

# FaceAppWithGPT2

## FaceAppWithGPT2

### Dependencies

- DlibDotNet v19.21.0.20220724
- Emgu.CV v4.9.0.5494
- Emgu.CV.runtime.windows v4.9.0.5494
- Xabe.FFmpeg v5.2.6

### /FaceAppWithGPT2/Program.cs

```
using Emgu.CV;
using ImageProcessingLibrary.Helpers;
using ImageProcessingLibrary.PictureSizeAdaptation;

namespace FaceAppWithGPT2
{
    internal class Program
    {
        static void Main(string[] args)
        {
            if (args.Length < 3)
            {
                Console.WriteLine("Usage: FaceAppWithGPT2 <inputDirectory> <outputDirectory>");
                Console.WriteLine("dimensionType: 'width' or 'height' (only required if");
                return;
            }

            string inputDirectory = args[0];
            string outputDirectory = args[1];
            string resizeOption = args[2];
            string dimensionType = args.Length > 3 ? args[3].ToLower() : string.Empty;

            try
            {
                // Validate directories
                DirectoryHelper.ValidateDirectory(inputDirectory);
                if (!Directory.Exists(outputDirectory))
                {
                    Directory.CreateDirectory(outputDirectory);
                }

                // Validate resize option at the beginning
                if (resizeOption.EndsWith("%"))
            }
        }
    }
}
```

```

    {
        if (!int.TryParse(resizeOption.TrimEnd('%'), out int percentage) ||
        {
            throw new ArgumentException("Invalid percentage value. It must be between 1 and 100.")
        }
    }
    else if (int.TryParse(resizeOption, out int fixedSize))
    {
        if (fixedSize <= 0)
        {
            throw new ArgumentException("Invalid size value. Width or height must be greater than 0.")
        }
        if (string.IsNullOrEmpty(dimensionType) || (dimensionType != "width" & dimensionType != "height"))
        {
            throw new ArgumentException("Dimension type must be specified as width or height.")
        }
    }
    else
    {
        throw new ArgumentException("Invalid resize option. Provide a percentage or fixed size with dimension type.")
    }

    // Get image files from the input directory
    var imageFiles = DirectoryHelper.GetImageFiles(inputDirectory);

    // Instantiate the ImageResizer
    var imageResizer = new ImageResizer();

    // Resize each image and save it to the output directory
    foreach (var imagePath in imageFiles)
    {
        string outputPath = Path.Combine(outputDirectory, Path.GetFileName(imagePath));
        imageResizer.ResizeImage(imagePath, outputPath, resizeOption, dimensionType);
        Console.WriteLine($"Resized image saved to: {outputPath}");
    }

    Console.WriteLine("Image resizing completed successfully.");
}
catch (Exception ex)
{
    Console.WriteLine($"Error: {ex.Message}");
}
}
}

```

```
}
```

## ImageProcessingLibrary

### Dependencies

- DlibDotNet v19.21.0.20220724
- Emgu.CV v4.9.0.5494
- Emgu.CV.runtime.windows v4.9.0.5494
- Xabe.FFmpeg v5.2.6

/ImageProcessingLibrary/Exceptions/ImageProcessingException.cs

```
using System;
```

```
namespace ImageProcessingLibrary.Exceptions
```

```
{
```

```
    public class ImageProcessingException : Exception
```

```
    {
```

```
        public ImageProcessingException(string message) : base(message) { }
```

```
        public ImageProcessingException(string message, Exception innerException) : base(message, innerException) { }
```

```
    }
```

```
}
```

/ImageProcessingLibrary/Helpers/DirectoryHelper.cs

```
using System;
```

```
using System.Collections.Generic;
```

```
using System.IO;
```

```
namespace ImageProcessingLibrary.Helpers
```

```
{
```

```
    public static class DirectoryHelper
```

```
    {
```

```
        /// <summary>
```

```
        /// Validates if the given directory path exists. If it doesn't exist, throws a DirectoryNotFoundException
```

```
        /// </summary>
```

```
        /// <param name="directoryPath">The path of the directory to validate.</param>
```

```
        public static void ValidateDirectory(string directoryPath)
```

```
        {
```

```
            if (directoryPath == null)
```

```
            {
```

```
                throw new ArgumentNullException(nameof(directoryPath), "Directory path cannot be null.");
```

```
            if (string.IsNullOrWhiteSpace(directoryPath))
```

```
            {
```

```
                throw new ArgumentException("Directory path cannot be empty.", nameof(directoryPath));
```

```
            }
```

```

        if (!Directory.Exists(directoryPath))
            throw new DirectoryNotFoundException($"Directory '{directoryPath}' not found");
    }

    /// <summary>
    /// Gets all image files (JPG, PNG) from the specified directory.
    /// </summary>
    /// <param name="directoryPath">The path of the directory to search for image files</param>
    /// <returns>A list of file paths for the images found in the directory.</returns>
    public static List<string> GetImageFiles(string directoryPath)
    {
        ValidateDirectory(directoryPath);

        // Define allowed image extensions
        string[] allowedExtensions = { ".jpg", ".jpeg", ".png" };

        // Get all files with allowed extensions
        var imageFiles = new List<string>();
        foreach (var file in Directory.GetFiles(directoryPath))
        {
            if (Array.Exists(allowedExtensions, ext => ext.Equals(Path.GetExtension(file)))
            {
                imageFiles.Add(file);
            }
        }

        return imageFiles;
    }
}

```

/ImageProcessingLibrary/Interfaces/IImageResizer.cs

```

using Emgu.CV;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ImageProcessingLibrary.Interfaces
{
    public interface IImageResizer
    {

```

```

    /// <summary>
    /// Resizes the image while maintaining the aspect ratio, based on a given fixed size.
    /// </summary>
    /// <param name="inputPath">The path of the input image.</param>
    /// <param name="outputPath">The path where the resized image will be saved.</param>
    /// <param name="resizeOption">The resize option, either a fixed size or percentage.
    /// <param name="dimensionType">Indicates whether the fixed size is for width (true) or height (false).
    void ResizeImage(string inputPath, string outputPath, string resizeOption, string dimensionType)

    /// <summary>
    /// Resizes the image while maintaining the aspect ratio, based on a given fixed size.
    /// </summary>
    /// <param name="image">The input image as a Mat object.</param>
    /// <param name="fixedSize">The fixed size for either width or height.</param>
    /// <param name="isWidth">Indicates whether the fixed size is for width (true) or height (false).
    Mat ResizeImageKeepingAspectRatio(Mat image, int fixedSize, bool isWidth);

    /// <summary>
    /// Resizes the image by a given percentage, maintaining the original aspect ratio.
    /// </summary>
    /// <param name="image">The input image as a Mat object.</param>
    /// <param name="percentage">The percentage by which the image should be resized.
    Mat ResizeImageByPercentage(Mat image, int percentage);
}
}

```

/ImageProcessingLibrary/Logging/Logger.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ImageProcessingLibrary.Logging
{
    public static class Logger
    {
        public static void LogInfo(string message)
        {
            Console.WriteLine($"[INFO] {DateTime.Now}: {message}");
        }

        public static void LogError(string message)
        {
            Console.WriteLine($"[ERROR] {DateTime.Now}: {message}");
        }
    }
}

```

```

    }

    public static void LogWarning(string message)
    {
        Console.WriteLine($"[WARNING] {DateTime.Now}: {message}");
    }
}
}

```

/ImageProcessingLibrary/PictureSizeAdaptation/ImageResizer.cs

```

using Emgu.CV;
using Emgu.CV.CvEnum;
using ImageProcessingLibrary.Interfaces;
using ImageProcessingLibrary.Logging;
using ImageProcessingLibrary.Exceptions;
using System;
using System.IO;

namespace ImageProcessingLibrary.PictureSizeAdaptation
{
    public class ImageResizer : IImageResizer
    {
        public void ResizeImage(string inputPath, string outputPath, string resizeModeOption)
        {
            try
            {
                // Log the start of the resize process
                Logger.LogInfo($"Starting resizing for image: {inputPath}");

                // Validate input paths
                if (!File.Exists(inputPath))
                {
                    throw new FileNotFoundException($"Input file not found: {inputPath}");
                }

                using (var image = CvInvoke.Imread(inputPath))
                {
                    if (image == null || image.IsEmpty)
                    {
                        throw new ImageProcessingException($"Failed to load image: {inputPath}");
                    }

                    if (resizeOption.EndsWith("%"))
                    {

```

```

        int percentage = int.Parse(resizeOption.TrimEnd('%'));
        using (var resizedImage = ResizeImageByPercentage(image, percent
        {
            CvInvoke.Imwrite(outputPath, resizedImage);
        }
    }
    else if (int.TryParse(resizeOption, out int fixedSize))
    {
        using (var resizedImage = dimensionType == "width"
            ? ResizeImageKeepingAspectRatio(image, fixedSize, isWidth: t
            : ResizeImageKeepingAspectRatio(image, fixedSize, isWidth: 1
        {
            CvInvoke.Imwrite(outputPath, resizedImage);
        }
    }
}

// Log the completion of the resize process
Logger.LogInfo($"Successfully resized image: {inputPath} -> {outputPath}
}
catch (FileNotFoundException ex)
{
    Logger.LogError($"File not found: {ex.Message}");
}
catch (ArgumentException ex)
{
    Logger.LogError($"Invalid argument: {ex.Message}");
}
catch (ImageProcessingException ex)
{
    Logger.LogError($"Image processing error: {ex.Message}");
}
catch (Exception ex)
{
    Logger.LogError($"Unexpected error resizing image {inputPath}: {ex.Messa
}
}

public Mat ResizeImageKeepingAspectRatio(Mat image, int fixedSize, bool isWidth)
{
    try
    {
        int newWidth, newHeight;

        if (isWidth)
        {

```

```

        newWidth = fixedSize;
        newHeight = (int)(image.Height * ((double)fixedSize / image.Width));
    }
    else
    {
        newHeight = fixedSize;
        newWidth = (int)(image.Width * ((double)fixedSize / image.Height));
    }

    var resizedImage = new Mat();
    CvInvoke.Resize(image, resizedImage, new System.Drawing.Size(newWidth, newHeight));

    return resizedImage;
}
catch (Exception ex)
{
    throw new ImageProcessingException("Error while resizing the image while " + ex.Message);
}
}

public Mat ResizeImageByPercentage(Mat image, int percentage)
{
    try
    {
        int newWidth = (int)(image.Width * (percentage / 100.0));
        int newHeight = (int)(image.Height * (percentage / 100.0));

        var resizedImage = new Mat();
        CvInvoke.Resize(image, resizedImage, new System.Drawing.Size(newWidth, newHeight));

        return resizedImage;
    }
    catch (Exception ex)
    {
        throw new ImageProcessingException("Error while resizing the image by percentage " + ex.Message);
    }
}
}
}

```

## FaceMorphingLibrary

### Dependencies

- DlibDotNet v19.21.0.20220724
- Emgu.CV v4.9.0.5494



- Emgu.CV.runtime.windows v4.9.0.5494
- Xabe.FFmpeg v5.2.6

/FaceMorphingLibrary/Class1.cs

```
namespace FaceMorphingLibrary
{
    public class Class1
    {

    }
}
```

## VideoGenerationLibrary

### Dependencies

- DlibDotNet v19.21.0.20220724
- Emgu.CV v4.9.0.5494
- Emgu.CV.runtime.windows v4.9.0.5494
- Xabe.FFmpeg v5.2.6

/VideoGenerationLibrary/Class1.cs

```
namespace VideoGenerationLibrary
{
    public class Class1
    {

    }
}
```

## ImageProcessingLibrary.Tests

### Dependencies

- coverlet.collector v6.0.0
- Emgu.CV.runtime.windows v4.9.0.5494
- Microsoft.NET.Test.Sdk v17.8.0
- NUnit v3.14.0
- NUnit.Analyzers v3.9.0
- NUnit3TestAdapter v4.5.0

/ImageProcessingLibrary.Tests/DirectoryHelperTests.cs

```
using System;
using System.Collections.Generic;
using System.IO;
```

```

using System.Linq;
using System.Text;
using System.Threading.Tasks;
using ImageProcessingLibrary.Helpers;
using NUnit.Framework;

namespace ImageProcessingLibrary.Tests
{
    [TestFixture]
    public class DirectoryHelperTests
    {
        [Test]
        public void ValidateDirectory_ShouldThrowArgumentNullException_WhenPathIsNull()
        {
            // Act & Assert
            Assert.Throws<ArgumentNullException>(() => DirectoryHelper.ValidateDirectory(null));
        }

        [Test]
        public void ValidateDirectory_ShouldThrowArgumentException_WhenPathIsEmpty()
        {
            // Act & Assert
            Assert.Throws<ArgumentException>(() => DirectoryHelper.ValidateDirectory(""));
        }

        [Test]
        public void ValidateDirectory_ShouldThrowDirectoryNotFoundException_WhenDirectoryDoesNotExist()
        {
            // Arrange
            string nonExistentDirectory = "C:\\\\NonExistentDirectory";

            // Act & Assert
            Assert.Throws<DirectoryNotFoundException>(() => DirectoryHelper.ValidateDirectory(nonExistentDirectory));
        }

        [Test]
        public void ValidateDirectory_ShouldNotThrowException_WhenDirectoryExists()
        {
            // Arrange
            string existingDirectory = Path.GetTempPath();

            // Act & Assert
            Assert.DoesNotThrow(() => DirectoryHelper.ValidateDirectory(existingDirectory));
        }

        [Test]
    }
}

```

```

public void GetImageFiles_ShouldReturnEmptyList_WhenNoImagesArePresent()
{
    // Arrange
    string tempDirectory = Path.Combine(Path.GetTempPath(), "EmptyDirectory");
    Directory.CreateDirectory(tempDirectory);

    try
    {
        // Act
        List<string> imageFiles = DirectoryHelper.GetImageFiles(tempDirectory);

        // Assert
        Assert.AreEqual(0, imageFiles.Count);
    }
    finally
    {
        // Cleanup
        Directory.Delete(tempDirectory);
    }
}

[Test]
public void GetImageFiles_ShouldReturnImageFiles_WhenImagesArePresent()
{
    // Arrange
    string tempDirectory = Path.Combine(Path.GetTempPath(), "ImageDirectory");
    Directory.CreateDirectory(tempDirectory);

    string imagePath1 = Path.Combine(tempDirectory, "image1.jpg");
    string imagePath2 = Path.Combine(tempDirectory, "image2.png");
    File.Create(imagePath1).Dispose();
    File.Create(imagePath2).Dispose();

    try
    {
        // Act
        List<string> imageFiles = DirectoryHelper.GetImageFiles(tempDirectory);

        // Assert
        Assert.AreEqual(2, imageFiles.Count);
        Assert.Contains(imagePath1, imageFiles);
        Assert.Contains(imagePath2, imageFiles);
    }
    finally
    {
        // Cleanup

```

```

        Directory.Delete(tempDirectory, true);
    }
}
}
}

```

/ImageProcessingLibrary.Tests/ImageResizerTests.cs

```

using System;
using System.Collections.Generic;
using System.Drawing.Imaging;
using System.Drawing;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using ImageProcessingLibrary.PictureSizeAdaptation;
using NUnit.Framework;

namespace ImageProcessingLibrary.Tests
{
    [TestFixture]
    public class ImageResizerTests
    {
        [Test]
        public void ResizeImageKeepingAspectRatio_ShouldResizeBasedOnWidth_WhenWidthIsProvided()
        {
            // Arrange
            var imageResizer = new ImageResizer();
            string tempDirectory = Path.GetTempPath();
            string inputPath = Path.Combine(tempDirectory, "input.jpg");
            string outputPath = Path.Combine(tempDirectory, "output.jpg");

            // Create a valid dummy image file
            using (Bitmap bitmap = new Bitmap(200, 100))
            {
                using (Graphics g = Graphics.FromImage(bitmap))
                {
                    g.Clear(Color.White);
                    g.DrawRectangle(Pens.Black, 10, 10, 180, 80);
                }
                bitmap.Save(inputPath, ImageFormat.Jpeg);
            }

            try
            {

```

```

        // Act
        imageResizer.ResizeImage(inputPath, outputPath, "100", "width");

        // Assert
        Assert.IsTrue(File.Exists(outputPath));
        using (var outputImage = Image.FromFile(outputPath))
        {
            Assert.AreEqual(100, outputImage.Width);
            Assert.AreEqual(50, outputImage.Height); // Aspect ratio maintained
        }
    }
    finally
    {
        // Cleanup
        File.Delete(inputPath);
        File.Delete(outputPath);
    }
}

[Test]
public void ResizeImageKeepingAspectRatio_ShouldResizeBasedOnHeight_WhenHeightIsPro
{
    // Arrange
    var imageResizer = new ImageResizer();
    string tempDirectory = Path.GetTempPath();
    string inputPath = Path.Combine(tempDirectory, "input.jpg");
    string outputPath = Path.Combine(tempDirectory, "output.jpg");

    // Create a valid dummy image file
    using (Bitmap bitmap = new Bitmap(200, 100))
    {
        using (Graphics g = Graphics.FromImage(bitmap))
        {
            g.Clear(Color.White);
            g.DrawRectangle(Pens.Black, 10, 10, 180, 80);
        }
        bitmap.Save(inputPath, ImageFormat.Jpeg);
    }

    try
    {
        // Act
        imageResizer.ResizeImage(inputPath, outputPath, "50", "height");

        // Assert
        Assert.IsTrue(File.Exists(outputPath));
    }
}

```

```

        using (var outputImage = Image.FromFile(outputPath))
        {
            Assert.AreEqual(100, outputImage.Width); // Aspect ratio maintained
            Assert.AreEqual(50, outputImage.Height);
        }
    }
    finally
    {
        // Cleanup
        File.Delete(inputPath);
        File.Delete(outputPath);
    }
}

[Test]
public void ResizeImageByPercentage_ShouldResizeImageCorrectly_WhenPercentageIsProvided
{
    // Arrange
    var imageResizer = new ImageResizer();
    string tempDirectory = Path.GetTempPath();
    string inputPath = Path.Combine(tempDirectory, "input.jpg");
    string outputPath = Path.Combine(tempDirectory, "output.jpg");

    // Create a valid dummy image file
    using (Bitmap bitmap = new Bitmap(200, 100))
    {
        using (Graphics g = Graphics.FromImage(bitmap))
        {
            g.Clear(Color.White);
            g.DrawRectangle(Pens.Black, 10, 10, 180, 80);
        }
        bitmap.Save(inputPath, ImageFormat.Jpeg);
    }

    try
    {
        // Act
        imageResizer.ResizeImage(inputPath, outputPath, "50%", "");

        // Assert
        Assert.IsTrue(File.Exists(outputPath));
        using (var outputImage = Image.FromFile(outputPath))
        {
            Assert.AreEqual(100, outputImage.Width); // 50% of original width
            Assert.AreEqual(50, outputImage.Height); // 50% of original height
        }
    }
}

```

```
    }  
    finally  
    {  
        // Cleanup  
        File.Delete(inputPath);  
        File.Delete(outputPath);  
    }  
}  
}
```

## Sonstige Dateien

### Dependencies

- No dependencies found