Transformer Application

Name: Ilyes

Surname: Saidi

Course: **Computer vision, pattern recognition & image retrieval**

Application Introduction:

The **Transformer** MATLAB app is designed to serve as a versatile image processing tool, allowing users to interactively apply a variety of transformations to their images. The application assumes that users may want to perform common image processing operations without the need for extensive knowledge of image processing algorithms or coding.

Purpose:

Image Transformation and Enhancement:

- The primary purpose of the app is to provide users with an intuitive interface to transform and enhance images. It includes a range of operations, such as grayscale conversion, rotation, contrast adjustment, Gaussian blur, edge detection, and more.

User-Friendly Interface:

- The app is designed with a user-friendly interface, featuring buttons for specific image processing tasks. This design aims to make image manipulation accessible to users who may not have expertise in image processing or MATLAB programming.

Interactive Image Editing:

- Users can import an image, visualize it in the original state, and apply various transformations with a simple button click. The interactive nature of the app allows users to experiment with different operations and observe the effects in real-time.

Assumptions:

Basic Image Processing Needs:

- The app assumes that users have common image processing needs and aims to fulfill those needs without overwhelming them with advanced options. It focuses on fundamental operations that are frequently used in image editing.

Limited Programming Knowledge:

- The target users may not have extensive programming knowledge, and the app provides a graphical interface to perform image processing tasks. Users can achieve desired effects without writing code or understanding complex algorithms.

Single Image Processing Workflow:

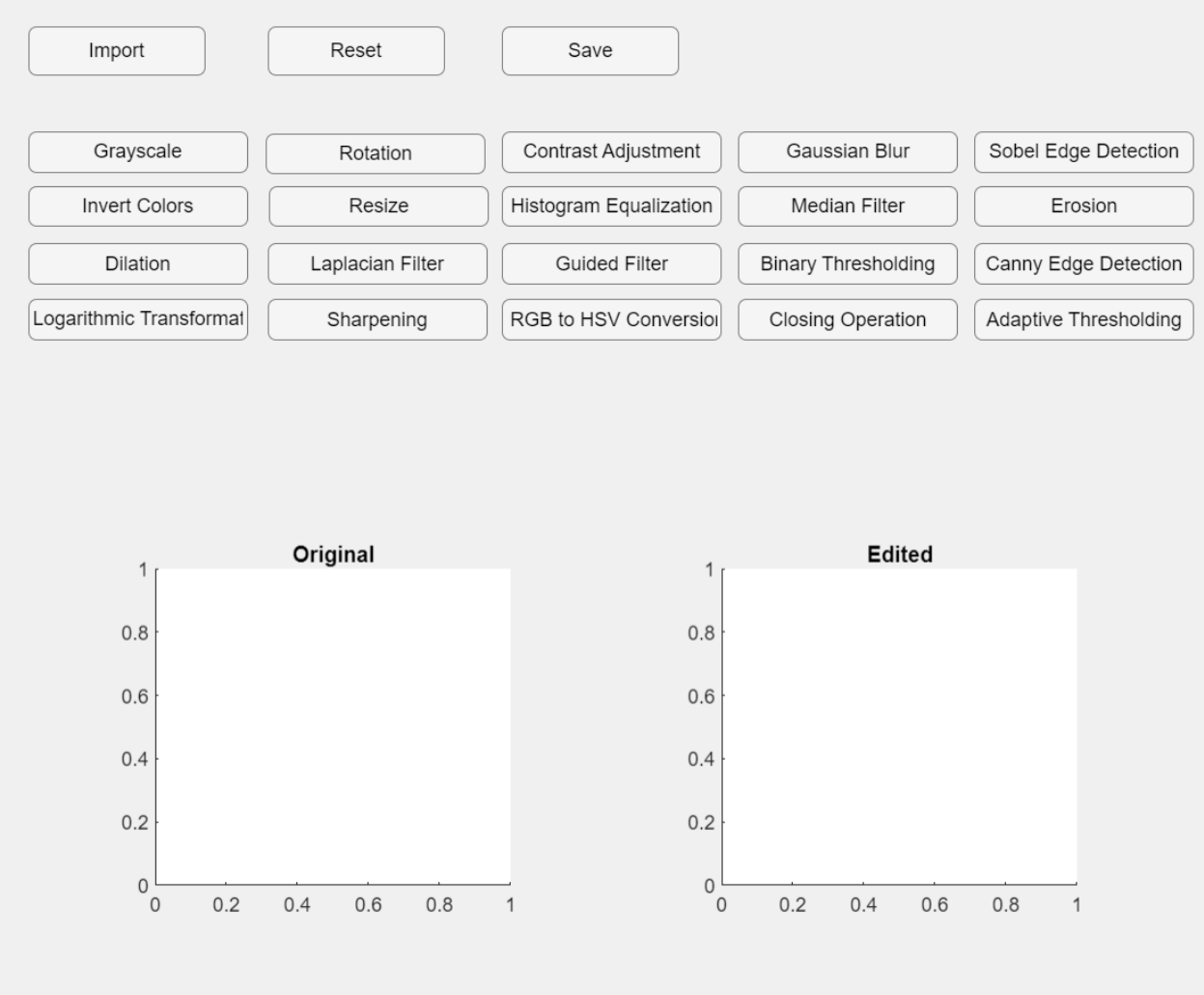
- The app assumes a linear workflow where users import an image, apply transformations sequentially, and save the edited image. It caters to users who prefer a straightforward approach to image editing.

Overall Theme:

The leading theme of the **Transformer** app revolves around simplicity, accessibility, and practicality in image processing. By offering a curated set of operations through a user-friendly interface, the app aims to empower users to enhance and manipulate their images effortlessly. It assumes a user base with diverse backgrounds, including individuals interested in quick image edits, hobbyists, and those who need a straightforward tool for common image transformations.

Application overview:

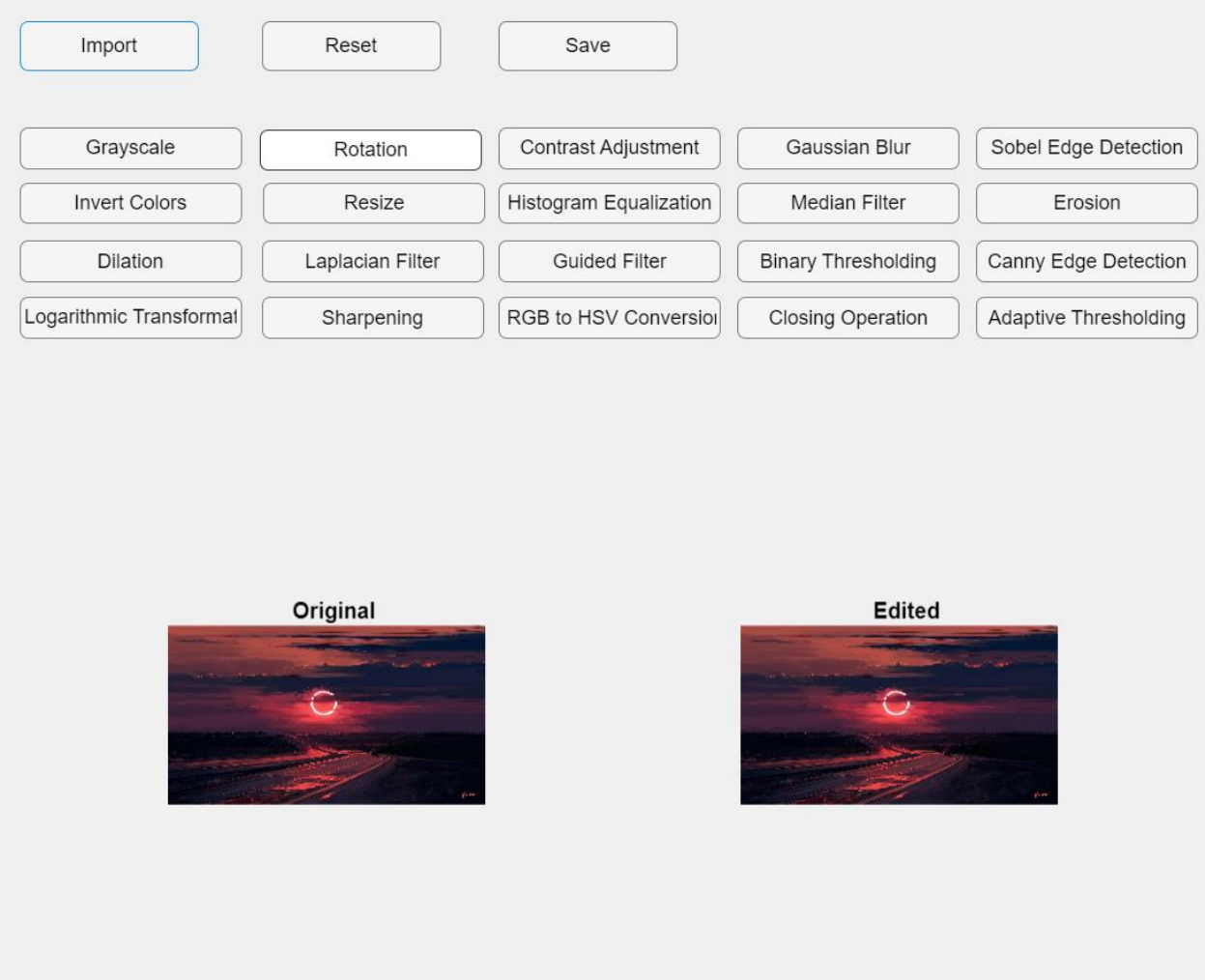
The program has a total of 23 Buttons and 2 Axes. 20 different transformation functions. A button to Import images of type jpg, jpeg, png and bnp. A reset button to reset our processed image to default. And finally, a save button to save our results.



Import Function:

```
[file1,path1]=uigetfile('*.jpg;*.png;*.bmp;*.jpeg','Load the image');
            Image=imread([path1,file1]);
            imshow(Image,'Parent',app.ImageViewer);
            imshow(Image,'Parent',app.ImageData);
```

- **Purpose:** Allows users to import an image from their file system.
- **Functionality:** Opens a file dialog for users to select an image file (supports formats like .jpg, .png, .bmp, .jpeg).



Reset button:

```
imshow(app.ImageViewer.Children.CData,'Parent',app.ImageData);
```

Purpose: Resets the edited image to the original imported image.

Functionality: Displays the original image in the "Edited" view.

Save button:

```
            edited_image = app.ImageData.Children.CData;
            [file2, path2] = uiputfile({'*.png', 'PNG Files'; '*.jpg', 'JPEG Files'},
'Save Edited Image As');
            if isequal(file2, 0) || isequal(path2, 0)
                return;
            end
            full_path = fullfile(path2, file2);
            imwrite(edited_image, full_path);
            msgbox('Edited image saved successfully!', 'Success', 'modal');
```

Purpose: Saves the edited image to a specified file.

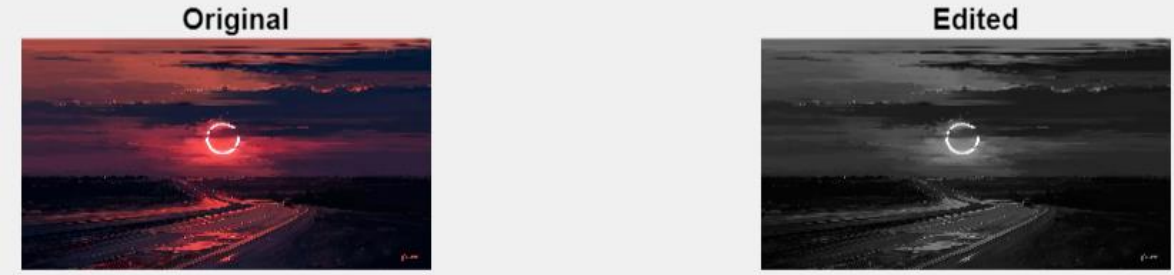
Functionality: Opens a file dialog for users to specify the file format and location for saving the edited image.

Grayscale button:

```
            iimage = app.ImageData.Children.CData;
            if ndims(iimage) == 3
                iimage = rgb2gray(iimage);
            end
            imshow(iimage, 'Parent', app.ImageData);
```

Purpose: Converts the image to grayscale.

Functionality: Applies the `rgb2gray` transformation to the image.

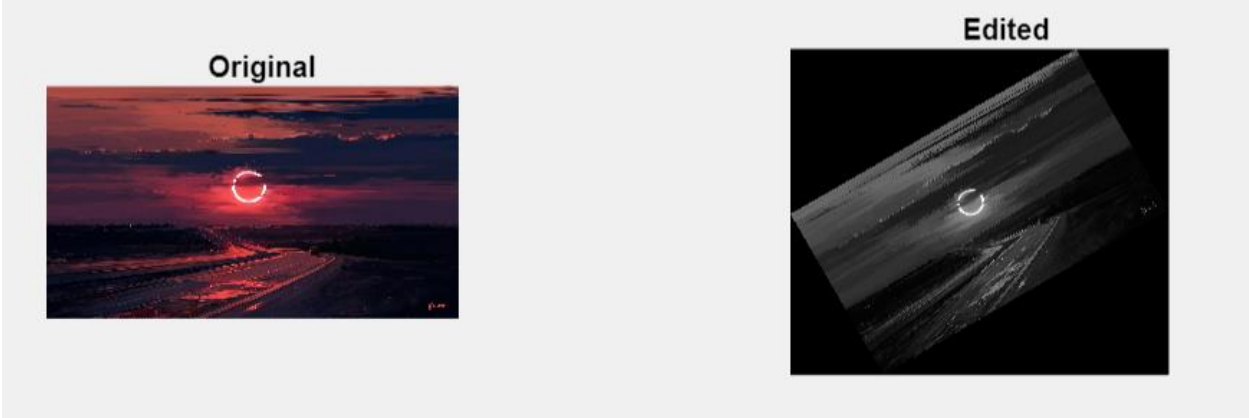


Rotation button:

```
angle = 30; % specify the rotation angle
            timage = imrotate(app.ImageData.Children.CData, angle);
            imshow(timage, 'Parent', app.ImageData);
```

Purpose: Rotates the image by a specified angle.

Functionality: Applies image rotation using the `imrotate` function with a predefined angle (e.g., 30 degrees).

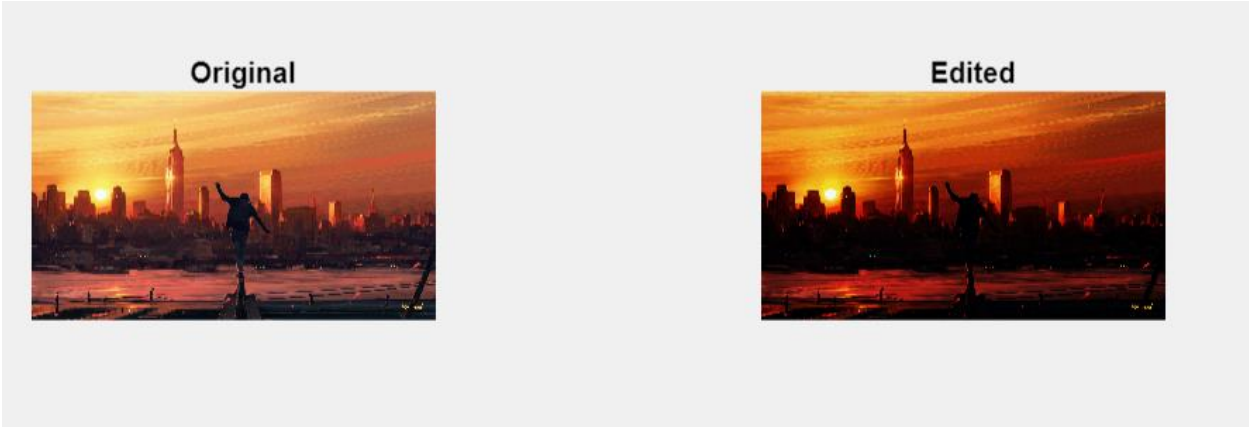


Contrast Adjustment button:

```
contrast_factor = 1.5; % adjust this value as needed
            timage = imadjust(app.ImageData.Children.CData, [], [], contrast_factor);
            imshow(timage, 'Parent', app.ImageData);
```

Purpose: Adjusts the contrast of the image.

Functionality: Applies contrast adjustment using the `imadjust` function with a predefined contrast factor (e.g., 1.5).

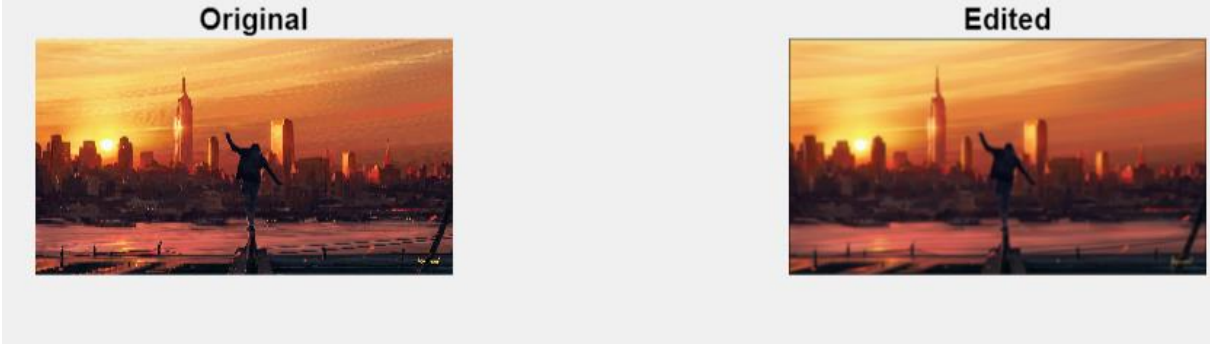


Gaussian Blur button:

```
h = fspecial('gaussian', [5 5], 2);
            timage = imfilter(app.ImageData ...
                .Children.CData, h);
            imshow(timage, 'Parent', app.ImageData
```

Purpose: Applies a Gaussian blur to the image.

Functionality: Uses a Gaussian filter for blurring via the `imfilter` function.



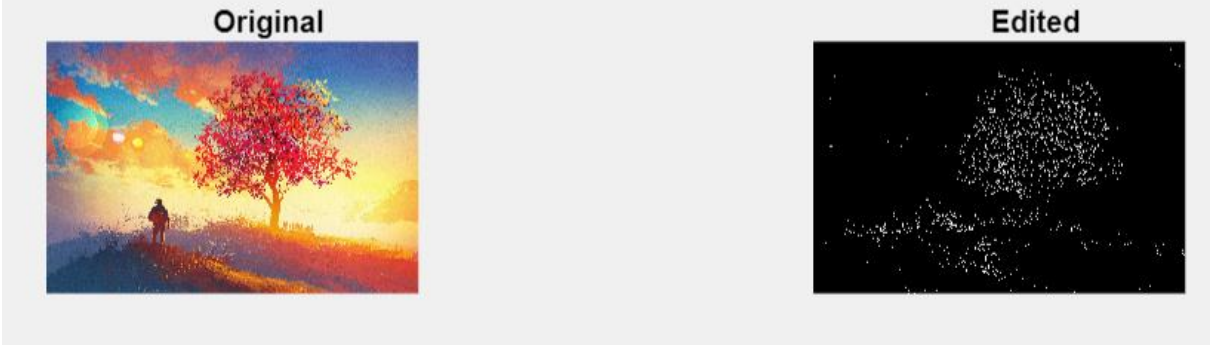
Sobel Edge Detection button:

```
iimage = app.ImageData.Children.CData;
```

```
% Check if the input image is RGB and convert to grayscale if needed
if ismatrix(iimage)
    % Already grayscale
    input_image_gray = iimage;
elseif ndims(iimage) == 3 && size(iimage, 3) == 3
    % RGB image, convert to grayscale
    input_image_gray = rgb2gray(iimage);
else
    % Unsupported format, return
    disp('Unsupported image format');
    return;
end
timage = edge(input_image_gray ...
    , 'sobel');
imshow(timage, 'Parent', app.ImageData);
```

Purpose: Performs edge detection using the Sobel operator.

Functionality: Applies edge detection using the `edge` function with the 'sobel' method.

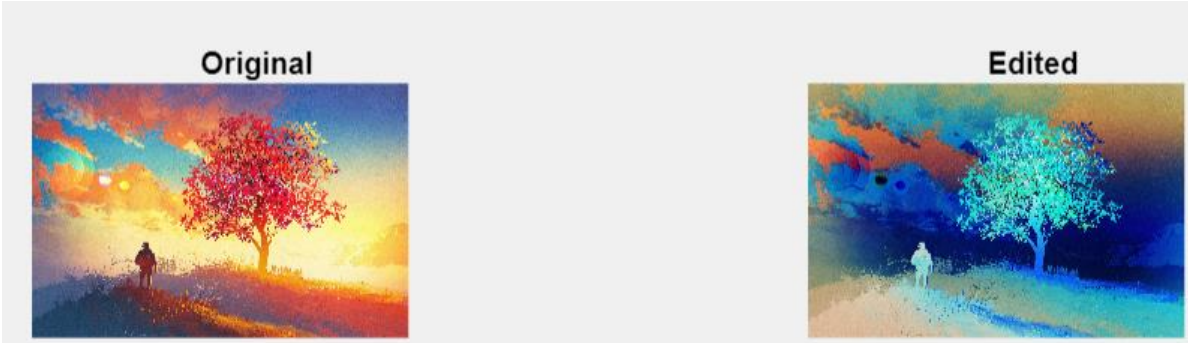


Invert Colors button:

```
timage = imcomplement(app.ImageData.Children.CData);
            imshow(timage, 'Parent', app.ImageData);
```

Purpose: Inverts the colors of the image.

Functionality: Applies color inversion using the `imcomplement` function.

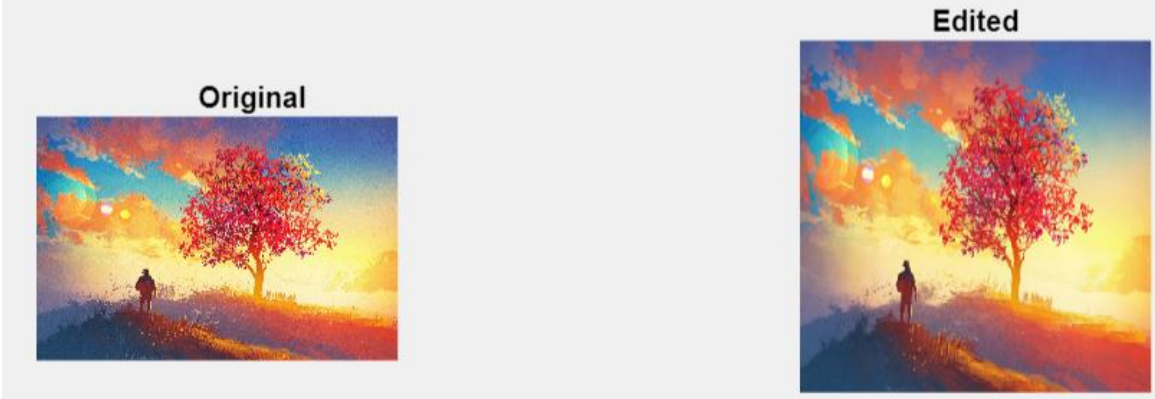


Resize button:

```
new_size = [300, 300]; % specify the new size
            timage = imresize(app.ImageData.Children.CData, new_size);
            imshow(timage, 'Parent', app.ImageData);
```

Purpose: Resizes the image to a specified new size.

Functionality: Uses the `imresize` function to resize the image to a predefined size (e.g., 300x300 pixels).

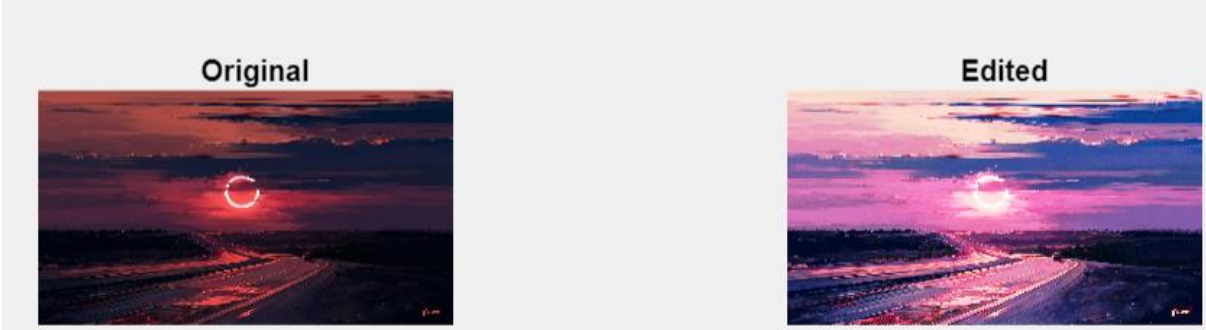


Histogram Equalization button:

```
iimage = im2double(app.ImageData.Children.CData);
            timage = histeq(iimage);
            imshow(timage, 'Parent', app.ImageData);
```

Purpose: Enhances the image contrast using histogram equalization.

Functionality: Applies histogram equalization using the `histeq` function.

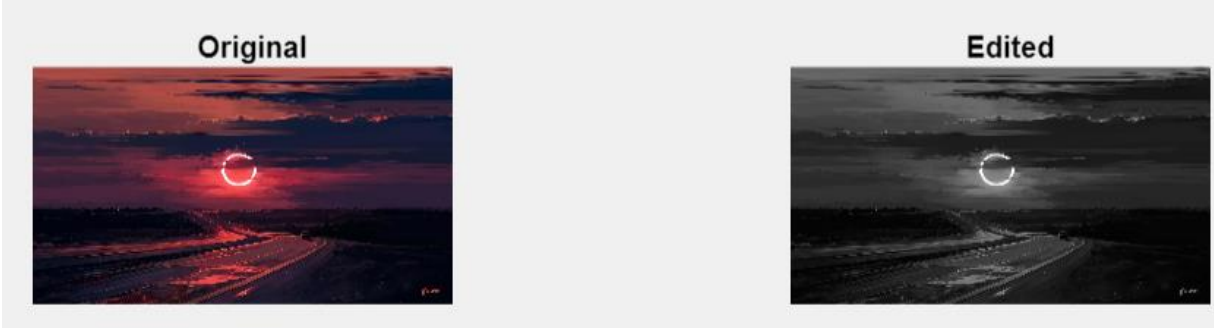


Median Filter button:

```
iimage = app.ImageData.Children.CData;
            if ndims(iimage) == 3
                iimage = rgb2gray(iimage);
            end
            timage = medfilt2(iimage, [3, 3]);
            imshow(timage, 'Parent', app.ImageData);
```

Purpose: Applies a median filter to the image.

Functionality: Uses a median filter with a 3x3 neighborhood via the `medfilt2` function.

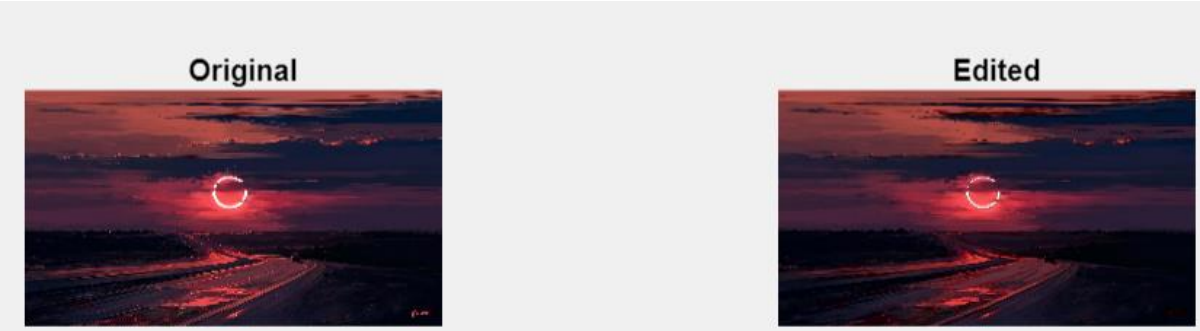


Erosion button:

```
se = strel('disk', 5);
            timage = imerode(app.ImageData ...
                .Children.CData, se);
            imshow(timage, 'Parent', app.ImageData);
```

Purpose: Performs image erosion using a disk-shaped structuring element.

Functionality: Applies erosion using the `imerode` function with a predefined structuring element.

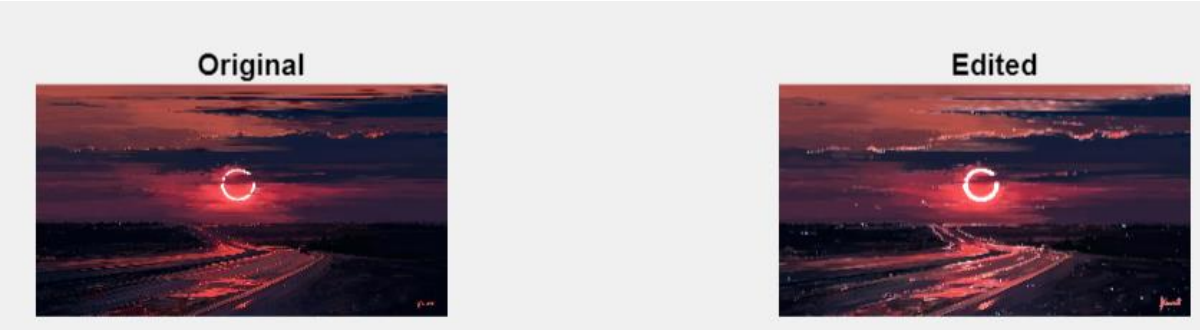


Dilation button:

```
se = strel('disk', 5);
            timage = imdilate(app.ImageData.Children.CData, se);
            imshow(timage, 'Parent', app.ImageData);
```

Purpose: Performs image dilation using a disk-shaped structuring element.

Functionality: Applies dilation using the `imdilate` function with a predefined structuring element.

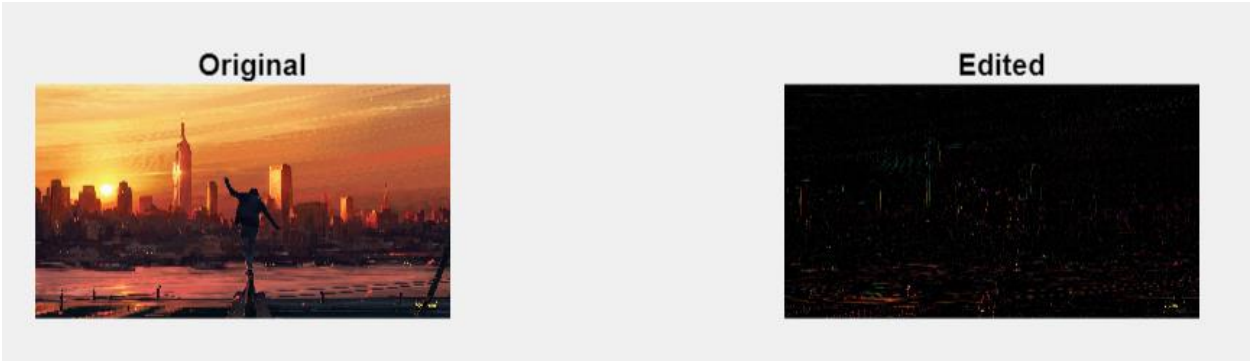


Laplacian Filter button:

```
timage = imfilter(app.ImageData.Children.CData, fspecial('laplacian'));
            imshow(timage, 'Parent', app.ImageData);
```

Purpose: Applies a Laplacian filter to enhance edges.

Functionality: Uses the Laplacian filter via the `imfilter` function with `fspecial('laplacian')`.

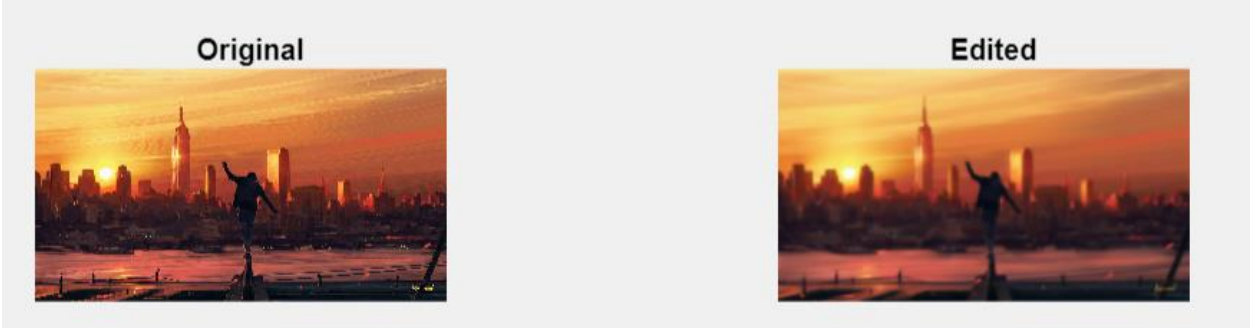


Guided Filter button:

```
input_image = app.ImageData.Children.CData;

            % Apply guided filter
            timage = applyGuidedFilter(app, input_image);

            % Display the result
            imshow(timage, 'Parent', app.ImageData);
```

Purpose: Applies a guided filter transformation.

Functionality: Uses the `imguidedfilter` function for guided filtering.

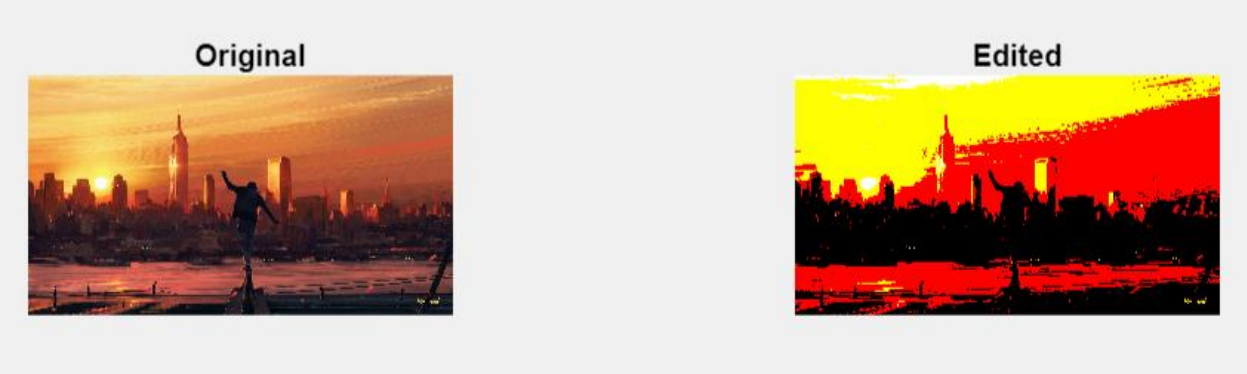


Binary Thresholding button:

```
input_image = app.ImageData.Children.CData;
            % Apply binary thresholding
            threshold_value = 0.5; % adjust as needed
            timage = imbinarize(input_image, threshold_value);

            % Display the result using imagesc instead of imshow
            imagesc(timage, 'Parent', app.ImageData);
```

Purpose: Applies binary thresholding to the image.

Functionality: Uses the `imbinarize` function with a predefined threshold value (e.g., 0.5).

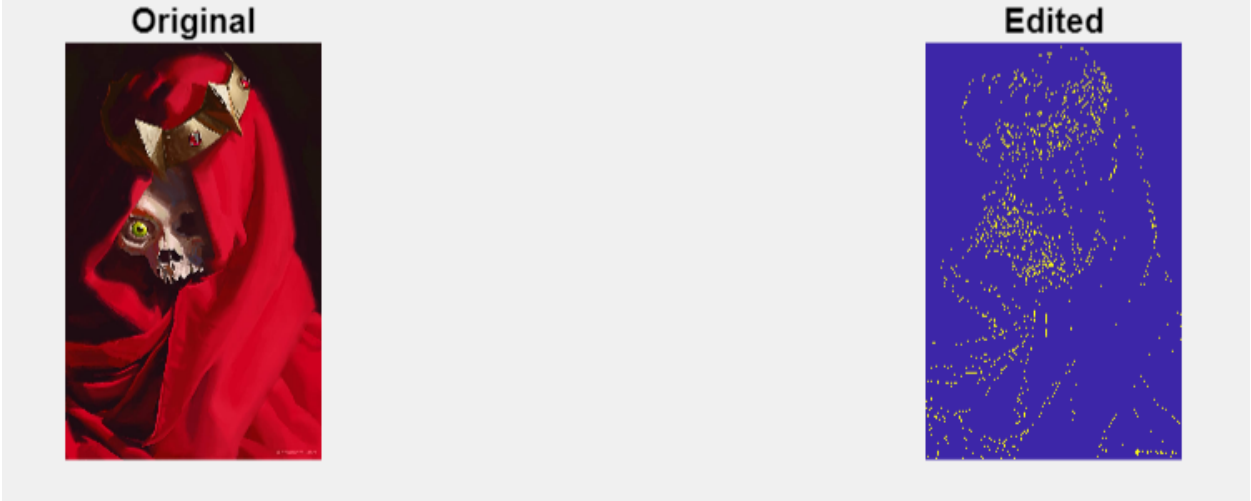


Canny Edge Detection button:

```
input_image = app.ImageData.Children.CData;

            % Convert the input image to grayscale if it's not already
            if size(input_image, 3) == 3
                input_image = rgb2gray(input_image);
            end

            % Apply Canny edge detection
            timage = edge(input_image, 'canny');

            % Display the result using imagesc instead of imshow
            imagesc(timage, 'Parent', app.ImageData);
```

Purpose: Applies edge detection using the Canny method.

Functionality: Applies Canny edge detection using the `edge` function with the 'canny' method.

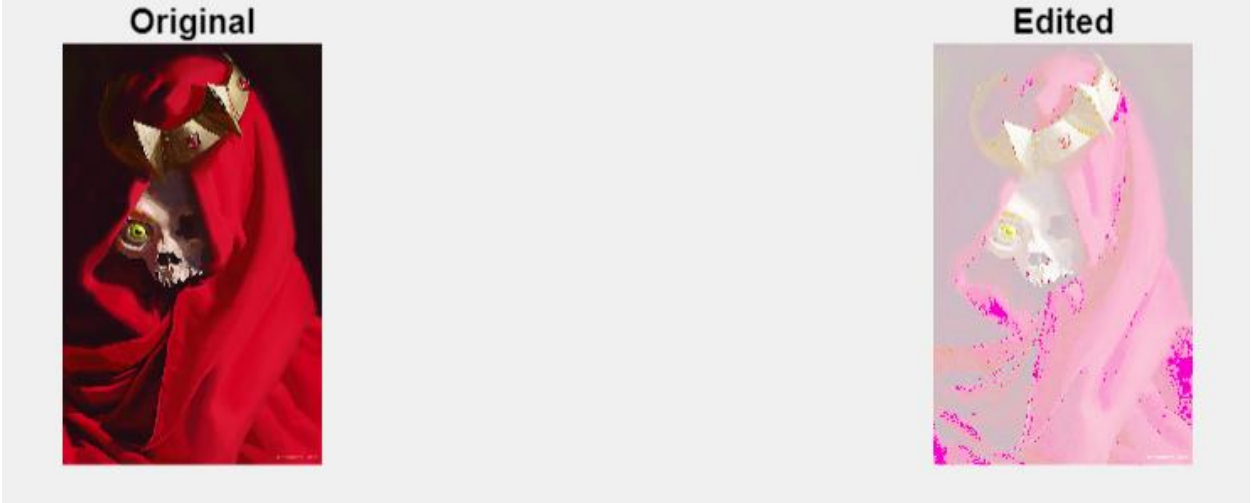


Logarithmic Transformation button:

```
timage = imadjust(app.ImageData.Children.CData, [], [], 0.1);
            imshow(timage, 'Parent', app.ImageData);
```

Purpose: Applies a logarithmic transformation to the image.

Functionality: Uses the `imadjust` function with a predefined gamma value (e.g., 0.1).

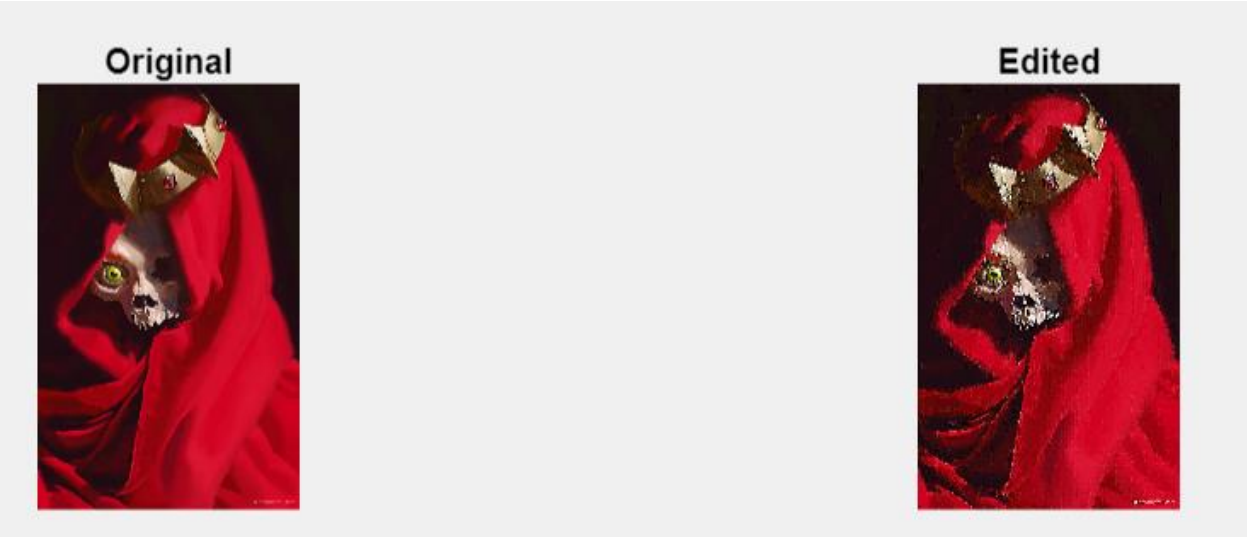


Sharpening button:

```
timage = imsharpen(app.ImageData.Children.CData);
            imshow(timage, 'Parent', app.ImageData);
```

Purpose: Sharpens the image.

Functionality: Applies image sharpening using the `imsharpen` function.

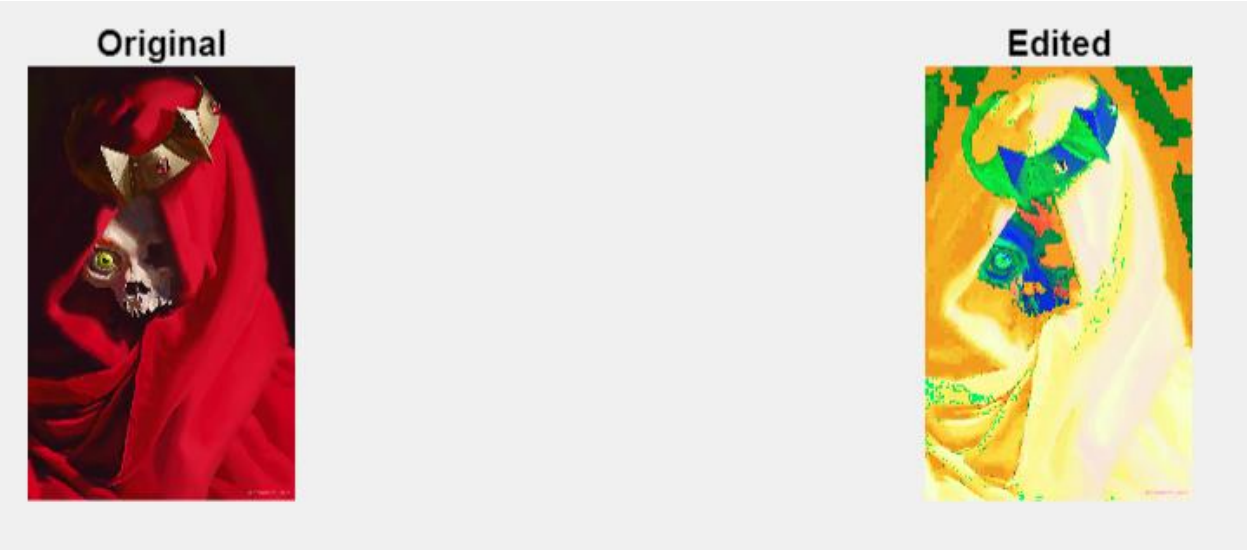


RGB to HSV Conversion button:

```
timage = rgb2hsv(app.ImageData.Children.CData);
            imshow(timage, 'Parent', app.ImageData);
```

Purpose: Converts the image from RGB to HSV color space.

Functionality: Uses the `rgb2hsv` function for color space conversion.

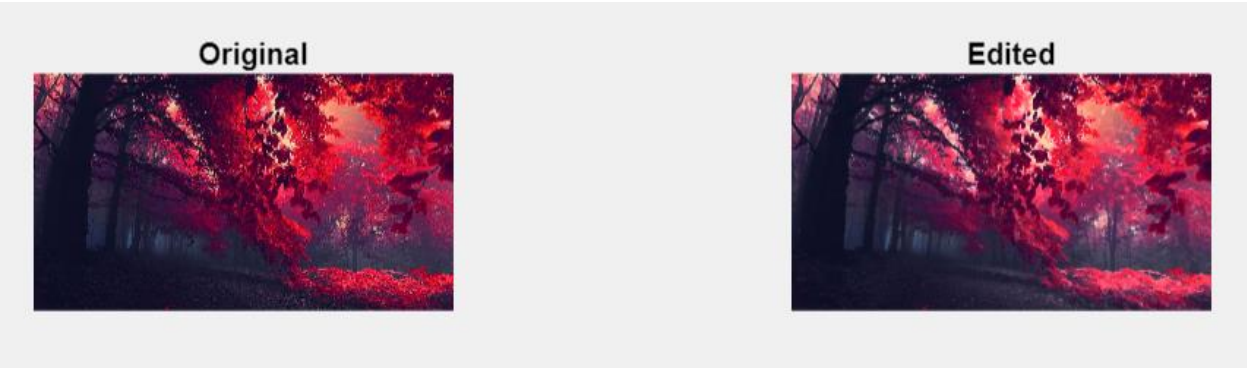


Closing Operation button:

```
se = strel('disk', 5);
            timage = imclose(app.ImageData.Children.CData, se);
            imshow(timage, 'Parent', app.ImageData);
```

Purpose: Applies a closing operation to the image.

Functionality: Uses morphological closing via the `imclose` function with a disk-shaped structuring element.

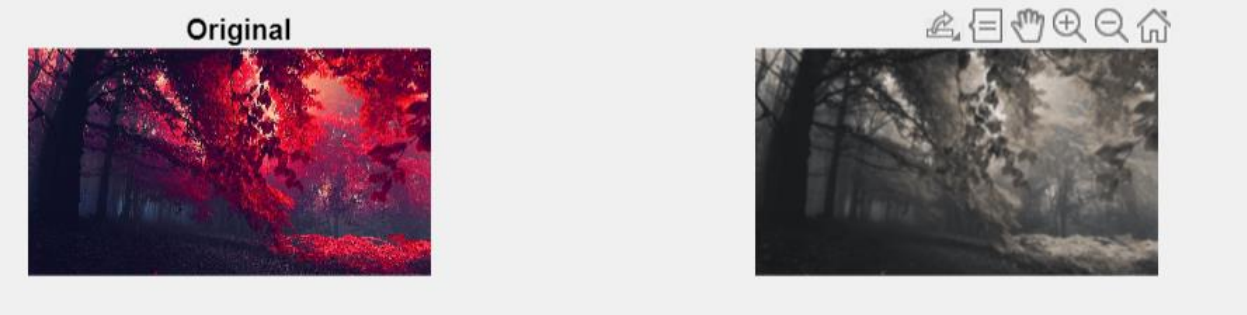


Adaptive Thresholding button:

```
timage = adaptthresh(app.ImageData.Children.CData, 'NeighborhoodSize', 15);
            imshow(timage, 'Parent', app.ImageData);
```

Purpose: Applies adaptive thresholding to the image.

Functionality: Uses adaptive thresholding with a specified neighborhood size via the `adaptthresh` function.



Code:

```
classdef Transformer < matlab.apps.AppBase

    % Properties that correspond to app components
    properties (Access = public)
        UIFigure                    matlab.ui.Figure
        SaveButton                  matlab.ui.control.Button
        AdaptiveThresholdingButton  matlab.ui.control.Button
        ClosingOperationButton      matlab.ui.control.Button
        RGBtoHSVConversionButton    matlab.ui.control.Button
        SharpeningButton            matlab.ui.control.Button
        LogarithmicTransformationButton  matlab.ui.control.Button
        CannyEdgeDetectionButton    matlab.ui.control.Button
        BinaryThresholdingButton    matlab.ui.control.Button
        GuidedFilterButton          matlab.ui.control.Button
        LaplacianFilterButton       matlab.ui.control.Button
        DilationButton              matlab.ui.control.Button
        ErosionButton               matlab.ui.control.Button
        MedianFilterButton          matlab.ui.control.Button
        HistogramEqualizationButton  matlab.ui.control.Button
        ResizeButton                matlab.ui.control.Button
        InvertColorsButton          matlab.ui.control.Button
        SobelEdgeDetectionButton    matlab.ui.control.Button
        GaussianBlurButton          matlab.ui.control.Button
        ContrastAdjustmentButton    matlab.ui.control.Button
        RotationButton              matlab.ui.control.Button
        GrayscaleButton             matlab.ui.control.Button
        ResetButton                 matlab.ui.control.Button
        ImportButton                matlab.ui.control.Button
        ImageData                   matlab.ui.control.UIAxes
        ImageViewer                 matlab.ui.control.UIAxes
    end


    methods (Access = public)
        % Transformation functions


function output_image = applyGuidedFilter(~, input_image)
    % Guided filter transformation

    % Ensure the input image is in double format
    input_image = im2double(input_image);

    % Apply guided filter
    guided_image = imguidedfilter(input_image);
    output_image = im2uint8(guided_image); % Convert back to uint8 if needed
end

    end


    % Callbacks that handle component events
    methods (Access = private)
```

```
% Button pushed function: ImportButton
function ImportButtonPushed(app, event)

    [file1,path1]=uigetfile('*.jpg;*.png;*.bmp;*.jpeg','Load the image');
    Image=imread([path1,file1]);
    imshow(Image,'Parent',app.ImageViewer);
    imshow(Image,'Parent',app.ImageData);
end

% Callback function
function TranformationDropDownValueChanged(app, event)

end

% Button pushed function: GrayscaleButton
function GrayscaleButtonPushed(app, event)
    iimage = app.ImageData.Children.CData;
    if ndims(iimage) == 3
        iimage = rgb2gray(iimage);
    end
    imshow(iimage, 'Parent', app.ImageData);
end

% Button pushed function: GaussianBlurButton
function GaussianBlurButtonPushed(app, event)
    h = fspecial('gaussian', [5 5], 2);
    timage = imfilter(app.ImageData ...
        .Children.CData, h);
    imshow(timage, 'Parent', app.ImageData);
end

% Button pushed function: ResetButton
function ResetButtonPushed(app, event)
    imshow(app.ImageViewer.Children.CData,'Parent',app.ImageData);
end

% Button pushed function: ErosionButton
function ErosionButtonPushed(app, event)
    se = strel('disk', 5);
    timage = imerode(app.ImageData ...
        .Children.CData, se);
    imshow(timage, 'Parent', app.ImageData);
end

% Button pushed function: ContrastAdjustmentButton
function ContrastAdjustmentButtonPushed(app, event)
    contrast_factor = 1.5; % adjust this value as needed
    timage = imadjust(app.ImageData.Children.CData, [], [], contrast_factor);
    imshow(timage, 'Parent', app.ImageData);
end

% Button pushed function: SobelEdgeDetectionButton
function SobelEdgeDetectionButtonPushed(app, event)
    iimage = app.ImageData.Children.CData;
```

```
    % Check if the input image is RGB and convert to grayscale if needed
    if ismatrix(iimage)
        % Already grayscale
        input_image_gray = iimage;
    elseif ndims(iimage) == 3 && size(iimage, 3) == 3
        % RGB image, convert to grayscale
        input_image_gray = rgb2gray(iimage);
    else
        % Unsupported format, return
        disp('Unsupported image format');
        return;
    end
    timage = edge(input_image_gray ...
        , 'sobel');
    imshow(timage, 'Parent', app.ImageData);
end

% Button pushed function: InvertColorsButton
function InvertColorsButtonPushed(app, event)
    timage = imcomplement(app.ImageData.Children.CData);
    imshow(timage, 'Parent', app.ImageData);
end

% Button pushed function: HistogramEqualizationButton
function HistogramEqualizationButtonPushed(app, event)
    iimage = im2double(app.ImageData.Children.CData);
    timage = histeq(iimage);
    imshow(timage, 'Parent', app.ImageData);
end

% Button pushed function: MedianFilterButton
function MedianFilterButtonPushed(app, event)
    iimage = app.ImageData.Children.CData;
    if ndims(iimage) == 3
        iimage = rgb2gray(iimage);
    end
    timage = medfilt2(iimage, [3, 3]);
    imshow(timage, 'Parent', app.ImageData);
end

% Button pushed function: DilationButton
function DilationButtonPushed(app, event)
    se = strel('disk', 5);
    timage = imdilate(app.ImageData.Children.CData, se);
    imshow(timage, 'Parent', app.ImageData);
end

% Button pushed function: LaplacianFilterButton
function LaplacianFilterButtonPushed(app, event)
    timage = imfilter(app.ImageData.Children.CData, fspecial('laplacian'));
    imshow(timage, 'Parent', app.ImageData);
end

% Button pushed function: GuidedFilterButton
```

```
function GuidedFilterButtonPushed(app, event)
    input_image = app.ImageData.Children.CData;

    % Apply guided filter
    timage = applyGuidedFilter(app, input_image);

    % Display the result
    imshow(timage, 'Parent', app.ImageData);
end

% Button pushed function: BinaryThresholdingButton
function BinaryThresholdingButtonPushed(app, event)
    input_image = app.ImageData.Children.CData;
    % Apply binary thresholding
    threshold_value = 0.5; % adjust as needed
    timage = imbinarize(input_image, threshold_value);

    % Display the result using imagesc instead of imshow
    imagesc(timage, 'Parent', app.ImageData);
end

% Button pushed function: CannyEdgeDetectionButton
function CannyEdgeDetectionButtonPushed(app, event)
    input_image = app.ImageData.Children.CData;

    % Convert the input image to grayscale if it's not already
    if size(input_image, 3) == 3
        input_image = rgb2gray(input_image);
    end

    % Apply Canny edge detection
    timage = edge(input_image, 'canny');

    % Display the result using imagesc instead of imshow
    imagesc(timage, 'Parent', app.ImageData);
end

% Button pushed function: LogarithmicTransformationButton
function LogarithmicTransformationButtonPushed(app, event)
    timage = imadjust(app.ImageData.Children.CData, [], [], 0.1);
    imshow(timage, 'Parent', app.ImageData);
end

% Button pushed function: SharpeningButton
function SharpeningButtonPushed(app, event)
    timage = imsharpen(app.ImageData.Children.CData);
    imshow(timage, 'Parent', app.ImageData);
end

% Button pushed function: RGBtoHSVConversionButton
function RGBtoHSVConversionButtonPushed(app, event)
    timage = rgb2hsv(app.ImageData.Children.CData);
    imshow(timage, 'Parent', app.ImageData);
end
```

```
        % Button pushed function: ClosingOperationButton
        function ClosingOperationButtonPushed(app, event)
            se = strel('disk', 5);
            timage = imclose(app.ImageData.Children.CData, se);
            imshow(timage, 'Parent', app.ImageData);
        end

        % Button pushed function: AdaptiveThresholdingButton
        function AdaptiveThresholdingButtonPushed(app, event)
            timage = adaptthresh(app.ImageData.Children.CData, 'NeighborhoodSize',
15);
            imshow(timage, 'Parent', app.ImageData);
        end

        % Button pushed function: RotationButton
        function RotationButtonPushed(app, event)
            angle = 30; % specify the rotation angle
            timage = imrotate(app.ImageData.Children.CData, angle);
            imshow(timage, 'Parent', app.ImageData);
        end

        % Button pushed function: ResizeButton
        function ResizeButtonPushed(app, event)
            new_size = [300, 300]; % specify the new size
            timage = imresize(app.ImageData.Children.CData, new_size);
            imshow(timage, 'Parent', app.ImageData);
        end

        % Button pushed function: SaveButton
        function SaveButtonPushed(app, event)
            edited_image = app.ImageData.Children.CData;
            [file2, path2] = uiputfile({'*.png', 'PNG Files'; '*.jpg', 'JPEG Files'},
'Save Edited Image As');
            if isequal(file2, 0) || isequal(path2, 0)
                return;
            end
            full_path = fullfile(path2, file2);
            imwrite(edited_image, full_path);
            msgbox('Edited image saved successfully!', 'Success', 'modal');
        end
    end

    % Component initialization
    methods (Access = private)

        % Create UIFigure and components
        function createComponents(app)

            % Create UIFigure and hide until all components are created
            app.UIFigure = uifigure('Visible', 'off');
            app.UIFigure.Position = [100 100 744 623];
            app.UIFigure.Name = 'MATLAB App';
            app.UIFigure.WindowStyle = 'modal';

            % Create ImageViewer
```

```
            app.ImageViewer = uiaxes(app.UIFigure);
            title(app.ImageViewer, 'Original')
            app.ImageViewer.Position = [81 63 246 230];

            % Create ImageData
            app.ImageData = uiaxes(app.UIFigure);
            title(app.ImageData, 'Edited')
            app.ImageData.Position = [426 63 246 230];

            % Create ImportButton
            app.ImportButton = uibutton(app.UIFigure, 'push');
            app.ImportButton.ButtonPushedFcn = createCallbackFcn(app,
@ImportButtonPushed, true);
            app.ImportButton.Position = [24 575 108 30];
            app.ImportButton.Text = 'Import';

            % Create ResetButton
            app.ResetButton = uibutton(app.UIFigure, 'push');
            app.ResetButton.ButtonPushedFcn = createCallbackFcn(app,
@ResetButtonPushed, true);
            app.ResetButton.Position = [170 575 108 30];
            app.ResetButton.Text = 'Reset';

            % Create GrayscaleButton
            app.GrayscaleButton = uibutton(app.UIFigure, 'push');
            app.GrayscaleButton.ButtonPushedFcn = createCallbackFcn(app,
@GrayscaleButtonPushed, true);
            app.GrayscaleButton.IconAlignment = 'center';
            app.GrayscaleButton.Position = [24 516 134 25];
            app.GrayscaleButton.Text = 'Grayscale';

            % Create RotationButton
            app.RotationButton = uibutton(app.UIFigure, 'push');
            app.RotationButton.ButtonPushedFcn = createCallbackFcn(app,
@RotationButtonPushed, true);
            app.RotationButton.Position = [169 515 134 25];
            app.RotationButton.Text = 'Rotation';

            % Create ContrastAdjustmentButton
            app.ContrastAdjustmentButton = uibutton(app.UIFigure, 'push');
            app.ContrastAdjustmentButton.ButtonPushedFcn = createCallbackFcn(app,
@ContrastAdjustmentButtonPushed, true);
            app.ContrastAdjustmentButton.IconAlignment = 'center';
            app.ContrastAdjustmentButton.Position = [313 516 134 25];
            app.ContrastAdjustmentButton.Text = 'Contrast Adjustment';

            % Create GaussianBlurButton
            app.GaussianBlurButton = uibutton(app.UIFigure, 'push');
            app.GaussianBlurButton.ButtonPushedFcn = createCallbackFcn(app,
@GaussianBlurButtonPushed, true);
            app.GaussianBlurButton.IconAlignment = 'center';
            app.GaussianBlurButton.Position = [457 516 134 25];
            app.GaussianBlurButton.Text = 'Gaussian Blur';

            % Create SobelEdgeDetectionButton
```

```
            app.SobelEdgeDetectionButton = uibutton(app.UIFigure, 'push');
            app.SobelEdgeDetectionButton.ButtonPushedFcn = createCallbackFcn(app,
@SobelEdgeDetectionButtonPushed, true);
            app.SobelEdgeDetectionButton.IconAlignment = 'center';
            app.SobelEdgeDetectionButton.Position = [601 516 134 25];
            app.SobelEdgeDetectionButton.Text = 'Sobel Edge Detection';

            % Create InvertColorsButton
            app.InvertColorsButton = uibutton(app.UIFigure, 'push');
            app.InvertColorsButton.ButtonPushedFcn = createCallbackFcn(app,
@InvertColorsButtonPushed, true);
            app.InvertColorsButton.Position = [24 483 134 25];
            app.InvertColorsButton.Text = 'Invert Colors';

            % Create ResizeButton
            app.ResizeButton = uibutton(app.UIFigure, 'push');
            app.ResizeButton.ButtonPushedFcn = createCallbackFcn(app,
@ResizeButtonPushed, true);
            app.ResizeButton.Position = [171 483 134 25];
            app.ResizeButton.Text = 'Resize';

            % Create HistogramEqualizationButton
            app.HistogramEqualizationButton = uibutton(app.UIFigure, 'push');
            app.HistogramEqualizationButton.ButtonPushedFcn = createCallbackFcn(app,
@HistogramEqualizationButtonPushed, true);
            app.HistogramEqualizationButton.Position = [313 483 134 25];
            app.HistogramEqualizationButton.Text = 'Histogram Equalization';

            % Create MedianFilterButton
            app.MedianFilterButton = uibutton(app.UIFigure, 'push');
            app.MedianFilterButton.ButtonPushedFcn = createCallbackFcn(app,
@MedianFilterButtonPushed, true);
            app.MedianFilterButton.Position = [457 483 134 25];
            app.MedianFilterButton.Text = 'Median Filter';

            % Create ErosionButton
            app.ErosionButton = uibutton(app.UIFigure, 'push');
            app.ErosionButton.ButtonPushedFcn = createCallbackFcn(app,
@ErosionButtonPushed, true);
            app.ErosionButton.Position = [601 483 134 25];
            app.ErosionButton.Text = 'Erosion';

            % Create DilationButton
            app.DilationButton = uibutton(app.UIFigure, 'push');
            app.DilationButton.ButtonPushedFcn = createCallbackFcn(app,
@DilationButtonPushed, true);
            app.DilationButton.Position = [24 448 134 25];
            app.DilationButton.Text = 'Dilation';

            % Create LaplacianFilterButton
            app.LaplacianFilterButton = uibutton(app.UIFigure, 'push');
            app.LaplacianFilterButton.ButtonPushedFcn = createCallbackFcn(app,
@LaplacianFilterButtonPushed, true);
            app.LaplacianFilterButton.Position = [170 448 134 25];
            app.LaplacianFilterButton.Text = 'Laplacian Filter';
```

```
            % Create GuidedFilterButton
            app.GuidedFilterButton = uibutton(app.UIFigure, 'push');
            app.GuidedFilterButton.ButtonPushedFcn = createCallbackFcn(app,
@GuidedFilterButtonPushed, true);
            app.GuidedFilterButton.Position = [313 448 134 25];
            app.GuidedFilterButton.Text = 'Guided Filter';

            % Create BinaryThresholdingButton
            app.BinaryThresholdingButton = uibutton(app.UIFigure, 'push');
            app.BinaryThresholdingButton.ButtonPushedFcn = createCallbackFcn(app,
@BinaryThresholdingButtonPushed, true);
            app.BinaryThresholdingButton.Position = [457 448 134 25];
            app.BinaryThresholdingButton.Text = 'Binary Thresholding';

            % Create CannyEdgeDetectionButton
            app.CannyEdgeDetectionButton = uibutton(app.UIFigure, 'push');
            app.CannyEdgeDetectionButton.ButtonPushedFcn = createCallbackFcn(app,
@CannyEdgeDetectionButtonPushed, true);
            app.CannyEdgeDetectionButton.Position = [601 448 134 25];
            app.CannyEdgeDetectionButton.Text = 'Canny Edge Detection';

            % Create LogarithmicTransformationButton
            app.LogarithmicTransformationButton = uibutton(app.UIFigure, 'push');
            app.LogarithmicTransformationButton.ButtonPushedFcn =
createCallbackFcn(app, @LogarithmicTransformationButtonPushed, true);
            app.LogarithmicTransformationButton.IconAlignment = 'center';
            app.LogarithmicTransformationButton.Position = [24 414 134 25];
            app.LogarithmicTransformationButton.Text = 'Logarithmic Transformation';

            % Create SharpeningButton
            app.SharpeningButton = uibutton(app.UIFigure, 'push');
            app.SharpeningButton.ButtonPushedFcn = createCallbackFcn(app,
@SharpeningButtonPushed, true);
            app.SharpeningButton.Position = [170 414 134 25];
            app.SharpeningButton.Text = 'Sharpening';

            % Create RGBtoHSVConversionButton
            app.RGBtoHSVConversionButton = uibutton(app.UIFigure, 'push');
            app.RGBtoHSVConversionButton.ButtonPushedFcn = createCallbackFcn(app,
@RGBtoHSVConversionButtonPushed, true);
            app.RGBtoHSVConversionButton.Position = [313 414 134 25];
            app.RGBtoHSVConversionButton.Text = 'RGB to HSV Conversion';

            % Create ClosingOperationButton
            app.ClosingOperationButton = uibutton(app.UIFigure, 'push');
            app.ClosingOperationButton.ButtonPushedFcn = createCallbackFcn(app,
@ClosingOperationButtonPushed, true);
            app.ClosingOperationButton.Position = [457 414 134 25];
            app.ClosingOperationButton.Text = 'Closing Operation';

            % Create AdaptiveThresholdingButton
            app.AdaptiveThresholdingButton = uibutton(app.UIFigure, 'push');
            app.AdaptiveThresholdingButton.ButtonPushedFcn = createCallbackFcn(app,
@AdaptiveThresholdingButtonPushed, true);
```

```
            app.AdaptiveThresholdingButton.Position = [601 414 134 25];
            app.AdaptiveThresholdingButton.Text = 'Adaptive Thresholding';

            % Create SaveButton
            app.SaveButton = uibutton(app.UIFigure, 'push');
            app.SaveButton.ButtonPushedFcn = createCallbackFcn(app,
@SaveButtonPushed, true);
            app.SaveButton.Position = [313 575 108 30];
            app.SaveButton.Text = 'Save';

            % Show the figure after all components are created
            app.UIFigure.Visible = 'on';
        end
    end

    % App creation and deletion
    methods (Access = public)

        % Construct app
        function app = Transformer

            % Create UIFigure and components
            createComponents(app)

            % Register the app with App Designer
            registerApp(app, app.UIFigure)

            if nargout == 0
                clear app
            end
        end

        % Code that executes before app deletion
        function delete(app)

            % Delete UIFigure when app is deleted
            delete(app.UIFigure)
        end
    end
end
```

Summary:

The implemented MATLAB app, named **Transformer**, serves as a practical and user-friendly tool for performing a variety of image processing operations. The app features a range of buttons, each corresponding to a specific transformation or enhancement, allowing users to interactively edit images without the need for extensive programming knowledge. Key functionalities include importing images, applying transformations, and saving the edited results.

The app covers fundamental image processing tasks such as grayscale conversion, rotation, contrast adjustment, filtering operations (Gaussian blur, median filter, Laplacian filter), edge detection (Sobel, Canny), color space conversion (RGB to HSV), and morphological operations (erosion, dilation, closing). Additionally, it provides options for inversion, resizing, histogram equalization, logarithmic transformation, and a guided filter.

Conclusions:

1. User-Friendly Interface:

- The app's design prioritizes user-friendliness, offering a straightforward interface with dedicated buttons for each operation. This approach caters to users with varying levels of expertise in image processing.

2. Versatility in Image Processing:

- The app provides a versatile set of image processing operations, allowing users to experiment with a wide range of transformations. This versatility makes it suitable for both beginners and users with specific image editing needs.

3. Interactive and Real-Time Preview:

- The app enables users to interactively apply transformations and observe real-time previews of the edited images. This feature enhances the user experience by providing instant feedback.

4. Assumptions and Limitations:

- The app assumes that users have basic image processing needs and may not be familiar with complex algorithms. While it covers a diverse set of operations, it does not include every possible image processing technique.

5. File Import and Save Functionality:

- The ability to import images and save edited results enhances the practicality of the app. Users can seamlessly load images, perform edits, and save the results in common image formats.

6. Clear Code Structure:

- The code is organized into well-defined methods, making it readable and modular. However, further documentation within the code or comments could enhance its understandability for future maintenance or collaboration.

In conclusion, the **Transformer** app successfully fulfills its purpose as an accessible image processing tool. Its simplicity, real-time feedback, and diverse functionalities make it a valuable resource for users seeking an intuitive way to enhance and transform their images. Future development could focus on expanding the range of transformations and refining the user interface for an even more seamless experience.