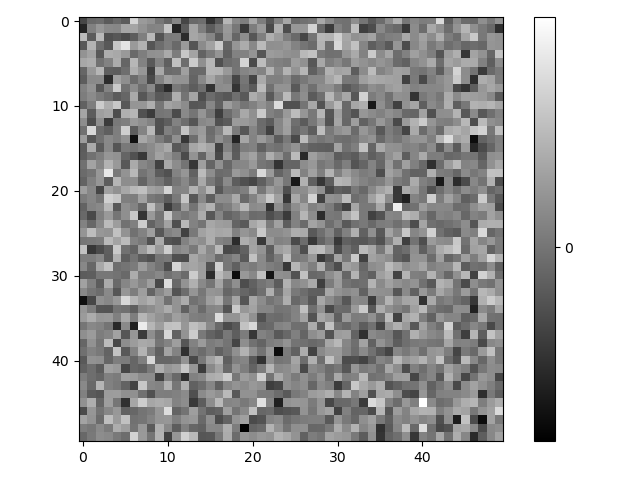
ממן 11 -מבוא לראייה ממוחשבת

סתיו כהן

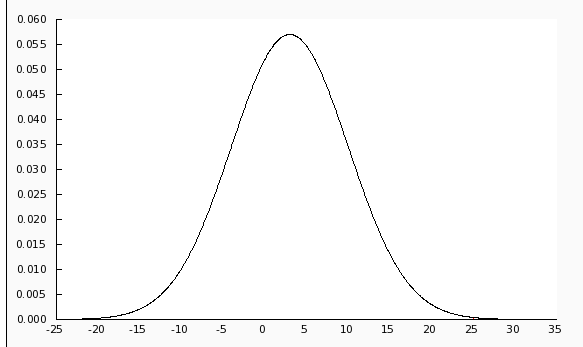
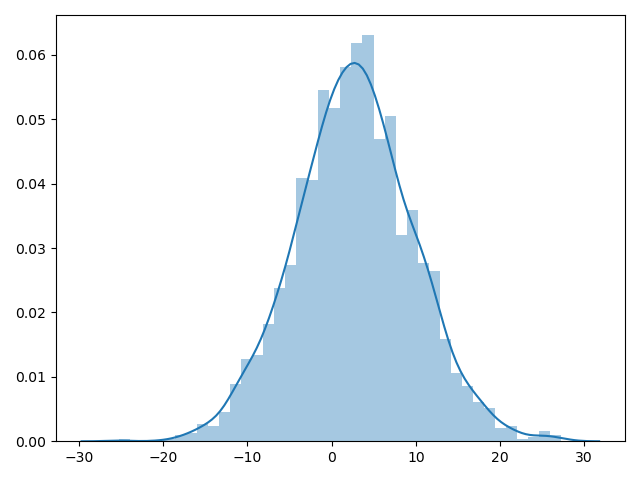


יצרתי מטריצה בגודל 50X50 עם התפלגות גאוסיינית עם ממוצע 3 וסטיית תקן 7 בעזרת שימוש בספרייה Numpy ולאחר מכן הצגתי את המטריצה בעזרת matplotlib כתמונת רמות אפור דו ממדית.





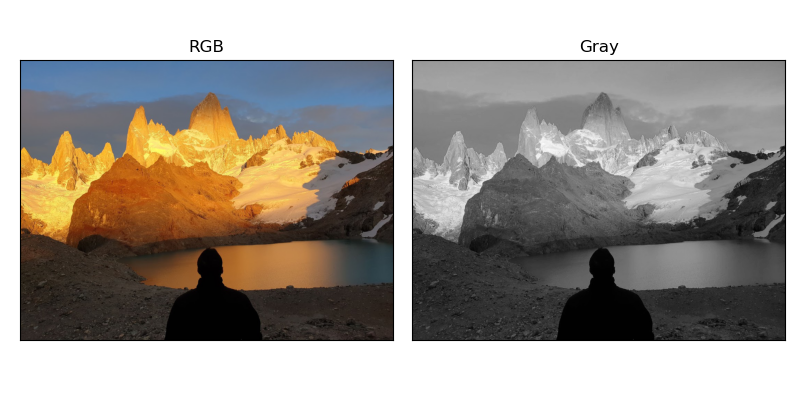
השתמשתי בספרייה Seaborn על מנת לצייר את ההיסטוגרמה של המטריצה(התמונה הימנית):

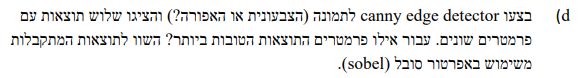


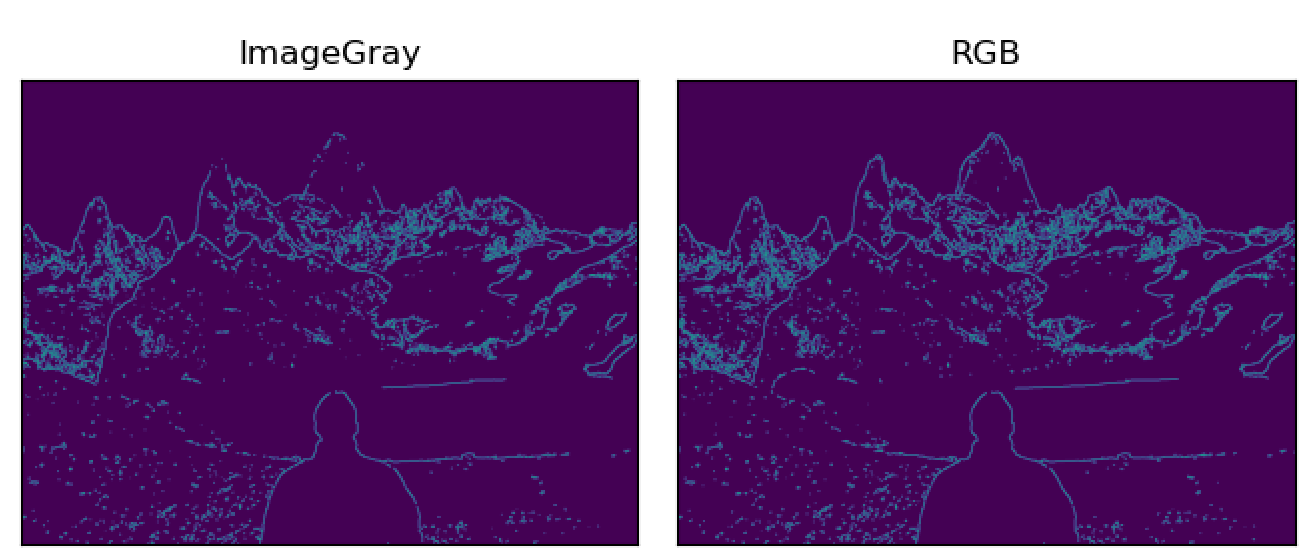
נראה שהיא בהחלט מתאימה לפונקציית הפילוג המתאימה הנורמלית.



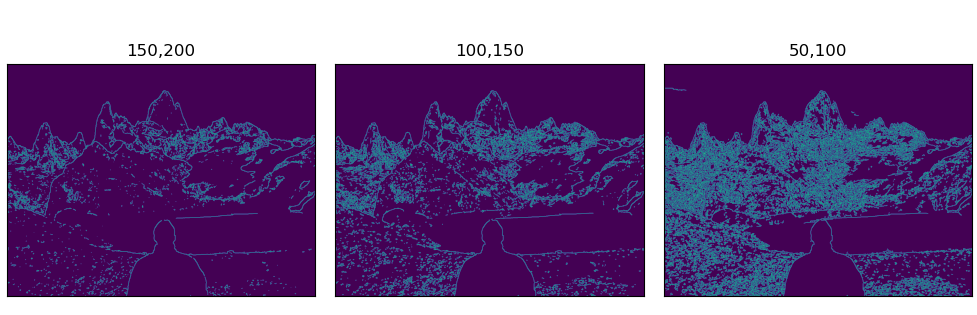
בסעיף זה בחרתי להשתמש בתמונה שלי מארגנטינה וקראתי אותה מהזיכרון באמצעות שימוש בספרייה OpenCv, לאחר מכן הצגתי אותה כתמונת RGB וכתמונה ברמות אפור:





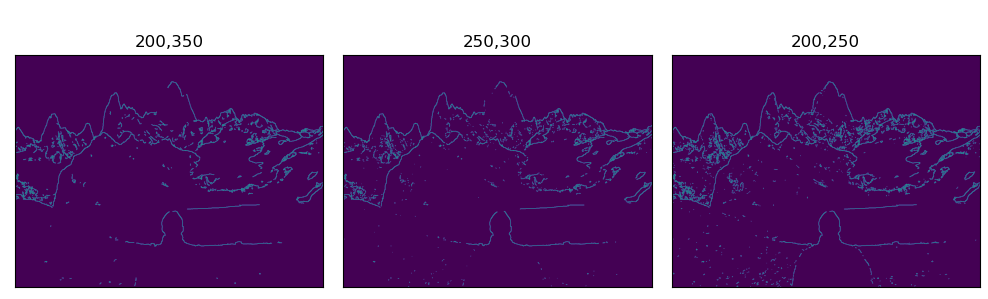
החלטתי להשתמש בפונקציית Canny של openCv בשביל לממש את השיטה והחלטתי לבדוק האם להשתמש בתמונה הצבעונית או להשתמש בתמונה האפורה, לכן השוואתי בין תמונה אפורה לצבעונית עם אותם פרמטרי סף: 

ניתן לראות כי דווקא התמונה הצבעונית זיהתה יותר טוב את ה-Edges (קל לראות יתרון ברור זה בהר המרכזי בתמונה), לכן החלטתי להמשיך לעבוד עם התמונה הצבעונית ולבדוק פרמטרים שונים לדרישות הסף של הפונקצייה Canny:

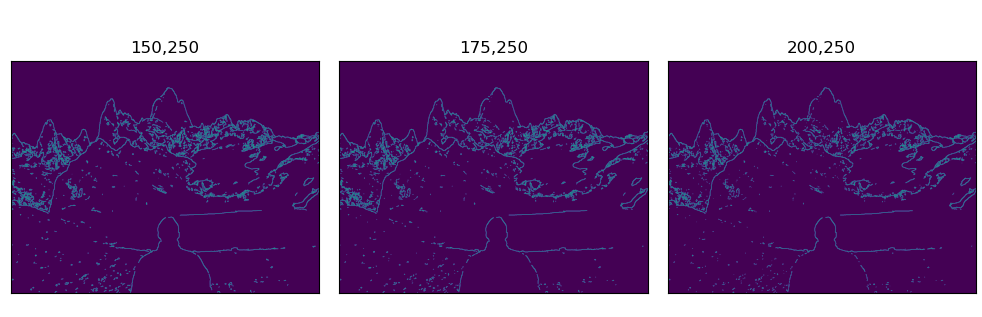


בחרתי את סף המינימום והמקסימום בהדרגה על מנת לקבל תחושה על הפרמטרים המתאימים לפונקצייה, ניתן לראות כי כאשר סף המנימום היה נמוך אז התגלו רעשים רבים בתמונה בנוסף ל-Edges המרכזיים.

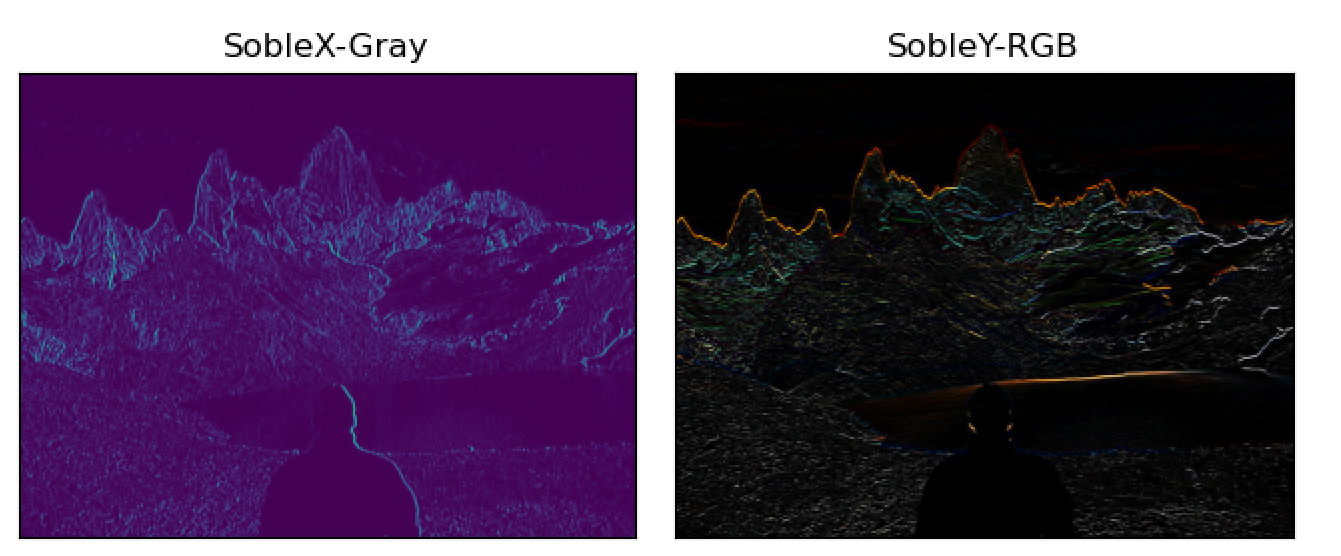
החלטתי להמשיך לבדוק פרמטרים נוספים אשר המינימום והמקסימום שלהם גדולים יותר:



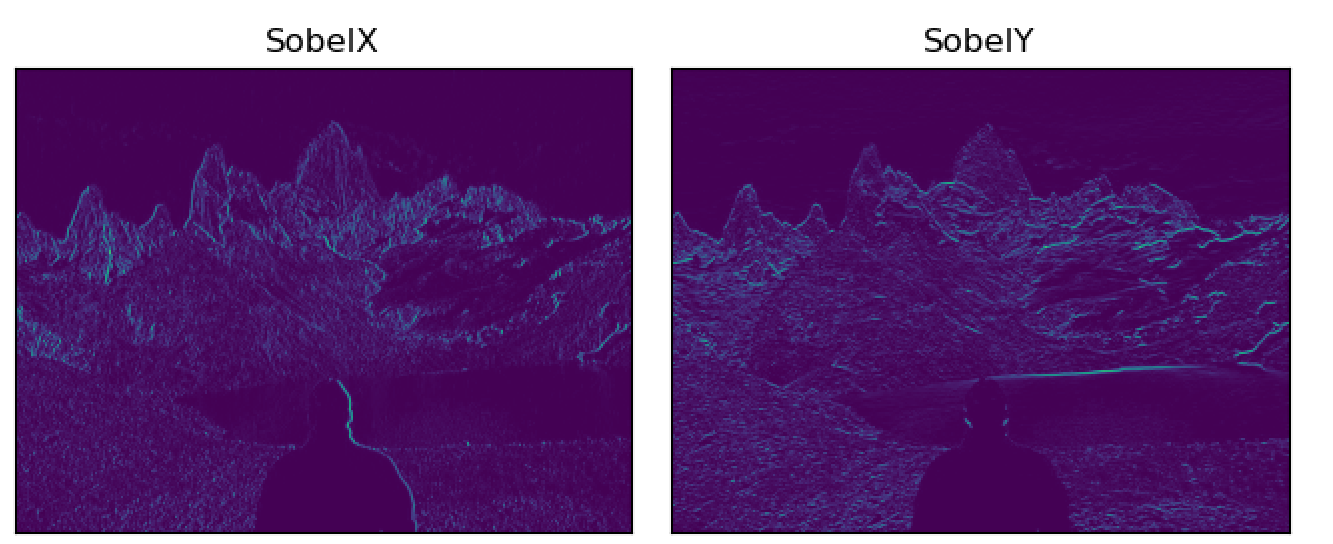
ניתן לראות כי בשני התמונות השמאליות אני מתחיל לאבד מידע על הEdges המרכזיים בתמונה, לבסוף הגעתי לשלוש תמונות אשר אותם ארצה להשוות מול אופרטור Sobel:



ניתן לראות הבדלים קטנים בין תמונות אלו אך לפי דעתי התמונה המוצלחת ביותר היא 200,250 אשר מתארת את ה-Edges המרכזיים עם כמה שפחות רעשים, אתקדם לשלב הבא למרות שנהנתי 😊:

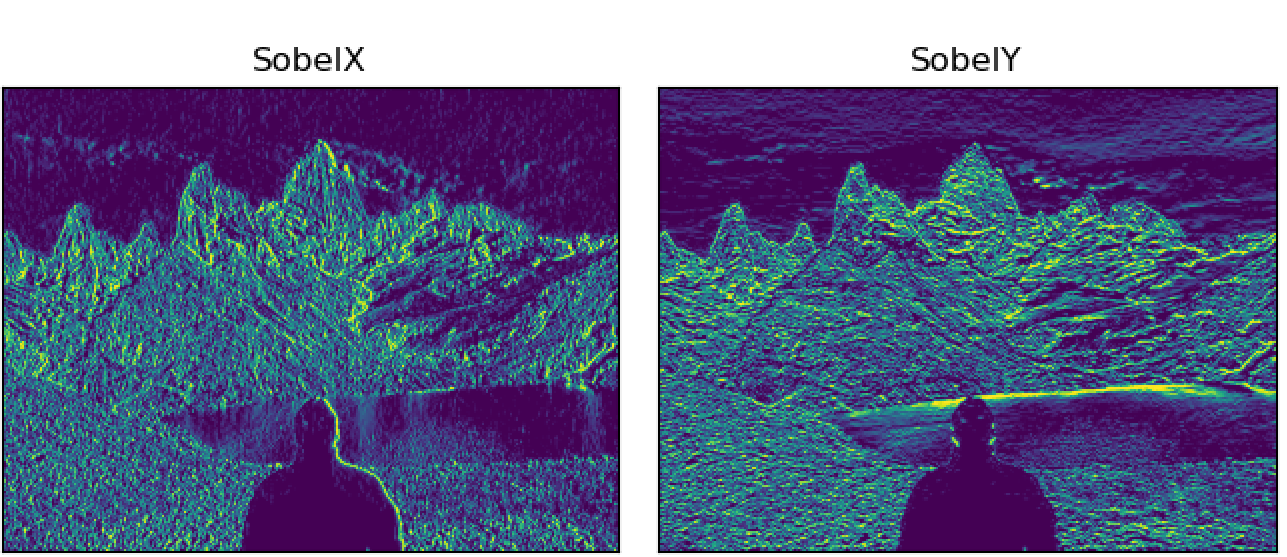
השתמשתי בפונקיצת Sobel על התמונה הצבעונית ועל התמונה העפורה : 

החלטתי להמשיך עם התמונה העפורה כאן על מנת שגווני הצבע יהיו דומים לפונקציית הCanny ,בנוסף השתמשתי בפרמטר Ksize=3 :

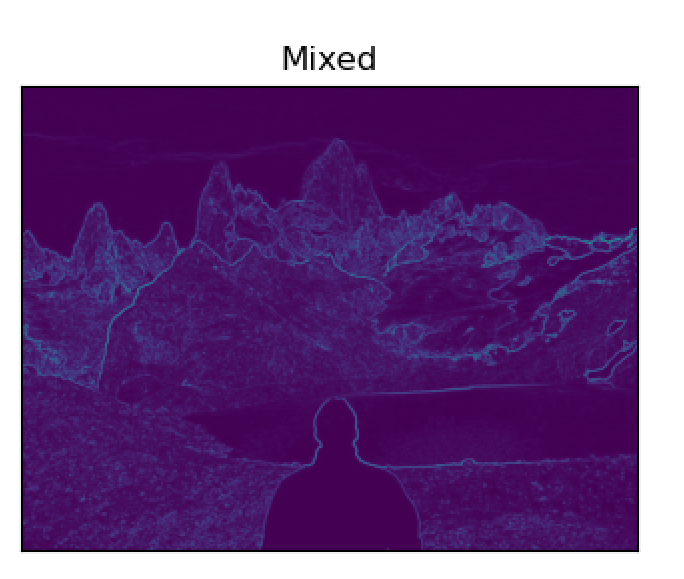
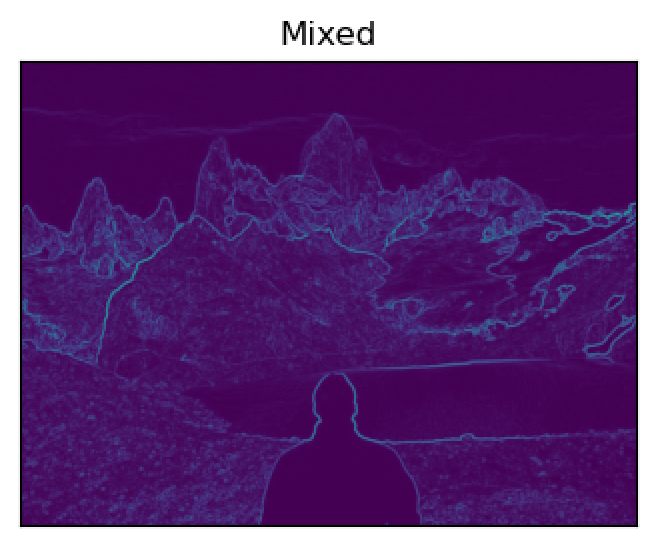


ניתן לראות כי בתמונת SobelX מודגשים לנו ה-Edges בציר הY לעומת התמונה השניה שבה מודגשים דווקא ה-Edges בציר הX, ניתן לראות זאת בבירור בהסתכלות על מבנה ההרים בשני התמונות.

ניסיתי להגדיל את גודל הKSize מ3 ל5 :

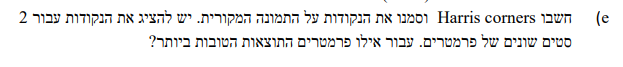


התוצאות דיי איכזבו אותי אך החלטתי לנסות לבצע מיקס בין התמונות האלו כמתואר בהרצאה הראשונה:

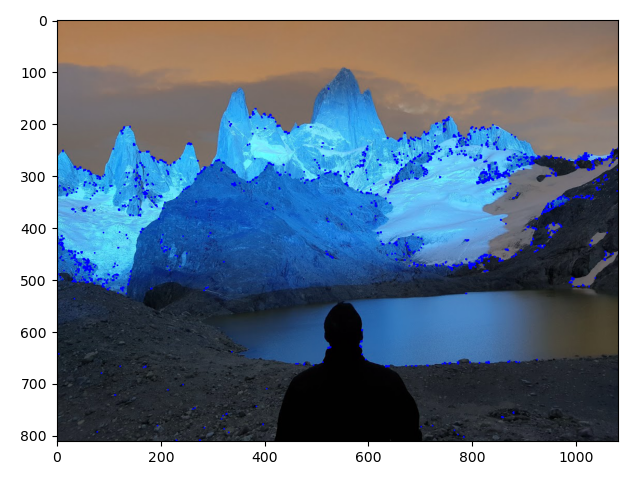


קיבלתי תוצאה דיי מרשימה בשני הנסיונות כאשר התמונה הימנית משתמשת בפרמטר kSize=5 לעומת השמאלית שמשמשת בפרמטר kSize=3.

ניתן לראות כי פונקציית Canny דווקא עבדה יותר טוב על תמונה זאת עם הפרמטרים שניסיתי מכיוון שהיא הצליחה לזהות את ה-Edges בצורה ברורה יותר ללא הרבה רעשי רק לעומת פונקציית Sobel שזיהתה Edges אנכי ואופקיים אך נראה שהייתה פחות חסינה לרעש.

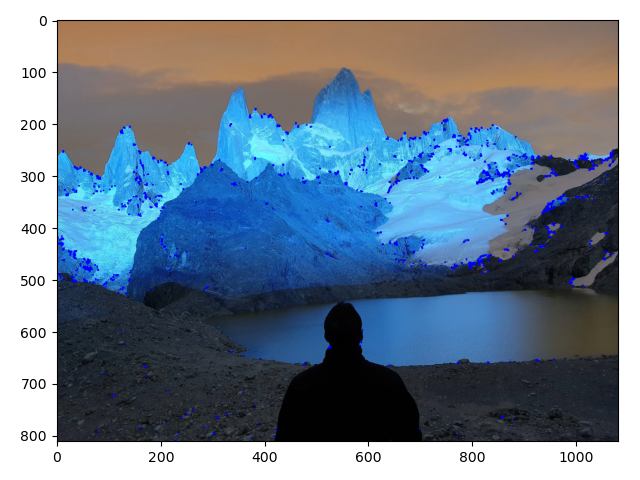
השתמשתי בפונקצית HarrisCorners אשר נמצאת בספריה OpenCv והתחלתי לנסות את הפרמטרים המופיעים בדוקמנטציה של OpenCv : (בסעיף זה אציג את התמונות בגדול על מנת לעזור לראות את סימוני הפינות)

תחילה בדקתי את הפונקציה עם Blocksize=2,kSize=3,k=0.04:

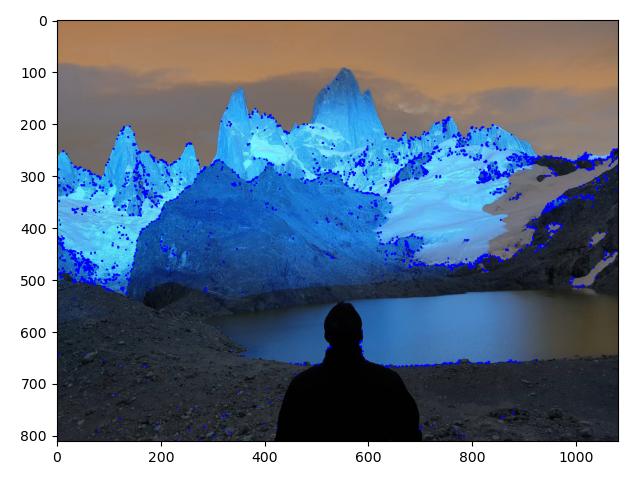


נראה כי בהחלט סומנו כמה פינות המופיעות בתמונה לדוגמא ניתן להסתכל על צורת רכסי ההרים והנקודות המסומנות, ניתן לראות גם כמה נקודות שאינם בהכרח פינות וגם פינות אשר ברורות בתמונה לא סומנו כגון ראשי ההרים המרכזיים בתמונה. אנסה לבדוק סט פרמטרים נוסף.

החלטתי להגדיל את פרמטר הkSize על מנת לנסות להימנע מסימון מקומות שבהכרח לא נראים כפינות לכן בדקתי kSize=7

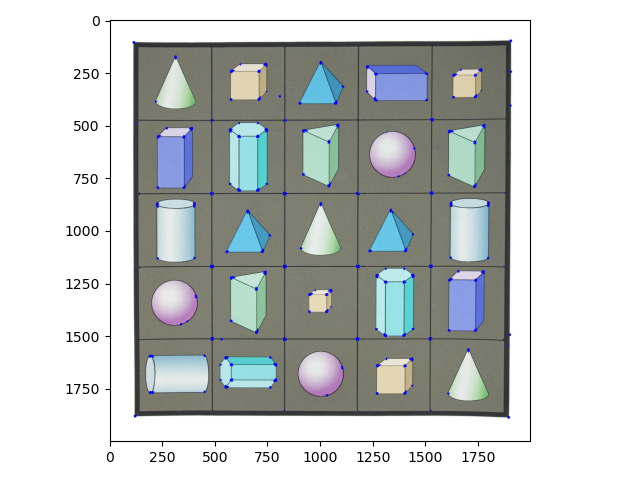


ניתן לראות שחלק מהפינות הלא אמיתיות נעלמו אך בהחלט גם חלק מהפינות האמיתות נעלמו וגם לא הצלחתי לאתר את הפינות בראש רכס ההרים, אנסה בדיקה עם kSize=1 :



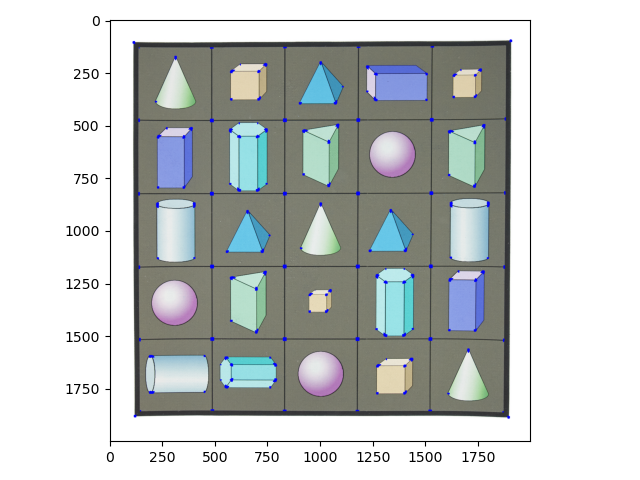
ניתן לראות כי כעת מסומנות עוד הרבה יותר פינות אך עדיין הפינות על ראש ההר לא מסומנות...

לאחר נסיונות חוזרים לא הצלחתי לאתר את הפינות בראש ההר אז אסיק כי סט הפרמטרים הראשון שניסיתי נראה הטוב ביותר לאיתור הפינות בתמונה (Blocksize=2,kSize=3,k=0.04) החלטתי לנסות לבדוק פרמטרים על תמונה נוספת : Blocksize=10,Ksize=3,k=0.04



כאן בהחלט נראה כי מצאנו המון פינות נכונות אך מפתיע שגם מצאנו פינות בצורת עיגול!

ניסיתי לפתור בעיה זו באמצעות הגדלת פרמטר : Blocksize=10,Ksize=11,k=0.04

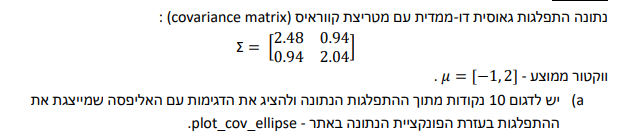


נראה שבהחלט הצלחתי כעת לא לסמן פינות בצורת עיגול ואכן גם לסמן את הפינות בצורות אחרות.

לכן בהחלט סט פרמטרים זה מתאים יותר לתמונה זאת.

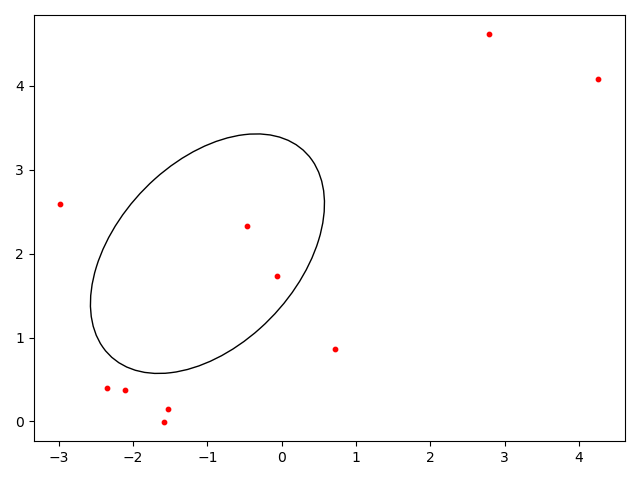
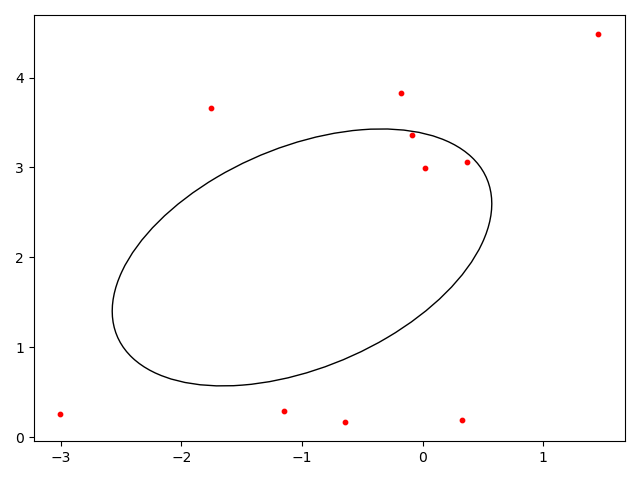
מסעיף זה אסיק כי בעיית מציאת הפינות בתמונה היא אכן לא בעיה קלה ונראה כי נדרש Fine-Tuning לפרמטרים על מנת שנקבל תוצאה טובה! ובקשר לפינת ההר החמקמקה אני עוד אצליח למצוא אותה בעתיד הקורס 😊.

שאלה 2:



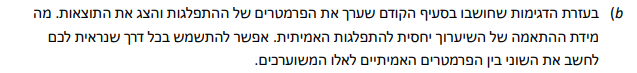
דגמתי 10 נקודות מתוך ההתפלגות בעזרת הפונקציה np.random.multivariate\_normal ולאחר מכן ציירתי את האליפסה בעזרתם ואז הצגתי גם את הנקודות הדגומות.

ביצעתי שינוי בפונקציה Plot\_cov\_ellipse בכך שהיא תחזיר גם את הקורדינטות של הנקודות הדגומות על מנת שאשתמש בהם בהמשך ולא אצור נקודות חדשות כפי שמתבקש בסעיף ב.



לדוגמא ניתן לראות שני סוגים שונים של דגימות של 10 נקודות ואת המיקומים השונים של האליפסה הנוצרת מהן.

אמשיך להציג נתונים המתאימים ל10 נקודות הנדגמו בתמונה הימנית.



בעזרת הדגימות שדגמתי שערכתי את ערכי הפרמטרים של ההתפלגות בעזרת שימוש בספריית Numpy

קיבלתי וקטור ממוצע ומטריצת Cov מתאימים :



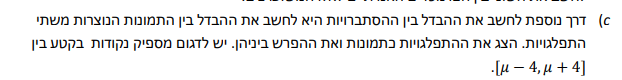
ניתן לראות כי מידת השערוך לא כזו טובה(זה משתנה מאיטרציה לאיטרציה כתלות בדגימות הנלקחות)

חישבתי את מידת השיערוך בין שני ההתפלגויות באמצעות שני טכניקות:

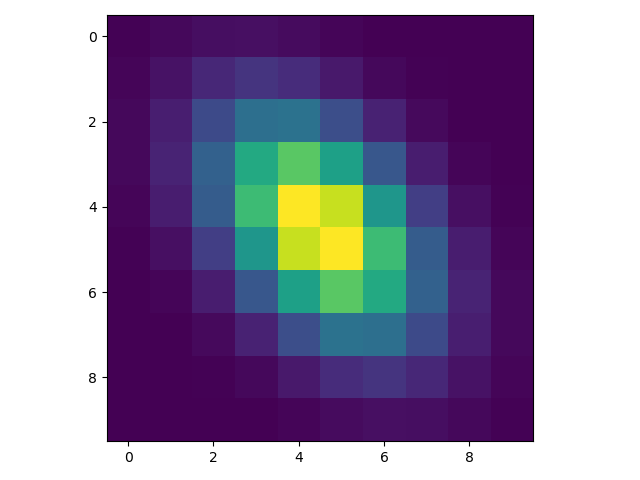
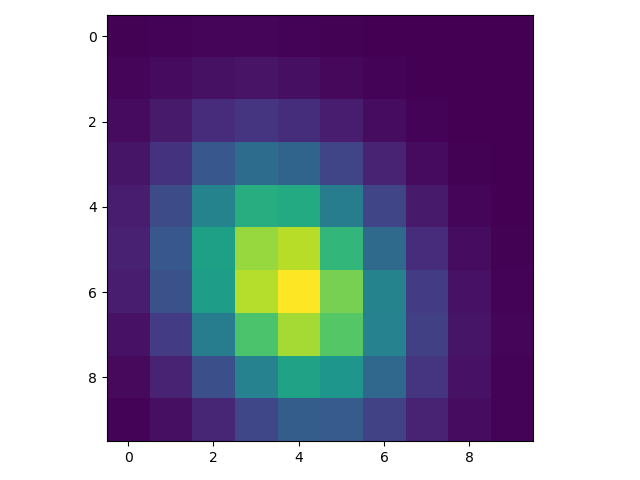
1. Frobenius Norm אשר מצאתי הסבר עליה באינטרנט.
2. Loss אשר הוסברה בשיעור האחרון (10.2)



ניתן לראות כי המרחקים דיי גדולים בין שני ההתפלגויות (בסעיף ד נראה כי דגימות רבות יותר יגרמו לשיערוך טוב יותר- מתחיל להיראות כמו עיקרון המפתח של למידת מכונה)

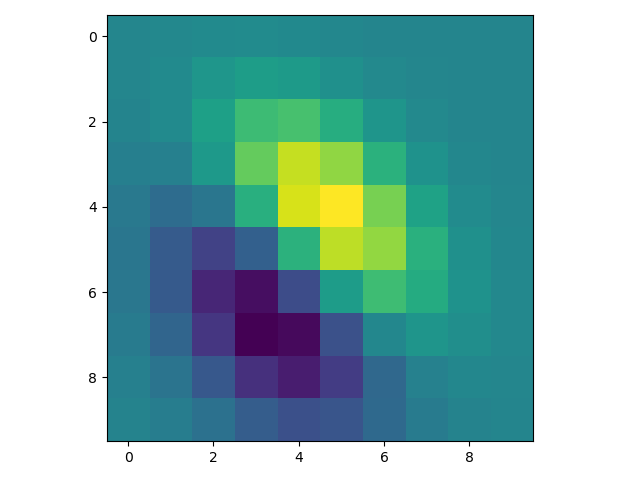


אחרי המון מאמצים הצלחתי ליצור Grid של 100 נקודות סביב הוקטור הממוצע, לאחר מכן השתמשתי בפונקציה Stats.multivariate\_normal.pdf על מנת לשערך את ההסתברות של נקודות אלו בהתפלגות והצגתי את התפלגויות אלו כתמונות:



ניתן לראות כי התמונה השמאלית היא התמונה של הסתברות הגאוסיין הנתונה לנו בשאלה והתמונה בימין היא שיערוך של ההסתברות.

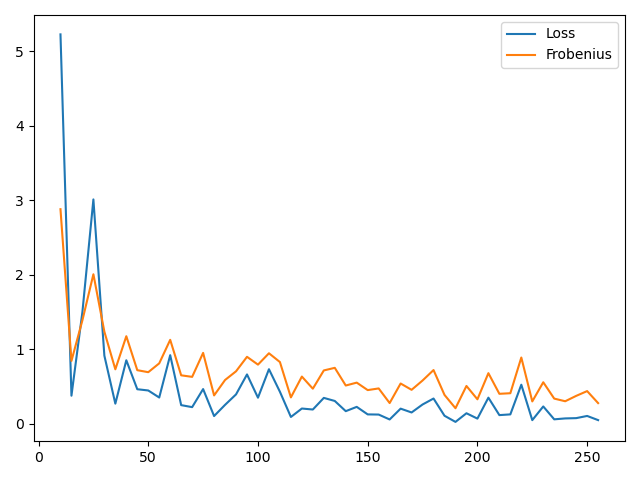
בעת חיסור המשוערכת מהתמונה המקורית הגעתי לתוצאה הבאה:





כעת החלטתי לדגום 50 פעמים דגימות של מספר נקודות אשר בעזרתן שעירכתי את ההתפלגות וחישבתי את המרחק בין ההתפלגות המשוערכת להתפלגות המקורית.

התחלתי לדגום מ10 נקודות עד 255 נקודות בקפיצות של 5 נקודות להלן התוצאות:

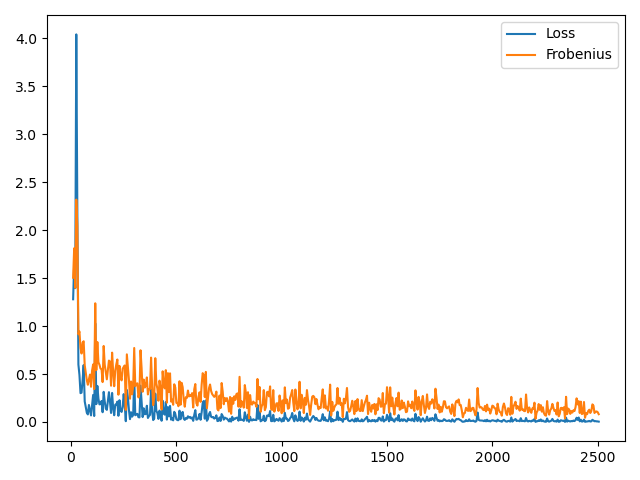


נראה כי ככל שנדגום יותר נקודות כך המרחק בין ההתפלגות המקורית להתפלגות המשעורכת יקטן

כאשר דגמתי 190 נקודות הגעתי לפונקציית המרחק הקטנה ביותר בין ההתפלוגויות (לפי מדד הLoss)

והיא 0.021 דבר זה מאמת את ההרעיון שככל שנדגום יותר נקודות כך נתקרב יותר לפונקצית ההתפלגות המקורית.

בחרתי לנסות זאת שוב הפעם עם 500 דגימות שונות של מספרי נקודות!

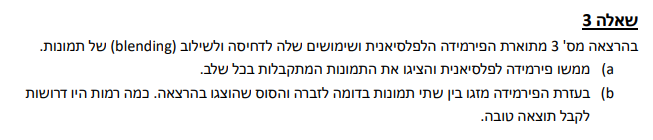


הפעם מצאתי כי כאשר דגמתי 2205 נקודות הגעתי למרחק הנמוך ביותר אשר הוא 0.0006 בהחלט מעניין.

בבדיקה על 5000 דגימות שונות הגעתי למספר אפסי כאשר דגמתי 11175 נקודות

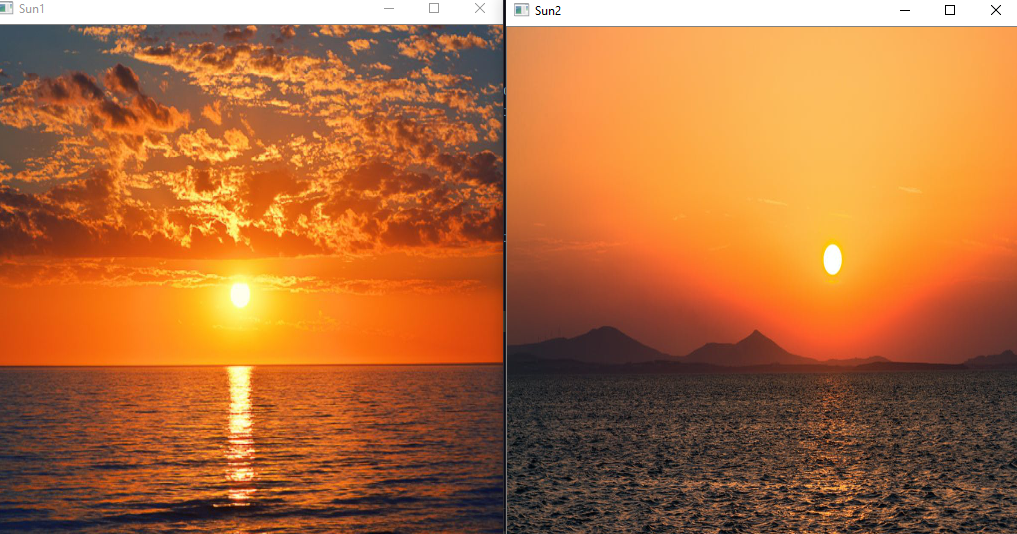


לכן אסיק כי ככל שמספר הדגימות שואף לאינסוף פונקציית הLoss שלנו שואפת ל0.



בניתי פירדמידה לפלסיאנית אשר איחדה בין שני תמונות של שקיעה בים שבחרתי בשביל ליצור אפקט של שתי שמשות, ניסיתי גם לחבר כמה שיותר את הים בין התמונות.

התמונות שבחרתי הם:



אשר חיבורם הטבעי (50 50 )נראה כך:



אך החלטתי להעצים את שתי השמשות לכן כשביצעתי את איחוד התמונות לא השתמשתי ברטיו של 50% לכל תמונה, התחלתי בלבנות 5 רמות והינה התוצאה:



לא משהו.... לאט לאט התקדמתי עד לבצוע 10 רמות והתוצאה הייתה דיי יפה! השגתי את 2 השמשות ואפילו המים נראים יחסית מתאימים!



דיי נהנתי לכן אנסה לעשות זאת על עוד 2 תמונות:

ניתן לראות שיצרתי משהו "מיוחד" ודווקא התמונה הימנית היא התמונה עם ה5 שכבות לעומת השמאלית עם 10!

לכן אסיק כי על מנת לקבל התאמה יפה הזדקקתי ל10 רמות בשקיעה לעומת 5 רמות בשילוב הכלבים. נראה כי כמות הרמות שנדרשות על מנת לקבל תוצאה טובה יכולה להשתנות מתמונה לתמונה.