Assignment 3:

Image Search

Due Date: 9/01/2017

הקדמה

בתרגיל זה נבנה מנוע חיפוש לתמונות, אשר מקבל אליו תמונת חיפוש ומציג תמונות דומות. האלגוריתם יתבסס על אלגוריתם ה-*Video Google* אשר נלמד בכיתה, ויפעל על תמונות בשחור-לבן. האלגוריתם מתואר בעמוד הבא.

על מנת להקל על תהליך פיתוח האלגוריתם, המימוש של האלגוריתם במהלך התרגיל יבוצע בסדר הפוך, קרי מהסוף להתחלה (פחות או יותר). שלבים אשר עושים שימוש במידע שטרם חושב יקבלו את הקלטים שלהם מתוך נתוני דמה אשר שמורים בקובץ test_data.mat שבקבצים המצורפים.

את תמונות יש להוריד מהאתר הבא: http://www.robots.ox.ac.uk/~vgg/data/oxbuildings/.

הורידו את 5k Dataset images (לינק ישיר מצורף גם בקובץ החומרים המצורפים) – כ-1.8GB.

שימו לב כי לא נעבוד עם כל התמונות הללו, אלא רק אלה שמופיעות בקובץ *ground_truth.mat*.

בקובץ החומרים המצורפים מופיע גם המאמר הבא:

BRIEF: Binary Robust Independent Elementary Features

המאמר מתאר חלופה ל-*SIFT Descriptor* כמנגנון לתיאור נקודות העניין בתמונה.

קראו את המאמר. במהלך התרגיל תידרשו לממש את ה-*Descriptor* המתואר בו.

שימו לב כי המאמר מכיל גם המלצות לפרמטרים ואופי השימוש במנגנון, אשר כדאי ליישמם בתרגיל.

התרגיל כולל מספר חלקים שזמן הריצה שלהם עלול להיות ארוך. לפיכך מומלץ שלא להשאיר את העבודה על התרגיל לרגע האחרון.

תיאור האלגוריתם

בניית Code Book

תחילה נרצה לבנות את ה-*Code-Book* אשר יכיל "מילים" אשר בעזרתן נתאר כל תמונה.

נבנה את ה-*Code-Book* על ידי ניתוח כלל התמונות שאיתן נעבוד - באופן הבא:

1. מציאת נקודות עניין
נחפש פינות בשיטת *Harris Corner Detector* (מקסימום N_{MP} נקודות לתמונה).
2. תיאור נקודות עניין
נבנה *descriptor* לכל נקודת עניין על ידי *BRIEF* (השיטה מהמאמר) באורך N_B ביטים.
3. בניית ה-Code Book
נאסוף את כל ה-*BRIEF-Descriptors* מכל התמונות שאנחנו עובדים איתן (לכל היותר $\mathrm{N_D}$).
נבנה את ה-*Code-Book* על-ידי שימוש באלגוריתם *k-means* וחלוקת כל ה-*BRIEF-Descriptors* ל-N_C צבירים (clusters), כאשר מתקיים כמובן $k = N_c$.
כל *cluster* יתאר לנו מילת קוד יחידה (מטריקת המרחק תהיה Hamming Distance).

תיאור תמונה

4. מציאת מאפייני תמונה
בהינתן ה-*Code-Book*, נוכל לתאר כל תמונה על ידי היסטוגרמת המילים המופיעות בה.
זאת נעשה על ידי מציאת נקודות העניין בתמונה (*Harris*), ותיאורן (*Brief*).
כל נקודת עניין שכזו, נשייך לאחת ממילות הקוד (*cluster*) מתוך ה-*Code-Book*, וזו תהיה המילה שתתאר את נקודת העניין (מטריקת המרחק תהיה *Hamming Distance*).
לבסוף, נספור כמה פעמים מופיעה כל מילת קוד בתמונה, ונבנה מזה היסטוגרמה (לא מנורמלת).
היסטוגרמה זו תהווה את ווקטור מאפייני התמונה (ווקטור באורך N_C).

למידת מחלקות

5. בניית מאפייני מחלקות
בתהליך חיפוש התמונות, נרצה לשייך תמונת קלט למחלקה מסוימת (כאשר מחלקה מכילה תמונות דומות), דבר הדורש לאפיין תחילה את המחלקות. לשם כך נעשה שימוש בתמונות שחולקו מראש למחלקות ובעזרתן נלמד את ווקטור המאפיינים שמתאר כל מחלקה, וזאת על ידי סיכום ההיסטוגרמות של כל התמונות בכל משפחה, ונירמולה (סכום כל ההיסטוגרמה שווה ל-1).

חיפוש תמונה

6. סיווג תמונה למשפחה
עכשיו כשבידינו ישנו גם ה-*Code-Book* וגם מאפייני המשפחות, ולאחר שמימשנו את כל האלגוריתמים הנ"ל, נוכל להפעיל את מנוע החיפוש.
בהינתן תמונת קלט, נמצא את ווקטור מאפייני התמונה (בדומה לתהליך הנ"ל), עבורו נחפש את המשפחה הכי מתאימה ע"י השוואה לווקטורי המאפיינים של כל משפחה.
7. קתרזיס
כשלב אחרון, נציג 9 תמונות מתוך המשפחה שסיווגנו אליה, וזו תהיה תוצאת החיפוש שלנו.

הסבר על הקבצים המצורפים

ground_truth.mat

קובץ המכיל את שמות קבצי התמונה, המחולקים למחלקות (כל מחלקה מכילה תמונות דומות).

כל מחלקה מכילה 3 שדות – *good*, *ok* ו-*junk*, המכילות שמות קבצי תמונה באיכות גבוהה, בינונית וגרועה בהתאמה.

מומלץ לטעון את תכולת הקובץ באופן הבא:

```
ground_truth = load('ground_truth.mat');
```

test_data.mat

קובץ המכיל קלטים נדרשים לחלקי התרגיל השונים. תעשו שימוש בקלטים אלה בסעיפים בהם נדרש שימוש במידע אשר טרם חושב (בעיקר בפונקציות הבדיקה).

הנכם מתבקשים ***לא לצרף קובץ זה בעת הגשת התרגיל.***

פונקציות בדיקה test_#_*.m

פונקציות לבדיקת תקינות הקוד והתוצאות שהתקבלו. תעשו שימוש בקבצים אלה בחלקי התרגיל השונים, ומטרתם שתוודאו שהקוד רץ באופן תקין. קבצים אלה או דומים להם ישמשו לבדיקת הפתרון שתגישו.

ניתן ומומלץ להשתמש בקוד שבקבצים אלה על מנת לבנות פונקציות הרצה משלכם.

test_init.m

פונקציית אתחול בדיקה, הטוענת מידע נדרש מה-test_data.mat וכן שואלת לתיקייה המכילה את התמונות.

```
[out1, out2, ...] = test_init('name1', 'name2', ...);
```

כל פלט מוציא את המידע הנדרש ממנו בקלט המתאים לו, כאשר אם הקלט הוא *'path'* אז המשתמש נשאל לתיקיית התמונות, אחרת – הפונקציה מוציאה את המשתנה מתוך test_data.mat.

dist_hamming.m (ראו תיעוד הפונקציה להסבר מפורט)

קלט: שתי מטריצות מסוג *logical* (בינאריות) בעלות מספר עמודות זהה.

מוצא את מרחק Hamming בין כל זוג איברים של שתי מטריצות בינאריות (*logical*) X ו-Y.

בכל קלט, שורה מייצגת את הביטים של איבר בודד.

אם X בגודל $M \times B$ ו-Y בגודל $N \times B$, אז המוצא בגודל $M \times N$.

bin2int.m (ראו תיעוד הפונקציה להסבר מפורט)

קלט: מטריצה מסוג *logical* (בינארית)

פונקציה לדחיסת מטריצה בינארית למטריצת *integers* מסוג *uint8*.

בהינתן שורה של ערכים בינאריים, מייצר שורה של *integers* כאשר כל 8 ערכים בינאריים רצופים הופכים לערך *uint8* יחיד (המכיל כמובן 8 ביטים). למשל השורה $[0,0,0,0,1,1,1,1,0,1,0,1,0,1,0,1]$ הופכת לשורה $[240,170]$ (הביטים נקראים משמאל לימין).

בהינתן מטריצה, דוחס שורה-שורה.

מטרת פונקציה זו לדחוס (בערך פי 8) את גודל המטריצה הבינארית, הן לטובת זמן ריצה והן לטובת אחסון, ומשמשת (במידת הצורך) בעיקר לדחיסת ה-*BRIEF-Descriptors* של תמונות.

int2bin.m (ראו תיעוד הפונקציה להסבר מפורט)

קלט: מטריצה מסוג *uint8* (בלבד!).

פונקציה לחילוץ המטריצה הבינארית מתוך מטריצת *uint8* דחוסה הנ"ל. בהינתן מטריצה, מחלץ שורה-שורה.

bit_count.m (ראו תיעוד הפונקציה להסבר מפורט)

קלט: מטריצה מסוג *uint8* (בלבד!).

פונקציה לספירת מספר הביטים השווים ל-1 בכל שורה של מטריצת *uint8* דחוס כנ"ל.

שקול ל-$sum(B,2)$ אם עובדים ישירות על מטריצה בינארית לא דחוסה.

bit_hamming.m (ראו תיעוד הפונקציה להסבר מפורט)

קלט: שתי מטריצות מסוג *uint8* (בלבד!) בעלות מספר עמודות זהה.

שקול לפונקציה dist_hamming.m, אך פועל ישירות על מטריצת *uint8* הדחוסה כנ"ל (דורש פחות זיכרון ממנה, אך זמן ריצה ארוך יותר).

תקציר פרמטרים

לאורך התרגיל מופיעים גדלים ופרמטרים שונים ומשונים.

מטרת הרשימה הבאה היא לרכז גדלים אלה לנוחיותכם.

K – מספר ה"מילים" השונות ב-*Code-Book* .

L – מספר הביטים המהווים *BRIEF-Descriptor* של נקודת עניין.

T – מספר המחלקות השונות של התמונות.

I – מספר נקודות העניין בתמונה אחת.

N – חסם עליון למספר נקודות העניין בתמונה אחת.

D – מספר ה-BRIEF-Descriptors השונים שמשמשים ללימוד ה-Code-Book.

J – מספר התמונות השונות שמשמשות ללימוד ווקטורי המאפיינים של המחלקות.

חלק א': סיווג תמונה למחלקה (5 נק')

בחלק הזה נתחיל תחת ההנחה שכבר חישבנו ווקטור מאפיינים עבור כל התמונות, וכן שיש בידינו את ווקטור מאפיינים של כל מחלקה.

כלומר, בהינתן *Code-Book* המכיל N_C מילים, כל תמונה מיוצגת על ידי היסטוגרמה הסופרת כמה פעמים מופיעה בה כל מילה (ווקטור באורך N_C של מספרים שלמים, כאשר במקום ה-k שמור מספר הפעמים שהופיעה המילה ה-k).

ווקטור המאפיינים של מחלקה הינה ווקטור שכיחויות עבור כל מילה (ווקטור באורך N_C, שסכום איבריו שווה ל-1, ובמקום ה-k שמורה השכיחות המילה ה-k מתוך כלל התמונות השייכות למחלקה). נניח סה"כ N_F מחלקות שונות.

1. בנו פונקציה המקבלת ווקטור מאפיינים של תמונה (היסטוגרמה לא מנורמלת), וכן את ווקטורי המאפיינים של כל המחלקות (היסטוגרמות מנורמלות), ומסווגת את התמונה למחלקה אליה היא הכי מתאימה.

   השתמשו ב-API הבא:

class_id = classify_hist (img_hist, class_hists)

<u>קלט:</u>

- img_hist – ווקטור בגודל $1 \times N_C$ המייצג היסטוגרמת מילים (לא מנורמלת) של תמונה.
- class_hists – מטריצה בגודל $N_F \times N_C$ המכילה את ווקטורי המאפיינים של כל המחלקות. כל שורה מייצגת ווקטור מאפיינים של מחלקה אחת (היסטוגרמה מנורמלת).

<u>פלט:</u>

- class_id – אינדקס המחלקה אליה התמונה מתאימה ביותר (כלומר, הפלט הוא k אם המחלקה המתאימה ביותר מיוצגת ע"י ווקטור מאפיינים בשורה ה-k).

<u>טיפ:</u> ניתן להימנע משימוש בלולאות על ידי שימוש בפונקציה *bsxfun*.

2. תארו בקצרה את אופן מימוש הפונקציה *classify_hist*.

   האם בחרתם לסווג על ידי המרחק המינימאלי בין היסטוגרמת התמונה להיסטוגרמות המחלקות, או שבחרתם בגישה ההסתברותית, לפיה חיפשתם את המחלקה שמספקת סבירות מקסימאלית?

   נמקו את בחירתכם.

3. הריצו את הקובץ test_1_check_classify.m.

   קובץ זה בודק את הפונקציה שמימשתם בחלק זה. וודאו שהרצת קובץ זה עוברת באופן תקין, וציינו במסמך את אחוז ההצלחה המתקבל (מודפס בסוף תהליך ההרצה). וודאו שאחוז ההצלחה המתקבל גבוה מ-80%.

   בבדיקה זו טוענים מ-test_data.mat את ווקטורי המאפיינים של תמונות (*images_hist*) וכן את ווקטורי המאפיינים של מחלקות (*class_hist*).

   שימו לב כי בפונקציה זו נעשה שימוש רק בתמונות באיכות גבוהה ובינונית (קרי, רק תמונות שמופיעות ב-*ground_truth* תחת *good* ו-*ok*), ולא נעשה שימוש בתמונות באיכות גרועה (*junk*).

חלק ב': בניית מאפייני מחלקה (10 נק')

בחלק זה נרצה לבנות בעצמנו את ווקטורי המאפיינים של המחלקות.

ברשותנו נתונים ווקטורי המאפיינים של כל התמונות (כל ווקטור באורך N_C), וכן ידע מקדים על המחלקה אליה שייכת כל תמונה. על סמך זה, נרצה ללמוד את ווקטור המאפיינים של כל מחלקה, על מנת לקבל תוצאה טובה ככל האפשר בעת סיווג תמונה למחלקה (בדומה למה שביצענו בחלק הקודם).

שימו לב שכדי להימנע מתלות במספר הדגימות שיש בכל מחלקה, יש לנרמל בסוף את ווקטור המאפיינים של המחלקה, כך שסכומו יהיה 1.

4. שיטה אפשרית ללימוד המחלקות היא לסכום את כל ווקטורי המאפיינים של התמונות בכל מחלקה, ולנרמל את התוצאה המתקבלת.

   מהי המוטיבציה בשיטה שכזו, כלומר מהו הקריטריון ששיטה זו ממזערת?

5. בנו פונקציה המקבלת את כל ווקטורי המאפיינים של כל התמונות, וכן את הסיווג של כל תמונה למחלקה שלה, ולומדת מתוך מידע זה את ווקטורי המאפיינים של המחלקות השונות.

   השתמשו ב-API הבא:

class_hists = learn_classes (sample_hists, class_ids)

<u>קלט:</u>

- sample_hists – מטריצה בגודל $N_S \times N_C$ המכילה את ווקטורי המאפיינים של כל התמונות. כל שורה מייצגת ווקטור מאפיינים של תמונה אחת (היסטוגרמה לא מנורמלת).
- class_ids – ווקטור בגודל $N_S \times 1$ המכיל את הסיווג של כל תמונה למחלקה. כל איבר מכיל מספר שלם בין 1 לבין N_F.

<u>פלט:</u>

- class_hists – מטריצה בגודל $N_F \times N_C$ המכילה את ווקטורי המאפיינים של כל המחלקות.
  כל שורה מייצגת ווקטור המאפיינים של מחלקה אחת.
  יש לוודא שסכום כל האיברים בכל שורה שווה ל-1.

6. תארו בקצרה את אופן הלימוד אותו בחרתם לממש.

   <u>בונוס:</u> במידה ובחרתם שיטה שונה מהשיטה בשאלה 4, נמקו את היתרונות בשיטה זו, וכן מהו הקריטריון אשר אותה שיטה באה למזער.

7. הריצו את הקובץ test_2_create_classes.m.

בבדיקה זו בודקים את תקינות הפונקציה שמימשתם בחלק זה. וודאו שהרצתה עוברת באופן תקין.

בבדיקה זו טוענים מ-test_data.mat את ווקטורי המאפיינים של תמונות (*images_hist*).

שימו לב כי בפונקציה זו נעשה שימוש רק בתמונות באיכות גבוהה ובינונית (קרי, רק תמונות שמופיעות ב-*ground_truth* תחת *good* ו-*ok*), ולא נעשה שימוש בתמונות באיכות גרועה (*junk*).

בסיום תהליך ההרצה, תוצג שאלה האם ברצונכם לשמור את התוצאה המתקבלת לקובץ 'my_class_hist.mat'. קובץ זה נדרש לשאלה הבאה, וכן גרסה עתידית שלו צריכה להיכלל בקבצים אשר אתם מגישים במהלך התרגיל (התוצאה נשמרת בכל מקרה בקובץ 'tmp_last_class_hist.mat').

8. הריצו את הקובץ test_3_check_classes.m.

בבדיקה זו בודקים את נכונות התוצאה שקיבלתם בחלק זה, על ידי שימוש בווקטורי המחלקות שנשמרו בקובץ 'my_class_hist.mat'. וודאו שהרצתה מניבה תוצאה נכונה, וציינו במסמך את אחוז ההצלחה המתקבל (מודפס בסוף תהליך ההרצה).

האם קיבלתם אחוז שונה מזה שהתקבל בחלק הקודם (שאלה 3)?

בבדיקה זו טוענים מ-test_data.mat את ווקטורי המאפיינים של תמונות (*images_hist*).

שימו לב כי בפונקציה זו נעשה שימוש רק בתמונות באיכות גבוהה ובינונית (קרי, רק תמונות שמופיעות ב-*ground_truth* תחת *good* ו-*ok*), ולא נעשה שימוש בתמונות באיכות גרועה (*junk*).

חלק ג': מציאת מאפייני תמונה (15 נק')

בחלק הזה נרצה למצוא עבור כל תמונה את ווקטור המאפיינים שלה (היסטוגרמת מילים לא מנורמלת).

נצא מנקודת הנחה שכבר מצאנו עבור כל תמונה את ה-*BRIEF-Descriptor* של כל נקודות העניין שלה, וכן כי בנינו כבר את ה-*Code-Book* (כרגע לא קריטי להבין מהו ה-*BRIEF-Descriptor*, רק חשוב להבין שזהו ווקטור בינארי באורך N_B).

מניחים כי כל תמונה מכילה N_P נקודות עניין (לכל היותר N_{MP} נקודות), וכל נקודה שכזו מתוארת על ידי *BRIEF-Descriptor*. מקבלים כי אוסף כל ה-*BRIEF-Descriptors* של תמונה זו מטריצה בגודל $N_P \times N_B$ (מטריצה בינארית), בה השורה ה-k היא ה-*BRIEF-Descriptor* של נקודת עניין ה-k. שימו לב כי N_P יכול להיות שונה מתמונה לתמונה.

ה-*Code-Book* הוא אוסף של N_C מילות קוד שונות, כאשר כל מילה מתוארת על ידי *BRIEF-Descriptor* שונה. מקבלים כי ה-*Code-Book* הוא מטריצה בגודל $N_C \times N_B$ (מטריצה בינארית), בה השורה ה-k היא ה-*BRIEF-Descriptor* של המילה ה-k.

על מנת לייצג *BRIEF-Descriptor* של תמונה כמילה, יש למצוא מהי המילה הקרובה ביותר מתוך ה-*Code-Book*, ואינדקס המילה הזו (מספר שלם בין 1 ל-N_C) הוא המילה המייצגת של *BRIEF-Descriptor* זה. מכיוון שאנו עוסקים בווקטורים בינאריים, נשתמש ב-*Hamming Distance* בתור מטריקת המרחק בין ווקטורים.

על מנת לבנות ווקטור מאפיינים לתמונה, תחילה נדרש למצוא עבור כל *BRIEF-Descriptor* בתמונה את מילת הקוד המייצגת שלו, ולבסוף לבנות היסטוגרמה של מספר החזרות של כל מילה (ווקטור בגודל $1 \times N_C$). הערך במקום ה-*k* בווקטור ההיסטוגרמה מייצג את מספר הפעמים שהמילה ה-*k* הופיעה בתמונה.

9. הסבירו מהו *Hamming Distance*, ומדוע במקרה שלנו זו מטריקה טובה למרחק בין ווקטורים.

   האם כדאי להשתמש ב-*Hamming Distance* לבעיית סיווג תמונה למשפחה (כפי שביצענו בחלק א' של התרגיל)? מדוע?

10. לתרגיל מצורפת הפונקציה dist_hamming.m, אשר מטרתה למצוא *Hamming Distance* בין שתי מטריצות, תחת הנחה שכל שורה במטריצה מייצגת ווקטור בינארי.

    עברו על הסבר הפונקציה ומימושה, והבינו את אופן פעולתה.

    וודאו שהפונקציה נותנת תוצאות נכונות.

    ניתן להיעזר בפונקציה זו בהמשך התרגיל.

11. ממשו פונקציה המקבלת אוסף של *BRIEF-Descriptors* וכן מקבלת *Code-Book*, ואשר מוצאת את המילה המייצגת של כל *BRIEF-Descriptor*.

השתמשו ב-API הבא:

[words, hdist] = assign_to_cluster (descriptors, code_book)

<u>קלט:</u>

- descriptors – מטריצה בינארית בגודל $N_P \times N_B$ המכילה את ה-*BRIEF-Descriptors* של כל נקודות העניין בתמונה. כל שורה מייצגת *BRIEF-Descriptor* של נקודת עניין אחת (ווקטור בינארי באורך N_B).
- code_book – מטריצה בינארית בגודל $N_C \times N_B$ המכילה את ה-*BRIEF-Descriptors* של כל המילים ב-*Code-Book*. כל שורה מייצגת *BRIEF-Descriptor* של מילה אחת.

<u>פלט:</u>

- words – ווקטור בגודל $N_P \times 1$ של מספרים שלמים (בין 1 ל-N_C) המציין את המילים המייצגות של כל *BRIEF-Descriptor*. ערך n במיקום ה-k בווקטור מציין ש-*BRIEF-Descriptor* ה-k מיוצג על ידי המילה ה-n מתוך ה-*Code-Book*.
- hdist – ווקטור בגודל $N_P \times 1$ המכיל את ה-*Hamming Distance* בין כל *BRIEF-Descriptor* למילה המייצגת שלו.

<u>טיפ:</u> אם משתמשים ב-dist_hamming.m המצורף, ניתן לממש פונקציה זו ללא לולאות.

12. תארו בקצרה את אופן מימוש הפונקציה assign_to_cluster.

13. ממשו פונקציה המקבלת את רשימת המילים בתמונה, ובונה ווקטור מאפיינים לתמונה על ידי בניית היסטוגרמה הסופרת את מספר הפעמים שמופיעה כל מילה.

השתמשו ב-API הבא:

word_hist = build_feature_vector (words, code_book)

<u>קלט:</u>

- words – ווקטור בגודל $N_P \times 1$ של מספרים שלמים (בין 1 ל-N_C) המציין את המילים המייצגות של כל *BRIEF-Descriptor*. ערך n במיקום ה-k בווקטור מציין ש-*BRIEF-Descriptor* ה-k מיוצג על ידי המילה ה-n מתוך ה-*Code-Book*.
- code_book – מטריצה בינארית בגודל $N_C \times N_B$ המכילה את ה-*BRIEF-Descriptors* של כל המילים ב-*Code-Book*. כל שורה מייצגת *BRIEF-Descriptor* של מילה אחת.

<u>פלט:</u>

- word_hist – ווקטור בגודל $1 \times N_C$ של מספרים שלמים המייצג את היסטוגרמת מילות הקוד שבקלט. כלומר אם הערך k מופיע c פעמים ברשימת ה-*words* שבקלט, צריך להתקיים כי $word\_hist(k) = c$.

<u>טיפ:</u> מומלץ להיעזר בפונקציה hist. בשימוש ב-hist, ניתן לממש פונקציה זו ללא לולאות.

14. הריצו את הקובץ test_4_create_features.m.

בבדיקה זו בודקים את תקינות הפונקציה שמימשתם בחלק זה. וודאו שהרצתה עוברת באופן תקין. צרפו למסמך את ההדפסות שמתקבלות בסוף תהליך ההרצה (סטטיסטיקה על מרחקי הערכים מהמילים. יש לצרף רק את הערכים הכוללים, ולא את רשימת המרחקים עבור כל קובץ בנפרד).

בסיום תהליך ההרצה, תוצג שאלה האם ברצונכם לשמור את התוצאה המתקבלת לקובץ 'tmp_images_hist.mat' (התוצאה נשמרת בכל מקרה בקובץ 'tmp_last_images_hist.mat'). קובץ זה נדרש לשאלה הבאה, אולם **הנכם מתבקשים שלא לצרף קבצים אלה לפתרון המוגש**.

בבדיקה זו טוענים מ-test_data.mat את ה-*Code-Book* (*code_book*) ואת ה-*BRIEF-Descriptors* של התמונות (*images_brief*).

שימו לב כי בפונקציה זו נעשה שימוש בתמונות בכל האיכויות (תמונות שמופיעות ב-*ground_truth* תחת *good*, *ok* ו-*junk*).

שימו לב כי ה-*BRIEF-Descriptors* שנטענים מתוך ה-*test_data* מגיעים דחוסים (כל 8 ערכים בינאריים דחוסים לערך *uint8* יחיד), וחילוץ הערכים הבינאריים מבוצע על ידי int2bin.m. עם זאת, הפונקציות שנדרשתם לממש בחלק זה מקבלות כבר ערכים בינאריים מחולצים.

15. בנו (1) פונקציה אשר בונה מאפייני משפחות, ו-(2) פונקציה המבצעת סיווג, בדומה למה שבוצע בחלקי התרגיל עד עכשיו, כאשר הפעם אתם משתמשים בווקטורי המאפיינים שקיבלתם בשאלה הקודמת, במקום בערכים ששמורים ב-*test_data* (שמורים בקובץ 'tmp_images_hist.mat').

מומלץ להתבסס על test_2_create_classes.m ליצירת ושמירת ווקטורי המאפיינים של המשפחות (1), ועל test_3_check_classes.m להרצת הסיווג (2). אין צורך לטעון אף מידע מ-test_data.mat.

שימרו את ווקטורי המאפיינים של המשפחות בקובץ 'my_class_hist.mat' בדומה למה שבוצע בשאלה 7.

ציינו במסמך את אחוז ההצלחה המתקבל (בדומה לנעשה בשאלות 3 ו-8).

חלק ד': תיאור נקודות עניין (25 נק')

כעת נרצה לבנות בעצמנו את המנגנון ליצירת תיאור נקודות העניין.

בהרצאות ראינו כי שיטה אחת שכזו היא *SIFT Descriptors*. אנחנו נממש בתרגיל זה שיטה חלופית.

קראו את המאמר המצורף לתרגיל: *BRIEF: Binary Robust Independent Elementary Features*.

16. תארו בקצרה במילים שלכם מהו ה-*BRIEF-Descriptor*.

מהם היתרונות של השימוש בשיטה זו על פני ה-*SIFT*? מהם החסרונות?

17. ממשו פונקציה לתיאור נקודות עניין בתמונה המתבססת על השיטה הנ"ל. הפונקציה תקבל בקלט תמונה (בגווני אפור), את מיקומי נקודות העניין בה (קואורדינטות של פינות), וכן רשימה המכילה את הקואורדינטות (היחסיות) של זוגות הנקודות שמשמשות לכל בדיקה בינארית.

השתמשו ב-API הבא:

descriptors = describe_interest_points (img, corners, brief_coordinates)

<u>קלט:</u>

- img – תמונה בגווני אפור (מטריצת ערכי האפור בתמונה).
- corners – מטריצה בגודל $N_P \times 2$ של מיקומי נקודות העניין בתמונה:

$$corners = \begin{pmatrix} x_1 & y_1 \\ x_2 & y_2 \\ \vdots & \vdots \\ x_{N_P} & y_{N_P} \end{pmatrix}$$

- brief_coordinates – מטריצה בגודל $4 \times N_B$ של הקואורדינטות היחסיות של זוגות הנקודות שמשמשות לכל בדיקה בינארית:

$$brief\ coordinates = \begin{bmatrix} x1_1 & x1_2 & \dots & x1_{N_B} \\ y1_1 & y1_2 & \dots & y1_{N_B} \\ x2_1 & x2_2 & \dots & x2_{N_B} \\ y2_1 & y2_2 & \dots & y2_{N_B} \end{bmatrix}$$

יש לטעון את הקואורדינטות היחסיות הנדרשות לבניית *BRIEF-Descriptors* (*brief_coordinates*) מ-test_data.mat

<u>פלט:</u>

- descriptors – מטריצה בינארית בגודל $N_P \times N_B$ של תיאורי כל נקודות העניין שבתמונה.

  השורה ה-k במטריצה מכילה ווקטור באורך N_B המהווה את ה-*BRIEF-Descriptor* של נקודת העניין ה-k.

<u>טיפ:</u> ניתן לממש את הפונקציה ללא לולאות.

18. הריצו את הקובץ test_5_create_brief.m.

בבדיקה זו בודקים את תקינות הפונקציה שמימשתם בחלק זה. וודאו שהרצתה עוברת באופן תקין.

בסיום תהליך ההרצה, תוצג שאלה האם ברצונכם לשמור את התוצאה המתקבלת לקובץ 'tmp_images_brief.mat' (התוצאה נשמרת בכל מקרה בקובץ 'tmp_last_images_brief.mat'). קובץ זה נדרש לשאלה הבאה, אולם **הנכם מתבקשים <u>שלא</u> לצרף קבצים אלה לפתרון המוגש**.

בבדיקה זו טוענים מ-test_data.mat את רשימת נקודות העניין בתמונות (*images_corner*) ואת הקואורדינטות היחסיות של זוגות הנקודות לבניית ה-*BRIEF-Descriptors* (*brief_coordinates*).

שימו לב כי בפונקציה זו נעשה שימוש בתמונות בכל האיכויות (תמונות שמופיעות ב-*ground_truth* תחת *good*, *ok* ו-*junk*).

שימו לב כי ה-*BRIEF-Descriptors* שנשמרים לקובץ בסיום התהליך עוברים תהליך דחיסה (כל 8 ערכים בינאריים דחוסים לערך *uint8* יחיד), ודחיסת הערכים הבינאריים מבוצע על ידי bin2int.m. עם זאת, הפונקציות שנדרשתם לממש בחלק זה מוציאות בפלט מטריצה בינארית "רגילה".

19. בנו פונקציה אשר מוצאת ושומרת ווקטור מאפיינים לתמונות, בדומה למה שבוצע בחלק ג', כאשר הפעם אתם משתמשים ב-*BRIEF Descriptors* שקיבלתם בשאלה הקודמת, במקום בערכים ששמורים ב-*test_data* (עדיין יש צורך לטעון מ-test_data.mat את ה-*Code-Book* (*code_book*)). כמו כן, חזרו על הרצת הפונקציות משאלה 15, בהתבסס על הנתונים החדשים.

מומלץ להתבסס על test_4_create_features.m ליצירת ושמירת ווקטורי המאפיינים של התמונות.

שימרו את ווקטורי המאפיינים של המשפחות בקובץ 'my_class_hist.mat' בדומה למה שבוצע בשאלה 7. במידה וזהו הסעיף האחרון שתספיקו לבצע, יש לצרף את הקובץ 'my_class_hist.mat' שמתקבל בשלב זה לפתרון המוגש.

ציינו במסמך את אחוז ההצלחה המתקבל (בדומה לנעשה בשאלות 3 ו-8). שימו לב שעקב מימוש שונה, עלולים להיווצר הבדלים באחוזי ההצלחה לעומת סעיפים קודמים.

חלק ה': מציאת נקודות עניין (15 נק')

כעת נרצה למצוא את נקודות העניין בתמונה.

לשם כך, נשתמש ב-*Harris Corner Detector*. חזרו על ההרצאות, והיעזרו במקורות נוספים, על מנת לממש את האלגוריתם.

כיוון שחלק זה הוא אינו עיקר התרגיל, ניתן להיעזר בפונקציה הקיימת ב-MATLAB.

20. ממשו גלאי פינות בתמונה המקבל כקלט תמונה בגווני אפור, חסם למספר נקודות עניין מקסימאלי, וכן חסם למרחק מינימאלי מקצה התמונה (עקב האופי של *BRIEF*, אנחנו לא רוצים להתעסק עם קצוות תמונה).

השתמשו ב-API הבא:

corners = find_interest_points (img, dist_from_edge, max_points)

<u>קלט:</u>

- img – תמונה בגווני אפור (מטריצת ערכי האפור בתמונה).
- dist_from_edge – מספר המייצג את המרחק המינימאלי (בפיקסלים) של נקודות העניין מקצה התמונה.
- max_points – מספר המהווה חסם עליון למספר נקודות העניין בתמונה.

<u>פלט:</u>

- corners – מטריצה בגודל $N_P \times 2$ של מיקומי נקודות העניין (פינות) בתמונה:

$$corners = \begin{pmatrix} x_1 & y_1 \\ x_2 & y_2 \\ \vdots & \vdots \\ x_{N_P} & y_{N_P} \end{pmatrix}.$$

21. הריצו את הקובץ test_6_create_corners.m.

בבדיקה זו בודקים את תקינות הפונקציה שמימשתם בחלק זה. וודאו שהרצתה עוברת באופן תקין.

בסיום תהליך ההרצה, תוצג שאלה האם ברצונכם לשמור את התוצאה המתקבלת לקובץ 'tmp_images_corner.mat' (התוצאה נשמרת בכל מקרה בקובץ 'tmp_last_images_corner.mat'). קובץ זה נדרש לשאלה הבאה, אולם **הנכם מתבקשים <u>שלא</u> לצרף קבצים אלה לפתרון המוגש**.

בבדיקה זו לא טוענים מידע ב-test_data.mat.

שימו לב כי בפונקציה זו נעשה שימוש בתמונות בכל האיכויות (תמונות שמופיעות ב-*ground_truth* תחת *good*, *ok* ו-*junk*).

22. בנו פונקציה אשר מוצאת ושומרת *BRIEF-Descriptors* לנקודות העניין בתמונות, בדומה למה שבוצע בחלק הקודם, כאשר הפעם אתם משתמשים בנקודות עניין שקיבלתם בשאלה הקודמת, במקום בערכים ששמורים ב-*test_data* (עדיין יש צורך לטעון מ-test_data.mat את הקואורדינטות היחסיות הנדרשות לבניית *BRIEF-Descriptors* (*brief_coordinates*)). כמו כן, חזרו על הרצת הפונקציות משאלה 19 (כולל הפונקציות משאלה 15), בהתבסס על הנתונים החדשים.

מומלץ להתבסס על test_5_create_brief.m ליצירת ושמירת *BRIEF-Descriptors* של התמונות.

שימרו את ווקטורי המאפיינים של המשפחות בקובץ 'my_class_hist.mat' בדומה למה שבוצע בשאלה 7. במידה וזהו הסעיף האחרון שתספיקו לבצע, יש לצרף את הקובץ 'my_class_hist.mat' שמתקבל בשלב זה לפתרון המוגש.

ציינו במסמך את אחוז ההצלחה המתקבל (בדומה לנעשה בשאלות 3 ו-8). שימו לב שעקב מימוש שונה, עלולים להיווצר הבדלים באחוזי ההצלחה לעומת סעיפים קודמים.

חלק ו': בניית ה-Code Book (15 נק')

בשלב זה נרצה לבנות בעצמנו את ה-*Code-Book*.

כזכור, ה-*Code-Book* הוא אוסף של N_C מילות קוד, אשר משתמשים בהן לתיאור נקודות העניין (מילה לכל נקודת עניין), כאשר ברצוננו לבנות אותו בצורה שתאפשר לתאר את נקודות העניין השונות בצורה הטובה ביותר.

לשם כך נבחר להשתמש באלגוריתם *K-Means*, בעזרתו נקבץ סט של *BRIEF-Descriptors* ל-N_C צבירים (*Clusters*), כאשר כל *Cluster* מגדיר לנו מילת קוד אחת.

גם כאן נשתמש במטריקת *Hamming Distance*.

23. תארו בקצרה את אלגוריתם ה-*K-Means*.

כיצד בונים את ה-*Code-Book* מתוך ה-*Cluster*-ים (כלומר, כיצד מחליטים מהו ה-*BRIEF-Descriptor* המתאר כל מילת קוד)?

24. מהי פונקציית המטרה באלגוריתם ה-*K-Means* (מהו הקריטריון אותו רוצים למזער)?

מדוע לדעתכם בחרנו ב-*K-Means* לבניית ה-*Code-Book*?

25. ניתן להסתכל על קידוד תמונה להיסטוגרמת מילים בתור תהליך דחיסה של האינפורמציה בתמונה.

נתון כי אנו דוחסים כל תמונה לווקטור היסטוגרמה באורך N_C, כאשר כל תא (*bin*) בהיסטוגרמה מיוצג על ידי *byte* בודד (8 bit-ים).

מהו יחס הדחיסה אם אנו דוחסים תמונה בגודל $H \times W$ בה כל פיקסל מיוצג על ידי *byte* בודד?

מהו יחס הדחיסה ביחס לשמירת *BRIEF-Descriptors* של תמונה, בהינתן שבתמונה יש N_P נקודות עניין כאשר כל נקודה מתוארת על ידי *BRIEF-Descriptor* באורך N_B (הניחו שערך בינארי מיוצג על ידי bit בודד)?

כעת נתון כי גודל תמונה הוא 1024×768, מספר מילות הקוד (N_C) הוא 2000, בתמונה יש 1000 נקודות עניין ($\mathrm{N_P}$), ואורך כל *BRIEF-Descriptor* (N_B) הוא 256. מהו יחס הדחיסה בשני המקרים הנ"ל?

26. נתון כי מריצים את אלגוריתם ה-*K-Means* על כל נקודות העניין של N_S תמונות (סה"כ $N_D = N_S \times N_P$ נקודות עניין), מספר מילות הקוד הוא N_C (כלומר $k = N_C$) ואורך כל *BRIEF-Descriptor* הוא N_B.

מהי הסיבוכיות החישובית של איטרציה אחת של ה-*K-Means*, בהינתן הגדלים האלה?

מהו הערך המתקבל בהינתן הנתונים מהשאלה הקודמת ובהינתן שכמות התמונות (N_S) היא 500?

העריכו (בצורה גסה) כמה זמן ייקח למחשב להריץ 100 איטרציות בהינתן מיליארד פעולות בשניה.

הציעו דרכים (2 ומעלה) להתמודד עם הבעיה שעולה פה (תיאור קצר מספיק).

27. ממשו פונקציה הבונה *Code-Book* על ידי הפעלת אלגוריתם *K-Means* על *BRIEF-Descriptors*.

שימו לב כי **לא** ניתן לעשות בסעיף זה שימוש בפונקציית *kmeans* המובנית ב-MATLAB.

השתמשו ב-API הבא:

[code_book, idx, hdist_mean] = kmeans_hamming (briefs, k_or_c, max_itter)

<u>קלט:</u>

briefs – מטריצה בינארית בגודל $N_D \times N_B$ המכילה את ה-*BRIEF-Descriptors* של כל נקודות העניין שמשמשות לבניית ה-*Code-Book*. כל שורה מייצגת *BRIEF-Descriptor* של נקודת עניין אחת (ווקטור בינארי באורך N_B).

ניתן להוסיף תמיכה לעבודה עם מטריצה בינארית דחוסה (ראו טיפים).

k_or_c – קלט המקבל אחת משתי אפשרויות:

1. במידה והתקבל מספר שלם בודד, מפרש את זה כ-k (מספר ה-*Cluster*-ים), ומאתחל את ה-*Cluster*-ים באופן הסטנדרטי.
2. במידה והתקבלה מטריצה בינארית בגודל $k \times N_B$ מפרש את k לפי מספר השורות במטריצה, ומאתחל את ה-*Cluster*-ים לפי המטריצה (כל שורה מתארת מילת קוד).

ניתן להוסיף גם כאן תמיכה במטריצה בינארית דחוסה.

max_itter – מספר המציין מספר מקסימאלי של איטרציות לימוד לאלגוריתם ה-*K-Means* שאחריהן האלגוריתם מסיים פעולתו גם אם לא הגיע להתכנסות.

<u>פלט:</u>

code_book – מטריצה בינארית בגודל $k \times N_B$ המכילה את ה-*BRIEF-Descriptors* של כל המילים ב-*Code-Book*. כל שורה מייצגת *BRIEF-Descriptor* של מילה אחת.

Idx – ווקטור בגודל $N_D \times 1$ המציין לכל נקודת אימון בקלט מהי מילת הקוד אליה היא משויכת (אינדקס השורה ב-*code_book*).

hdist_mean – מספר המייצג הערך הממוצע של המרחקים (*Hamming*) בין כל *BRIEF-Descriptor* לבין מילת הקוד אליה הוא משויך.

<u>טיפים:</u>

- ניתן להיעזר בפונקציה *assign_to_cluster* (משאלה 11) ו/או ב-dist_hamming.m.
- נסו לממש את הפונקציה ללא לולאות (פרט לאיטרציות).
- מומלץ להוסיף הדפסות בסוף כל איטרציה עם נתונים מעניינים כמו מספר העדכונים באיטרציה, מרחק ממוצע וכד', על מנת לראות אם תהליך הריצה מתכנס ואת קצב ההתכנסות שלו.
- ניתן לעבוד במטריצה בינארית דחוסה על מנת להקטין כמות זיכרון נדרשת להרצה (על חשבון זמן ריצה ארוך יותר). היעזרו ב-bin2int.m, ב-int2bin.m וב-bit_hamming.m המסופקים.
  במקרה זה, יש לדאוג שהפונקציה תדע לקבל הן קלט בינארי רגיל, והן קלט בינארי דחוס (הפונקציה *islogical()* תעזור פה).

28. הריצו את הקובץ test_7_check_kmeans.m.

בבדיקה זו בודקים את הממשק של הפונקציה בלבד (ולא את נכונותה). וודאו שהרצתה עוברת באופן תקין.

חלק ז': קתרזיס (25 נק')

בחלק זה נעשה אינטגרציה לכל הפונקציות שמימשנו עד כה, למערכת אחת של "מנוע חיפוש" תמונות.

בחלק זה יהיה עליכם לבחור לבד את הפרמטרים איתם תעבדו.

29. ממשו פונקציה המייצרת סט קואורדינטות יחסיות המשמשות ליצירת ה-*BRIEF-Descriptor*. היעזרו במאמר המצורף לקביעת מנגנון ההגרלה של הקואורדינטות.

השתמשו ב-API הבא:

brief_coordinates = generate_brief_coordinates (desc_length, win_size)

קלט:

desc_length – מספר שלם המכיל את האורך הרצוי של ה-*BRIEF-Descriptor* (N_B).

win_size – גודל החלון להגרלת הקואורדינטות (במאמר מופיע כ-S). הקוארדינטות היחסיות (בערך מוחלט) שיוגרלו חייבות להיות קטנות או שוות לחצי ממנו.

פלט:

brief_coordinates – מטריצה בגודל $4 \times N_B$ של הקוארדינטות היחסיות של זוגות הנקודות שמשמשות לכל בדיקה בינארית:

כאשר כל אלמנט במטריצה a_{ij} מקיים $-\frac{win\ size}{2} \leq a_{ij} \leq \frac{win\ size}{2}$.

30. הריצו את הקובץ test_8_check_briefs_gen.m.

בבדיקה זו בודקים את תקינות הפונקציה שמימשתם בחלק זה. וודאו שהרצתה עוברת באופן תקין.

31. בנו פונקציה (אפשר יותר מאחת) המגרילה קואורדינטות ל-*BRIEF*, מוצאת נקודות עניין (חלק ה') בכל התמונות, ומחשבת עבורן *BRIEF-Descriptors* (חלק ד'). בחרו בעצמכם את הפרמטרים השונים.

ניתן לעבוד רק על התמונות שמופיעות בקובץ ground_truth.mat.

יש לשמור את קוארדינטות ה-*BRIEF* היחסיות האחרונות (אלה שעבורן אתם מייצרים את שאר הקבצים להגשה) בקובץ 'my_brief_coordinates.mat' תחת המשתנה *'brief_coordinates'*. **יהיה עליכם לצרף קובץ זה לפתרון.**

כמו כן, עליכם לשמור קובץ המכיל את ה-*BRIEF-Descriptors* של כל התמונות לקובץ (מומלץ לשמור לקובץ בשם 'tmp_images_brief.mat' תחת המשתנה *'images_brief'*). **אין** להגיש קובץ זה.

מומלץ להיעזר בקבצי הבדיקה המצורפים לתרגיל (בעיקר test_5 ו-test_6).

ניתן לדחוס את מטריצות ה-*BRIEF-Descriptors* בעזרת הפונקציה bin2int.m.

32. בנו פונקציה להפעלת *k-means* על ה-*BRIEF-Descriptors* שקיבלתם בסעיף הקודם, לשם לימוד ה-*Code-Book*. בחרו בעצמכם את הפרמטרים השונים.

יש לשמור את ה-*Code-Book* האחרון (זה שעבורו אתם מייצרים את שאר הקבצים להגשה) בקובץ 'my_code_book.mat' תחת המשתנה *'code_book'* (יש לשמור מטריצה בינארית רגילה ולא מטריצה דחוסה). **<u>יהיה עליכם לצרף קובץ זה לפתרון.</u>**

שימו לב כי לא חובה להריץ את הלימוד על כלל הנקודות כולן.

מומלץ שלא להריץ על נקודות מתמונות באיכות junk.

33. הסבירו את משטר הפעלת *k-means* שמימשתם בשאלה הקודמת: מה מכניסים כקלט, איך מריצים וכמה פעמים וכד'.

34. בנו פונקציה (אפשר יותר מאחת) אשר עבור כל התמונות משאלה 31, ממירה את ה-*BRIEF-Descriptors* למילות קוד לפי ה-*Code-Book* משאלה 32 ובונה לכל תמונה ווקטור מאפיינים על ידי היסטוגרמת המילים (חלק ג'), וכן לומדת את ווקטורי המאפיינים של כל מחלקה בהתאם לחלוקה הנתונה ב-ground_truth.mat (חלק ב').

בחרו בעצמכם את הפרמטרים השונים.

יש לשמור את ווקטורי המאפיינים של המחלקות (זה שעבורם אתם מייצרים את שאר הקבצים להגשה) בקובץ 'my_class_hist.mat' תחת המשתנה *'class_hist'* (יש לשמור משתנה מסוג *struct* במבנה זהה למבנה המופיע ב-test_data.mat במשתנה *'class_hist'*). **<u>יהיה עליכם לצרף קובץ זה לפתרון.</u>**

ניתן לשמור שלבי ביניים לקבצים, אולם **<u>אין</u>** להגיש קבצים אלה.

מומלץ שלא להריץ לימוד על תמונות באיכות junk.

מומלץ להיעזר בקבצי הבדיקה המצורפים לתרגיל (בעיקר test_4 ו-test_2).

מומלץ לבדוק איזה אחוז מתמונות הלימוד מסווגות בצורה נכונה (בדומה לנעשה ב-test_1).

35. הריצו את הקובץ test_9_check_files.m.

בבדיקה זו בודקים את תקינות הקבצים ששמרתם. וודאו שהרצתה עוברת באופן תקין.

36. בנו פונקציה image_search.m המקבלת בקלט תמונה לא מעובדת (לא שם קובץ אלא תמונה טעונה) בגווני אפור, מעבדת אותה, משייכת אותה למחלקה ומציגה 9 תמונות אקראיות מאותה המחלקה.

על הפונקציה לעשות שימוש במידע השמור בקבצים הבאים **<u>בלבד</u>**:

my_brief_coordinates.mat ; my_code_book.mat ; my_class_hist.mat

כמו גם במידע השמור ב-ground_truth.mat (לבחירת 9 תמונות מאותה מחלקה).

37. סכמו את הפרמטרים שבהם השתמשתם לבניית מערכת חיפוש תמונות:

- N_B – גודל *BRIEF-Descriptor* בודד.
- N_C – גודל ה-*Code-Book*.
- N_{MP} – מספר נקודות העניין (פינות) המקסימלי לתמונה.
- N_D – מספר הנקודות הכולל שהשתמשתם לבניית ה-*Code-Book*.

תארו בקצרה את המוטיבציה מאחורי בחירת כל פרמטר.

מה דעתכם עכשיו על הביטוי "תמונה אחת שווה אלף מילים"?

חלק ח': סעיפי בונוס

מזל טוב. עכשיו כשסיימתם את התרגיל, לפניכם מספר סעיפים להעשרה והתנסות נוספת.

שימו לב כי ניקוד יינתן על סעיפים אלה רק במידה והושלמו כל סעיפי התרגיל עד כה. כמו כן, הניקוד מותנה בכך שהאלגוריתם שתציעו עובד באופן מלא ותקין, ומצורף לפתרון הקוד המלא שלו (אין צורך להעתיק קוד למסמך) כמו גם הסבר על אופן פעולתו והפעלתו.

בכל אופן, ציון תרגיל הבית לא יחרוג מ-100 נקודות.

38. הציעו אלגוריתם חלופי לאחד החלקים א'-ו', ממשו אותו, ובדקו את ביצועיו (שימו לב לא לדרוס את הפונקציות והקבצים הנדרשים להגשה במסגרת התרגיל). תארו בקצרה במסמך את האלגוריתם, אופן פעולתו ואופן הפעלתו.

39. בעבודתנו על תמונות בגווני אפור, איבדנו מידע רב הקיים בצבעים של התמונות.

עדכנו את המערכת שהוצגה בתרגיל בית זה לעבודה עם תמונות צבעוניות, ונתחו את הביצועים (שימו לב לא לדרוס את הפונקציות והקבצים הנדרשים להגשה במסגרת התרגיל).

הוראות הגשה:

1. ההגשה בזוגות.

2. יש להגיש מסמך המכיל התייחסות לכלל הסעיפים בתרגיל, המציג את כל התוצאות ועונה על כל השאלות.

3. יש לצרף את כל פונקציות ה-MATLAB שהוגדר להן API בתרגיל, וכן כל הפונקציות הנלוות אליהן שכתבתם. בדיקת התרגיל תכלול הרצה אוטומטית של פונקציות אלו ובדיקתן.

4. יש לצרף לפתרון המוגש את תוצרי הביניים הבאים:
   1. my_brief_coordinates.mat
   2. my_code_book.mat
   3. my_class_hist.mat

הנכם מתבקשים **שלא** לצרף קבצי ביניים נוספים (בדגש על קבצי tmp_*.mat), שתופסים נפח זיכרון גדול. הנ"ל תופס גם לקבצים שנעשה בהם שימוש בחלק מקבצי הבדיקה.
(בקיצור, אל תצרפו את קבצי tmp_*.mat ואת test_data.mat).

5. את כל הקבצים המצורפים למייל (כולל המסמך) יש לכווץ ל-zip.

6. יש לבדוק שאתם מצליחים להריץ את כל קבצי הבדיקה (test_#_*.m) המצורפים, ללא ביצוע שינויים בהם, וכאשר תיקיית העבודה של MATLAB היא התיקייה בה נמצאים הקבצים.

7. בכדי לא לאבד נקודות, מומלץ לבדוק שהקוד שכתבתם פועל גם על מחשב בלתי תלוי במחשב בו כתבתם אותו (למשל במעבדות באוניברסיטה), ולוודא שהרצה של כל אחת מפונקציות הבדיקה מניבה תוצאה רצויה.

8. **את הפתרון עם כל הקבצים הרלוונטיים (zip) יש להגיש למייל: ATICV2016@gmail.com.**
   את הנושא של המייל יש לנסח באופן הבא:
   Assignment #3 ID: id-number1_id-number2

9. איחור במועד ההגשה יגרור הורדה בציון.

בהצלחה!