ECE 593-01: Homework #1- Exploration of MATLAB Image Processing Toolbox Features

Kaleb Byrum
Computer Vision and Image Processing Laboratory
University of Louisville
Louisville, KY, USA
kabyru01@louisville.edu

***Abstract*—This report will explore different features available for use in the MATLAB Image Processing Toolbox, including histogram equalization, edge detection and enhancement, and noise filtering.**

Keywords—noise filtering, histogram equalization, edge detection, image processing

I. Introduction

The MATLAB Image Processing Toolbox provides a multitude of included image manipulation commands that are useful for Computer Vision scientists. This report will first explore the commands included that allow for the manipulation of image contrast through Histogram of Intensities equalization Next, commands that enhance and detect edges in images through different included algorithms will be explored, including the Roberts' Cross method to reveal and enhance edges, and the Canny algorithm to reveal both strong and weak edges in an image. Finally, this report will explore the use of median and Gaussian noise filters to clarify images subjected to different intensities of noise.

II. Part One: Equalizing the Histogram of Intensities to Even Out Image Contrast

A. Defining the Histogram of Intensities & Image Contrast

In MATLAB, greyscale images are described as 2-dimensional matrices, which can be visualized using a histogram. The histogram generated from MATLAB image data describes the "intensity" of each pixel in the image in relation to surrounding data points. In relation to images, this is analogous to *image contrast*, which describes the difference in color (or gray levels in this case) that make an object distinguishable.

First, an example image will be shown alongside its respective Histogram of Intensities, where then the image will be equalized using the MATLAB command *histeq*, which equalizes the Histogram of Intensities to resemble a uniform distribution, which in turn provides equal contrast to all pixels in the image [1].

B. Generating the Histogram of Intensities for the Original Image

The following image will be used to demonstrate the effects of Histogram of Intensities equalization:

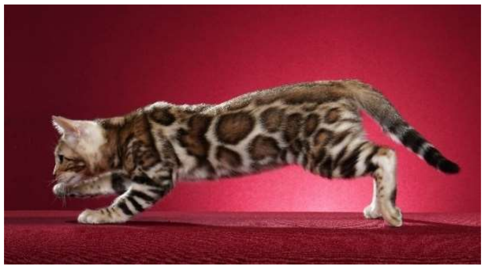

Figure 1: Original photo that will be equalized, "cat.jpg."

The resulting Histogram of Intensities from this photo's MATLAB data is as follows:

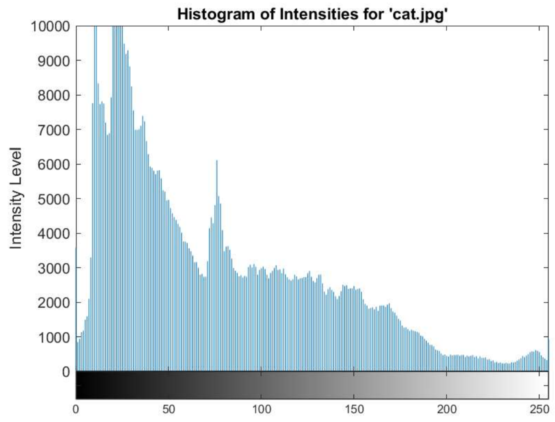

Figure 2: Resulting Histogram of Intensities from Figure 1.

C. Generating the Equalized Image and Corresponding Histogram of Intensities

Using the *histeq* command, the following equalized Histogram of Intensities is generated:

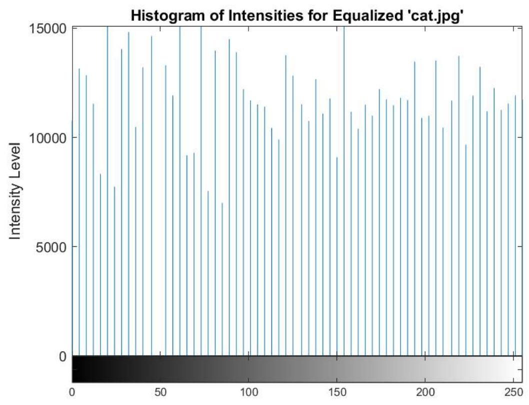

Figure 3: Equalized Histogram of Intensities for Figure 4.

This results in the following generated image:

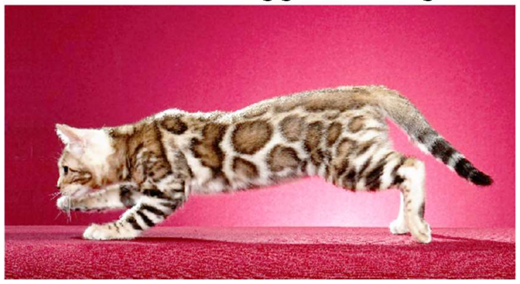

Figure 4: Equalized form of "cat.jpg" generated from the *histeq* command

As seen in Figure 4, the *histeq* command successful equalizes the contrast of images by making direct manipulations to their underlying matrix data in the MATLAB workspace, which is an efficient means to ensure edges can be properly detected by other commands within the Toolbox.

III. Sharpening Images to Reduce Blur/Enhance Edges

A. Introduction

Before continuing on to explore the algorithms available to detect and enhance edges, MATLAB's ability to *sharpen* images will be explored, which is a quick and computationally lax means to reduce blur in images, as well as emphasize edges in a blurry image. This is possible through the Image Processing Toolbox's *imsharpen* command, which uses *unsharp masking* to emphasize borders around objects in images, which in turn reduces blur in those areas [2]. Under the hood of this algorithm is the use of a Gaussian Low-Pass Filter, which can be used to filter fuzz in images.

B. Using the imsharpen Command

The following photo will be used to demonstrate the effects of using the *imsharpen* command:

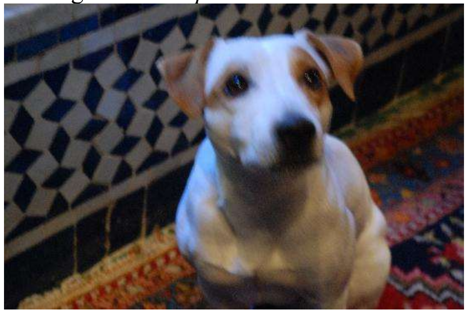

Figure 5: Photo that will be sharpened, "blurryDog.jpg"
Note that Figure 5 is a slightly blurry image on purpose, and *imsharpen* will be used to reduce blur around the main object in the image (the dog) which will emphasize the edges around them as well.

The following image is created by sending Figure 5 through the *imsharpen* command:

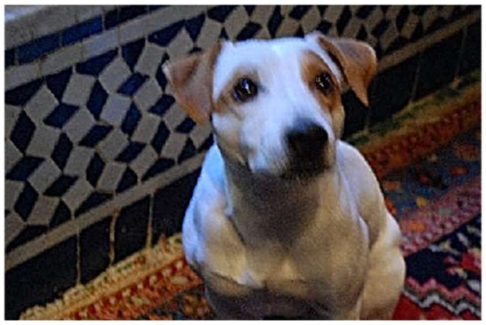

Figure 6: Sharpened version of "blurryDog.jpg" resulting from the execution of the *imsharpen* command.

As seen from Figure 6, *imsharpen* provides a quick and easy way to reduce fuzz and blur in images, which in turn enhances the edges of the image for later computations.

IV. Enhancing Edges using the Roberts Cross Algorithm

A. Introduction

In MATLAB, the Roberts Cross algorithm can be used to reveal the major edges in a photo, which then can be used with later computations for various tasks. This is possible using the *edge* command in the Image Processing Toolbox, which provides various algorithms to detect and enhance edges of images [3]. Alongside the Roberts Cross algorithm, the *edge* command also includes the Sobel algorithm, the Prewitt algorithm, the Laplacian of Gaussian algorithm, the zerocross method, and the Canny algorithm.

The Roberts Cross algorithm uses the Roberts approximation to the derivative to generate the edges around the main object in the image.

B. Using the Roberts Cross Algorithm to Reveal/Emhance the Main Edges in an Image

The following image will be used to demonstrate the output of the Roberts Cross algorithm:

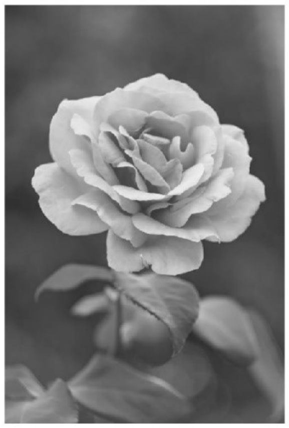

Figure 7: Image that will be used in the Roberts Cross algorithm edge enhancement, "flower.jpg" Note that images must be in greyscale to meet the appropriate matrix size restrictions.

The following image is created by the algorithm:

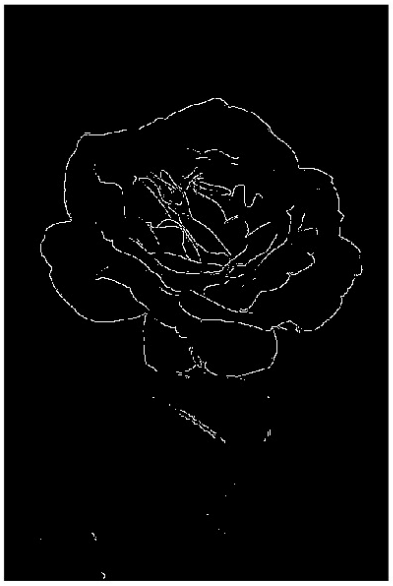

Figure 8: Output of the Roberts Cross algorithm, which reveals the main edges of the main object in the photo, which is the outer border of the flower.

As seen from Figure 8, the Roberts Cross algorithm successfully reveals the main edges of the image, as expected. The results of this algorithm can be used in later computations that require the knowledge of where the edges in an image are located.

V. Detecting Main and Weak Edges in an Image using the Canny Algorithm

A. Introduction

In addition to the Roberts Cross algorithm, the *Canny algorithm* can also be used to detect edges in an image. In comparison to other algorithms available for use in the *edge* command, the Canny algorithm provides more robust results, reveals *weak edges* that are not normally detected or enhanced by other algorithms, including the Roberts Cross algorithm.

The Canny algorithm finds edges my looking for local maxima in the gradient of the image, which are found by taking the derivative of a Gaussian filter. Two thresholds are used in this algorithm to respectively detect strong (main) and weak edges, as well as weak edges that are connected to strong edges. The use of two thresholds is the main reason behind the Canny algorithm's robustness in detecting edges in images.

B. Using the Canny Algorithm to Detect Edges

Figure 7 will be used to demonstrate the output of the Canny algorithm, to allow for comparisons between the Canny and Roberts Cross algorithms. The output of Figure 7 being subjected to the Canny algorithm is as follows:

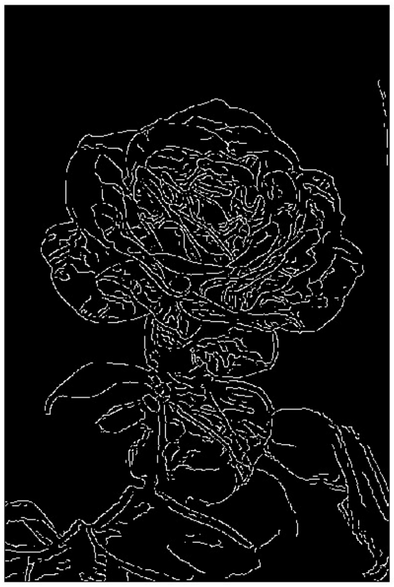

Figure 9: Output of the Canny algorithm, which reveals both main and weak edges in an image. This includes the petals of the flower, as well as the veins slightly shown within the leaves.

As seen by Figure 9, the Canny algorithm proves to be a more robust method of revealing *all* edges in an image, while the Roberts Cross algorithm is a good method for revealing only the *outer borders* of the main object in an image.

VI. Filtering Image Noise using Median and Gaussian Filters

A. Introduction

Within the Image Processing Toolbox are various methods to filter unwanted image *noise,* which is defined as any pixel definition that is an unintended result. Two main filter types that are available for use are *median* and *Gaussian* filters.

Median filters perform filtering of an image by creating the output value of each pixel the median value in a 3x3 neighborhood around the pixel. For small amounts of signal noise, this method is extremely efficient.

Gaussian filters use a 2D Gaussian smoothing kernel with a default standard deviation of 0.5 to return each output pixel. This method is efficient in generating image blur as well as removing various levels of noise. MATLAB can implement a Gaussian filter using the *imgaussfilt* command, and a median filter using the *medfilt2* command. [4] [5].

B. Using Median Filters on Noise-afflicted Images

To demonstrate the output of median filters on noisy images, the following image will be used:

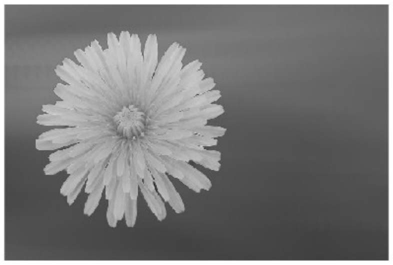

Figure 10: Greyscale version of “dandelion.jpg,” which will be used to demonstrate the output of median and Gaussian image filters.

To demonstrate its noise filtering abilities, Figure 10 will first be subjected to artificial “salt and pepper” noise at SNR levels of 0.02, 0.04 and 0.08, respectively:

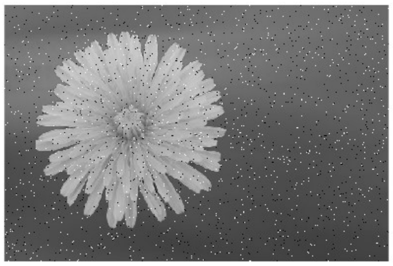

Figure 11: Figure 10 subjected to “salt and pepper” noise at SNR = 0.02

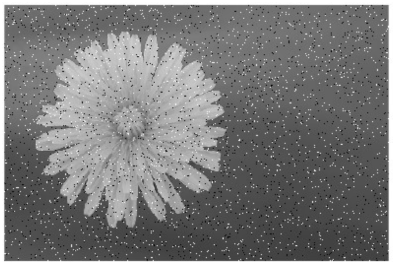

Figure 12: Figure 10 subjected to “salt and pepper” noise at SNR = 0.04

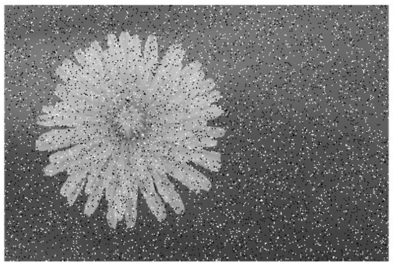

Figure 13: Figure 10 subjected to “salt and pepper” noise at SNR = 0.08

The following images result from subjecting Figures 11 through 13 to a median image filter:

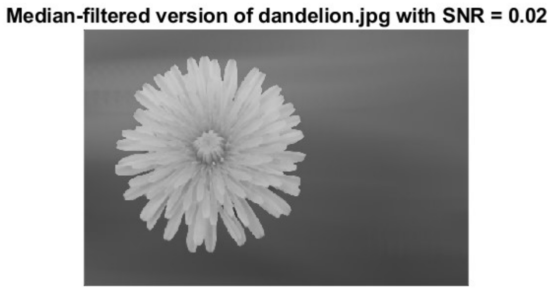

Figure 14: Output of Figure 11 being subjected to a median image filter.

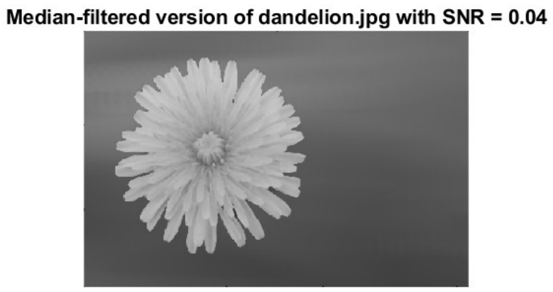

Figure 15: Output of Figure 12 being subjected to a median image filter.

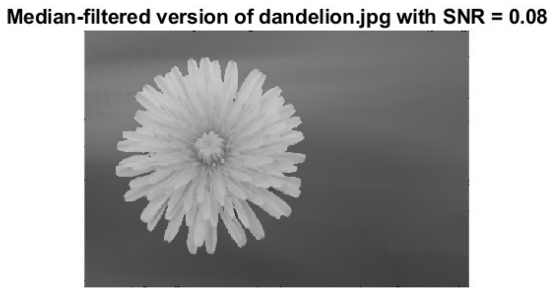

Figure 16: Output of Figure 13 being subjected to a median filter.

As seen from Figures 14 through 16, median filters are a robust method of reducing noise, especially at low SNR levels. For images afflicted by higher levels of SNR, other filtering methods may be necessary.

C. Using Gaussian Filters on Noise-afflicted Images

To demonstrate the output of Gaussian filters on noisy images, Figure 10 will be used to allow for comparisons of results between median and Gaussian filters. Salt and pepper noise at SNR levels of 0.02, 0.04 and 0.08 will be used once again to demonstrate the differences between the two filters; Figures 11 through 13 demonstrate what images will be subjected to a Gaussian filter.

The following images result from subjecting Figures 11 through 13 to a Gaussian image filter:

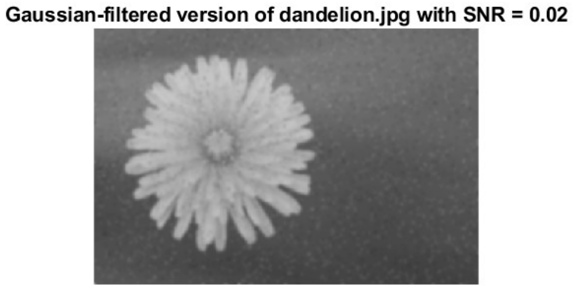

Figure 17: Output of Figure 11 being subjected to a Gaussian filter with 0.5 standard deviation.

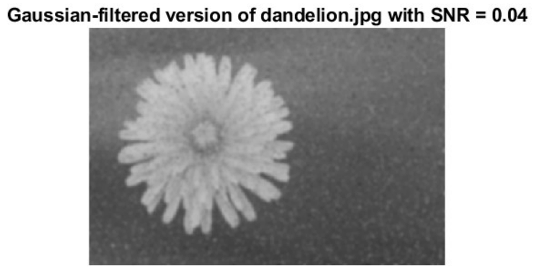

Figure 18: Output of Figure 12 being subjected to a Gaussian filter with 0.5 standard deviation.

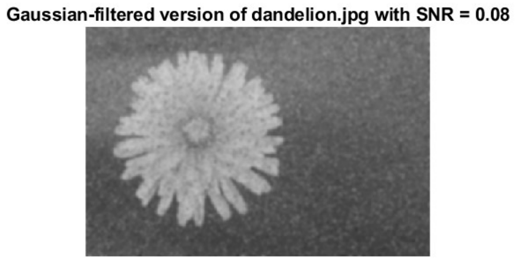

Figure 19: Output of Figure 13 being subjected to a Gaussian filter with 0.5 standard deviation.

As seen by Figures 17 through 19, Gaussian filters provide a somewhat robust method for reducing noise. It's efficiency in reducing noise may also vary by the type of noise applied and is evidently not as efficient in removing salt and pepper noise compared to median filters.

VII. Conclusion

In conclusion, the MATLAB Image Processing Toolbox includes an amass of tools to manipulate images as necessary, including contrast balancing, edge detection and enhancement and noise filtering. These tools are important starting points for later computations that this course will explore.

VIII. References

[1] The MathWorks Inc., "Enhance contrast using histogram equalization - MATLAB histeq," The MathWorks Inc., 2019. [Online]. Available: https://www.mathworks.com/help/images/ref/histeq.html. [Accessed 11 January 2020].

[2] The MathWorks Inc., "Sharpen image using unsharp masking - MATLAB imsharpen," The MathWorks Inc., 2019. [Online]. Available: https://www.mathworks.com/help/images/ref/imsharpen.html. [Accessed 11 January 2020].

[3] The MathWorks Inc., "Find edges in intensity image - MATLAB edge," The Mathworks Inc., 2019. [Online]. Available: https://www.mathworks.com/help/images/ref/edge.html. [Accessed 11 January 2020].

[4] The MathWorks Inc., "2-D Gaussian filtering of images - MATLAB imgaussfilt," The MathWorks Inc., 2019. [Online]. Available: https://www.mathworks.com/help/images/ref/imgaussfilt.html. [Accessed 11 January 2020].

[5] The MathWorks Inc., "2-D median filtering - MATLAB medfilt2," The MathWorks Inc., 2019. [Online]. Available: https://www.mathworks.com/help/images/ref/medfilt2.html. [Accessed 11 January 2020]

```matlab
%CatHistogram, used in ECE 593-01 HW1, written by Kaleb Byrum on 1/10/2019

clear all
close all

I = imread('cat.jpg');
%Original cat photo
figure, imshow(I);
%Histogram of Intensities for original cat photo
figure, imhist(I);
title("Histogram of Intensities for 'cat.jpg'"), xlabel("Spectrum of Intensities"), ylabel("Intensity Level");
saveas(gcf, 'CatHistogram/originalCatIntensityHistogram.png');
%Create new photo object with equalized Histogram of Intensities
I2 = histeq(I);
figure, imshow(I2), saveas(gcf, 'CatHistogram/equalizedCat.png');
%Show equalized Histogram of Intensities
figure, imhist(I2);
title("Histogram of Intensities for Equalized 'cat.jpg'"), xlabel("Spectrum of Intensities"), ylabel("Intensity Level");
saveas(gcf, 'CatHistogram/equalizedCatIntensityHistogram.png');


```

```matlab
%DogSharpen, used in ECE 593-01 HW1, written by Kaleb Byrum on 1/10/2019

clear all
close all

%Read blurry image.
I = imread("blurryDog.jpg");
figure, imshow(I);
%Generate sharpened version of image using imsharpen
I2 = imsharpen(I, 'Radius', 2, 'Amount', 2);
figure, imshow(I2), saveas(gcf, "DogSharpen/sharpenedDog.png");
```

```matlab
%usingCannyEdge, used in ECE 593-01 HW1, written by Kaleb Byrum on 1/10/2019

clear all
close all

I = imread("flower.png");
%Generate grayscale version of input image.
I2 = rgb2gray(I);
figure, imshow(I2), saveas(gcf,"usingCannyEdge/grayScaleFlower.png");
%Create Canny Edge Enhanced Image
BW1 = edge(I2, "Canny");
figure, imshow(BW1), saveas(gcf, "usingCannyEdge/cannyFlower.png");

```

```
%usingCrossEdge, used in ECE 593-01 HW1, written by Kaleb Byrum on 1/10/2019

clear all
close all

I = imread("flower.png");
%Generate grayscale version of input image.
I2 = rgb2gray(I);
figure, imshow(I2), saveas(gcf,"usingCrossEdge/grayScaleFlower.png");
%Create Roberts Cross Edge Enhanced Image
BW1 = edge(I2, "Roberts");
figure, imshow(BW1), saveas(gcf, "usingCrossEdge/crossFlower.png");
```

```matlab
%usingGaussianFilters, used in ECE 593-01 HW1, written by Kaleb Byrum on 1/10/2019

I = imread("dandelion.jpg");
%Generate grayscale version of input image.
I2 = rgb2gray(I);
figure, imshow(I2), saveas(gcf,"usingGaussianFilters/grayScaleDandelion.png");

%SNR = 0.02
I3 = imnoise(I2, 'salt & pepper', 0.02);
figure, imshow(I3), saveas(gcf, "usingGaussianFilters/noisyDandelionSNR002.png");
I4 = imgaussfilt(I3, 2);
figure, imshow(I4), title("Gaussian-filtered version of dandelion.jpg with SNR = 0.02");
saveas(gcf,"usingGaussianFilters/gaussianDandelionImageSNR002.png");

%SNR = 0.04
I3 = imnoise(I2, 'salt & pepper', 0.04);
figure, imshow(I3), saveas(gcf, "usingGaussianFilters/noisyDandelionSNR004.png");
I4 = imgaussfilt(I3, 2);
figure, imshow(I4), title("Gaussian-filtered version of dandelion.jpg with SNR = 0.04");
saveas(gcf,"usingGaussianFilters/gaussianDandelionImageSNR004.png");

%SNR = 0.08
I3 = imnoise(I2, 'salt & pepper', 0.08);
figure, imshow(I3), saveas(gcf, "usingGaussianFilters/noisyDandelionSNR008.png");
I4 = imgaussfilt(I3, 2);
figure, imshow(I4), title("Gaussian-filtered version of dandelion.jpg with SNR = 0.08");
saveas(gcf,"usingGaussianFilters/gaussianDandelionImageSNR008.png");
```

```matlab
%usingMedianFilters, used in ECE 593-01 HW1, written by Kaleb Byrum on 1/10/2019

I = imread("dandelion.jpg");
%Generate grayscale version of input image.
I2 = rgb2gray(I);
figure, imshow(I2), saveas(gcf,"usingMedianFilters/grayScaleDandelion.png");

%SNR = 0.02
I3 = imnoise(I2, 'salt & pepper', 0.02);
figure, imshow(I3), saveas(gcf, "usingMedianFilters/noisyDandelionSNR002.png");
I4 = medfilt2(I3);
figure, imshow(I4), title("Median-filtered version of dandelion.jpg with SNR = 0.02");
saveas(gcf,"usingMedianFilters/medianDandelionImageSNR002.png");

%SNR = 0.04
I3 = imnoise(I2, 'salt & pepper', 0.04);
figure, imshow(I3), saveas(gcf, "usingMedianFilters/noisyDandelionSNR004.png");
I4 = medfilt2(I3);
figure, imshow(I4), title("Median-filtered version of dandelion.jpg with SNR = 0.04");
saveas(gcf,"usingMedianFilters/medianDandelionImageSNR004.png");

%SNR = 0.08
I3 = imnoise(I2, 'salt & pepper', 0.08);
figure, imshow(I3), saveas(gcf, "usingMedianFilters/noisyDandelionSNR008.png");
I4 = medfilt2(I3);
figure, imshow(I4), title("Median-filtered version of dandelion.jpg with SNR = 0.08");
saveas(gcf,"usingMedianFilters/medianDandelionImageSNR008.png");
```