magick.NET	Not Natively Supported / Requires Custom Implementation
Image Loading and Creation	- MagickImage: Loads images from byte arrays, files, or streams. - MagickImageCollection: Handles multiple images, useful for formats like GIFs. - MagickImage: Creates new images with specified dimensions and colors.	- No built-in asynchronous image loading (no equivalent to Image.LoadAsync).

Feature Category	Supported by Magick.NET	Not Natively Supported / Requires Custom Implementation
Image Processing and Manipulation	<ul style="list-style-type: none"> - MagickImage.Clone: Clones the image. - MagickImage.Resize: Resizes images with various filtering and resampling options. - MagickImage.Crop: Crops images. - MagickImage.Resize: Resizes images with customizable resampling options. - MagickImage.Grayscale: Converts images to grayscale. - MagickImage.ColorSpace: Converts between different color spaces. - MagickImage.Rotate, MagickImage.Flip, MagickImage.Flop: Performs various image transformations. 	<ul style="list-style-type: none"> - Magick.NET supports extensive image processing, similar to ImageSharp's Mutate method. Custom implementation is rarely needed.
Pixel Formats	<ul style="list-style-type: none"> - Supports a wide range of pixel formats, including RGBA, RGB, Gray, CMYK, and more. - MagickColor: Handles color conversions and supports various color profiles. 	<ul style="list-style-type: none"> - Fully supports advanced pixel formats and color management, so custom implementation is minimal.
Pixel Access and Manipulation	<ul style="list-style-type: none"> - MagickImage.GetPixels: Provides access to individual pixels or pixel regions. - MagickImage.SetPixels: Sets individual pixels. - MagickImage.ToByteArray: Converts pixel data to a byte array. 	<ul style="list-style-type: none"> - No direct method like ProcessPixelRows; however, pixel manipulation is flexible and powerful.
Image Metadata and Conversion	<ul style="list-style-type: none"> - MagickImage.Attribute: Accesses and manipulates image metadata such as EXIF, IPTC, and XMP. - MagickImage.Format: Converts images to various formats. 	<ul style="list-style-type: none"> - Magick.NET offers comprehensive metadata handling and format conversion.

4.1. Evaluation of Alternatives

Feature Category	Supported by Magick.NET	Not Natively Supported / Requires Custom Implementation
Creating and Disposing Instances	<ul style="list-style-type: none"> - MagickImage: Creates and manages image instances. - Proper resource management using the Dispose method is necessary to free resources. - MagickImageCollection: Manages multiple image instances, useful for animations or multi-layer images. 	<ul style="list-style-type: none"> - Fully supports instance creation and disposal, with comprehensive memory management.
Cropping and Resizing	<ul style="list-style-type: none"> - MagickImage.Crop: Crops images. - MagickImage.Resize: Resizes images with customizable resampling options. - MagickImage.AdaptiveResize: Provides advanced resizing techniques. 	<ul style="list-style-type: none"> - Fully supports cropping and resizing with advanced options, no need for custom implementation.
Encoding Images in Various Formats	<ul style="list-style-type: none"> - MagickImage.Write: Encodes images in a wide array of formats including PNG, JPEG, TIFF, BMP, GIF, WebP, and more. - MagickImage.Format: Specifies the output format. 	<ul style="list-style-type: none"> - Supports a wider range of formats than ImageSharp, with built-in encoding capabilities.
Composing Image Layers	<ul style="list-style-type: none"> - MagickImage.Composite: Composes one image over another. - MagickImage.Mosaic: Combines multiple images into a mosaic. - MagickImageCollection: Handles layering for complex compositions. 	<ul style="list-style-type: none"> - Fully supports complex image compositions, with built-in methods for layering and merging.
Resampling Methods	<ul style="list-style-type: none"> - MagickImage.Resample: Provides advanced resampling methods and filtering options. - MagickImage.AdaptiveResize: Offers specialized resampling techniques. 	<ul style="list-style-type: none"> - Extensive support for resampling, surpassing basic needs and requiring no custom implementation.

Feature Category	Supported by Magick.NET	Not Natively Supported / Requires Custom Implementation
Saving the Image	<ul style="list-style-type: none"> - MagickImage.Write: Saves images to files, streams, or byte arrays in the desired format. - MagickImage.Save: Provides simple saving options. 	<ul style="list-style-type: none"> - No built-in asynchronous saving method; async saving requires custom implementation.

4.1.4. 4. Emgu CV

- **Type:** Open-source
- **Key Features:** .NET wrapper for OpenCV, robust image processing and computer vision
- **Licensing:** 799\$ (for version 4, with additional costs for upgrades).
- **Performance:** High performance, suitable for advanced computer vision tasks
- **Integration Effort:** Moderate to high, depending on complexity of use
- **Community and Support:** Active community, extensive tutorials
- **Restricted Countries contributors:** Need a check

Feature Category	Supported by Emgu CV	Not Natively Supported / Requires Custom Implementation
Image Loading and Creation	<ul style="list-style-type: none"> - CvInvoke.Imread: Loads an image from a file. - CvInvoke.Imdecode: Loads an image from a byte array. - Mat: Creates a new image with specified dimensions and type. - Image<TColor, TDepth>: Generic class for creating images with specific color and depth. 	<ul style="list-style-type: none"> - No built-in asynchronous image loading (no equivalent to Image.LoadAsync).
Image Processing and Manipulation	<ul style="list-style-type: none"> - Mat.Clone: Clones the image matrix. - CvInvoke.Resize: Resizes the image with various interpolation methods. - CvInvoke.CvtColor: Converts the image to grayscale or other color spaces. - CvInvoke.Rotate: Rotates the image. - CvInvoke.Flip: Flips the image vertically or horizontally. - CvInvoke.WarpAffine: Applies affine transformations. 	<ul style="list-style-type: none"> - Emgu CV provides extensive support for image processing operations, similar to ImageSharp's Mutate method.

Feature Category	Supported by Emgu CV	Not Natively Supported / Requires Custom Implementation
Pixel Formats	<ul style="list-style-type: none"> - Supports various pixel formats including BGR, RGBA, RGB, and Gray. - Mat.Depth and Mat.NumberOfChannels: Specify pixel depth and channels. - Image<TColor, TDepth>: Allows pixel data manipulation in a type-safe manner. 	<ul style="list-style-type: none"> - Default BGR format might differ from ImageSharp's RGBA, requiring format conversion in some cases.
Pixel Access and Manipulation	<ul style="list-style-type: none"> - Mat.GetData: Accesses individual pixels or pixel regions. - Mat.SetTo: Sets individual pixels or regions with specific values. - Image<TColor, TDepth>.Data: Provides access to pixel data. 	<ul style="list-style-type: none"> - Pixel manipulation is supported, but the approach differs from ImageSharp's more abstracted methods.
Image Metadata and Conversion	<ul style="list-style-type: none"> - CvInvoke.Imencode: Converts the image to various formats (PNG, JPEG, etc.). - CvInvoke.Imwrite: Saves images to files. 	<ul style="list-style-type: none"> - Does not extensively handle metadata like EXIF or IPTC, focusing more on basic properties and format conversion.
Creating and Disposing Instances	<ul style="list-style-type: none"> - Mat: Can be used to create empty images, with proper disposal via Dispose to free resources. - Image<TColor, TDepth>: Manages images with type safety. 	<ul style="list-style-type: none"> - Fully supports instance creation and disposal, requiring careful memory management due to OpenCV's low-level handling.
Cropping and Resizing	<ul style="list-style-type: none"> - CvInvoke.GetRectSubPix: Crops the image. - CvInvoke.Resize: Resizes the image with advanced interpolation options. 	<ul style="list-style-type: none"> - Emgu CV supports cropping and resizing extensively, similar to ImageSharp's Mutate method.
Encoding Images in Various Formats	<ul style="list-style-type: none"> - CvInvoke.Imwrite: Saves images in formats like PNG, JPEG, BMP, WebP, etc. - CvInvoke.Imencode: Encodes images for various formats and uses. 	<ul style="list-style-type: none"> - none.

4.1. Evaluation of Alternatives

Feature Category	Supported by Emgu CV	Not Natively Supported / Requires Custom Implementation
Composing Image Layers	<ul style="list-style-type: none"> - CvInvoke.AddWeighted: Blends two images together, allowing for composition. - CvInvoke.CopyMakeBorder: Combines images into a larger canvas. 	<ul style="list-style-type: none"> - Supports basic layer composition, though complex operations may require additional code or OpenCV functions.
Resampling Methods	<ul style="list-style-type: none"> - CvInvoke.Resize: Provides multiple resampling methods (linear, cubic, nearest-neighbor). 	<ul style="list-style-type: none"> - Resampling techniques are fully supported, similar to ImageSharp's capabilities.
Saving the Image	<ul style="list-style-type: none"> - CvInvoke.Imwrite: Saves images to files in the desired format. 	<ul style="list-style-type: none"> - No built-in asynchronous saving method; async operations require custom implementation.

4.1.5. 5. MagicScaler

- **Type:** Open-source
- **Key Features:** High-performance image processing, optimized for resizing
- **Licensing:** Free
- **Performance:** Excellent for image resizing with high quality
- **Integration Effort:** Easy, designed for high-performance scenarios
- **Community and Support:** Active, good documentation
- **Restricted Countries contributors:** No

Feature Category	Supported by MagicScaler	Not Natively Supported / Requires Custom Implementation
Image Loading and Creation	<ul style="list-style-type: none"> - MagicImageProcessor.ProcessImage: Loads and processes an image from a file, stream, or byte array. - ImageFileInfo: Provides basic details about the image without fully loading it into memory. 	<ul style="list-style-type: none"> - No direct method for creating new images from scratch. - Lacks asynchronous methods like Image.LoadAsync in ImageSharp.
Image Processing and Manipulation	<ul style="list-style-type: none"> - MagicImageProcessor.ProcessImage: Supports resizing, cropping, and color adjustments. - ProcessImageSettings: Allows specifying transformations such as resizing, cropping, and color adjustments. 	<ul style="list-style-type: none"> - Lacks complex manipulation like cloning, rotating, flipping, or direct pixel manipulation. - Does not support advanced image editing features found in ImageSharp.
Pixel Formats	<ul style="list-style-type: none"> - Supports various pixel formats including RGBA, RGB, Gray, etc. - Automatically handles color space conversions. 	<ul style="list-style-type: none"> - Handling of pixel formats is abstracted, with no direct pixel manipulation like in ImageSharp.
Pixel Access and Manipulation	<ul style="list-style-type: none"> - Provides high-level processing but no direct pixel access. 	<ul style="list-style-type: none"> - No direct access or manipulation of individual pixels, unlike ImageSharp's ProcessPixelRows.

4.1. Evaluation of Alternatives

Feature Category	Supported by MagicScaler	Not Natively Supported / Requires Custom Implementation
Image Metadata and Conversion	<ul style="list-style-type: none"> - Handles basic image properties and metadata. - Supports conversion between different image formats. 	<ul style="list-style-type: none"> - Limited metadata handling compared to libraries like ImageSharp, focusing more on image processing efficiency.
Creating and Disposing Instances	<ul style="list-style-type: none"> - Focuses on processing existing images rather than creating new ones. - Managed disposal of resources. 	<ul style="list-style-type: none"> - Does not support creating images from scratch. - No manual instance creation like in ImageSharp.
Cropping and Resizing	<ul style="list-style-type: none"> - ProcessImageSettings.Crop: Specifies cropping options. - ProcessImageSettings.Width and Height: Specifies resizing dimensions. - Uses high-quality resampling algorithms for resizing. 	<ul style="list-style-type: none"> - Cropping and resizing are part of the processing pipeline, not standalone operations.
Encoding Images in Various Formats	<ul style="list-style-type: none"> - Supports encoding images into formats like PNG, JPEG, BMP, WebP, etc. - Can control compression, quality, and other encoding parameters. 	<ul style="list-style-type: none"> - Lacks format-specific customization that ImageSharp provides through various encoders.
Composing Image Layers	<ul style="list-style-type: none"> - Not supported; focuses on single-image processing. 	<ul style="list-style-type: none"> - Lacks capabilities for composing or layering multiple images, unlike ImageSharp's Stitch or DrawImage.
Resampling Methods	<ul style="list-style-type: none"> - Provides high-quality resampling techniques for resizing. - ProcessImageSettings.Interpolation: Allows specifying the interpolation method. 	<ul style="list-style-type: none"> - Resampling is integrated into the processing pipeline, without a direct interface like IResampler in ImageSharp.
Saving the Image	<ul style="list-style-type: none"> - MagicImageProcessor.ProcessImage: Saves the processed image to a file, stream, or byte array. 	<ul style="list-style-type: none"> - No asynchronous saving method, with all operations handled synchronously.

4.1.6. 6. SimpleITK

- **Type:** Open-source
- **Key Features:** Interface to the Insight Toolkit (ITK), simplifies complex image analysis
- **Licensing:** Free
- **Performance:** Suitable for medical and scientific image processing
- **Integration Effort:** Moderate, specialized usage
- **Community and Support:** Good community support, extensive resources
- **Restricted Countries contributors:** Need a check!

Feature Category	Supported by SimpleITK	Not Natively Supported / Requires Custom Implementation
Image Loading and Creation	<ul style="list-style-type: none"> - sitk.ReadImage: Loads an image from a file. - sitk.Image: Creates new images with specified dimensions and pixel types. - sitk.ImportImageFilter: Loads image data from NumPy arrays. 	<ul style="list-style-type: none"> - No asynchronous loading methods like Image.LoadAsync. - More focused on scientific image formats and lacks direct support for common web image formats.
Image Processing and Manipulation	<ul style="list-style-type: none"> - sitk.Clone, sitk.Resample, sitk.Cast: For cloning, resizing, and pixel type conversion. - sitk.Transform, sitk.Crop: Supports geometric transformations and cropping. - sitk.Grayscale: Converts the image to grayscale. 	<ul style="list-style-type: none"> - Lacks the high-level abstraction of ImageSharp's Mutate for chaining operations. - More tailored to medical image processing than artistic or graphical manipulations.
Pixel Formats	<ul style="list-style-type: none"> - Supports a variety of formats, including scalar, vector, and label images. - Automatic pixel type conversions. 	<ul style="list-style-type: none"> - Limited in handling non-medical specific color spaces and formats compared to ImageSharp.
Pixel Access and Manipulation	<ul style="list-style-type: none"> - sitk.GetPixel/SetPixel: Access and manipulate individual pixels. - Full image data accessible via NumPy arrays for batch processing. 	<ul style="list-style-type: none"> - Less intuitive pixel manipulation compared to ImageSharp's ProcessPixelRows. - Focus on scientific data rather than general-purpose image formats.

4.1. Evaluation of Alternatives

Feature Category	Supported by SimpleITK	Not Natively Supported / Requires Custom Implementation
Image Metadata and Conversion	<ul style="list-style-type: none"> - sitk.Image: Handles metadata such as image origin, spacing, and direction. - sitk.Cast: Converts images to various pixel types. 	<ul style="list-style-type: none"> - Limited advanced metadata handling, focusing on medical image properties.
Creating and Disposing Instances	<ul style="list-style-type: none"> - sitk.Image: Creates empty images. - Automatic resource management with Python's garbage collector. 	<ul style="list-style-type: none"> - Does not offer the fine-grained control over image creation seen in ImageSharp.
Cropping and Resizing	<ul style="list-style-type: none"> - sitk.Crop: Crops the image. - sitk.Resample: Provides resizing with multiple interpolation methods. 	<ul style="list-style-type: none"> - Requires more manual setup compared to the intuitive API of ImageSharp's Mutate.
Encoding Images in Various Formats	<ul style="list-style-type: none"> - sitk.WriteImage: Supports encoding and saving images in formats like PNG, JPEG, and TIFF. 	<ul style="list-style-type: none"> - Focuses more on medical image formats, and does not natively support WebP.
Composing Image Layers	<ul style="list-style-type: none"> - Can handle multi-channel images, simulating layer composition. - sitk.LabelOverlay: Can overlay labels on grayscale images. 	<ul style="list-style-type: none"> - Lacks direct support for layer-based compositions found in ImageSharp. - Limited artistic composition tools.
Resampling Methods	<ul style="list-style-type: none"> - sitk.Resample: Offers various resampling methods with advanced interpolation options. 	<ul style="list-style-type: none"> - Resampling is powerful but less user-friendly than ImageSharp's high-level options.
Saving the Image	<ul style="list-style-type: none"> - sitk.WriteImage: Saves images to files in various formats. - Supports scientific formats like NIfTI, DICOM. 	<ul style="list-style-type: none"> - Limited optimization for modern web formats compared to ImageSharp.

4.1.7. 7. Structure.Sketching

- **Type:** Open-source
- **Key Features:** Image transformations, filters, and format support
- **Licensing:** Free (Apache 2.0 License)
- **Performance:** Competitive, with various resampling filters and transformations
- **Integration Effort:** Easy to moderate, supports .NET Core and .NET Framework
- **Community and Support:** Growing community, good initial documentation
- **Restricted Countries contributors:** No

Feature Category	Supported by Image Package	Not Natively Supported / Requires Custom Implementation
Image Loading and Creation	<ul style="list-style-type: none"> - png.loadPNG: Typically read an image file and decode it using the appropriate decoder from a format-specific sub-package. - image.NewRGBA: Can create new images using the image package, which provides several types like image.RGBA, image.NRGBA, image.Gray, etc. - png.Encode: Can save it using an encoder from the appropriate format-specific package. 	<ul style="list-style-type: none"> - Limited support for creating images from scratch in more diverse formats. - Requires third-party libraries for formats like WebP. - No inherent support for asynchronous operations, can achieve asynchronous behavior by running image-related operations in separate goroutines.
Image Processing and Manipulation	<ul style="list-style-type: none"> - Clone, Resize, ConvertToGrayscale: Supports advanced image processing tasks. - Transformations, Annotations: Extensive transformations and annotation capabilities. - Image Enhancement: High-level enhancement tools for noise reduction, contrast adjustment, etc. 	<ul style="list-style-type: none"> - Requires custom code implementation.

4.1. Evaluation of Alternatives

Feature Category	Supported by Image Package	Not Natively Supported / Requires Custom Implementation
Pixel Formats	<ul style="list-style-type: none"> - image.RGB: Represents a color image with 8-bit RGBA values per pixel. - image.Gray: Represents a grayscale image with 8-bit gray values per pixel. - image.YCbCr: Represents a color image using the Y'CbCr color model, typically used in JPEG images. 	<ul style="list-style-type: none"> - Basic support for different image formats but does not explicitly handle pixel formats like image processing libraries.
Pixel Access and Manipulation	<ul style="list-style-type: none"> - No direct methods like getpixel, setpixel or RasterImageData. 	<ul style="list-style-type: none"> - No direct pixel access, unlike ImageSharp's ProcessPixelRows.
Image Metadata and Conversion	<ul style="list-style-type: none"> - Does not support comprehensive metadata handling directly. - Focuses on image manipulation and format conversion but lacks built-in tools for managing or accessing metadata like EXIF data or other detailed image properties. 	<ul style="list-style-type: none"> - Lack of support requires custom implementations.
Creating and Disposing Instances	<ul style="list-style-type: none"> - Use functions from the image package along with specific image formats (like image/png, image/jpeg). - Go's garbage collector handles memory management, so no need to manually dispose of images. 	<ul style="list-style-type: none"> - Does not directly provide CreateImage or Dispose functions as in other libraries or languages with more explicit image management.
Cropping and Resizing	<ul style="list-style-type: none"> - Does not include built-in support for more advanced operations like cropping and resizing directly. 	<ul style="list-style-type: none"> - Requires custom implementations.
Encoding Images in Various Formats	<ul style="list-style-type: none"> - Save: Create a file using os.Create, then use png.Encoder or jpeg.Encoder to encode and save the image. 	<ul style="list-style-type: none"> - For other image formats like WebP, need to import the corresponding encoding package.
Composing Image Layers	<ul style="list-style-type: none"> - Does not directly support complex operations like combining images or drawing one image onto another. 	<ul style="list-style-type: none"> - Requires custom implementations.

Feature Category	Supported by Image Package	Not Natively Supported / Requires Custom Implementation
Resampling Methods	- Does not natively support advanced resampling techniques.	- Provides basic image manipulation capabilities, including resizing, but does not include advanced resampling algorithms.
Saving the Image	- Does not directly support saving images or providing asynchronous save functionality.	- To perform asynchronous saving, need to use Go's concurrency features such as goroutines.

4.1.8. 8. OpenCvSharp

- **Type:** Open-source
- **Key Features:** .NET wrapper for OpenCV, similar to Emgu CV
- **Licensing:** Free (Apache-2.0 license)
- **Performance:** High performance, supports a wide range of tasks
- **Integration Effort:** Moderate, different API style from Emgu CV
- **Community and Support:** Active community, good documentation
- **Restricted Countries contributors:** Need a check!

Feature Category	Supported by OpenCvSharp	Not Natively Supported / Requires Custom Implementation
Image Loading and Creation	<ul style="list-style-type: none"> - Cv2.ImRead, Cv2.ImDecode: Loads images from files or byte arrays. - Mat, Cv2.ImCreate: Creates new images with specified dimensions and types. 	<ul style="list-style-type: none"> - No asynchronous loading methods like Image.LoadAsync in ImageSharp.
Image Processing and Manipulation	<ul style="list-style-type: none"> - Mat.Clone, Cv2.Resize: For cloning and resizing. - Cv2.CvtColor: Converts between color spaces. - Cv2.Rotate, Cv2.Flip, Cv2.WarpAffine: Provides extensive geometric transformations. - Cv2.Crop: Crops images using a Rect object. 	<ul style="list-style-type: none"> - Lacks some of the more advanced image effects and filters available in ImageSharp.
Pixel Formats	<ul style="list-style-type: none"> - Supports multiple formats, including CvType.CV_8UC3 (BGR), CvType.CV_8UC1 (Grayscale), CvType.CV_8UC4 (BGRA). - Handles automatic color space conversion. 	<ul style="list-style-type: none"> - May require custom implementations for less common pixel formats.

Feature Category	Supported by OpenCvSharp	Not Natively Supported / Requires Custom Implementation
Pixel Access and Manipulation	<ul style="list-style-type: none"> - Mat.At, Mat.Set: Direct pixel access and manipulation. - Mat.Data: Provides low-level access to pixel data arrays. 	<ul style="list-style-type: none"> - Less intuitive pixel manipulation compared to ImageSharp's ProcessPixelRows.
Image Metadata and Conversion	<ul style="list-style-type: none"> - Mat: Stores image properties like size and type. - Cv2.ImEncode, Cv2.ImWrite: Converts and saves images in various formats. 	<ul style="list-style-type: none"> - Limited metadata handling compared to ImageSharp's extensive metadata support.
Creating and Disposing Instances	<ul style="list-style-type: none"> - Mat: Creates and initializes images. - Dispose: Required for freeing unmanaged resources. - Effective memory management through manual disposal. 	<ul style="list-style-type: none"> - More complex resource management due to reliance on unmanaged resources.
Cropping and Resizing	<ul style="list-style-type: none"> - Cv2.GetRectSubPix: For precise cropping. - Cv2.Resize: Resizing with various interpolation methods. 	<ul style="list-style-type: none"> - Lacks some of the more advanced cropping techniques like those in ImageSharp.
Encoding Images in Various Formats	<ul style="list-style-type: none"> - Cv2.ImWrite: Saves images in multiple formats including WebP. - Cv2.ImEncode: For encoding images to byte arrays. 	<ul style="list-style-type: none"> - Less flexible format optimization compared to ImageSharp's encoders.
Composing Image Layers	<ul style="list-style-type: none"> - Cv2.AddWeighted, Cv2.Add, Cv2.Subtract: For image blending and compositing. 	<ul style="list-style-type: none"> - Less comprehensive layering system than ImageSharp.
Resampling Methods	<ul style="list-style-type: none"> - Cv2.Resize: Offers multiple interpolation methods. - Cv2.PyrUp, Cv2.PyrDown: Pyramid methods for scaling. 	<ul style="list-style-type: none"> - Fewer options for specialized resampling methods compared to ImageSharp.
Saving the Image	<ul style="list-style-type: none"> - Cv2.ImWrite, Cv2.ImEncode: Saves images to files or byte arrays. 	<ul style="list-style-type: none"> - Lacks some advanced saving options like those in ImageSharp.

4.1.9. 9. Microsoft.Maui.Graphics

- **Type:** Open-source
- **Key Features:** Graphics functionalities across platforms, uses SkiaSharp
- **Licensing:** Free (MIT)
- **Performance:** Optimized for cross-platform use within MAUI framework
- **Integration Effort:** Easy if using MAUI
- **Community and Support:** Active support from Microsoft, good documentation
- **Restricted Countries contributors:** No

This repository has been archived by the owner on Dec 21, 2023. It is now read-only.

Feature Category	Supported by Microsoft.Maui.Graphics	Not Natively Supported / Requires Custom Implementation
Image Loading and Creation	<ul style="list-style-type: none"> - ImageLoadingService.LoadImageAsync: Asynchronous loading from various sources. - GraphicsPlatform.CreateImage: Creates images with specified dimensions and formats. 	- Unlike ImageSharp, there is limited support for creating images from scratch in more diverse formats.
Image Processing and Manipulation	<ul style="list-style-type: none"> - Image.Clone, ICanvas.DrawImage: For cloning, resizing, cropping, and applying transformations. - ICanvas.SetFillColor, ICanvas.DrawRectangle: Supports color transformations and selective drawing for cropping. 	- Lacks extensive image manipulation tools available in libraries like ImageSharp.
Pixel Formats	<ul style="list-style-type: none"> - Supports RGBA, RGB and other common formats. - Designed for higher-level graphics tasks rather than extensive pixel format diversity. 	- Limited to basic pixel formats compared to the wide range in ImageSharp.

Feature Category	Supported by Microsoft.Maui.Graphics	Not Natively Supported / Requires Custom Implementation
Pixel Access and Manipulation	<ul style="list-style-type: none"> - Focuses on high-level operations, without direct pixel access. - Does not offer low-level pixel manipulation like GetPixel and SetPixel. 	<ul style="list-style-type: none"> - No direct pixel access, unlike ImageSharp's ProcessPixelRows.
Image Metadata and Conversion	<ul style="list-style-type: none"> - Handles basic image properties and conversion via Image.Save. - Can save to various formats. 	<ul style="list-style-type: none"> - Less detailed metadata handling compared to ImageSharp.
Creating and Disposing Instances	<ul style="list-style-type: none"> - GraphicsPlatform.CreateImage: Easily creates new images. - Disposal managed via .NET's garbage collection. 	<ul style="list-style-type: none"> - Lacks manual resource management options, unlike some lower-level libraries.
Cropping and Resizing	<ul style="list-style-type: none"> - ICanvas.DrawImage: Supports cropping and resizing with simple interfaces. 	<ul style="list-style-type: none"> - Less advanced cropping and resizing techniques compared to ImageSharp.
Encoding Images in Various Formats	<ul style="list-style-type: none"> - Image.Save: Supports encoding in multiple formats. - Provides functionality to save directly to files, streams, or byte arrays. 	<ul style="list-style-type: none"> - Limited format-specific encoding settings compared to ImageSharp and does not mention WebP support.
Composing Image Layers	<ul style="list-style-type: none"> - ICanvas.DrawImage: Allows layering images over one another. - Basic support for image compositing and blending. 	<ul style="list-style-type: none"> - More limited compositing features compared to ImageSharp.
Resampling Methods	<ul style="list-style-type: none"> - ICanvas.DrawImage: Automatically handles resampling during resizing. 	<ul style="list-style-type: none"> - Fewer advanced resampling methods compared to ImageSharp.
Saving the Image	<ul style="list-style-type: none"> - Image.Save: Saves images to files, streams, or byte arrays in various formats. 	<ul style="list-style-type: none"> - Lacks advanced saving options and optimizations available in ImageSharp.

4.1. Evaluation of Alternatives

4.1.10. 10. LeadTools

- **Type:** Commercial
- **Key Features:** Extensive image processing features, supports numerous formats
- **Licensing:** Paid (More than 17k\$)
- **Performance:** High performance, enterprise-grade reliability
- **Integration Effort:** Moderate to high, depending on features used
- **Community and Support:** Excellent support, extensive documentation
- **Restricted Countries contributors:** Need a check!

Feature Category	Supported by LEADTOOLS	Not Natively Supported / Requires Custom Implementation
Image Loading and Creation	<ul style="list-style-type: none"> - LoadImage, LoadAsync: Robust image loading with asynchronous support. - RasterImage.Create: Flexible image creation with control over dimensions and pixel formats. 	<ul style="list-style-type: none"> - None; LEADTOOLS provides comprehensive loading and creation features.
Image Processing and Manipulation	<ul style="list-style-type: none"> - Clone, Resize, ConvertToGrayscale: Supports advanced image processing tasks. - Transformations, Annotations: Extensive transformations and annotation capabilities. - Image Enhancement: High-level enhancement tools for noise reduction, contrast adjustment, etc. 	<ul style="list-style-type: none"> - While comprehensive, some specific manipulation tasks might require custom scripting or code.
Pixel Formats	<ul style="list-style-type: none"> - Supports multiple formats: Extensive support for various pixel formats such as RGB, RGBA, Grayscale, CMYK, and more. 	<ul style="list-style-type: none"> - None; pixel format support is exhaustive.
Pixel Access and Manipulation	<ul style="list-style-type: none"> - GetPixel, SetPixel, RasterImageData: Direct access and manipulation of pixel data. - Lock/Unlock: For precise pixel-level operations. 	<ul style="list-style-type: none"> - Pixel data manipulation can be complex, especially when locking/unlocking is required.

Feature Category	Supported by LEADTOOLS	Not Natively Supported / Requires Custom Implementation
Image Metadata and Conversion	<ul style="list-style-type: none"> - Comprehensive Metadata Handling: Extracts, edits, and writes metadata for a wide range of formats. - Advanced Conversion: Extensive format support including DICOM. 	<ul style="list-style-type: none"> - None; LEADTOOLS excels in metadata and conversion capabilities.
Creating and Disposing Instances	<ul style="list-style-type: none"> - CreateImage, Dispose: Facilitates creation and proper resource disposal of images. 	<ul style="list-style-type: none"> - Proper disposal requires careful management, similar to .NET libraries.
Cropping and Resizing	<ul style="list-style-type: none"> - Crop, Resize: Advanced cropping and resizing with various resampling methods. 	<ul style="list-style-type: none"> - None; cropping and resizing are robustly supported.
Encoding Images in Various Formats	<ul style="list-style-type: none"> - Save: Encodes images into numerous formats including WebP, including advanced options for compression and format-specific settings. - Multimedia and Document Formats: Extends to multimedia and document encoding. 	<ul style="list-style-type: none"> - Format-specific settings might require additional configuration.
Composing Image Layers	<ul style="list-style-type: none"> - Combine, DrawImage: Supports complex image layering with detailed control over blending and transparency. 	<ul style="list-style-type: none"> - Layer management might need additional custom code for complex use cases.
Resampling Methods	<ul style="list-style-type: none"> - Advanced Resampling: Provides high-quality resampling techniques during resizing. 	<ul style="list-style-type: none"> - Resampling methods are extensive but need careful selection for optimal results.
Saving the Image	<ul style="list-style-type: none"> - Save, SaveAsync: Saves images with extensive control over parameters. - Asynchronous Saving: Supports performance-enhanced operations. 	<ul style="list-style-type: none"> - None; saving features are comprehensive and powerful.

4.2. Evaluation of Alternatives in GO

4.2.1. 11. Golang Image Package

- **Type:** Standard library
- **Key Features:** Basic image processing (decoding, encoding, and simple manipulations)
- **Licensing:** Free (BSD-style license)
- **Performance:** Suitable for basic image processing tasks
- **Integration Effort:** Minimal, as it's part of the standard library
- **Community and Support:** Large community, extensive documentation

Feature Category	Supported by Image Package	Not Natively Supported / Requires Custom Implementation
Image Loading and Creation	<ul style="list-style-type: none"> - png.loadPNG: Typically read an image file and decode it using the appropriate decoder from a format-specific sub-package. - image.NewRGBA: Can create new images using the image package, which provides several types like image.RGBA, image.NRGBA, image.Gray, etc. - png.Encode: Can save it using an encoder from the appropriate format-specific package. 	<ul style="list-style-type: none"> - Limited support for creating images from scratch in more diverse formats. - BMP: The image/bmp package is available in the Go ecosystem but is not included in the standard library. - TIFF: The image/tiff package exists as a separate package but is not part of the standard library. - WebP: Requires third-party libraries such as golang.org/x/image/webp. - Does not inherently support asynchronous operations; can achieve asynchronous behavior by running image-related operations in separate goroutines.

Feature Category	Supported by Image Package	Not Natively Supported / Requires Custom Implementation
Image Processing and Manipulation	<ul style="list-style-type: none"> - Clone, Resize, ConvertToGrayscale: Supports advanced image processing tasks. - Transformations, Annotations: Extensive transformations and annotation capabilities. - Image Enhancement: High-level enhancement tools for noise reduction, contrast adjustment, etc. 	<ul style="list-style-type: none"> - Requires custom code implementation.
Pixel Formats	<ul style="list-style-type: none"> - Image.RGB: Represents a color image with 8-bit RGBA values per pixel. - Image.Gray: Represents a grayscale image with 8-bit gray values per pixel. - Image.YCbCr: Represents a color image using the Y'CbCr color model, typically used in JPEG images. 	<ul style="list-style-type: none"> - Provides basic support for different image formats but does not explicitly handle pixel formats like image processing libraries.
Pixel Access and Manipulation	<ul style="list-style-type: none"> - Does not have direct methods like getpixel, setpixel or RasterImageData. 	<ul style="list-style-type: none"> - No direct pixel access, unlike ImageSharp's ProcessPixelRows.
Image Metadata and Conversion	<ul style="list-style-type: none"> - Does not support comprehensive metadata handling directly. This package focuses on image manipulation and format conversion but lacks built-in tools for managing or accessing metadata like EXIF data or other detailed image properties. - Does not support advanced conversion. 	<ul style="list-style-type: none"> - Lack of support requires custom implementations.

4.2. Evaluation of Alternatives in GO

Feature Category	Supported by Image Package	Not Natively Supported / Requires Custom Implementation
Creating and Disposing Instances	<ul style="list-style-type: none"> - To create an image in Go, you generally use functions from the image package along with specific image formats (like image/png, image/jpeg). - Go's garbage collector handles memory management, so you don't need to manually dispose of images. 	<ul style="list-style-type: none"> - The Go image package does not directly provide CreateImage or Dispose functions as we might find in other libraries or languages with more explicit image management.
Cropping and Resizing	<ul style="list-style-type: none"> - Does not include built-in support for more advanced operations like cropping and resizing directly. 	<ul style="list-style-type: none"> - Requires custom implementations.
Encoding Images in Various Formats	<ul style="list-style-type: none"> - Save: Create a file using os.Create, then use png.Encoder or jpeg.Encoder to encode and save the image. - Does not directly support more advanced multimedia or document formats. 	<ul style="list-style-type: none"> - For other image formats like WebP, need to import the corresponding encoding package.
Composing Image Layers	<ul style="list-style-type: none"> - Does not directly support complex operations like combining images or drawing one image onto another. 	<ul style="list-style-type: none"> - Requires custom implementations.
Resampling Methods	<ul style="list-style-type: none"> - Does not natively support advanced resampling techniques. 	<ul style="list-style-type: none"> - Provides basic image manipulation capabilities, including resizing, but does not include advanced resampling algorithms.
Saving the Image	<ul style="list-style-type: none"> - Does not directly support saving images or providing asynchronous save functionality. 	<ul style="list-style-type: none"> - To perform asynchronous saving, need to use Go's concurrency features such as goroutines.

4.2.2. 12. Bild

- **Type:** Open-source
- **Key Features:** Basic image processing functions like resizing, cropping, and rotation
- **Licensing:** Free (MIT License)
- **Performance:** Suitable for lightweight image processing tasks
- **Integration Effort:** Easy, simple API
- **Community and Support:** Active community, well-documented

Feature Category	Supported by Bild	Not Natively Supported / Requires Custom Implementation
Image Loading and Creation	- Does not support image loading and creation directly. Instead, it focuses on image processing, including operations like resizing.	- Does not support image loading and creating; need third-party package.
Image Processing and Manipulation	- Resize: Is supported. - Does not include advanced image processing capabilities.	- Requires custom code implementation.
Pixel Formats	- Supports several basic pixel formats.	- Primarily works with the standard Go image package types like image.RGBA , image.NRGBA .
Pixel Access and Manipulation	- RGB and RGBA: These are the primary formats supported. The bild package can handle images in these formats for manipulation and conversion tasks.	- Primarily deals with RGB and RGBA formats. We can manipulate and convert images using this package, but we may need to handle pixel formats explicitly.
Image Metadata and Conversion	- Does not provide image metadata and conversion features for handling image metadata or advanced image format conversion.	- Lack of support requires custom implementations.
Creating and Disposing Instances	- Does not explicitly manage the lifecycle of instances.	- The Go bild package does not directly provide CreateImage or Dispose functions as we might find in other libraries or languages with more explicit image management.

4.2. Evaluation of Alternatives in GO

Feature Category	Supported by Bild	Not Natively Supported / Requires Custom Implementation
Cropping and Resizing	- Crop, Resize: Provides functions for cropping and resizing images.	- None, Resize and crop is supported by bild .
Encoding Images in Various Formats	- Does not directly handle encoding images to various formats.	- Requires third-party packages to handle formats like WebP .
Composing Image Layers	- blend.Overlay : The function is used to combine two images. - The bild library offers various blending modes such as Add , Multiply , Screen .	- None.
Resampling Methods	- Provides various resampling methods: - ResizeNearestNeighbor - ResizeBilinear - ResizeBicubic - ResizeLanczos	- None; Resampling robustly supported.
Saving the Image	- Does not directly support saving images or providing asynchronous save functionality.	- To perform asynchronous saving, need to use Go's concurrency features such as goroutines.

4.2.3. 13. GoCV

- **Type:** Open-source
- **Key Features:** Go wrapper for OpenCV, advanced image processing and computer vision
- **Licensing:** Free (MIT License)
- **Performance:** High performance, suitable for advanced image processing tasks
- **Integration Effort:** Moderate to high, depending on usage complexity
- **Community and Support:** Active community, extensive documentation

Feature Category	Supported by GoCV	Not Natively Supported / Requires Custom Implementation
Image Loading and Creation	<ul style="list-style-type: none"> - gocv.IMRead: Reads an image file from disk into a gocv.Mat object. - gocv.IMWrite: Writes images, and uses the Mat type to manipulate them in memory. 	- None; supports read and write of images.
Image Processing and Manipulation	<ul style="list-style-type: none"> - Clone, Resize, ConvertToGrayscale: Supports advanced image processing tasks. - Transformations, Annotations: Transformations such as resizing, rotating, and cropping using OpenCV functions available in gocv. - Image Enhancement: High-level enhancement tools for noise reduction, contrast adjustment, etc. 	- None.
Pixel Formats	- Supports multiple formats, including 8-bit grayscale, 16-bit grayscale, RGB (24-bit color), RGBA (32-bit color), etc.	- The gocv library provides support for various pixel formats through its integration with OpenCV.
Pixel Access and Manipulation	<ul style="list-style-type: none"> - img.Cols(), img.Rows(): Provide the dimensions of the image. - img.At(x, y): Gets the pixel value at the specified coordinates. - img.Set(): Modifies pixel values. 	- GoCV , a Go wrapper for the OpenCV library, allows pixel access and manipulation in images.

4.2. Evaluation of Alternatives in GO

Feature Category	Supported by GoCV	Not Natively Supported / Requires Custom Implementation
Image Metadata and Conversion	- Supports various operations, including image metadata extraction and conversion.	- For more advanced metadata extraction (such as EXIF data), additional libraries or tools might be needed, as gocv itself doesn't handle metadata like EXIF.
Creating and Disposing Instances	- Mat : Use Close() to release the matrix object when done. - gocv.NewMat() : Creates a new Mat object to store the captured frame.	- Use defer for cleanup to make code cleaner and more reliable.
Cropping and Resizing	- gocv.Resize : Resizes the cropped image to the specified dimensions.	- Crop is not supported.
Encoding Images in Various Formats	- gocv.IMEncode : Supports various formats like JPEG, PNG, etc. The format can be specified by changing the file extension in the function parameter.	- Supports various formats like JPEG, PNG, etc. - Like OpenCV, WebP is not supported by default.
Composing Image Layers	- gocv.AddWeighted : Blends the images based on the specified weights. - Use OpenCV functions like AddWeighted for blending or Add for overlaying images.	- None.
Resampling Methods	- Support for image resampling methods through its bindings to OpenCV such as cv2.INTER_NEAREST , cv2.INTER_LINEAR , cv2.INTER_CUBIC , cv2.INTER_LANCZOS4 .	- None; resampling robustly supported.
Saving the Image	- gocv.IMWrite : Saves the processed image.	- To perform asynchronous saving, need to use Go's concurrency features such as goroutines.

4.2.4. 14. Gift

- **Type:** Open-source
- **Key Features:** A package for image filtering, supports many common filters
- **Licensing:** Free (MIT License)
- **Performance:** Efficient for applying various filters
- **Integration Effort:** Easy, straightforward API
- **Community and Support:** Moderate, good documentation

Feature Category	Supported by Gift	Not Natively Supported / Requires Custom Implementation
Image Loading and Creation	- Does not support Image Loading and Creation.	- Does not support Image Loading and Creation.
Image Processing and Manipulation	- Clone, Resize, ConvertToGrayscale: Supports advanced image processing tasks. - Transformations, Annotations: Transformations supported but Annotations not supported. - Image Enhancement: Supports operations like sharpening, blurring, adjusting brightness and contrast, and more.	- To clone an image, you can create a new image with the same dimensions and copy the pixel data. - Need to use gift for image transformations and another library like gg for annotations.
Pixel Formats	- Does not directly provide pixel format manipulation or conversion functionalities.	- Need to write custom implementation.
Pixel Access and Manipulation	- Does not provide pixel-level access and manipulation.	- gift 's built-in functions aren't sufficient.
Image Metadata and Conversion	- Does not directly handle image metadata or conversion between different formats.	- The gift package does not provide direct support for format conversion.

4.2. Evaluation of Alternatives in GO

Feature Category	Supported by Gift	Not Natively Supported / Requires Custom Implementation
Creating and Disposing Instances	- Does not directly support creating and disposing of instances.	- We create a gift processor with a specific filter. We need to apply the filter to an image. Once done, Go's garbage collector will handle memory management.
Cropping and Resizing	- gift.Resize : Resizes image to the specified dimensions. - gift.Crop : Crops image to the specified dimensions.	- For cropping, need to use a combination of gift.Crop and gift.Resize . While gift does not have a direct Crop function, we can achieve cropping by resizing and then applying the crop manually.
Encoding Images in Various Formats	- Does not inherently include encoding images in different formats.	- Does not directly support WebP .
Composing Image Layers	- Does not natively support composing image layers.	- Can achieve layer composition by manually handling image layers using Go's image processing libraries.
Resampling Methods	- Provides a variety of resampling methods for image.	- None; Resampling robustly supported.
Saving the Image	- Does not directly handle file I/O for saving.	- To perform asynchronous saving, need to use Go's concurrency features such as goroutines.

4.2.5. 15. ImageMagick (via Go bindings)

- **Type:** Open-source
- **Key Features:** Extensive image manipulation capabilities, leveraging ImageMagick
- **Licensing:** Free (Apache 2.0 License)
- **Performance:** Excellent for complex image processing
- **Integration Effort:** Moderate, requires use of Go bindings
- **Community and Support:** Large user base, comprehensive documentation

Feature Category	Supported by ImageMagick	Not Natively Supported / Requires Custom Implementation
Image Loading and Creation	<ul style="list-style-type: none"> - imagemagick.NewMagickWand: Load images from various file formats like PNG, JPEG, BMP, and others. - NewImage, SetFormat, SetSize: Create new images from scratch by setting up dimensions, colors, and other properties. 	- None
Image Processing and Manipulation	<ul style="list-style-type: none"> - Clone, Resize, ConvertToGrayscale: Supports advanced image processing tasks. - Transformations, Annotations: Supports image processing tasks. - Image Enhancement: Supports operations like sharpening, blurring, contrast, and noise reduction. 	- All advanced image processing and manipulation supported.
Pixel Formats	- Supports various image pixel formats: RGB (Red, Green, Blue), RGBA (Red, Green, Blue, Alpha), Gray (Grayscale), etc.	- None

4.2. Evaluation of Alternatives in GO

Feature Category	Supported by ImageMagick	Not Natively Supported / Requires Custom Implementation
Pixel Access and Manipulation	<ul style="list-style-type: none"> - Provides the PixelWand API, which allows you to manipulate individual pixels in an image. - A PixelIterator is used to access and modify the pixels of the image. - The SetRed, SetGreen, and SetBlue methods are used to change the color of the pixels. 	- None
Image Metadata and Conversion	<ul style="list-style-type: none"> - Reading Metadata: Allows reading metadata from images, such as EXIF data, image properties, and other relevant information. - Writing Metadata: Allows manipulating and writing metadata back to the image file. - Format Conversion: Supports conversion between a wide range of image formats (e.g., BMP, PNG, JPEG, TIFF). - Quality and Compression: Allows adjusting quality and compression settings when converting images. 	- Robust support for image metadata and conversion.
Creating and Disposing Instances	<ul style="list-style-type: none"> - imagemick.NewMagickWand: Create image instances. - imagemick.Destroy: Free resources when the image is no longer needed. 	- This process is crucial for managing memory usage, especially when working with large images or in a long-running application.
Cropping and Resizing	<ul style="list-style-type: none"> - imagemick.MagickResizeImage: Resize the image. - imagemick.MagickCropImage: Crop the image to the specified dimensions. 	- None
Encoding Images in Various Formats	- Allows encoding and decoding images in various formats. Some of the most commonly supported formats include JPEG, BMP, PNG, WebP, GIF, etc.	- None

Feature Category	Supported by ImageMagick	Not Natively Supported / Requires Custom Implementation
Composing Image Layers	<ul style="list-style-type: none"> - base.png: The base image. - overlay.png: The image to be composited on top of the base image. - - imagemick.COMPOSITE_OP_OVER: Specifies the compositing operation (in this case, overlay). 	<ul style="list-style-type: none"> - None
Resampling Methods	<ul style="list-style-type: none"> - Provides a variety of resampling methods for images. 	<ul style="list-style-type: none"> - None; resampling robustly supported.
Saving the Image	<ul style="list-style-type: none"> - SetImageFormat(): Sets the format of the image (e.g., PNG). - WriteImage: Writes the image to a new file. 	<ul style="list-style-type: none"> - Does not directly support asynchronous operations for image saving. - To perform asynchronous saving, need to use Go's concurrency features such as goroutines.

4.3. Summary of Evaluations

4.3.1. Suggestion 1: OpenCvSharp + SkiaSharp

Strong points:

- **OpenCvSharp:** Image processing, pixel access, extensive pixel formats, high-performance manipulation metadata handling.
- **SkiaSharp:** High-quality 2D graphics rendering, image creation, layer composition, efficient drawing operations.

Why this combination:

- **OpenCvSharp:** Lacks advanced 2D graphics and layer composition.
- **SkiaSharp:** Complements with robust 2D rendering and drawing features.

Pricing:

- **OpenCvSharp:** Free for commercial use.
- **SkiaSharp:** Free for commercial use.

4.3.2. Suggestion 2: Magick.NET + MagicScaler

Strong points:

- **Magick.NET:** Comprehensive image processing, wide format support, advanced metadata handling, image composition.
- **MagicScaler:** High-quality, efficient image resizing and resampling.

Why this combination:

- **Magick.NET:** Lacks optimized high-quality resampling.
- **MagicScaler:** Ensures top-tier resizing and resampling, covering Magick.NET's limitations.

Pricing:

- **Magick.NET:** Apache-2.0 license
- **MagicScaler:** Free for commercial use.

4.3.3. Suggestion 3: LEADTOOLS (Single Library Solution)

Strong points:

- Comprehensive image processing, extensive format support, advanced metadata handling, versatile imaging features.

Pricing:

- Paid (More than \$17,000).

4.3.4. Suggestion 4: Emgu CV + Structure.Sketching

Strong points:

- **Emgu CV:** Advanced image processing, pixel manipulation, built on OpenCV.
- **Structure.Sketching:** Efficient drawing, resizing, and layer composition.

Why this combination:

- **Emgu CV:** Lacks comprehensive drawing and layer composition.
- **Structure.Sketching:** Complements with effective graphics rendering and basic image processing.

Pricing:

- **Emgu CV:** \$799 (for version 4, with additional costs for upgrades).
- **Structure.Sketching:** Free for commercial use.

5. Analysis and Discussion

5.1. Performance Benchmarking Results

5.1.1. Benchmark Overview

This benchmark evaluates the performance of image processing tasks through three key metrics: Image Conversion, Pixel Iteration, and their associated times. These metrics provide insight into both the initial setup time and the efficiency of processing during steady-state operations.

Benchmark Metrics

Pixel Iteration:

- **Warm-Up Time:** Time taken for the initial pixel operation during the first setup.
- **Average Time:** Average time measured across 100 iterations, excluding warm-up; indicates processing efficiency.
- **Total Time:** Cumulative time for all 100 iterations.

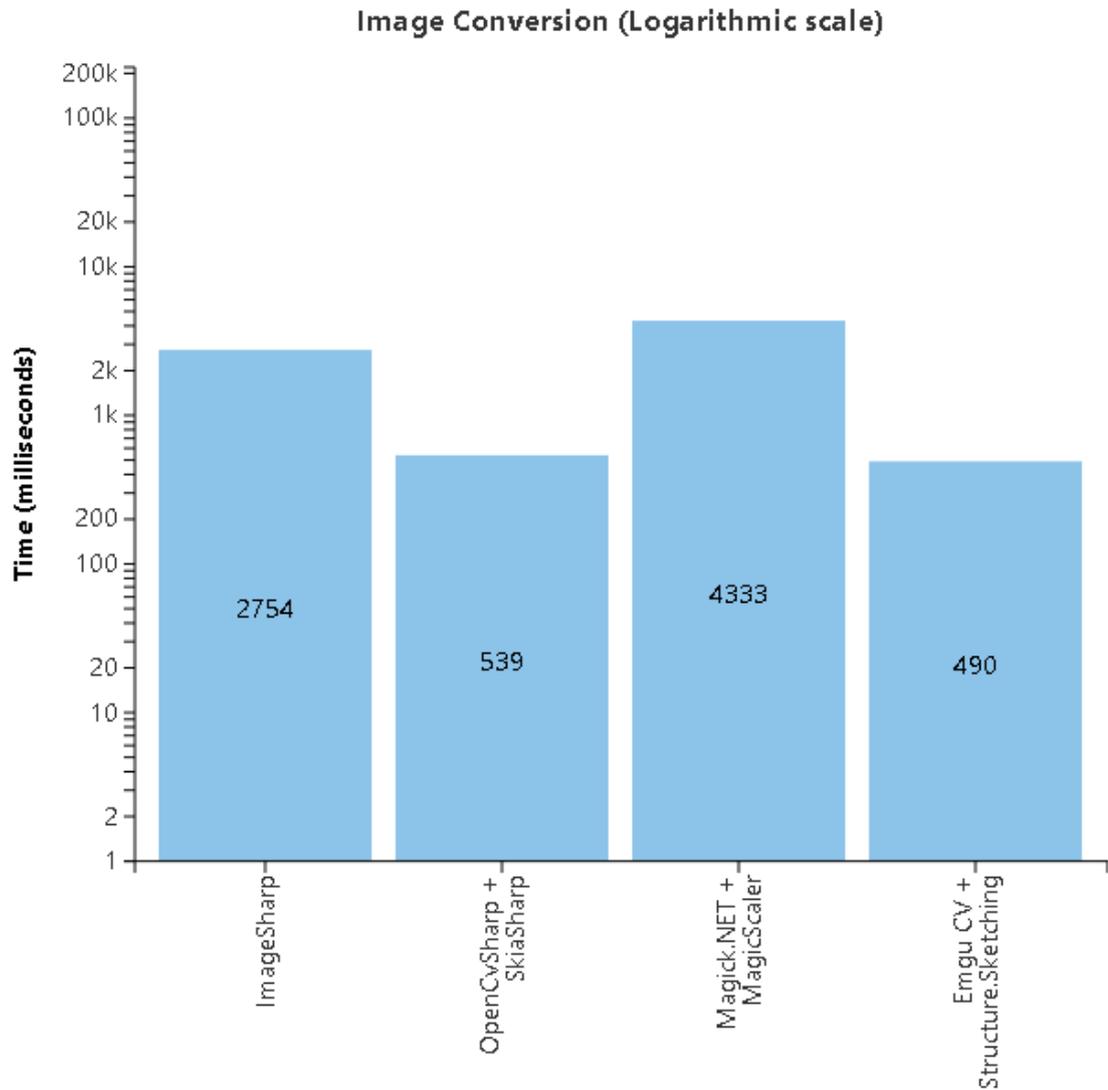
Image Conversion: Time taken to load the image into memory and iterate through every pixel in the image, applying a simple operation (converting to grayscale) and then save the image.

Benchmarking Resources

Image used for this benchmarking: [🔗 Image Link](#)

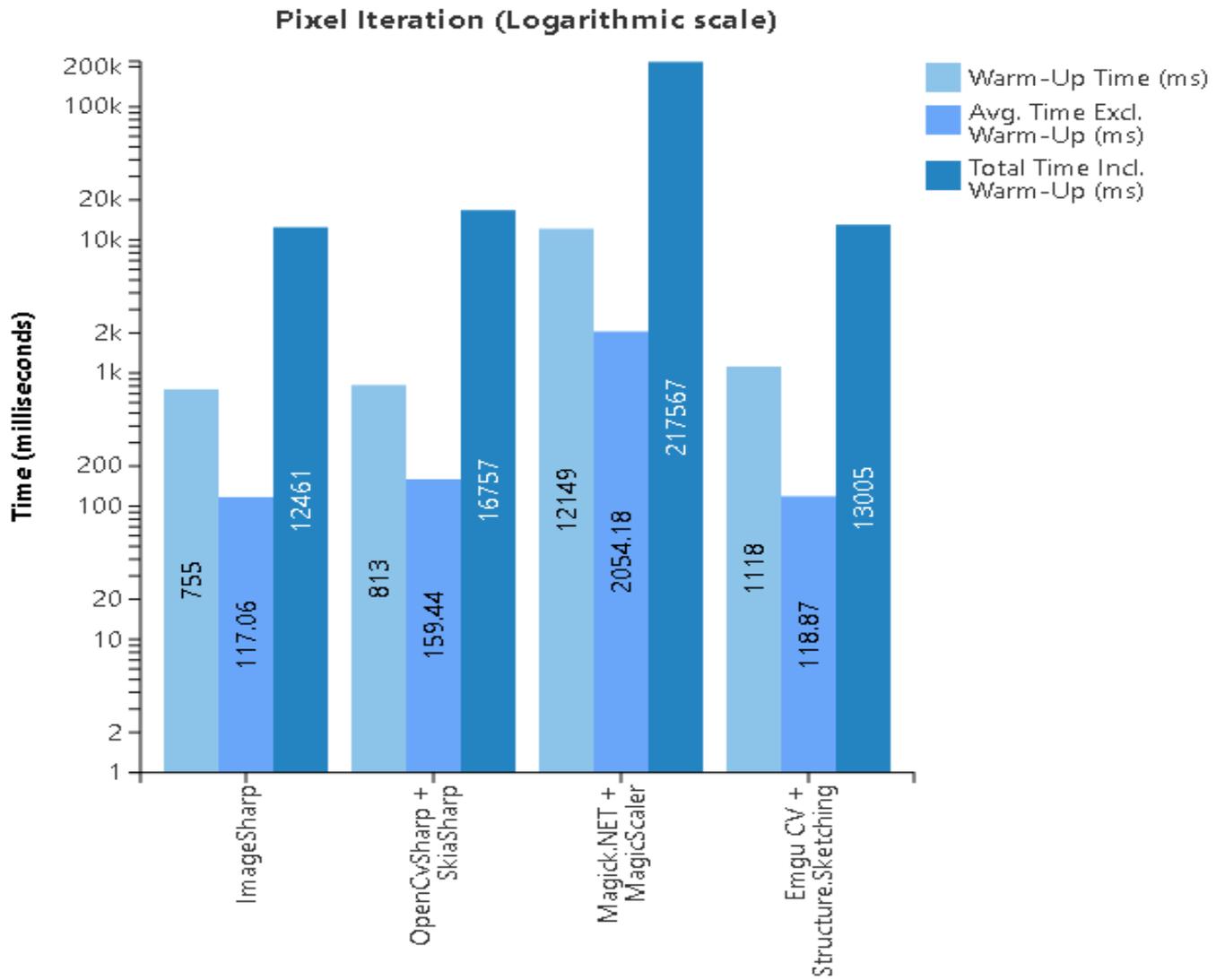
Benchmarking Implementation repository: [🔗 Benchmarking Repository](#)

5.1.2. Benchmark Results:



Library	Benchmark	Time (ms)
ImageSharp	Image Conversion	2754
OpenCvSharp + SkiaSharp	Image Conversion	539
Magick.NET + MagicScaler	Image Conversion	4333
Emgu CV + Structure.Sketching	Image Conversion	490

5.1. Performance Benchmarking Results



Library	Benchmark	Warm-Up Time (ms)	Avg. Time Excl. Warm-Up (ms)	Total Time Incl. Warm-Up (ms)
ImageSharp	Pixel Iteration	755	117.06	12461
OpenCvSharp + SkiaSharp	Pixel Iteration	813	159.44	16757
Magick.NET + MagicScaler	Pixel Iteration	12149	2054.18	217567
Emgu CV + Structure.Sketching	Pixel Iteration	1118	118.87	13005

5.2. Memory Benchmarking

5.2.1. Benchmarking Overview:

In this benchmark, we utilized BenchmarkDotNet to measure the performance of various image processing libraries. BenchmarkDotNet is a powerful .NET library that provides accurate and detailed performance metrics. It handles the complexities of benchmarking, such as warm-up, iteration, and statistical analysis, ensuring reliable results.

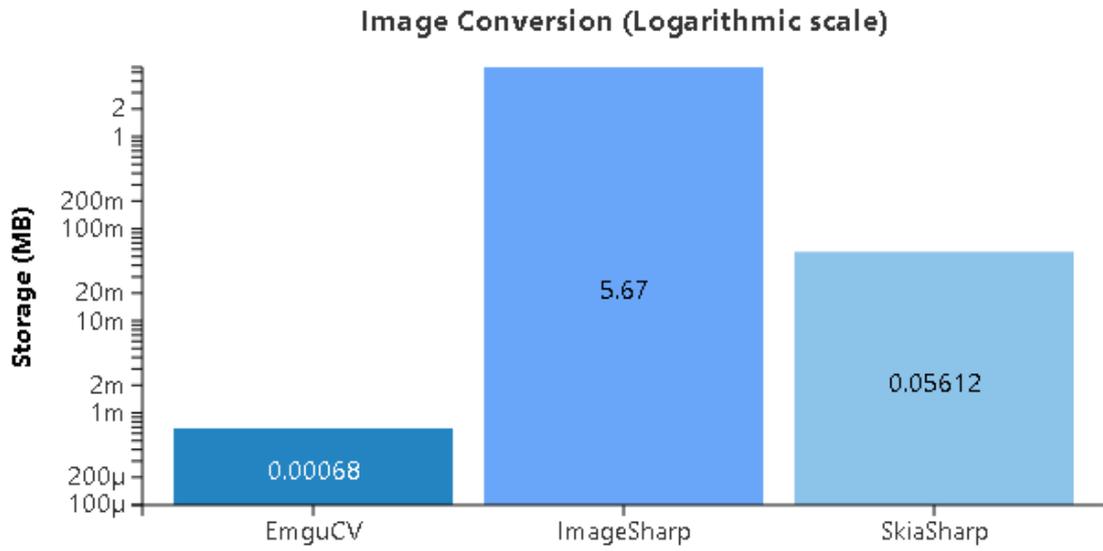
The benchmarking process involved the following steps:

- **Setup:** Initialize the environment and load the image.
- **Warm-Up:** Perform initial iterations to stabilize the environment.
- **Measurement:** Execute the image processing tasks (Image Conversion and Pixel Iteration) and record the time and memory usage.
- **Analysis:** Analyze the collected data to determine the average, total, and warm-up times, as well as memory allocations and garbage collections.

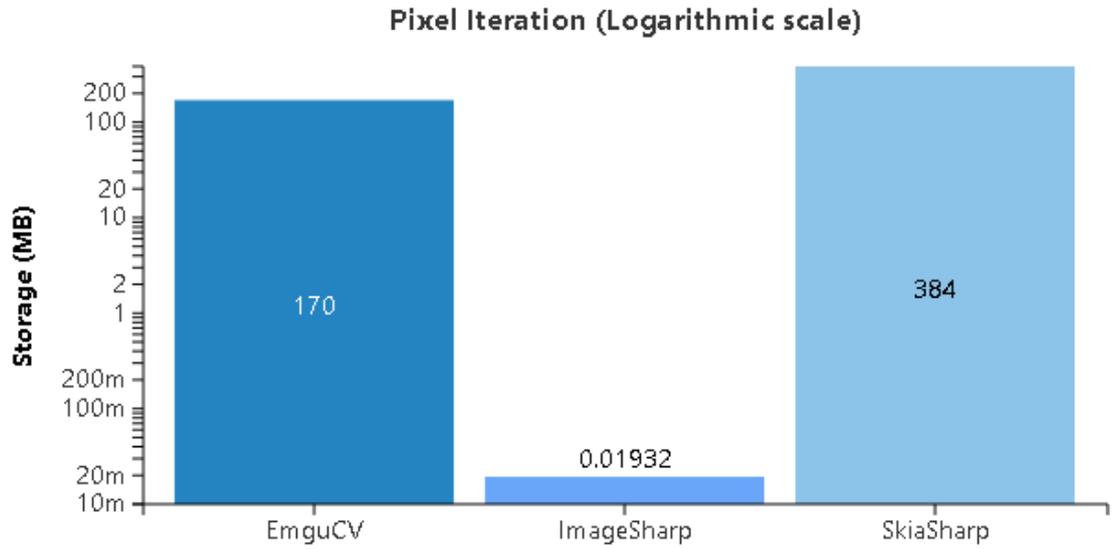
The attached images and tables provide a detailed comparison of the performance metrics for different libraries, highlighting their strengths and weaknesses in various tasks.

5.2. Memory Benchmarking

5.2.2. Benchmark Results:



Library	Task	Mean Time	Allocated Memory	Gen0/Gen1/Gen2 Collections
EmguCV	Image Conversion	52.53 ms	0.00068 MB (712 bytes)	- / - / -
ImageSharp	Image Conversion	475.71 ms	5.67 MB (5,805.41 KB)	1,000 / 1,000 / 1,000
SkiaSharp	Image Conversion	63.97 ms	0.05612 MB (58,864 bytes)	- / - / -



Library	Task	Mean Time	Allocated Memory	Gen0/Gen1/Gen2 Collections
EmguCV	Pixel Iteration	85.49 ms	170.00 MB (177,976,185 bytes)	33,142 / 1,571 / 1,571
ImageSharp	Pixel Iteration	86.56 ms	0.01932 MB (20.26 KB)	- / - / -
SkiaSharp	Pixel Iteration	2.82 s	384.00 MB (403,300,552 bytes)	85 / - / -

5.3. Development Effort Estimation

5.3.1. Overview

In this section, we outline the development effort estimation for implementing image processing functionalities within our project. We evaluate two primary approaches: custom development of an image processing library and the integration of existing external libraries such as OpenCV or ImageMagick.

5.3.2. Custom Development of Image Processing Library

During our team discussions, we evaluated the possibility of developing our own image processing library. While this would give us complete control over the feature set, we found that it would require a significant amount of effort **+100 story points**. For example:

- Implementing basic functionality like converting **BMP to JPEG** alone would require approximately **40 story points**.
- Expanding to cover the full range of necessary image processing features would take an impractically long time for the scope of the project.

Given these factors, developing an internal library from scratch was deemed too resource-intensive, and we opted for an external solution.

We referenced the following resources for additional insights into the complexity of image conversions:

- YouTube Video 1
- YouTube Video 2

5.3.3. Use alternative Image Processing Library

Based on our discussions, the implementation effort for integrating either OpenCV or ImageMagick into our Imagegen and ImageProcessor NuGet packages can only be estimated roughly at this stage. However, we anticipate that the development effort will be approximately **20 or more story points** for either solution. This estimation takes into account the necessary feature scope and the complexity involved in replacing our current image processing functionality.

5.4. Overall Comparison and Key Insights

5.4.1. Overall Comparison:

The benchmarking results highlight the strengths and weaknesses of each library in different tasks. SkiaSharp excels in image conversion tasks due to its fast processing time and low memory usage. EmguCV, while consuming more memory, provides the best performance for

pixel iteration tasks, making it suitable for complex image processing operations. ImageSharp, although efficient in memory usage for pixel iteration, falls short in image conversion performance.

The decision to adopt SkiaSharp for image conversion and EmguCV for complex image processing is based on a balance between performance and cost. SkiaSharp's superior performance in image conversion and EmguCV's comparable performance to ImageSharp, combined with cost savings, make them the preferred choices for their respective tasks.

5.4.2. Key Insights:

- **SkiaSharp** showed excellent performance for **image conversion**, with both the fastest time (63.97 ms) and the least memory allocation (~58 KB). This makes it an ideal choice for **image conversion tasks**.
- **EmguCV** performed best for **pixel iteration** with a fast mean time of **85.49 ms**, though its memory consumption was higher (~170 MB). The extensive **garbage collections** in EmguCV indicate high memory usage, but the performance benefits outweigh the memory cost for more complex operations.
- **ImageSharp** consumed minimal memory for **pixel iteration** (~20 KB), but its **image conversion** performance lagged behind, taking significantly more time than SkiaSharp and consuming more memory (~5.67 MB).

5.4.3. Meeting Outcome and Final Decision:

- **SkiaSharp** will be adopted for **image conversion** tasks due to its superior performance.
- **EmguCV** will be the preferred choice for **complex image processing**. Although its memory consumption is higher than **ImageSharp**, the performance between the two is similar. However, the increased memory usage of **EmguCV** is within a manageable range for our needs. The primary reason for choosing **EmguCV** is the significant cost savings, as **ImageSharp** is considerably more expensive, and switching to **EmguCV** allows us to reduce licensing costs while maintaining comparable performance.

6. Conclusion and Recommendations

6.1. Summary of Findings

A summary of key findings from the evaluation and analysis.

6.2. Final Recommendation

After evaluating the various alternatives and the effort required for custom development, the team has decided to use a combination of **Emgu CV** and **SkiaSharp** for our image processing needs.

- **Reason for Decision:**

- **Emgu CV** provides powerful image processing capabilities, including pixel manipulation, resizing, and format conversions. It is built on OpenCV, which ensures high performance for tasks such as image resizing, cropping, color conversion, and basic filtering.
- **SkiaSharp** complements Emgu CV by providing robust 2D graphics rendering, efficient image creation, and handling of different image formats. It excels at drawing operations and layer composition, making it suitable for both simple and complex image manipulation scenarios.

- **Why This Combination?**

- The combination of Emgu CV and SkiaSharp covers both the low-level, performance-critical image manipulation provided by Emgu CV and the high-level, efficient rendering and image handling offered by SkiaSharp. Together, they meet the project's feature requirements with minimal integration effort. The one-time cost of **\$799** for Emgu CV, while notable, seems reasonable given its extensive capabilities and the absence of recurring subscription fees.

6.3. Future Work

Suggestions for future research include exploring additional libraries and evaluating for specific use cases.

A. Appendices

A.1. Appendix A: Detailed Benchmarking Results

Include detailed benchmarking tables and data here.

A.2. Appendix B: Implementation Details

Code snippets, setup configurations, and other technical details go here.

A.3. Appendix C: Resource Links and Additional Documentation

Provide links, references, and documentation resources here.