מבוא לעיבוד ספרתי של תמונות

עבודת הגשה מס' 1

נתן דוידוב 211685300

ניקולאי קרוחמל 320717184

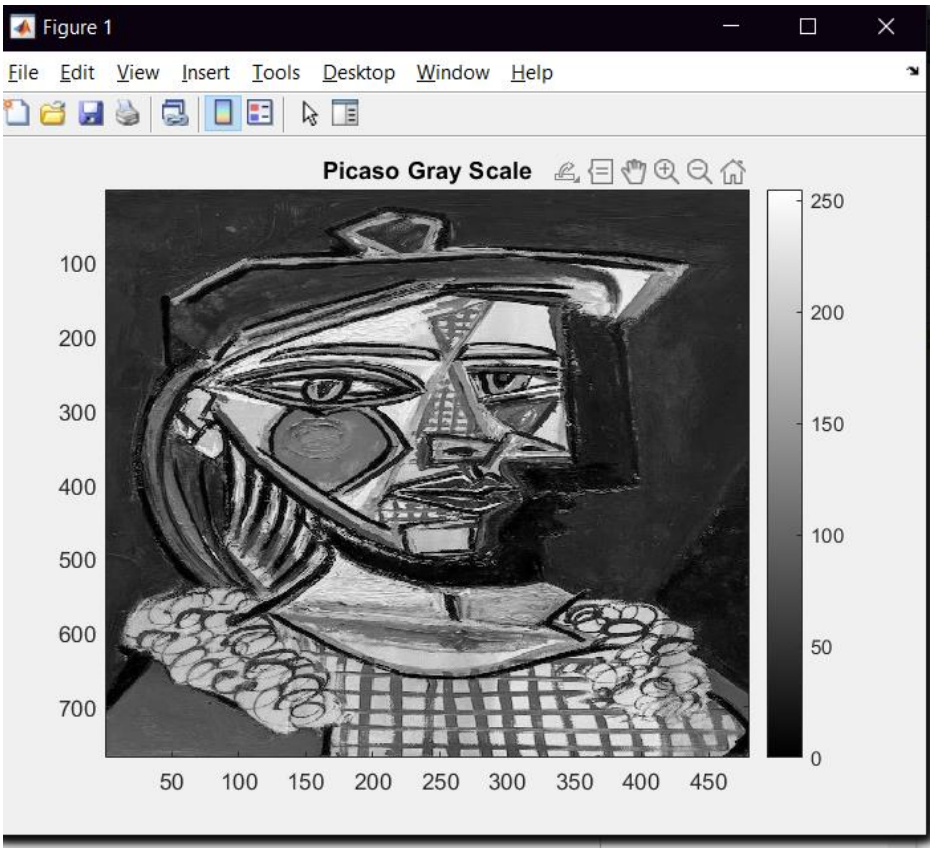
1. Histogram Manipulation

1.1 Image Reading

קראנו את התמונה Picasso.jpg מתוך תקיית התמונות, העברנו לגווני אפור ונירמלנו אותה ע"פ הנדרש. להלן הקוד:

```
%Natan Davidov 211685300, Nikolai Krokhmal 320717184

function normalizedGrayScale = dip_GN_imread(filename)
   path = fullfile('Images',filename);
   grayScale = double(rgb2gray(imread(path))); % make gray scale image
   minImg = min(grayScale(:));
   maxImg = max(grayScale(:));
   normalizedGrayScale = ((grayScale-minImg)/(maxImg-minImg)); % here we
   % normalize the values of the pixels
end
```

להלן התוצאה:



1.2 Histogram Construction

בסעיף זה בנינו פונקציה שמקבל תמונה ומספר סלסלות כפרמטרים ומחזירה היסטוגרמה של התמונה לפי מספר הסלסלות שניתן.

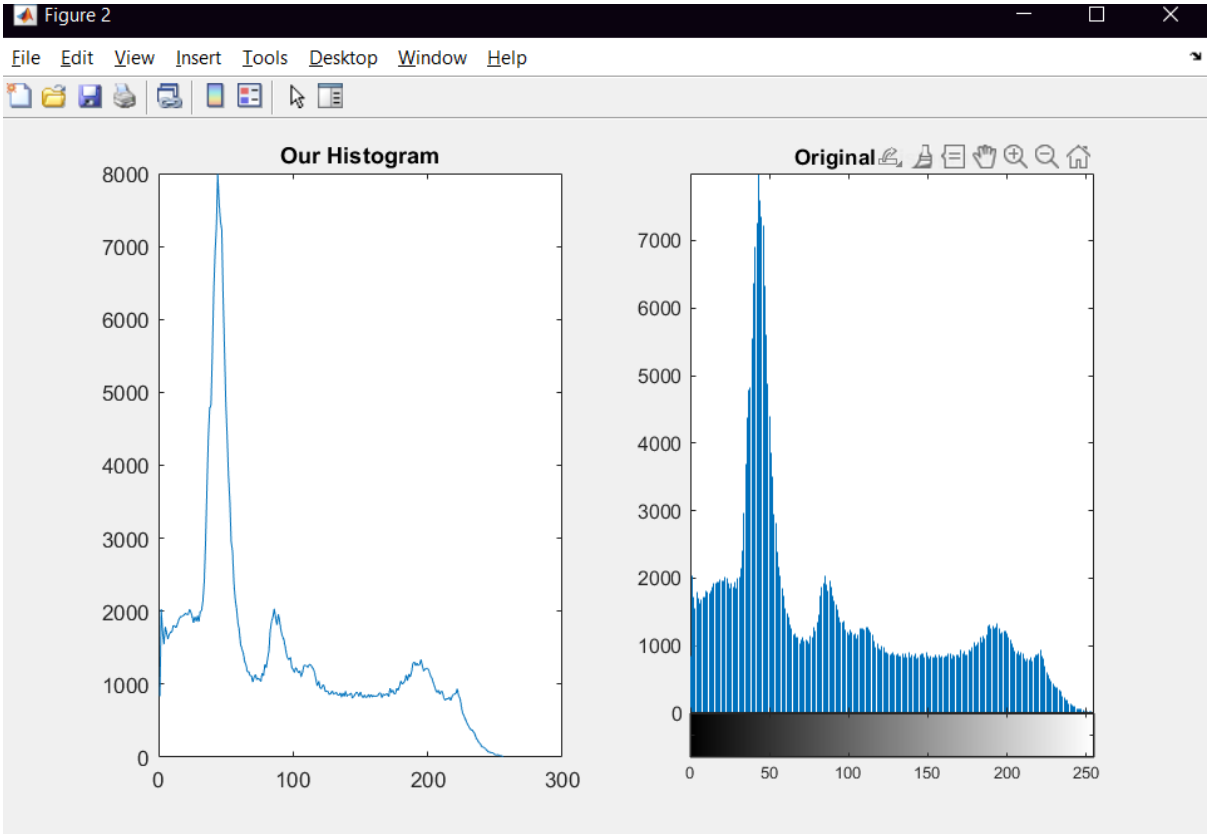
על מנת להשיג זאת ראשית בפונקציה שלנו אנחנו בונים היסטוגרמה רגילה של 256 רמות אפור. לאחר מכן אנחנו משתמשים בפונקצייה עזר כדי לחלק את 256 רמות האפור למקטעים כמספר הסלסלות שניתן. את המקטעים אנחנו מציגים בתור וקטור המכיל את הגבולות של כל מקטע. החישוב נעשה ע"י חלוקה במספר הסלסלות ועיגול כלפי מטה. לבסוף אנחנו סוכמים את כל הערכים של ההיסטוגרמה המקורית בין הגבולות של כל סלסלה. עובד גם אם 256 לא מתחלק במספר הסלסלות. להלן הקוד:

Editor - D:\Uni\TimeSkip\Semester G\Image processing\Ex1\HW1\dip_histogram.m *

Ex1.m | dip_histogram.m *

```
%Natan Davidov 211685300, Nikolai Krokhmal 320717184

function new_hist= dip_histogram(image, nbins)
img = imread(image);  % here we read img and make it grayscale
img = double(rgb2gray(img));
n = numel(img);
hist_vec = zeros(1, 256);
mat2vec = reshape(img, 1, []); % we reshape it into a vector to iterate
% over and build a histogram
for i = 1:1:n
    hist_vec(mat2vec(i)+1) = hist_vec(mat2vec(i)+1) + 1;
end
mySections = get_sections(nbins); % we get the sections from our sections
    % function
new_hist = zeros(1, nbins);
for j = 1:1:nbins % now we calculate the new histogram summing the amount
    % of pixels in each bin
    for i = mySections(j*2-1):mySections(j*2)
        new_hist(j) = new_hist(j) + hist_vec(i);
    end
end
end


function secVec = get_sections(nbins) % this function creates a vector
% with the section borders for each bin
    secVec = ones(1,2*nbins);
    for i = 1:nbins
        secVec(i*2) = floor(256*i/nbins);
    end
    for j = 3:2:2*nbins-1
        secVec(j) = secVec(j-1) +1;
    end
```

לבסוף כנדרש ביצענו בדיקה ע"י הכנסת 256 סלסלות לפונקציה שלנו, והשוונו אל מול ההיסטוגרמה האמיתי המתקבל ע"י פקודת imhist() .



כצפוי ניתן לראות ששתי התוצאות זהות.

1.3 Brightness

בסעיף זה כמתבקש בנינו פונקציה שמשנה את הבהירות של התמונה ע"י שינוי ערך של כל הפיקסלים באופן זהה, כאשר הגדלת הערכים של כל הפיקסלים תגרום לתמונה להיראות בהירה והקטנת הערכים של כל הפיקסלים תגרום לתמונה להיראות יותר כהה. הפונקציה שלנו יודעת להוסיף ערך קבוע לכל הפיקסלים או להכפיל את כל הפיקסלים בערך קבוע, ובנוסף את ערך הפיקסל עולה על 1 או יורד מתחת ל 0 אז הוא מוחלף ב1 או 0. להלן הקוד:

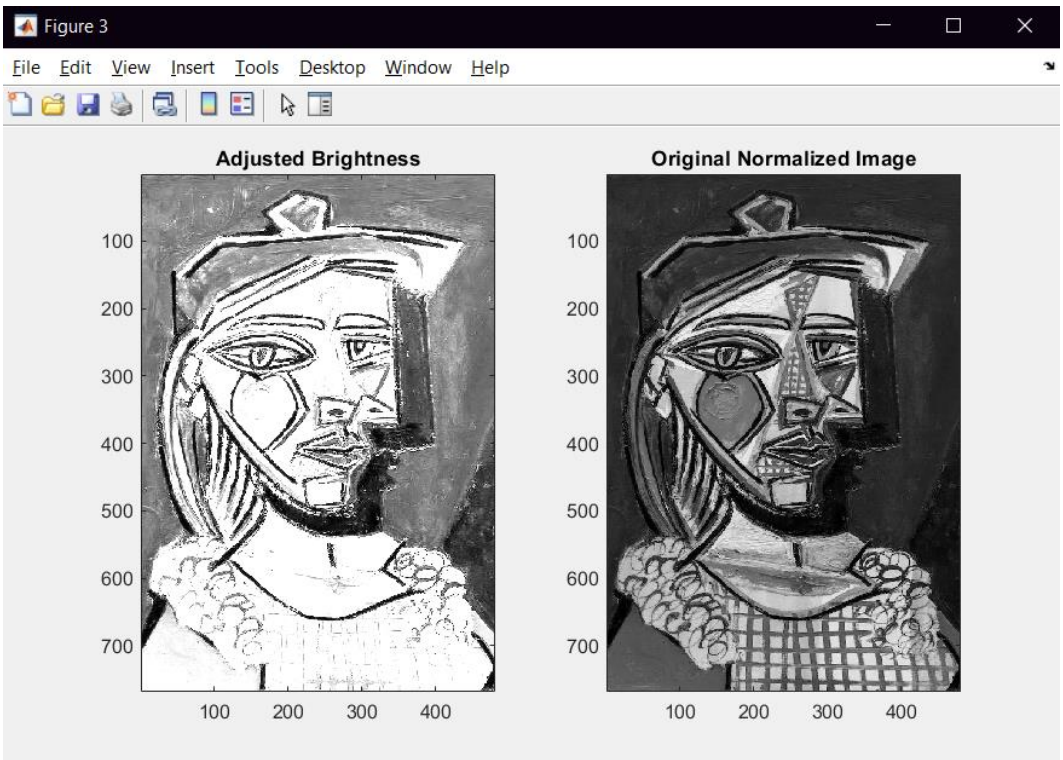
Editor - D:\Uni\TimeSkip\Semester G\Image processing\Ex1\HW1\adjust_brightness.m

Ex1.m | adjust_brightness.m

```
%Natan Davidov 211685300, Nikolai Krokhmal 320717184

% this main function send to the appropriate one dependeing on "action"
function modImage = adjust_brightness(img,action,parameter)
   if action == "mul"
        modImage = multiple_brightness(img,parameter);
   else
       modImage = add_brightness(img,parameter);
   end
end

% this function multiplies every pixel of img by parameter and truncks if
% greater the 1 or less then zero
function mulImage = multiple_brightness(img,parameter)
    [rows, cols] = size(img);
    for row = 1:rows
```

```
    for row = 1:rows
        for col = 1:cols
            img(row,col) = img(row,col)*parameter;
            if img(row,col) > 1
                img(row,col) = 1;
            elseif img(row,col) < 0
                img(row,col) = 0;
            end
        end
    end
    mulImage = img;
end


% this function add to every pixel of img thr parameter and trunckes if
% greater the 1 or less then zero
function addImage = add_brightness(img,parameter)
    [rows, cols] = size(img);
    for row = 1:rows
        for col = 1:cols
            img(row,col) = img(row,col)+parameter;
            if img(row,col) > 1
                img(row,col) = 1;
            elseif img(row,col) < 0
                img(row,col) = 0;
            end
        end
    end
    addImage = img;
end
```

בנוסף נצרף דוגמא של תמונה לה הגברנו את הבהירות פי 3 לעומת התמונה המקורית:



1.4 Contrast

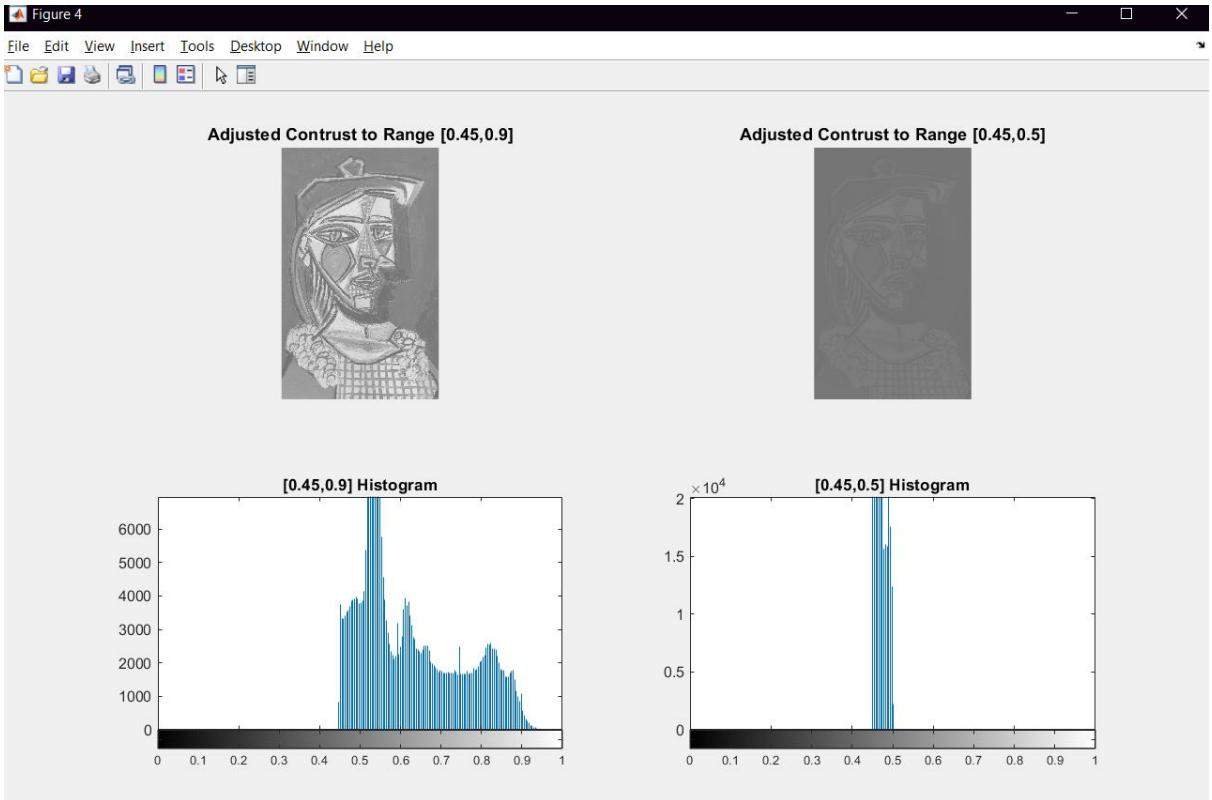
בסעיף זה התבקשנו לבנות פונקציה שתשנה את הContrast של התמונה לטווח ערכים חדש.
הcontrast (הניגודיות) בתמונה מתייחס למידת יכולת ההבחנה שלנו בין פרטים בתמונה לעומת הרקע שלהם וקשור לטווח הערכים של ההסיטוגרמה.

הפונקציה שלנו מקבל תמונה וטווח ערכים חדש עבור ההיסטוגרמה שלה ומחזירה תמונה עם הטווח החדש. את המיפוי אנחנו מבצעים בצורה לינארית, לכל פיקסל עפ הנוסחה הבאה:

$$P_{new} = \frac{(P_{old} - L_{old})}{(H_{old} - L_{old})} * (H_{new} - L_{new}) + L_{new}$$

כאשר P מייצג את הערך של הפיקסל, H את הגבול העליון של ההיסטוגרמה, L הגבול התחתון של ההיסטוגרמה וOld וnew מסמנים את הערכים הישנים והחדשים בהתאמה.

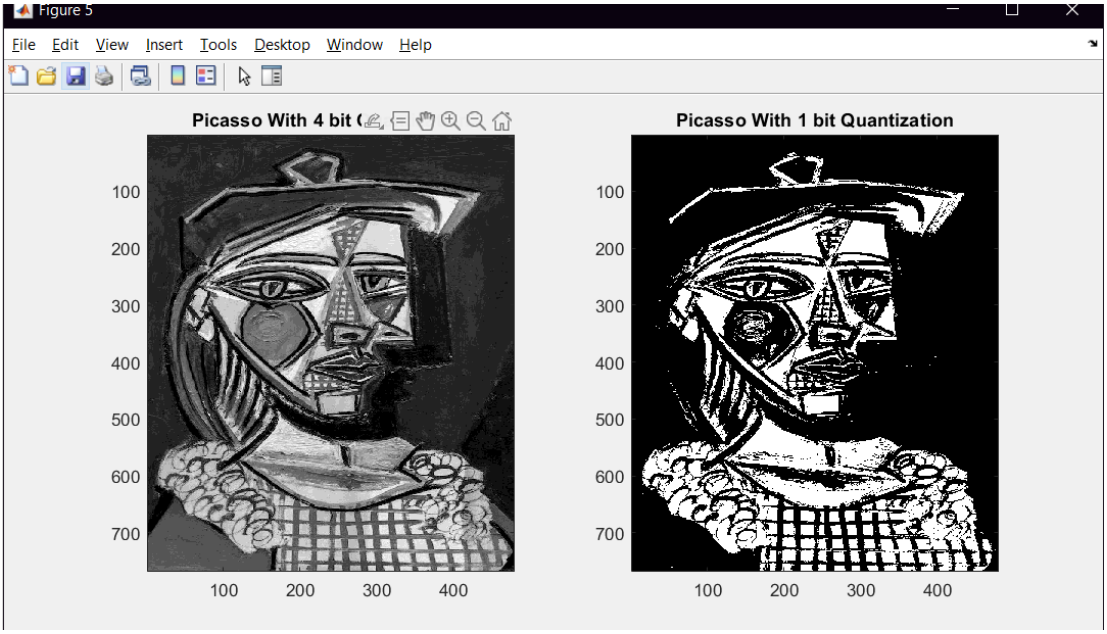
להלן שתי תמונות שעברו עיבוד בעזרת הפונקציה שלנו, אחת לטווח ערכים של [0.45,0.9] והשנייה לטווח ערכים של [0.45,0.5]. ניתן לראות כיצד ההיסטוגרמות שלהם מופו לטווחים אלו:



כצפוי, מיפוי ההיסטוגרמה לטווח ערכים הקטן של [0.45,0.5] ממש מקשה עלינו לראות את התמונה הייטב, מאחר וכמעט אין מנעד של גווני אפור. המיפוי השני פחות מקשה עלינו להבחין בפרטים, אך מאחר שכל ההיסטוגרמה מופתה לטווח ערכים גבוהה נראה כאילו הוספנו לה המון בהירות.

1.6 Quantization

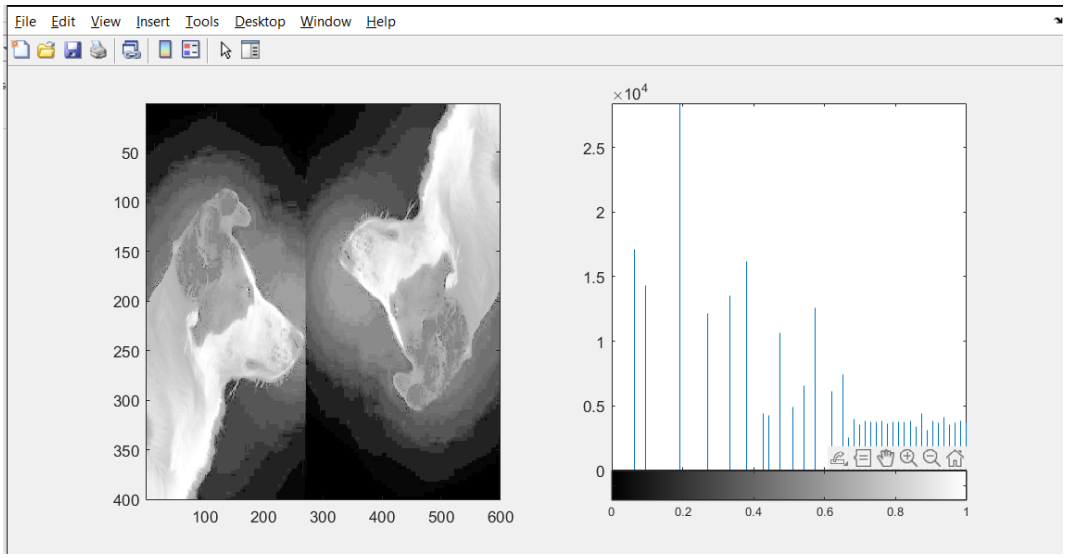
בסעיף זה התבקשנו לבצע קוונטיזציה של רמות האפור. כלומר לייצג את התמונה ע"י פחות רמות אפור, כתלות בכמות הביטים שאנחנו רוצים להקצות לכל רמה. פעם הבקשנו להקצות לכל רמת אפור 4 ביטים (כלומר $2^4=16$ רמות אפור) ובפעם השנייה להקצות ביט יחיד (כלומר 2 רמות אפור בלבד). את הקוונטיזציה ביצענו ע"י חלוקה של ערך של כל פיקסל ב $256-2^i$ (כאשר i הוא מספר הביטים). לאחר מכן ביצענו עיגול כלפי מטה והכפלנו באותו ערך חזרה. כך הקצענו כל פיקסל לרמת אפור מסויימת מתוך רמות האפור שאנחנו מרשים. להלן התוצאות:



ניתן לראות בבירור שמאחר ובתמונה הימנית אנחנו מאפשרים רק שתי רמות אפור היא נצבעה בשחור ולבן ואילו בתמונה השמאלית קיימות 16 רמות אפור מה שאמנם מוריד מעט מהאיכות שלה אך מאפשר שימוש בפחות זיכרון.

1.7 Histogram Equalization

בסעיף זה התשמשנו על התמונה dog.jpg בפקודה המובנית histeq() . להלן התוצאות:



פעולת histogram equalization מטרת לשטח את ההיסטוגרמה כך שההתפלגות של כל רמות האפור תיהיה אחידה עד כמה שניתן על כל הספקטרום. בעזרת פעולה זה ניתן לשפר את ניגודית של התמונה אם עיקר הפיקסלים בהיסטוגרמה שלה נעים בטווח ערכים צר. במקרה של התמונה dog.jpg הכלבים נראים בצבע לבן כמעט (למעט הכתם החמוד על העין) והרקע שחור. במקרה זה ההיסטוגרמה של התמונה כבר נעה על כל טווב הערכים כאשר רוב הפיקסלים יהיו בערכי קצה. Histogram equalization במקרה כזה קשה לבצע (אין מחליטים איזה פיקסלים נעים לאילו רמות אפור מבלי לפגוע בתמונה) ובנוסף הפעולה רק תוריד את הניגודיות בתמונה מה שיפגע באיכות שלה וביכולת שלנו להבחין בפרטים בה (מה שבאמת קרה עם הכלב, וניתן לראות שמטלאב לא ייצר התפלגות אחידה כמו שהיינו רוצים).

2. Spatial Filters and Noise

2.1 Read the Image

```
% 2.1 Spatial Filters and Noise
dog_gn = dip_GN_imread('dog.jpg');
```

שמרנו את התמונה המנורמלת ב gray scale ב- dog_gn.

2.2 Mean vs Median filter

2.2.1

```
function mean_filtered_img = mean_filter(img, k)
    %% Zero padding

    % Zero pad the image using integer k for kernel size
    sz = size(img);
    imgPadded = zeros(sz(1)+(k-1), sz(2)+(k-1));
    imgPadded(((floor((k-1)/2)):(sz(1))+(floor((k-1)/2)-1)), ...
        (((floor((k-1)/2)):(sz(2))+(floor((k-1)/2)-1)))) = img;


    % Mean filtering
    %X = ones(k);
    mean_filtered_img = zeros(size(img));  % Pre-allocate output image
    for row = 1:size(img, 1)
        for col = 1:size(img, 2)
            % Define neighborhood for filtering
            rowStart = row;
            rowEnd = row + k - 1;
            colStart = col;
            colEnd = col + k - 1;

            % Extract neighborhood and handle boundary cases
            neighborhood = imgPadded(rowStart:rowEnd, colStart:colEnd);

            % Calculate mean of non-zero elements in the neighborhood
            mean_value = mean(neighborhood(:));

            % Assign mean value to the output image
            mean_filtered_img(row, col) = mean_value;
        end
    end
end
```

בסעיף זה יצרנו פונקציה אשר מבצעת פילטר מיצוע בגודל k על k.
בשביל להתגבר על הבעיה בקצוות ריפדנו באפסים את התמונה כך: יצרנו מטריצת אפסים בגודל של התמונה המקורית פלוס k-1 שורות ועמודות, לאחר מכן באמצעות slicing הכנסנו את התמונה המקורית למרכז מטריצת האפסים. לאחר מכן באמצעות לולאה מקוננת עברנו על כל התמונה וביצענו מיצוע של k*k איברים.

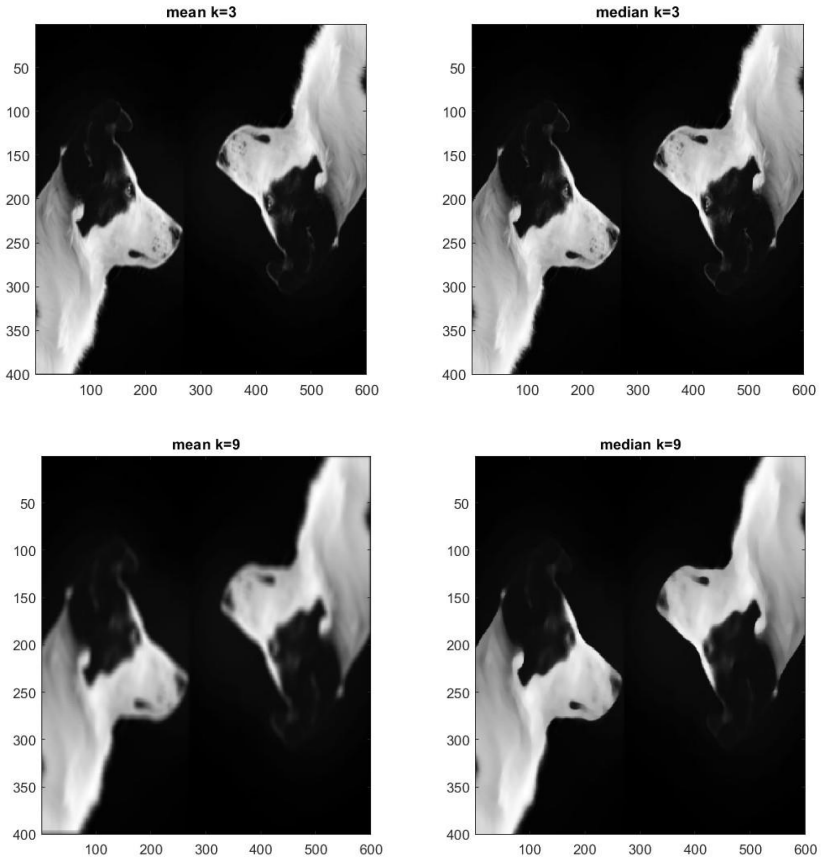
2.2.2 <u>סעיף אופציונלי</u>

2.2.3

```
% 2.2.3
mean3 = mean_filter(dog, 3);
mean9 = mean_filter(dog, 9);
med3 = medfilt2(dog,[3,3]);
med9 = medfilt2(dog,[9,9]);
figure;
subplot(1, 2, 1);
colormap('gray');
imagesc(mean3);
title('mean k=3');
subplot(1, 2, 2);
colormap('gray');
imagesc(med3);
title('median k=3');

figure;
subplot(1, 2, 1);
colormap('gray');
imagesc(mean9);
title('mean k=9');
subplot(1, 2, 2);
colormap('gray');
imagesc(med9);
title('median k=9');
```

השתמשנו בפונקציית המיצוע שכתבנו בסעיף 2.2.1 ובפונקציית medfilt2() המוכנה עבור k=3,9

להלן התוצאות:



כמצופה ניתן לראות שעבור k=9 התמונה יותר מטושטשת מ k=3 כי אנחנו מבצעים את המיצוע על יותר פיקסלים

2.3 Gaussian Filter

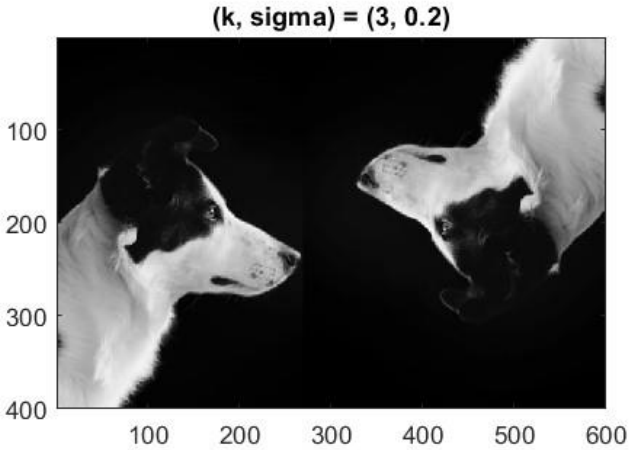
2.3.1

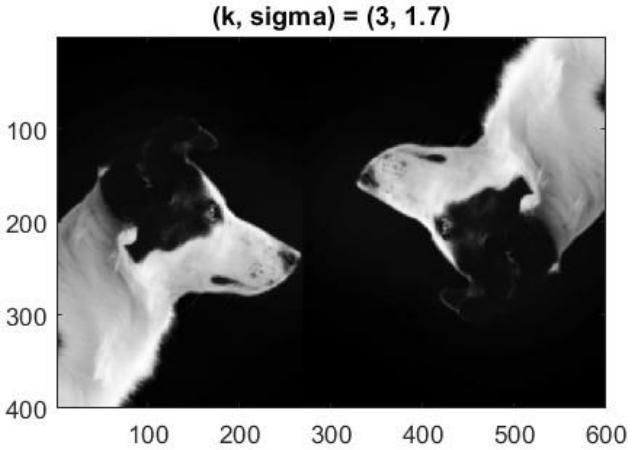
```
%Natan Davidov 211685300, Nikolai Krokhmal 320717184

function gauss_filtered_img = dip_gaussian_filter(img, k, sigma)
    [X, Y] = meshgrid(-(k-1)/2:(k-1)/2, -(k-1)/2:(k-1)/2);
    Filter = (1/(2*pi*sigma^2)).*exp(-(X.^2+Y.^2)/(2*sigma^2));


     % Zero pad the image using integer k for kernel size
    sz = size(img);
    imgPadded = zeros(sz(1)+(k-1), sz(2)+(k-1));
    imgPadded(((floor((k-1)/2)):(sz(1))+(floor((k-1)/2)-1)), ...
        (((floor((k-1)/2)):(sz(2))+(floor((k-1)/2)-1)))) = img;



    % Mean filtering
    gauss_filtered_img = zeros(size(img));  % Pre-allocate output image
    for row = 1:size(img, 1)
        for col = 1:size(img, 2)
            % Define neighborhood for filtering
            rowStart = row;
            rowEnd = row + k - 1;
            colStart = col;
            colEnd = col + k - 1;

            % Extract neighborhood and handle boundary cases
            neighborhood = imgPadded(rowStart:rowEnd, colStart:colEnd); %k*k matrix

            % Calculate mean of non-zero elements in the neighborhood
            gauss_value = sum(neighborhood.*Filter, "all");

            % Assign mean value to the output image
            gauss_filtered_img(row, col) = gauss_value;
        end
    end
end
```

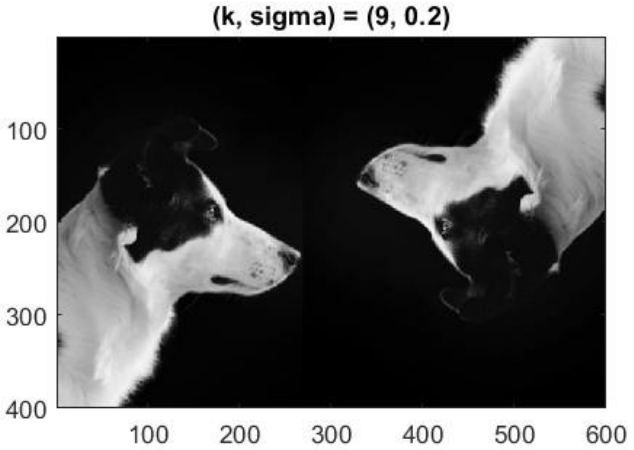
בסעיף זה כתבנו פונקציה אשר מקבלת בארגומנט את התמונה, את k ואת סיגמא ויוצרת באמצעות פונקציית meshgrid את הפילטר הגאוסי ולאחר מכן באמצעות לולאה מקוננת נעבור על התמונה המרופדת האפסים ונכפיל את הפילטר בחלק המתאים בתמונה.

2.3.2

```
% 2.3.2
gauss1 = dip_gaussian_filter(dog, 3, 0.2);
gauss2 = dip_gaussian_filter(dog, 3, 1.7);
gauss3 = dip_gaussian_filter(dog, 9, 0.2);
figure;
subplot(1,3,1);
colormap('gray');
imagesc(gauss1);
title('(k, sigma) = (3, 0.2)');
subplot(1,3,2);
imagesc(gauss2);
title('(k, sigma) = (3, 1.7)');
subplot(1,3,3);
imagesc(gauss3);
title('(k, sigma) = (9, 0.2)');
```



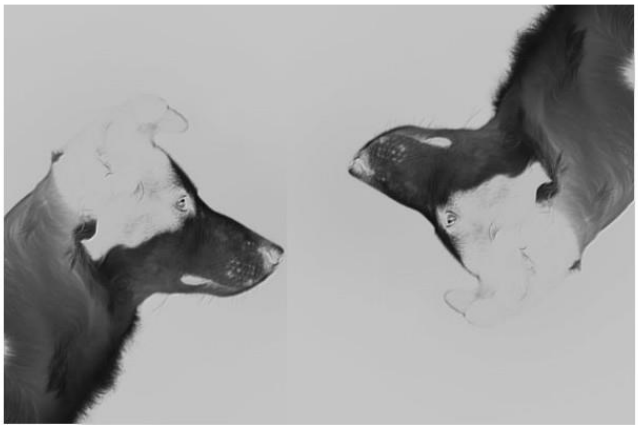




קשה לראות הבדלים בין התמונות אך אם נסתכל טוב נראה שעבור k גבוה יותר התמונה יותר מטושטשת וזה הגיוני מכיוון שאנחנו סוכמים פיקסלים יותר רחוקים, וגם עבור sigma גבוהה התמונה נראית מטושטשת יותר וזה הגיוני מכיוון שאנחנו נותנים יותר משקל לפיקסלים שמסביב.

2.3.3

```
% 2.3.3
sub = dog - gauss1;
figure;
imshow(sub, [])
```



כאן ניתן לראות את החיסור של התמונה המקורית מהתמונה המפולטרת. ניתן לראות שהקיבלנו מאיין inverse של התמונה המקורית

2.4 Noise Filtering

2.4.1

בסעיף זה הוספנו 2 סוגים של רעשים לתמונה של הכלב

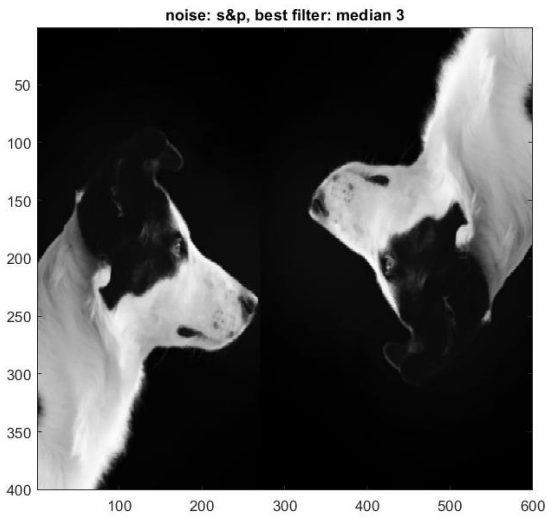
```
% 2.4 Noise Filtering
noised1 = imnoise(dog, "salt & pepper");
noised2 = imnoise(dog, "gaussian");
```

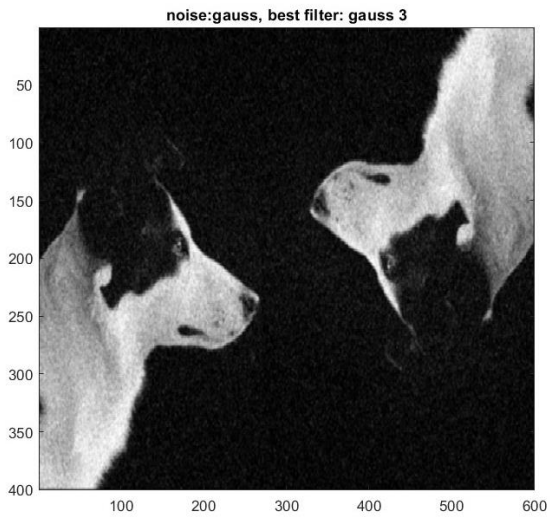
2.4.2

בסעיף זה פילטרנו כל אחת מהתמונות המורעשות באמצעות כל אחד מהפילטרים: mean, median, gaussian עבור k=3,9

לאחר שהסתכלנו על כל התמונות ראינו כי הפילטרים שעשו את העבודה הכי טובה הם ה median פילטר עם k=3 עבור הרעש salt&pepper, והפילטר הגאוסי עם k=3 עבור התמונה עם הרעש הגאוסי.

```
% 2.4.1
noised1mean3 = mean_filter(noised1, 3);
noised1mean9 = mean_filter(noised1, 9);
noised1med3 = medfilt2(noised1,[3,3]);
noised1med9 = medfilt2(noised1,[9,9]);
noised1gauss3 = dip_gaussian_filter(noised1, 3 ,1);
noised1gauss9 = dip_gaussian_filter(noised1,9 ,1);

noised2mean3 = mean_filter(noised2, 3);
noised2mean9 = mean_filter(noised2, 9);
noised2med3 = medfilt2(noised2,[3,3]);
noised2med9 = medfilt2(noised2,[9,9]);
noised2gauss3 = dip_gaussian_filter(noised2, 3 ,1);
noised2gauss9 = dip_gaussian_filter(noised2,9 ,1);

figure;
subplot(1,2,1);
colormap("gray");
imagesc(noised1med3);
title("noise: s&p, best filter: median 3");
subplot(1,2,2);
imagesc(noised2gauss3);
title("noise:gauss, best filter: gauss 3");
```





2.4.3

פילטר ממוצע:
השפעה על רעש s&p: מטשטש את התמונה, עבור k=3 עדיין ניתן לראות בבירור את הרעש, עבור k=9 לא ניתן לראות את הרעש אך התמונה מטושטשת לכן בחירת k תלויה במטרה שלנו.
השפעה על רעש גאוסי: עבור k=3 עדיין ניתן להבחין ברעש ועבור k=9 הרעש סונן אך שוב התמונה מטושטשת.

פילטר median:
השפעה על רעש s&p: לוקח את הערך החציוני ולכן לדעתנו הוא מושלם לרעש salt&pepper מכיוון שרוב התמונה אינה מורעשת ובכך שהוא לוקח את הערך החציוני הוא כמעט אף פעם לא ייקח את הערך של הרעש. עבור k=3 התוצאה מעולה. k=9 זאת בחירה גדולה מידי ולא הכרחית וכבר פוגעת באיכות התמונה.
השפעה על רעש גאוסי: התוצאה עבור k=3 לא כל-כך מעלימה את הרעש, ועבור k=9 אנחנו מצליחים להעלים את הרעש בצורה טובה אך שוב איכות התמונה נפגעת.

פילטר גאוסי:
השפעה על רעש s&p: הפילטר הגאוסי הוא הכי פחות טוב עבור הרעש הזה
השפעה על רעש גאוסי: קשה להחליט איזה k יותר מתאים לנו מכיוון שעבור שתי האופציות קיבלנו תוצאות טובות, גם כאן יש טריידאוף בין איכות התמונה לרמת העלמת הרעש, במקרה הזה בחרנו ב k=3 כי התמונה נראתה לנו טוב יותר.