# Evaluation of Alternatives to ImageSharp for Image Processing

**Introduction**

The purpose of this investigation is to identify and evaluate potential alternatives to ImageSharp for image processing in our applications. Currently, ImageSharp costs $5,000 per year, which impacts our pricing structure. This review aims to explore cost-effective and efficient alternatives that can replace or complement ImageSharp, thereby optimizing our overall expenditure and performance.

* [Functionalities](#scroll-bookmark-3)(see page 1)
  + [Tasks: (Current Usage of ImageSharp in Common.ImageProcessor and LayerStacking.Base)](#scroll-bookmark-6)(see page 1)
  + [Performance Metrics](#scroll-bookmark-9)(see page 3)
  + [Evaluation of Alternatives](#scroll-bookmark-12)(see page 4)
  + [Evaluation of Alternatives in GO](#scroll-bookmark-15)(see page 18)
  + [Suggestions](#scroll-bookmark-18)(see page 27)
  + [Image Processing Benchmark](#scroll-bookmark-21)(see page 28)
  + [Development Effort Estimation](#scroll-bookmark-24)(see page 32)
  + [Memory Benchmarking](#scroll-bookmark-27)(see page 32)
  + [Final Decision on Image Processing Library](#scroll-bookmark-30)(see page 34)

# Functionalities

Imagegen, Show, Create

## Tasks: (Current Usage of ImageSharp in Common.ImageProcessor and LayerStacking.Base)

### 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.

### 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.

### 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.

### 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.

### 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.

### 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).

### Cropping and Resizing

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

### 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.

### Composing Image Layers

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

### 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.

### Saving the image

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

## Performance Metrics

* **Image Conversion Test:**

Measure the time taken to load an image and convert its format using each library.

Steps:

* 1. Load the image into memory.
  2. Convert the image to another format (JPG to PNG).
  3. Save the converted image to disk.

Metric: Time taken to load the image and to convert the image format.

* **Pixel Iteration Test:**

Measure the time taken 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:

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

Metric: Time taken to complete the pixel iteration process.

## Evaluation of Alternatives

### 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** | - **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. | - Asynchronous image loading (no equivalent to Image.LoadAsync). |
| **Image Processing and Manipulation** | - Basic pixel manipulation through scanlines/tiles. - **ImageBufAlgo** provides some algorithms for manipulation. | - No built-in functions for Clone, Mutate, Resize, Grayscale. - Requires external libraries (e.g., OpenCV) or custom code for advanced processing. |
| **Pixel Formats** | - Supports various pixel formats including RGBA, RGB, L (grayscale), YUV, etc. - **ImageBufAlgo::colorconvert** allows conversion between formats. | - Specific formats like Byte4, YCbCr may require manual handling. |
| **Pixel Access and Manipulation** | - Provides pixel access through **ImageInput::read\_scanline** and **ImageInput::read\_tile**. - Can manipulate pixels by reading and writing scanlines/tiles. | - No direct method like ProcessPixelRows; manual processing required. |
| **Image Metadata and Conversion** | - **ImageSpec**: Handles image metadata. - **ImageBuf**: Manages pixel data and metadata. - **ImageBufAlgo**: Offers conversion algorithms. | - Handling complex metadata and conversions might require custom implementation depending on needs. |
| **Creating and Disposing Instances** | - **ImageBuf**: Creates and manages image instances. - Resources automatically managed in C++ (via destructors). | - Explicit disposal may be needed for resource-intensive operations, especially in languages without automatic garbage collection. |
| **Cropping and Resizing** | - **ImageBufAlgo::crop**: Crops images. - **ImageBufAlgo::resize**: Resizes images with various techniques. | - No direct equivalent to Mutate for fluent transformations. |
| **Encoding Images in Various Formats** | - Supports encoding in multiple formats (BMP, JPEG, PNG, TIFF, **WebP,** etc.) via **ImageOutput**. | - none. |
| **Composing Image Layers** | - **ImageBufAlgo::paste**: Combines image layers or tiles. | - No built-in methods equivalent to ImageSharp's Stitch method. |
| **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. |

### 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** | - **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. | - No built-in asynchronous image loading (no equivalent to Image.LoadAsync). |
| **Image Processing and Manipulation** | - **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. | - No high-level fluent API like ImageSharp's Mutate for chaining multiple operations. |
| **Pixel Formats** | - Supports various pixel formats like SKColorType.Rgba8888, SKColorType.Bgra8888, SKColorType.Gray8, etc. - **SKColorSpace**: Manages color space conversions. | - Specific formats like Byte4, YCbCr may require custom conversion. |
| **Pixel Access and Manipulation** | - **SKBitmap.GetPixel** / **SetPixel**: Accesses and sets individual pixels. - **SKBitmap.Pixels**: Provides access to the pixel data for bulk manipulation. | - No direct method like ProcessPixelRows; manual processing of pixel rows is required. |
| **Image Metadata and Conversion** | - **SKImageInfo**: Manages basic image properties such as dimensions and color type. - **SKImage.Encode**: Converts images into various formats like PNG, JPEG, **WebP**, etc. | - Limited support for complex metadata handling compared to ImageSharp and OIIO. |
| **Creating and Disposing Instances** | - **SKBitmap** and **SKImage**: Create and manage image instances. - Proper resource management using the Dispose method is necessary to free resources. | - No direct equivalent to Image<Rgba32>—pixel format and dimensions must be managed manually. |
| **Cropping and Resizing** | - **SKBitmap.ExtractSubset**: Crops images. - **SKBitmap.Resize**: Resizes images with different resampling techniques. | - No built-in equivalent to ImageSharp's Mutate method for complex transformations. |
| **Encoding Images in Various Formats** | - **SKImage.Encode**: Encodes images in multiple formats (e.g., PNG, JPEG, BMP, WEBP). | - Custom handling might be needed for less common formats. |
| **Composing Image Layers** | - **SKCanvas.DrawBitmap**: Composes images by drawing one bitmap onto another. - **SKPicture**: Records a sequence of drawing commands for later playback and compositing. | - No built-in method equivalent to ImageSharp's Stitch for seamless image stitching. |
| **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. |

### 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). |
| **Image Processing and Manipulation** | - **MagickImage.Clone**: Clones the image. - **MagickImage.Resize**: Resizes images with various filtering and resampling options. - **MagickImage.Crop**: Crops images. - **MagickImage.Grayscale**: Converts images to grayscale. - **MagickImage.ColorSpace**: Converts between different color spaces. - **MagickImage.Rotate, MagickImage.Flip, MagickImage.Flop**: Performs various image transformations. | - Magick.NET supports extensive image processing, similar to ImageSharp’s Mutate method. Custom implementation is rarely needed. |
| **Pixel Formats** | - Supports a wide range of pixel formats, including RGBA, RGB, Gray, CMYK, and more. - **MagickColor**: Handles color conversions and supports various color profiles. | - Fully supports advanced pixel formats and color management, so custom implementation is minimal. |
| **Pixel Access and Manipulation** | - **MagickImage.GetPixels**: Provides access to individual pixels or pixel regions. - **MagickImage.SetPixels**: Sets individual pixels. - **MagickImage.ToByteArray**: Converts pixel data to a byte array. | - No direct method like ProcessPixelRows; however, pixel manipulation is flexible and powerful. |
| **Image Metadata and Conversion** | - **MagickImage.Attribute**: Accesses and manipulates image metadata such as EXIF, IPTC, and XMP. - **MagickImage.Format**: Converts images to various formats. | - Magick.NET offers comprehensive metadata handling and format conversion. |
| **Creating and Disposing Instances** | - **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. | - Fully supports instance creation and disposal, with comprehensive memory management. |
| **Cropping and Resizing** | - **MagickImage.Crop**: Crops images. - **MagickImage.Resize**: Resizes images with customizable resampling options. - **MagickImage.AdaptiveResize**: Provides advanced resizing techniques. | - Fully supports cropping and resizing with advanced options, no need for custom implementation. |
| **Encoding Images in Various Formats** | - **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. | - Supports a wider range of formats than ImageSharp, with built-in encoding capabilities. |
| **Composing Image Layers** | - **MagickImage.Composite**: Composes one image over another. - **MagickImage.Mosaic**: Combines multiple images into a mosaic. - **MagickImageCollection**: Handles layering for complex compositions. | - Fully supports complex image compositions, with built-in methods for layering and merging. |
| **Resampling Methods** | - **MagickImage.Resample**: Provides advanced resampling methods and filtering options. - **MagickImage.AdaptiveResize**: Offers specialized resampling techniques. | - Extensive support for resampling, surpassing basic needs and requiring no custom implementation. |
| **Saving the Image** | - **MagickImage.Write**: Saves images to files, streams, or byte arrays in the desired format. - **MagickImage.Save**: Provides simple saving options. | - No built-in asynchronous saving method; async saving requires custom implementation. |

### 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** | - **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. | - No built-in asynchronous image loading (no equivalent to Image.LoadAsync). |
| **Image Processing and Manipulation** | - **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. | - Emgu CV provides extensive support for image processing operations, similar to ImageSharp's Mutate method. |
| **Pixel Formats** | - 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. | - Default BGR format might differ from ImageSharp’s RGBA, requiring format conversion in some cases. |
| **Pixel Access and Manipulation** | - **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. | - Pixel manipulation is supported, but the approach differs from ImageSharp's more abstracted methods. |
| **Image Metadata and Conversion** | - **CvInvoke.Imencode**: Converts the image to various formats (PNG, JPEG, etc.). - **CvInvoke.Imwrite**: Saves images to files. | - Does not extensively handle metadata like EXIF or IPTC, focusing more on basic properties and format conversion. |
| **Creating and Disposing Instances** | - **Mat**: Can be used to create empty images, with proper disposal via **Dispose** to free resources. - **Image<TColor, TDepth>**: Manages images with type safety. | - Fully supports instance creation and disposal, requiring careful memory management due to OpenCV’s low-level handling. |
| **Cropping and Resizing** | - **CvInvoke.GetRectSubPix**: Crops the image. - **CvInvoke.Resize**: Resizes the image with advanced interpolation options. | - Emgu CV supports cropping and resizing extensively, similar to ImageSharp’s Mutate method. |
| **Encoding Images in Various Formats** | - **CvInvoke.Imwrite**: Saves images in formats like PNG, JPEG, BMP, **WebP** etc. - **CvInvoke.Imencode**: Encodes images for various formats and uses. | -  none, |
| **Composing Image Layers** | - **CvInvoke.AddWeighted**: Blends two images together, allowing for composition. - **CvInvoke.CopyMakeBorder**: Combines images into a larger canvas. | - Supports basic layer composition, though complex operations may require additional code or OpenCV functions. |
| **Resampling Methods** | - **CvInvoke.Resize**: Provides multiple resampling methods (linear, cubic, nearest-neighbor). | - Resampling techniques are fully supported, similar to ImageSharp's capabilities. |
| **Saving the Image** | - **CvInvoke.Imwrite**: Saves images to files in the desired format. | - No built-in asynchronous saving method; async operations require custom implementation. |

### 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** | - **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. | - No direct method for creating new images from scratch. - Lacks asynchronous methods like Image.LoadAsync in ImageSharp. |
| **Image Processing and Manipulation** | - **MagicImageProcessor.ProcessImage**: Supports resizing, cropping, and color adjustments. - **ProcessImageSettings**: Allows specifying transformations such as resizing, cropping, and color adjustments. | - Lacks complex manipulation like cloning, rotating, flipping, or direct pixel manipulation. - Does not support advanced image editing features found in ImageSharp. |
| **Pixel Formats** | - Supports various pixel formats including RGBA, RGB, Gray, etc. - Automatically handles color space conversions. | - Handling of pixel formats is abstracted, with no direct pixel manipulation like in ImageSharp. |
| **Pixel Access and Manipulation** | - Provides high-level processing but no direct pixel access. | - No direct access or manipulation of individual pixels, unlike ImageSharp's ProcessPixelRows. |
| **Image Metadata and Conversion** | - Handles basic image properties and metadata. - Supports conversion between different image formats. | - Limited metadata handling compared to libraries like ImageSharp, focusing more on image processing efficiency. |
| **Creating and Disposing Instances** | - Focuses on processing existing images rather than creating new ones. - Managed disposal of resources. | - Does not support creating images from scratch. - No manual instance creation like in ImageSharp. |
| **Cropping and Resizing** | - **ProcessImageSettings.Crop**: Specifies cropping options. - **ProcessImageSettings.Width and Height**: Specifies resizing dimensions. - Uses high-quality resampling algorithms for resizing. | - Cropping and resizing are part of the processing pipeline, not standalone operations. |
| **Encoding Images in Various Formats** | - Supports encoding images into formats like PNG, JPEG, BMP, **WebP**, etc. - Can control compression, quality, and other encoding parameters. | - Lacks format-specific customization that ImageSharp provides through various encoders. |
| **Composing Image Layers** | - Not supported; focuses on single-image processing. | - Lacks capabilities for composing or layering multiple images, unlike ImageSharp's Stitch or DrawImage. |
| **Resampling Methods** | - Provides high-quality resampling techniques for resizing. - **ProcessImageSettings.Interpolation**: Allows specifying the interpolation method. | - Resampling is integrated into the processing pipeline, without a direct interface like IResampler in ImageSharp. |
| **Saving the Image** | - **MagicImageProcessor.ProcessImage**: Saves the processed image to a file, stream, or byte array. | - No asynchronous saving method, with all operations handled synchronously. |

### 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** | - **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. | - 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** | - **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. | - 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** | - Supports a variety of formats, including scalar, vector, and label images. - Automatic pixel type conversions. | - Limited in handling non-medical specific color spaces and formats compared to ImageSharp. |
| **Pixel Access and Manipulation** | - **sitk.GetPixel/SetPixel:** Access and manipulate individual pixels. - Full image data accessible via NumPy arrays for batch processing. | - Less intuitive pixel manipulation compared to ImageSharp's ProcessPixelRows. - Focus on scientific data rather than general-purpose image formats. |
| **Image Metadata and Conversion** | - **sitk.Image:** Handles metadata such as image origin, spacing, and direction. - **sitk.Cast:** Converts images to various pixel types. | - Limited advanced metadata handling, focusing on medical image properties. |
| **Creating and Disposing Instances** | - **sitk.Image:** Creates empty images. - Automatic resource management with Python's garbage collector. | - Does not offer the fine-grained control over image creation seen in ImageSharp. |
| **Cropping and Resizing** | - **sitk.Crop:** Crops the image. - **sitk.Resample:** Provides resizing with multiple interpolation methods. | - Requires more manual setup compared to the intuitive API of ImageSharp's Mutate. |
| **Encoding Images in Various Formats** | - **sitk.WriteImage:** Supports encoding and saving images in formats like PNG, JPEG, and TIFF. | - Focuses more on medical image formats, and does not natively support **WebP**.​ |
| **Composing Image Layers** | - Can handle multi-channel images, simulating layer composition. - **sitk.LabelOverlay:** Can overlay labels on grayscale images. | - Lacks direct support for layer-based compositions found in ImageSharp. - Limited artistic composition tools. |
| **Resampling Methods** | - **sitk.Resample:** Offers various resampling methods with advanced interpolation options. | - Resampling is powerful but less user-friendly than ImageSharp's high-level options. |
| **Saving the Image** | - **sitk.WriteImage:** Saves images to files in various formats. - Supports scientific formats like NIfTI, DICOM. | - Limited optimization for modern web formats compared to ImageSharp. |

### 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 Structure.Sketching** | **Not Natively Supported / Requires Custom Implementation** |
| **Image Loading and Creation** | - **Image.Load:** Loads images from various sources. - **Image:** Creates new images with specified dimensions and formats. - **ImageConstructor:** Loads images from byte arrays, files, or streams. - **NewImage:** Creates blank images. | - No asynchronous loading methods like Image.LoadAsync in ImageSharp. |
| **Image Processing and Manipulation** | - **Image.Clone, Image.Resize, Image.Crop:** For cloning, resizing, and cropping. - **Image.Grayscale:** Converts to grayscale. - **Image.Mutate:** Allows chainable transformations. - **Image.Rotate, Image.Flip:** Provides basic geometric transformations. | - Limited specialized filters and effects compared to ImageSharp's extensive collection. |
| **Pixel Formats** | - Supports multiple formats, including **Rgba32, Rgb24, L8, RGBA, RGB, Grayscale**. - Handles color space conversions. | - Less advanced color space handling for professional-grade imaging compared to ImageSharp. |
| **Pixel Access and Manipulation** | - **GetPixel, SetPixel:** Provides direct pixel manipulation. - **Image.Pixels:** Allows batch processing of pixel data. | - Less intuitive than ImageSharp’s ProcessPixelRows for complex pixel operations. |
| **Image Metadata and Conversion** | - Handles basic image properties and metadata. - **Image.Metadata:** Manages EXIF and other metadata. - **Image.Save, Image.Format:** Converts images between formats. | - Lacks advanced metadata handling options available in specialized image libraries. |
| **Creating and Disposing Instances** | - **Image:** Allows creation of empty images. - **Dispose:** Ensures proper resource management. - Disposal managed by the library. | - Simpler resource management but may require more manual handling than ImageSharp’s built-in garbage collection. |
| **Cropping and Resizing** | - **Image.Crop:** Crops images. - **Image.Resize:** Resizes with different scaling methods and resampling techniques. | - Resizing features are robust but lack some of the more advanced resampling techniques found in ImageSharp. |
| **Encoding Images in Various Formats** | - **Image.Save:** Saves images in various formats. - **Image.Export:** Offers additional export options. | - Limited optimization for specific formats compared to ImageSharp’s finely tuned encoders.  - Supports **WebP** through additional encoders. |
| **Composing Image Layers** | - **Image.DrawImage:** Supports composing image layers. - **LayeredImage:** Supports multiple image layers and blending. | - Less flexible layer manipulation compared to ImageSharp’s comprehensive layering system. |
| **Resampling Methods** | - **Image.Resize, Image.Resample:** Provides various interpolation methods for resampling and resizing. - Ensures high-quality scaling. | - Limited interpolation methods compared to the advanced options available in ImageSharp. |
| **Saving the Image** | - **Image.Save:** Saves images to files or streams. - **Image.Export:** Provides flexible saving options. | - Does not offer the same level of control over compression and quality settings as ImageSharp. |

### 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** | - **Cv2.ImRead, Cv2.ImDecode:** Loads images from files or byte arrays. - **Mat, Cv2.ImCreate:** Creates new images with specified dimensions and types. | - No asynchronous loading methods like Image.LoadAsync in ImageSharp. |
| **Image Processing and Manipulation** | - **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. | - Lacks some of the more advanced image effects and filters available in ImageSharp. |
| **Pixel Formats** | - Supports multiple formats, including **CvType.CV\_8UC3** (BGR), **CvType.CV\_8UC1** (Grayscale), **CvType.CV\_8UC4** (BGRA). - Handles automatic color space conversion. | - May require custom implementations for less common pixel formats. |
| **Pixel Access and Manipulation** | - **Mat.At, Mat.Set:** Direct pixel access and manipulation. - **Mat.Data:** Provides low-level access to pixel data arrays. | - Less intuitive pixel manipulation compared to ImageSharp’s ProcessPixelRows. |
| **Image Metadata and Conversion** | - **Mat:** Stores image properties like size and type. - **Cv2.ImEncode, Cv2.ImWrite:** Converts and saves images in various formats. | - Limited metadata handling compared to ImageSharp’s extensive metadata support. |
| **Creating and Disposing Instances** | - **Mat:** Creates and initializes images. - **Dispose:** Required for freeing unmanaged resources. - Effective memory management through manual disposal. | - More complex resource management due to reliance on unmanaged resources. |
| **Cropping and Resizing** | - **Cv2.GetRectSubPix:** For precise cropping. - **Cv2.Resize:** Resizing with various interpolation methods. | - Lacks some of the more advanced cropping techniques like those in ImageSharp. |
| **Encoding Images in Various Formats** | - **Cv2.ImWrite:** Saves images in multiple formats including **WebP**. - **Cv2.ImEncode:** For encoding images to byte arrays. | - Less flexible format optimization compared to ImageSharp’s encoders. |
| **Composing Image Layers** | - **Cv2.AddWeighted, Cv2.Add, Cv2.Subtract:** For image blending and compositing. | - Less comprehensive layering system than ImageSharp. |
| **Resampling Methods** | - **Cv2.Resize:** Offers multiple interpolation methods. - **Cv2.PyrUp, Cv2.PyrDown:** Pyramid methods for scaling. | - Fewer options for specialized resampling methods compared to ImageSharp. |
| **Saving the Image** | - **Cv2.ImWrite, Cv2.ImEncode:** Saves images to files or byte arrays. | - Lacks some advanced saving options like those in ImageSharp. |

### 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** | - **IImageLoadingService.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** | - **IImage.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** | - 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. |
| **Pixel Access and Manipulation** | - Focuses on high-level operations, without direct pixel access. - Does not offer low-level pixel manipulation like GetPixel and SetPixel. | - No direct pixel access, unlike ImageSharp's ProcessPixelRows. |
| **Image Metadata and Conversion** | - Handles basic image properties and conversion via **IImage.Save**. - Can save to various formats. | - Less detailed metadata handling compared to ImageSharp. |
| **Creating and Disposing Instances** | - **GraphicsPlatform.CreateImage:** Easily creates new images. - Disposal managed via .NET’s garbage collection. | - Lacks manual resource management options, unlike some lower-level libraries. |
| **Cropping and Resizing** | - **ICanvas.DrawImage:** Supports cropping and resizing with simple interfaces. | - Less advanced cropping and resizing techniques compared to ImageSharp. |
| **Encoding Images in Various Formats** | - **IImage.Save:** Supports encoding in multiple formats. - Provides functionality to save directly to files, streams, or byte arrays. | - Limited format-specific encoding settings compared to ImageSharp and do not mention **WebP** support. |
| **Composing Image Layers** | - **ICanvas.DrawImage:** Allows layering images over one another. - Basic support for image compositing and blending. | - More limited compositing features compared to ImageSharp. |
| **Resampling Methods** | - **ICanvas.DrawImage:** Automatically handles resampling during resizing. | - Fewer advanced resampling methods compared to ImageSharp. |
| **Saving the Image** | - **IImage.Save:** Saves images to files, streams, or byte arrays in various formats. | - Lacks advanced saving options and optimizations available in ImageSharp. |

### 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** | - **LoadImage, LoadAsync:** Robust image loading with asynchronous support. - **RasterImage.Create:** Flexible image creation with control over dimensions and pixel formats. | - None; LEADTOOLS provides comprehensive loading and creation features. |
| **Image Processing and Manipulation** | - **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. | - While comprehensive, some specific manipulation tasks might require custom scripting or code. |
| **Pixel Formats** | - **Supports multiple formats:** Extensive support for various pixel formats such as RGB, RGBA, Grayscale, CMYK, and more. | - None; pixel format support is exhaustive. |
| **Pixel Access and Manipulation** | - **GetPixel, SetPixel, RasterImageData:** Direct access and manipulation of pixel data. - **Lock/Unlock:** For precise pixel-level operations. | - Pixel data manipulation can be complex, especially when locking/unlocking is required. |
| **Image Metadata and Conversion** | - **Comprehensive Metadata Handling:** Extracts, edits, and writes metadata for a wide range of formats. - **Advanced Conversion:** Extensive format support including DICOM. | - None; LEADTOOLS excels in metadata and conversion capabilities. |
| **Creating and Disposing Instances** | - **CreateImage, Dispose:** Facilitates creation and proper resource disposal of images. | - Proper disposal requires careful management, similar to .NET libraries. |
| **Cropping and Resizing** | - **Crop, Resize:** Advanced cropping and resizing with various resampling methods. | - None; cropping and resizing are robustly supported. |
| **Encoding Images in Various Formats** | - **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. | - Format-specific settings might require additional configuration. |
| **Composing Image Layers** | - **Combine, DrawImage:** Supports complex image layering with detailed control over blending and transparency. | - Layer management might need additional custom code for complex use cases. |
| **Resampling Methods** | - **Advanced Resampling:** Provides high-quality resampling techniques during resizing. | - Resampling methods are extensive but need careful selection for optimal results. |
| **Saving the Image** | - **Save, SaveAsync:** Saves images with extensive control over parameters. - **Asynchronous Saving:** Supports performance-enhanced operations. | - None; saving features are comprehensive and powerful. |

## Evaluation of Alternatives in GO

### 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** | - **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 | - Unlike ImageSharp, there is limited support for creating images from scratch in more diverse formats.   ex. **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.  - Do not inherently support asynchronous operations ,can achieve asynchronous behavior by running image-related operations in separate goroutines |
| **Image Processing and Manipulation** | - **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. | - Require custom code implementation |
| **Pixel Formats** | -I**mage.RGB** : Represents a color image with 8-bit RGBA values per pixel.  -I**mage.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. | -  Package provides basic support for different image formats but does not explicitly handle pixel formats like image processing libraries |
| **Pixel Access and Manipulation** | - It doesn't have direct method like getpixel,setpixel or RasterImageData | -No direct pixel access, unlike ImageSharp's ProcessPixelRows. |
| **Image Metadata and Conversion** | - it 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 - It dose not support Advance conversion | - Lack of support require custom implementations. |
| **Creating and Disposing Instances** | -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 don't need to manually dispose of images | - The Go Imagepackage 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** | -  Doesn't include built-in support for more advanced operations like cropping and resizing directly. | - Require custom implementations. |
| **Encoding Images in Various Formats** | - **Save:** Create a file using os.Create , then use png.Encoder or jpeg.Encoder to encode and save the image. - It does not directly support more advanced multimedia or document formats. | - For other image formats like **WebP**, need import the corresponding encoding package. |
| **Composing Image Layers** | - Does not directly support complex operations like combining images or drawing one image onto another. | - Require custom implementations.. |
| **Resampling Methods** | - Does not natively support advanced resampling techniques. | - It provides basic image manipulation capabilities, including resizing, but it 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 |

### 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 | -  dose not support image loading and creating need need to third part package |
| **Image Processing and Manipulation** | -Resize **:** Is supported  -it does not include advanced image processing capabilities | - Require custom code implementation |
| **Pixel Formats** | -it supports several basic pixel formats, | -The bild package in Go supports several basic pixel formats but it 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 | -it 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** | - it doesn't provide image metadata and conversion features for handling image metadata or advanced image format conversion | - Lack of support require custom implementations. |
| **Creating and Disposing Instances** | -it doesn't 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. |
| **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** | - it doesn't directly handle encoding images to various formats. | - Requires third-party packages to handle formats like **WebP**. |
| **Composing Image Layers** | - **B**lend.Overlay **:**The function is used to combine the two images.  -The bild library offers various blending modes such as Add, Multiply, Screen, | - None |
| **Resampling Methods** | Provide various resampling method :  -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 |

### 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** | -**gocv.IMRead :** function, which reads an image file from disk into a gocv.Mat object.  -**gocv.IMWrite** :  functions write images, and use the Mat type to manipulate them in memory | -  None ; dose support read and write of image. |
| **Image Processing and Manipulation** | - **Clone, Resize, ConvertToGrayscale:** Supports advanced image processing tasks. - **Transformations, Annotations:** Transformations such as resizing, rotating, and cropping can be done using the 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 for Go provides support for various pixel formats through its integration with OpenCV. |
| **Pixel Access and Manipulation** | -  img.Cols(),img.Rows() : provide the dimensions of the image. -**img.At(x, y) :** to get the pixel value at the specified coordinates.  **-img.Set() :**Modify Pixel Values | -GoCV, a Go wrapper for the OpenCV library, allows pixel access and manipulation in images. |
| **Image Metadata and Conversion** | - It supports various operations, including image metadata extraction and conversion. | - For more advanced metadata extraction (such as EXIF data), might need additional libraries or tools, as gocv itself doesn’t handle metadata like EXIF. |
| **Creating and Disposing Instances** | -**Mat**: Use Close() to release the matrix object when you are done with it.  -**gocv.NewMat()** Create 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. we can specify the format by changing the file extension in the function parameter.. | -  supports various formats like JPEG, PNG, etc. we can specify the format by changing the file extension in the function parameter.  -  Like OpenCV, **WebP** is not supported by default. |
| **Composing Image Layers** | **- gocv.AddWeighted:** blends the images based on the specified weights. In this case, both images are given equal weight.  -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 :** Save the processed image | - To perform asynchronous saving, need to use Go's concurrency features such as goroutines |

### 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** | -it dose not supportImage Loading and Creation | -  dose not supportImage Loading and Creation |
| **Image Processing and Manipulation** | - **Clone, Resize, ConvertToGrayscale:** Supports advanced image processing tasks. - **Transformations, Annotations:** Transformations supported but Annotations dose not supported - **Image Enhancement:**  It 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.  - we need use gift for image transformations and another library like gg for annotations. |
| **Pixel Formats** | - it does not directly provide pixel format manipulation or conversion functionalities**.** | -need to write custom implementation . |
| **Pixel Access and Manipulation** | -  it does not provide pixel-level access and manipulation | -gift's built-in functions aren't sufficient. |
| **Image Metadata and Conversion** | -  it does not directly handle image metadata or conversion between different formats. | - The gift package does not provide direct support for format conversion. |
| **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 :** crop 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** | -it dose 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** | -  it doesn't directly handle file I/O for saving | - To perform asynchronous saving, need to use Go's concurrency features such as goroutines |

### 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** | -**imagick.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** | - **Clone, Resize, ConvertToGrayscale:** Supports advanced image processing tasks. - **Transformations, Annotations:** Supports image processing tasks. - **Image Enhancement:**  It supports operations like sharpening, blurring, and contrast, and Noise reduction | -  All Advance 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 |
| **Pixel Access and Manipulation** | -it 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** | -  **Reading Metadata**: Imagick allows to read metadata from images, such as EXIF data, image properties, and other relevant information.  **-Writing Metadata**: we can also manipulate and write metadata back to the image file using Imagick. This includes adding or modifying existing metadata entries  **-Format Conversion**: Imagick supports conversion between a wide range of image formats (e.g., BMP, PNG, JPEG, TIFF).  **-Quality and Compression**: When converting images, Imagick allows you to adjust the quality and compression settings, which can be important for optimizing the size and appearance of the output image. | - Robust support for image metadata and conversion |
| **Creating and Disposing Instances** | -**imagick.NewMagickWand :** to create image instances.  -**imagick.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** | -  **imagick.MagickResizeImage:**resizes  image the  image  -**imagick.MagickCropImage:** crop image to the specified dimensions | - None |
| **Encoding Images in Various Formats** | -This library allows  to encode and decode images in various formats. Some of the most commonly supported formats include jpeg/jpg, bpm, png, **WebP**, gif etc. | None |
| **Composing Image Layers** | **-base.png :**is the base image  **-overlay.png :** is the image to be composited on top of the image  **-imagick.COMPOSITE\_OP\_OVER :** specifies the compositing operation (in this case, overlay). | -  None |
| **Resampling Methods** | -provides a variety of resampling methods for image | -  None; Resampling robustly supported. |
| **Saving the Image** | -  **SetImageFormat()** :sets the format of the image (in this case, PNG). - **WriteImage** : writes the image to a new file | -does not directly support asynchronous operations for image saving. ImageMagick itself operates synchronously  - To perform asynchronous saving, need to use Go's concurrency features such as goroutines |

## Suggestions

### 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.

### 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](https://github.com/dlemstra/Magick.NET?tab=Apache-2.0-1-ov-file#Apache-2.0-1-ov-file)
    - **MagicScaler**: Free for commercial use.

### Suggestion 3: LEADTOOLS (Single Library Solution)

* + **Strong points**: Comprehensive image processing, extensive format support, advanced metadata handling, versatile imaging features.
  + **Pricing**: Paid (More than 17k$).

### 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.

## Image Processing Benchmark

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.

**\*\* Image used for this benchmarking:  link**[**Image Link**](https://dsext001-eu1-215dsi0708-3dswym.3dexperience.3ds.com/api/media/streammedia/id/E_f8pedmRKKh2YOrjjqJoQ/type/picture/key/l1/update/f79c98dbee35f056b55aeabe4f2b6845)

**\*\*Benchmarking Implementation repository: link**[**Benchmarking Repository**](https://gd-git.dsone.3ds.com/SFI19/sc-176)



|  |  |  |
| --- | --- | --- |
| **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 |



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **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 |

## Development Effort Estimation

### 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](https://www.youtube.com/watch?v=Kv1Hiv3ox8I)
* [YouTube Video 2](https://www.youtube.com/watch?v=0me3guauqOU)

### 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.

## Memory Benchmarking

In our evaluation of alternatives to ImageSharp, we benchmarked EmguCV, SkiaSharp, and ImageSharp using **BenchmarkDotNet**. The focus was on memory consumption and performance in two key metrics: **image conversion** and **pixel iteration**.

### Benchmark Results:





|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Library | Task | Mean Time | Allocated Memory | Gen0/Gen1/Gen2 Collections |
| **EmguCV** | Image Conversion | 52.53 ms | 0.00068 MB (712 bytes) | - / - / - |
| Pixel Iteration | 85.49 ms | 170.00 MB (177,976,185 bytes) | 33,142 / 1,571 / 1,571 |
| **ImageSharp** | Image Conversion | 475.71 ms | 5.67 MB (5,805.41 KB) | 1,000 / 1,000 / 1,000 |
| Pixel Iteration | 86.56 ms | 0.01932 MB (20.26 KB) | - / - / - |
| **SkiaSharp** | Image Conversion | 63.97 ms | 0.05612 MB (58,864 bytes) | - / - / - |
| Pixel Iteration | 2.82 s | 384.00 MB (403,300,552 bytes) | 85 / - / - |

### 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).

### 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.

## Final Decision on Image Processing Library

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.