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 prov
                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);
                }

                // 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(image
                    imageResizer.ResizeImage(imagePath, outputPath, resizeOption, dimension
                    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/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 Dire
        /// </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 canno
            if (string.IsNullOrWhiteSpace(directoryPath))
                throw new ArgumentException("Directory path cannot be empty.", nameof(direct

            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
        /// <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 fixe
            /// </summary>
            /// <param name="inputPath">The path of the input image.</param>
            /// <param name="outputPath">The path where the resized image will be saved.</pa
            /// <param name="resizeOption">The resize option, either a fixed size or percen
            /// <param name="dimensionType">Indicates whether the fixed size is for width (
            void ResizeImage(string inputPath, string outputPath, string resizeOption, strin

            /// <summary>
            /// Resizes the image while maintaining the aspect ratio, based on a given fixe
            /// </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)
            Mat ResizeImageKeepingAspectRatio(Mat image, int fixedSize, bool isWidth);

            /// <summary>
            /// Resizes the image by a given percentage, maintaining the original aspect ra
            /// </summary>
            /// <param name="image">The input image as a Mat object.</param>
            /// <param name="percentage">The percentage by which the image should be resize
            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 System;
using System.IO;

namespace ImageProcessingLibrary.PictureSizeAdaptation
{
    public class ImageResizer : IImageResizer
    {
        public void ResizeImage(string inputPath, string outputPath, string resizeOption, st
        {
            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 (resizeOption.EndsWith("%"))
                    {
                        // Resize by percentage
```

```
                int percentage = int.Parse(resizeOption.TrimEnd('%'));
                using (var resizedImage = ResizeImageByPercentage(image, percentage)
                {
                    CvInvoke.Imwrite(outputPath, resizedImage);
                }
            }
            else if (int.TryParse(resizeOption, out int fixedSize))
            {
                if (string.IsNullOrEmpty(dimensionType))
                {
                    throw new ArgumentException("Dimension type must be specified wh
                }
                using (var resizedImage = dimensionType == "width"
                    ? ResizeImageKeepingAspectRatio(image, fixedSize, isWidth: true)
                    : ResizeImageKeepingAspectRatio(image, fixedSize, isWidth: false
                {
                    CvInvoke.Imwrite(outputPath, resizedImage);
                }
            }
            else
            {
                throw new ArgumentException("Invalid resize option. Provide a percen
            }
        }

        // Log the completion of the resize process
        Logger.LogInfo($"Successfully resized image: {inputPath} -> {outputPath}");
    }
    catch (Exception ex)
    {
        // Log any errors that occur
        Logger.LogError($"Error resizing image {inputPath}: {ex.Message}");
        throw;
    }
}

public Mat ResizeImageKeepingAspectRatio(Mat image, int fixedSize, bool isWidth)
{
    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;
        }

        public Mat ResizeImageByPercentage(Mat image, int percentage)
        {
            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;
        }
    }
}
```

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(nul
}

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

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

    // Act & Assert
    Assert.Throws<DirectoryNotFoundException>(() => DirectoryHelper.ValidateDirecto
}

[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_WhenWidthIsProvid
        {
            // 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_WhenHeightIsProv
{
    // 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_WhenPercentageIsProvi
        {
            // 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