מבוא לראייה ממוחשבת (22928) – ממ"ן 11

מגיש: דותן אסלמן 301372975

<u>שאלה 1</u>

מצ"ב קוד לשאלה 1 בקובץ question1.py

a. *צור מטריצה בגודל 100x100 שכל איבר בה מתפלג גאוסיאנית עם ממוצע 3 וסטיית תקן 2. הצג את המטריצה כתמונת רמות אפור דו מימדית*

על מנת לייצר את המטריצה השתמשתי בספריית numpy וייצרתי מטריצה עם מספרים רדנומליים בהתפלגות נורמלית. על מנת להציג את המטריצה בתור תמונה השתמשתי בopencv ובפרט בפונקציה imshow. כיוון שהמטריצה היא בייצוג שברי (float) הפונקציה imshow התיחסה למיפוי של 0 – שחור ו1- לבן, כך שכל האיברים השליליים במטריצה הוצגו בתור שחור מוחלט (נמצאים בקיטעון) וכל הערכים הגדולים מ1 יוצגו בתור לבן מוחלט (נמצאים ברוויה) להלן התוצאה :

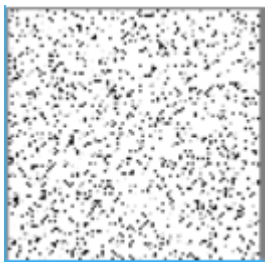


Figure 1 -תצוגת "תמונת רמות אפור" למטריצה המוגדרת בסעיף א

b. *מיין את איברי המטריצה וצייר את הוקטור המתקבל*

לשם כך שוב השתמשנו בnumpy לבצע את השיטוח של המטריצה (flatten) וכן את המיון. לאחר מכן בספריית matplotlib על מנת להציג את התוצאה. להלן התוצאה :

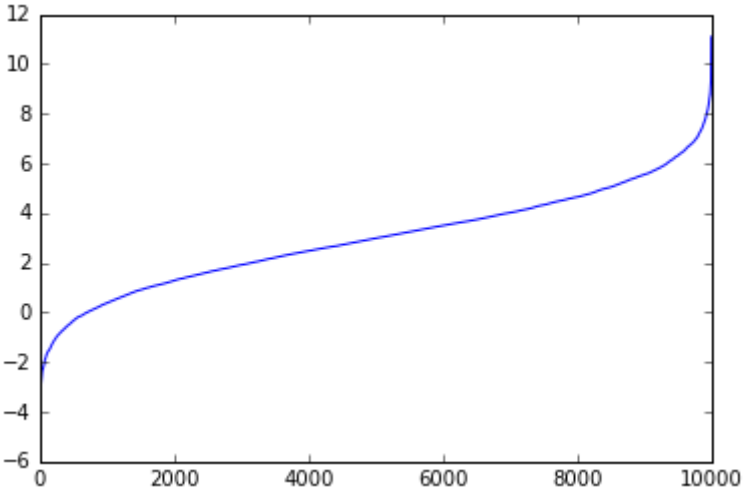


Figure 2 - plot של ערכי המטריצה מסעיף a בהתפלגות נורמלית לאחרי מיון

c. *צייר את ההיסטוגרמה של המטריצה, מחולקת ל32 בינים והשווה לפונקציית הפילוג המתאימה*

ההיסטוגרמה של ערכי המטריצה מבטאת את התפלגות הערכים של המטריצה. כיוון שהמטריצה נוצרה ע"י הגרלה בהתפלגות נורמלית עם ממוצע 3 וסטיית תקן 2, מצופה מההיסטוגרמה להתאים לפונקציית ההתפלגות N(3,2). להלן הצגה של ההיסטוגרמה לצד פונק' ההתפלגות. יש לציין שערכי ההיסטוגרמה נורמלו כך שסכום ערכי הבינים יהיה שווה ל1 (בהתאם להתפלגות)

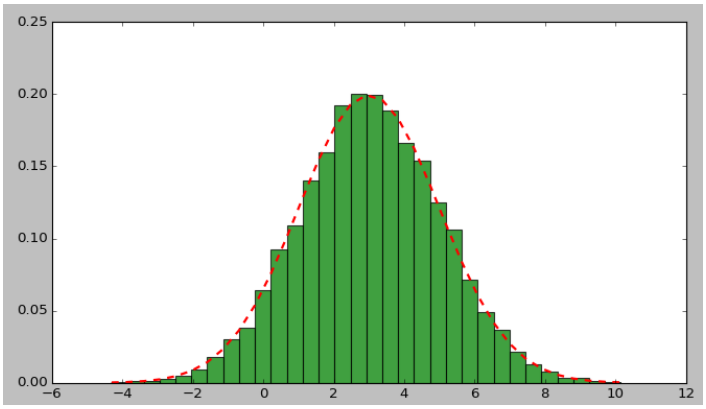


Figure 3 - בירוק, היסטוגרמת המטריצה לאחר כימות. באדום פונק' התפלגות נורמלית עם ממוצע 3 וסטיית תקן2

d. *כמת את איברי המטריצה ל256 רמות באופן אחיד, כאשר הערך הנמוך ביותר ימופה ל0 והגבוה ביותר ל255. הסבר את אופן המימוש. הצג את ההיסטורגמה של התמונה לאחר הכימות.*

הנוסחה שהשתמשתי בה על מנת 'למתוח' את היסטורגמת התמונה היא :

$$g(x,y) = \frac{(f(x,y) - fmin)}{fmax - fmin} * 255$$

כאשר f – תמונת המקור, g– תמונת התוצאה. Fmin,fmax הערך המינימלי והמקסימלי בתמונת המקור(בהתאמה).

הרעיון מאחורי הנוסחה הוא לנרמל את ערכי המטריצה להיות בין 0 ל1 (לחסר במינימום, מה שיביא את המינימום ל0 ולחלק במקסימום החדש מה שיביא את המקסימום ל1) ולאחר מכן להכפיל במקסימום הנדרש שהוא 255.

לאחר מכן נעשה עיגול למספרים שלמים ע"י המרה מfloat ל uint8 באמצעות פונקציית המרה של numpy.

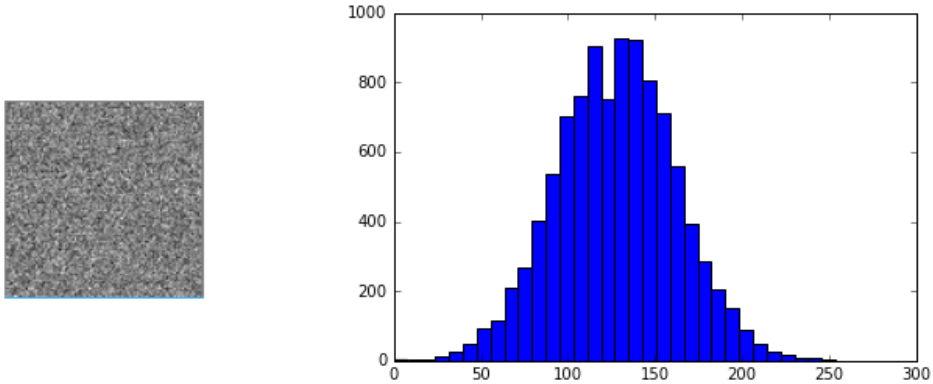


Figure 4 -משמאל, התמונה לאחר מתיחה והמרה לuint8, ימין - היסטוגרמת התמונה (32 בינים)

e. *מה הממוצע וסטיית התקן של האיברי המטריצה לאחר הכימות?*

מן הסתם משתנה בגלל האופי האקראי של המטריצה אבל באחת ההרצות התקבל :
Mean: 136.5948
Std: 33.472926567

f. *קרא תמונה צבעונית כלשהי מהזכרון והצג אותה כתמונה צבעונית וכתמונת רמות אפור.*

מצאתי 2 דרכים לקרוא את התמונה, באמצעות opencv או באמצעות הספרייה matplotlib. בסופו של דבר החלטתי להשתמש בmatplotlib. ההבדל העיקרי בניהם הוא ש opencv קורא את התמונה בפורמט BGR וmatplotlib ב RGB. את ההמרה עשיתי באמצעות פונקציית opencv בשם cvtColor וההצגה באמצעות imshow של matplotlib (גם במקרה הזה יש פונקציה מקבילה בOpencv ולא היה ברור לי מי עדיפה)

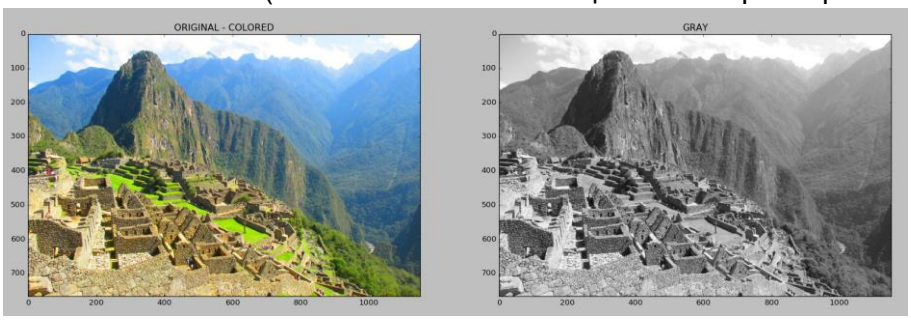


Figure 5 - שמאל - התמונה המקורית. ימין- התמונה ברמות אפור

g. *חתוך אזור מסויים מהתמונה*

בחרתי אזור אקראי וביצעתי את הcrop ללא פונקציה אלא באמצעות שימוש בכלים על האינדקסים של מערכים בפייתון :

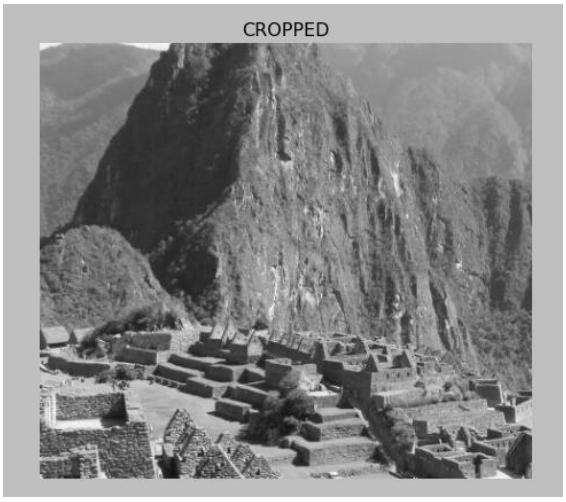


Figure 6 - האזור ה'חתוך' בתמונה

h. *הוסף רעש גאוסיאני עם ממוצע 0 ושונות 3 לתמונה*

יצרתי מטריצת 'רעש' בגודל של התמונה עם התפלגות נורמלית ממוצע 0 וסטיית תקן **1.732** ו'חיברתי' אותה לתמונה :

$$g(x,y) = f(x,y) + noise(x,y)$$

כאשר -noise מטריצת הרעש, f- תמונת המקור. ו g – תמונת התוצאה (תמונה + רעש)

לאחר החיבור השתמשתי שוב בפונקציית Uint8 של numpy על מנת לחזור לייצוג בשלמים (בגלל שהרעש היה float אז התוצאה גם היא יצא בfloat). להלן התוצאה המורעשת :

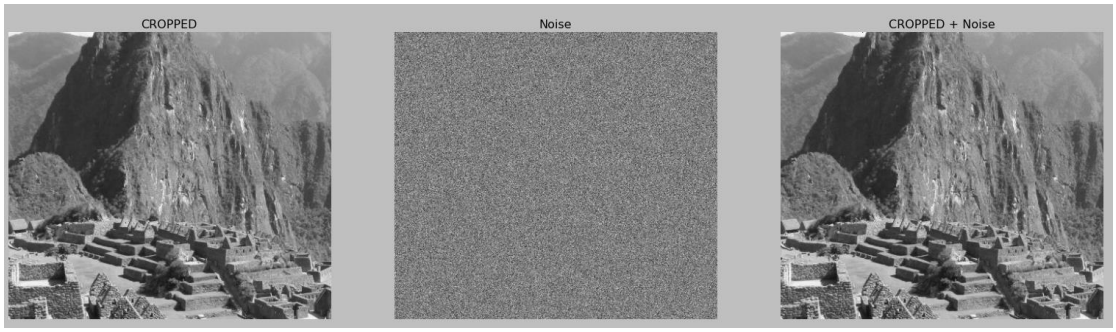


Figure 7 משמאל לימין : תמונת המקור, הרעש, והתוצאה. הרעש 'נורמל' (נמתח) לערכים 0 עד 1 לצורך המחשה(אחרת היה בלתי נראה)

ניתן לראות שהרעש מאד חלש וכמעט ולא משפיע על התמונה (במקרה קיצון הרעש מוריד או מעלה עד 8 רמות אפור לפיקסל.

i. *הפעל פילטר גאוסיאני להחלקת התמונה, מהם הפרמטרים של הפילטר הנותנים תוצאה אופטימלית.*

על מנת להריץ את פעולת המריחה השתמשתי בפונקציה GaussianBlur של opencv.

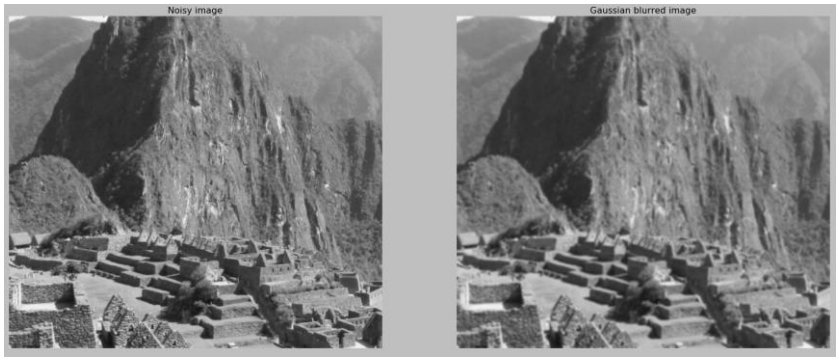
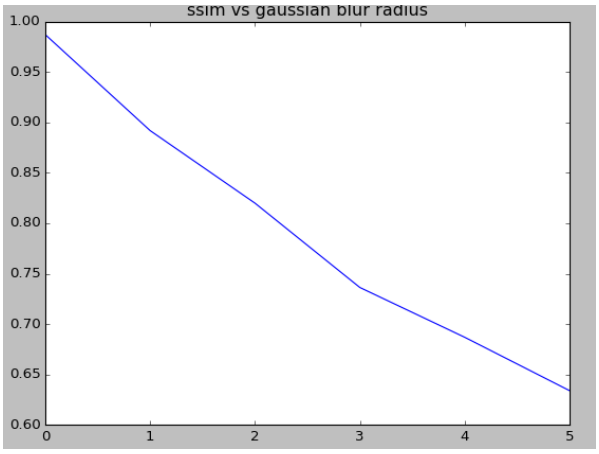


Figure 8 שמאל - התמונה לאחר הוספת הרעש, ימין - התמונה לאחר מריחה גאוסיאנית בחלון של 3על3 וסטיית תקן 2

קשה היה לי להגדיר מה בדיוק הכוונה ל"אופטימלי" לאיכות תמונה שזה מושג סובייקטיבי. ועל מנת לנסות בכל זאת לבצע בדיקה 'אוביקטיבית' החלטתי להשתמשבמדד structural similarity(ssim) על מנת לכמת את מידת הקירבה של התמונה לאחר המריחה לתמונה המקורית לפני הוספת הרעש. בחנתי קודם לכן שיטות בסיסיות יותר כגון MSE וכן PSNR אך זו היתה נראית לי מתאימה ביותר. את השיטה לא מימשתי בעצמי והתבססתי על ספריית skimage.

המסקנה מההרצה היא שלפי SSIM הכי טוב לא למרוח בכלל כיוון שכל מריחה שהיא תפגע בתוצאת ההתאמה ביחס להשארת הרעש כמו שהוא. הרצתי קרנלים בגודל של 3על3 עד 11על11 והמגמה ברורה. להלן גרף של ערך הssim כתלות ברדיוס של הקרנל (רדיוס 0 זהה ל-לא לבצע מריחה כלל)



לצערי (למרות שנראה שזו לא כוונת התרגיל) המסקנה המתבקשת שלי היא או שעשיתי איזשהי טעות או שהדבר נכון ביותר הוא לא לבצע מריחה על מנת לתקן את הרעש. כמובן ברור שמריחה פוגעת באיכות התמונה ובמיוחד בקצוות לכן יש סיכוי באמת שאפקט המריחה פוגע יותר ממה שהוא מועיל לרעש הלבן. לפחות במקרה שלי נראה שזה המצב.

שאלה 2

1. *בניית הmosaic : צור תמונת mosaic ע"פ תבנית הbayer הנתונה מתמונה צבעונית שצילמת בעצמך.*

הקוד לסעיף זה נמצא בקובץ question2.py בפונקציה rgb2bayer

על מנת ליצר תבנית bayer מתמונת הצבע, דגמתי כל צבע בדילוגים המתאימים והרכבתי תמונת 'שחור לבן' על פי התבנית. להלן התוצאה:

Figure 9 - שמאל : התמונה המקורית. ימין : התמונה בתצורתbayer . קשה לראות ברזולוציה הזו אבל יש תבנית משבצות

2. *אינטרפולציה לינארית : בצע אינטרפולציה לשחזור התמונה והצג את השגיאה*

הקוד לסעיף זה נמצא בקובץ question2.py בפונקציה demosaic_blinear

כדי לבצע את האינטרפולציה ראשית פירקתי את תמונת הmosaic שיצרתי בסעיף 1 ל3 תמונות: blueMask, greenMask, redMask, אחד עבור כל ערוץ צבע, כאשר כל התמונות ברזולוציה של התמונה המקורית אך למשל במקרה של blueMask, בכל מקום שהיה ערך כחול בתמונת הmosaic הערך יופיע באותו מקום גם בblueMask אך בכל מקום שהיה בmosaic ערך של צבע אחר בblueMask יופיע 0. וכנ"ל כך לגבי שאר הצבעים. את אותם "חורים" (אזורים בהם יש ערך 0) אני משלים באמצעות קונבולוציה בשימוש בפונקציה cv2.filter2d.

לצבע הירוק השתמשת בקרנל :

$$\begin{pmatrix} 0 & 1 & 0 \\ 1 & 4 & 1 \\ 0 & 1 & 0 \end{pmatrix}/4$$

כאשר הרציונל הוא שאם יש כבר ערך ירוק במרכז הקרנל אז הערך שלו יילקח (מהגדרת התבנית זה גם אומר שב4-שכנים שלו יהיה 0 במסיכה הירוקה) ואם אין אז יילקח הממוצע של ארבעת השכנים הירוקים שגם הם מהגדרת התבנית צריכים להיות בעלי ערך.

ולצבעים אדום וכחול השתמשתי באותו הקרנל אשר נראה כך:

$$\begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix}/4$$

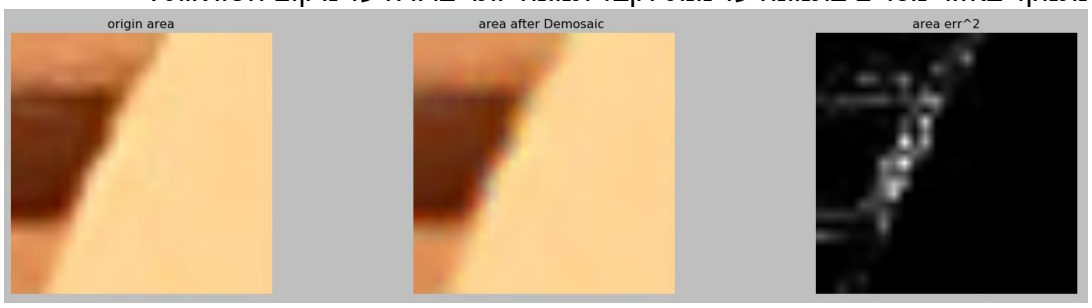
הרציונל מאחורי זה הוא שלצבע הכחול/אדום או שיש 4 שכנים באלכסון או שיש 2 שכנים למעלה-למטה או שיש 2 שכנים ימין ושמאל כל זה בהתאם לשורה שהם נמצאים אך אף פעם לא גם וגם. ה4 באמצע בדומה לירוק- איפה שכבר יש ערך של כחול/אדום מובטח שכל ה8-שכנים יהיו 0 ולכן הוא יילקח בשלמותו בלי הפרעות.

את תמונת השגיאה חישבתי ע"י החסרה בין תמונת המקור לתמונת התוצאה בריבוע (כלומר, לקחתי את ריבועי ההפרשים) ולאחר מכן על מנת לקבל תמונת רמות אפור סכמתי את כל מרכיבי הצבע של כל פיקסל וכך קיבלתי את עוצמת ההפרש הכוללת

*Figure 10 - משמאל לימין : תמונת המקור, התמונה לאחר demosaic , תמונת **שורש** ריבועי ההפרשים*

ניתן לראות בבירור(במסמך פחות) שהמקומות בהם התקבלה שגיאה גדולה יחסית הינם קצוות בתמונה, כדי להבליט את השגיאה לקחתי שורש של התוצאה אחרת כמעט הכל היה שחור ולא היה ניתן להבחין. התופעה "הגיונית" כיוון שאזורי קצוות מאופיינים בשינוי מהיר של עוצמות הצבע אך בגלל שיש לנו בתמונת bayer בין חצי לרבע מהרזולוציה והשחזור שלנו מתבצע על בסיס מיצוע שכנים (משהו שמתבסס על הנחת חלקות בתמונה) קצוות הם בדיוק המקומות שהשחזור 'נופל' בהם ויוצר אפקט טשטוש בדומה למה שקורה כשמריצים box filter וכדומה.

נתמקד באזור מסויים בתמונה על מנת לקבל תמונה יותר ברורה על מיקום השגיאות :



אפשר גם לראות שבתמונת השחזור, במקומות שבהם ישנה שגיאה גדולה גם יש מעין צבע 'מלאכותי' . כמו כן גם פה ניתן לראות בבירור שהשגיאה נמצאת באזורי הקצוות בתמונה.

נוסף על תמונת השגיאה חישבתי גם MSE, VARIANCE, MAX ERROR בשניהם לא הייתי בטוח האם לחשב את הערך הTOTAL שהתבסס על סכימה של 3 ערוצי הצבע או לבצע את החישוב על בסיס תמונת הצבע, לשתי הגישות יש היגיון – לבצע את החישוב על הערך הסופי לאחר סכימת ערוצי הצבע מבטא יותר טוב שגיאות בפיקסלים שיש בהם שגיאה גדולה ביותר מערוץ צבע אחד, מדד שנעלם במדידת שגיאה על בסיס תמונת הצבע שמתייחסת לכל ערוץ באופן בלתי תלוי. לכיוון ההפוך ראיתי שרוב הספריות המקובלות בPython וmatlab לחישוב mse פועלות ברמת תמונת הצבע ולא מבצעות סכימה. למען השלמות החלטתי לשים פה את 2 התוצאות:

	MSE	VAR	MAX ERR
TOTAL	86.985	64.6052	137.29
COLORED	28.995	24.89	123.0

3. *שיטת פרימן : ממש את האלגוריתם המשופר. מצא k מתאים והצג גרף של mse כתלות בk*

הקוד לסעיף זה נמצא בקובץ question2.py בפונקציות demosaic_freeman וכן ב create_freeman_report

להלן תוצאות האלג' :

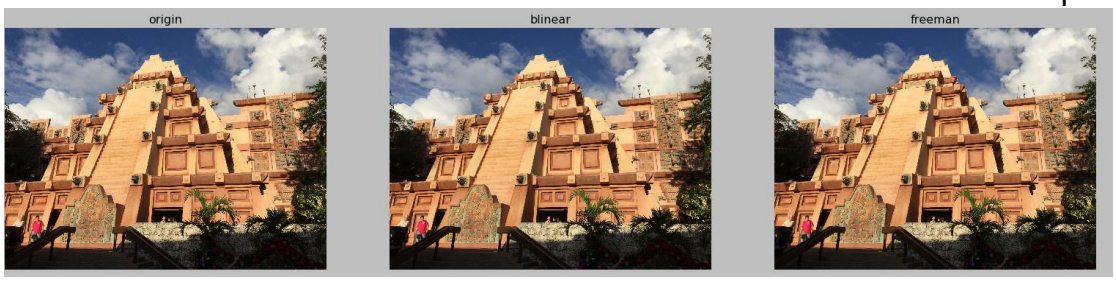
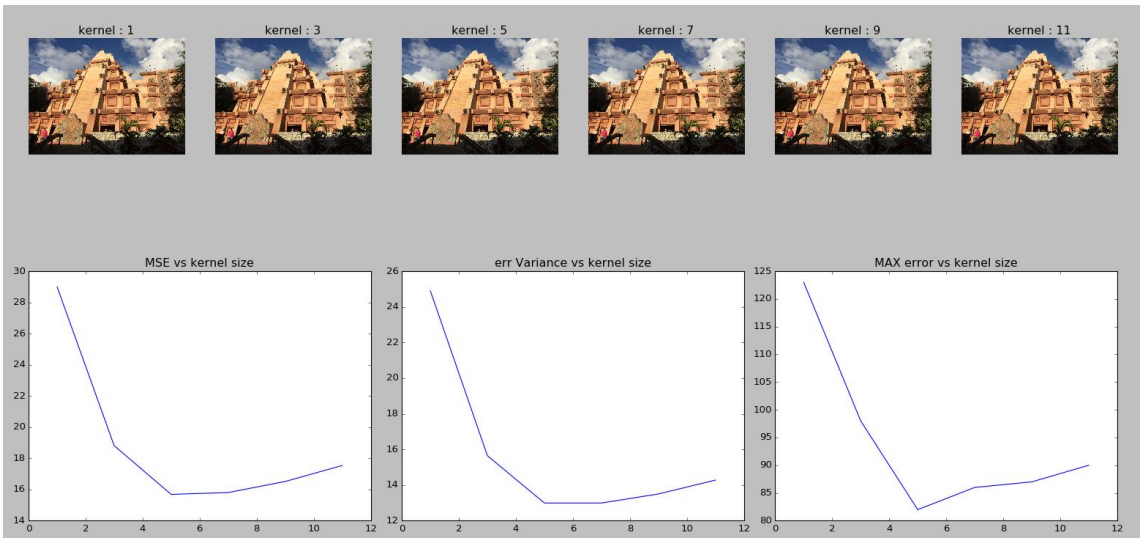


Figure 11- משמאל לימין : תמונת המקור, תמונה לאחר demosaic לינארי ללא תיקון פרימן, תמונה לאחר demosaic לינארי עם תיקון פרימן

ניתן לראות (בקושי) שהתמונה חדה יותר באמצעות שיטת פרימן.
הסיבה לmedian filter היא כדי לתקן אזורים בהם שינויים חדים בתמונה 'נמרחו' כתוצאה מהאינטרפולציה, במצב כזה median ישמר את השינוי החד ע"י קביעת ערכי הביניים להיות באחד מהערכים. למשל אם יש edge שנע בבת אחת מ10 ל 150 , עם אינטרפולציה כנראה שיהיה פיקסל או 2 שיהיו איפשהו באזור ה80 ובעצם 'ימרחו' את הedge , לאחר median הערכים באמצע ייקבעו להיות 10 או 150 וכך חדות הedge תישמר.

הרצתי את האלגוריתם עם פרמטרים שונים לקרנל של median filter כדי למצוא אופטימלי והשוותי MSE, , שונות ושגיאה מקסימלית:



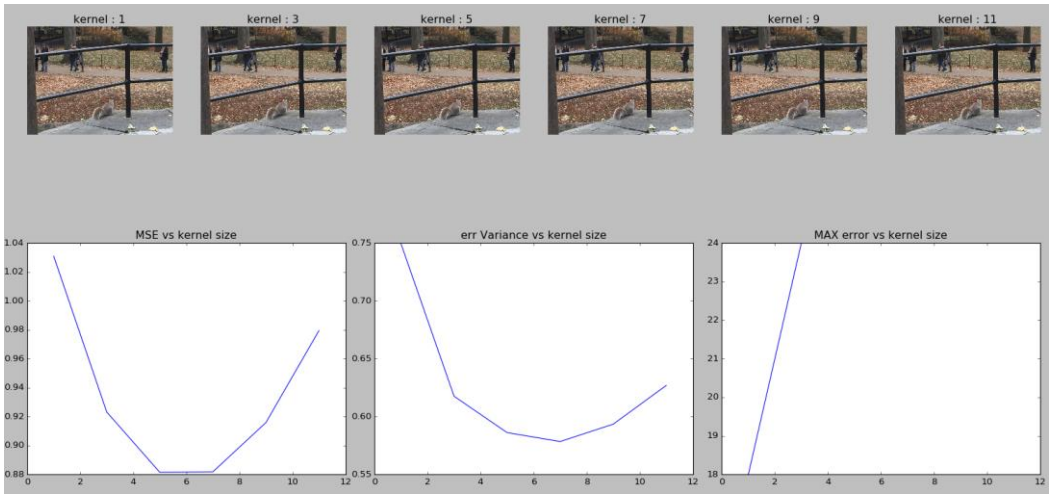
המסקנה שלי מההרצה שkernel בגודל 5x5 נותן את התוצאות הטובות ביותר(עבור תמונה זו)

4. *הרץ את האלגוריתם על תמונות נוספות והשווה תוצאות:*

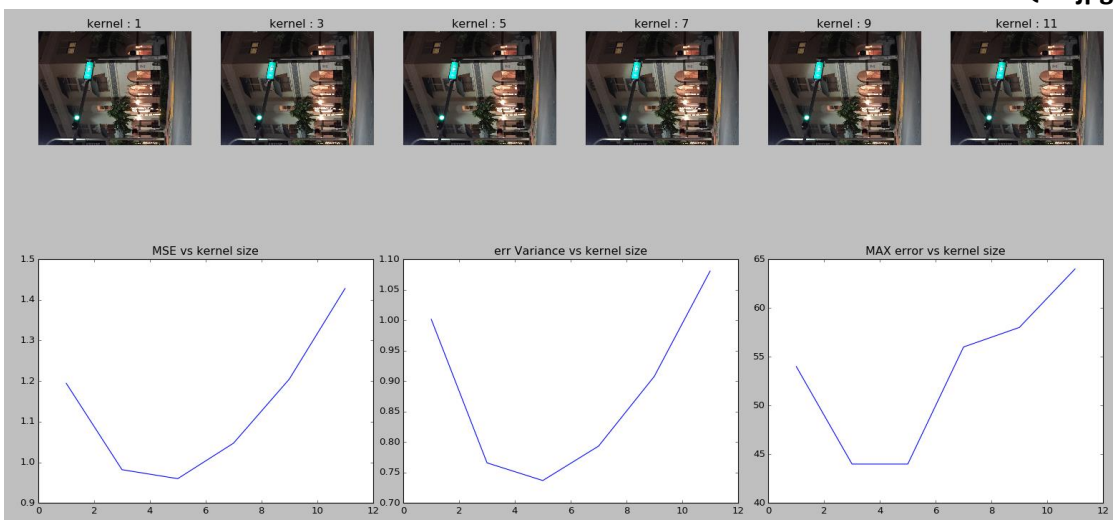
הקוד לסעיף זה נמצא בקובץ question2.py בפונקציה create_freeman_report

בחרתי עוד 3 תמונות ואני אריץ עליהם את ניתוח הגרף מסעיף 3

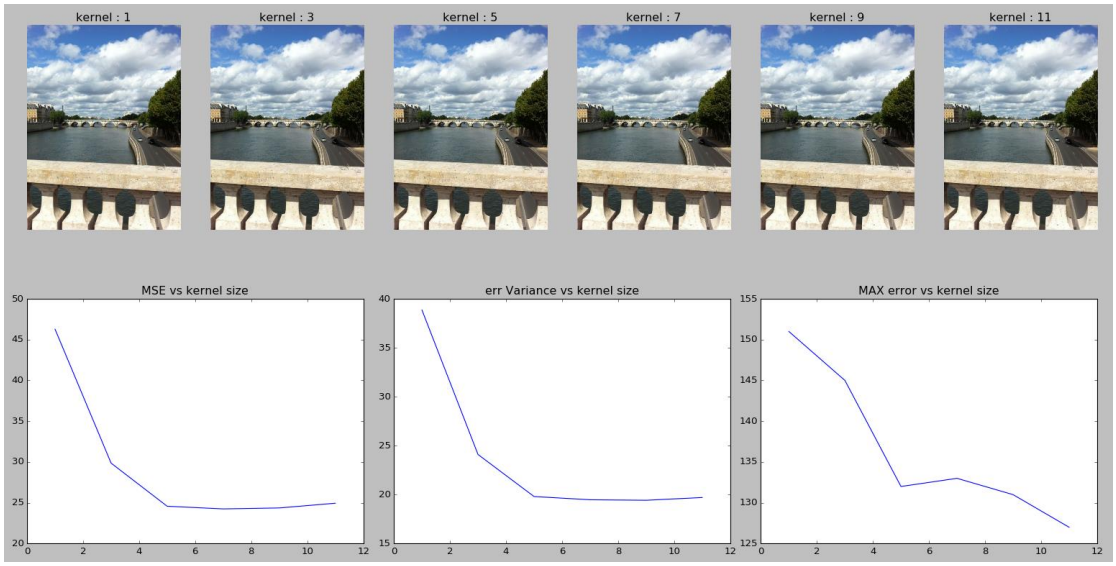
Q4a.jpg:



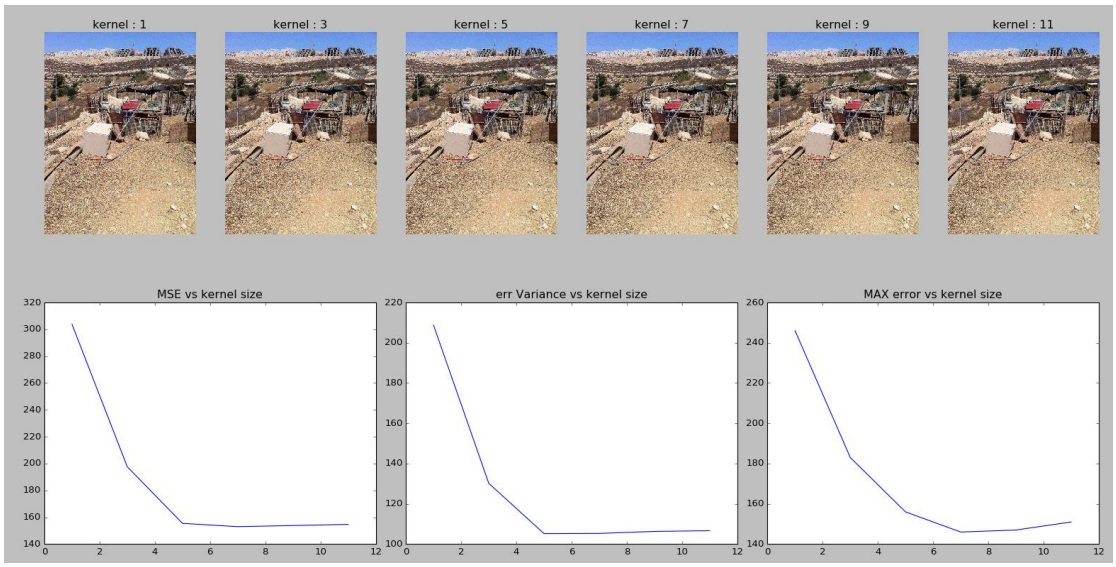
Q4b.jpg:



Q4c.jpg:



:Q4d.jpg



המסקנה שלי מההרצות היא שככל שיש יותר שינויים חדים בתמונה כך שיטת פרימן עוזרת יותר בהיבטי mse , כל עוד משתמשים בה במידה כי מגודל קרנל 6 והלאה מתחילה פגיעה.