בית הספר להנדסה ומדעי המחשב ע"ש רחל וסלים בנין

מבוא למדעי המחשב 67101 – סמסטר ב' 2021/2022

תרגיל 6 – עיבוד תמונה ועבודה עם רשימות רב-מימדיות

להגשה בתאריך **8/5/2022** בשעה 22:00

הקדמה

בתרגיל זה נתרגל שימוש בלולאות ורשימות רב מימדיות וניחשף לכלים בסיסיים בעיבוד תמונה. התרגיל מורכב בצורה מובנית ממספר משימות, שבסופו של דבר יתחברו ביחד וירכיבו את המוצר הסופי. שימו לב - **אין צורך בידע קודם בעיבוד תמונה**, את כל הידע הנדרש ריכזנו עבורכם בסרטון וידאו הנגיש באתר הקורס, צפו בו ורק לאחר מכאן המשיכו בקריאה.

עקבו אחר ההוראות והשלבים של התרגיל, והקפידו לכתוב את הקוד שלכם במדויק על פי הנחיות התרגיל. כמו כן, מומלץ בחום לקרוא את כלל התרגיל לפני תחילת הפתרון.

אנו ממליצים להתחיל לעבוד על התרגיל בשלב מוקדם שכן התרגיל ארוך מקודמיו.

לרשותכם קובץ בשם **cartoonify.py** עם חתימות הפונקציות שעליכם לממש בתרגיל. **אין לשנות חתימות אלו**.

הוספנו type hints לחתימות הפונקציות כדי להבהיר מה הם הטיפוסים של הפרמטרים וערכי ההחזרה של הפונקציות.

אתם רשאים לממש פונקציות עזר נוספות משלכם, ולהשתמש בהן בקוד שלכם. לאורך כל התרגיל ניתן להניח קלט תקין.

הקובץ ex6_helper.py

בקובץ העזר מימשנו עבורכם מספר פונקציות שיעזרו לכם בטעינת התמונה ובצפייה בתמונות השונות הנוצרות במהלך התוכנית שתכתבו:

```
load_image(image_path)
```

מקבלת ניתוב לתמונה צבעונית ומחזירה ייצוג של התמונה כרשימה תלת-מימדית כפי שתיארנו בתרגול.

```
save_image(image, path)
```

מקבלת תמונה וניתוב ושומרת את התמונה בניתוב הנתון.

```
show_image(image)
```

מקבלת תמונה ומציגה אותה.

<u>שימו לב</u>: אין לעשות שום שינוי בקובץ זה ואין להגישו.

הספרייה PIL

הקובץ ex6_helper.py – הנתון לכם בתרגיל זה עושה שימוש בספריה PIL של פייתון, ולכן צריך אותה על מנת להריץ את התרגיל. **במעבדת המחשבים של האוניברסיטה (האקוריום) כבר מותקנת ספריה זו, ואין צורך להתקין שום דבר.**

לעבודה על המחשב האישי, בדקו אם החבילה מותקנת (היא כלולה ב-WinPython) ואם לא התקינו אותה באמצעות הפקודה:

```
python3 -m pip install Pillow
```

במידה וההתקנה נכשלה, יתכן שעליכם לעדכן את גרסת ה pip. הריצו את הפקודות הבאות:

```
python3 -m pip install --upgrade pip
python3 -m pip install --upgrade Pillow
```

עבודה עם תמונות צבעוניות

הפרדה לערוצי צבע

תמונה צבעונית מורכבת מרשימה תלת-מימדית כאשר המימד האחרון מתאר את כל הצבעים של פיקסל מסוים. במקרה הנפוץ עובדים עם ייצוג RGB – כלומר ישנם שלושה ערכים שמייצגים את רמת הבהירות של אדום, ירוק וכחול.

בתרגיל, כשנעבוד עם תמונות צבעוניות, נעבוד על כל **ערוץ צבע** בנפרד, ולכן ניצור מטריצה אחת שמכילה את כל הגוונים האדומים, אחת שמכילה את כל הירוקים ואחת שמכילה את כל כחולים, וכך נעבוד על רשימות דו-מימדיות.

1. נפריד תמונות צבעוניות לערוצים נפרדים, ונחבר אותם חזרה לתמונה צבעונית.

   i. ממשו את הפונקציה:

`separate_channels(image)`

שמקבלת תמונה (רשימה תלת-מימדית) שמימדיה rows × columns × channels ומחזירה רשימה תלת-מימדית, שמימדיה channels × rows × columns, כלומר רשימה של תמונות דו-מימדיות שכל אחת מייצגת ערוץ צבע בודד.

ניתן להניח שהפונקציה מקבלת תמונה צבעונית חוקית. אין לשנות את תמונת המקור.

לדוגמא:

```
separate_channels([[[1, 2]]]) → [[[1]], [[2]]]
separate_channels([[[1, 2, 3]]*3]*4) → [[[1]*3]*4, [[2]*3]*4, [[3]*3]*4]
```

   ii. ממשו את הפונקציה:

`combine_channels(channels)`

שעושה את ההפך מהפונקציה הקודמת, כלומר מקבלת רשימה באורך channels של תמונות דו-מימדיות המורכבות מערוצי צבע בודדים, ומאחדת אותם לכדי תמונה צבעונית אחת שמימדיה rows × columns × channels.

ניתן להניח שהפונקציה מקבלת קלט תקין, של רשימה לא ריקה של תמונות דו- מימדיות (כלומר קיים לפחות ערוץ אחד). אין לשנות את רשימת המקור.

לדוגמא:

```
combine_channels([[[1]], [[2]]]) → [[[1, 2]]]
combine_channels([[[1]*3]*4, [[2]*3]*4, [[3]*3]*4]) → [[[1, 2, 3]]*3]*4
```

שימו לב: במקרה שתיארנו (RGB) יש שלושה ערוצי צבע, אבל לא תמיד זהו המצב. יש עוד מרחבי צבעים ועוד ייצוגים של תמונות בהם עשוי להיות מספר שונה של ערוצי צבע. כתבו את הפונקציות באופן גנרי.

מעבר לגווני שחור לבן

תמונה בגווני שחור לבן היא תמונה המורכבת מערוץ יחיד, כמו בדוגמא הבאה `[[[1], [2]], [[3], [4]]]`. **נהוג לוותר על המימד השלישי (של ה channels) בייצוג של תמונות עם ערוץ יחיד ולהשתמש ברשימה דו-מימדית (במקום תלת-מימדית). למשל את התמונה בדוגמא נייצג ע"י הרשימה הדו-מימדית הבאה** `[[1, 2], [3, 4]]` **במקום. מעתה ואילך בתרגיל, נייצג תמונות המורכבות מערוץ יחיד, ובפרט תמונות בגווני שחור לבן <u>ברשימות דו-מימדיות</u>.**

2. ממשו את הפונקציה:

```
RGB2grayscale(colored_image)
```

הפונקציה מקבלת תמונה צבעונית (**רשימה תלת מימדית** כפי שהוסבר לעיל) ומחזירה תמונה בגווני שחור לבן (**רשימה דו-מימדית**).

ניתן להניח שהפונקציה מקבלת תמונה צבעונית חוקית בפורמט **RGB** (כלומר כל פיקסל מורכב מ-3 ערכים).

<u>אין לשנות את תמונת המקור.</u>

הפיכת תמונה צבעונית לתמונה בגווני שחור לבן נעשית על ידי מיצוע מסויים של ערכי הפיקסלים הצבעוניים לכדי ערך אחד. הנוסחא בה נשתמש היא:

$$\text{RED} \cdot 0.299 + \text{GREEN} \cdot 0.587 + \text{BLUE} \cdot 0.114$$

כאשר <u>יש לעגל את התוצאה לשלם הקרוב ביותר</u>.

לדוגמא:

```
RGB2grayscale([[[100, 180, 240]]]) → [[163]]
RGB2grayscale([[[200, 0, 14], [15, 6, 50]]]) → [[61, 14]]
```

טשטוש

ראינו בתרגול כיצד מטשטשים תמונה באמצעות שימוש בקרנל. כעת נממש טכניקה זו.

3. ממשו את הפונקציה:

```
blur_kernel(size)
```

המחזירה קרנל החלקה בגודל size × size כרשימה של רשימות.

קרנל ההחלקה בו נשתמש בתרגיל לא יהיה הקרנל הגאוסיאני שראינו בסרטון, אלא קרנל הממצע את כל השכנים בצורה שווה, כלומר כל תא בקרנל מכיל את הערך $\frac{1}{\text{size}^2}$.

ניתן להניח כי size הינו מספר שלם, חיובי **<u>ואי זוגי.</u>**

לדוגמא:

```
blur_kernel(3) → [[1/9, 1/9, 1/9], [1/9, 1/9, 1/9], [1/9, 1/9, 1/9]]
```

4. ממשו את הפונקציה:

`apply_kernel(image , kernel)`

הפונקציה מקבלת תמונה בעלת ערוץ צבע יחיד (קרי רשימה דו-מימדית) וקרנל (גם הוא רשימה דו-מימדית), ומחזירה תמונה בגודל זהה לזה של התמונה המקורית, כאשר כל פיקסל בתמונה החדשה מחושב באמצעות הפעלת הקרנל עליו.

כלומר:

מזהים את הפיקסל image[row][column] עם הכניסה המרכזית במטריצה kernel, וסוכמים את ערכי שכניו (כולל הפיקסל עצמו) כפול הכניסה המתאימה להם ב-kernel (ראו סרטון).

ניתן להניח שהפונקציה מקבלת תמונה חוקית בעלת ערוץ צבע יחיד (רשימה דו-מימדית), ושהקרנל בגודל $size \times size$ כאשר $size$ מספר טבעי **אי-זוגי**.

אין לשנות את תמונת המקור.

לדוגמא:

```
apply_kernel([[0, 128, 255]], blur_kernel(3)) → [[14, 128, 241]]
apply_kernel([[10, 20, 30, 40, 50], [8, 16, 24, 32, 40], [6, 12, 18, 24, 30], [4, 8, 12, 16, 20]] , blur_kernel(5)) → [[12, 20, 26, 34, 44], [11, 17, 22, 27, 34], [10, 16, 20, 24, 29], [7, 11, 16, 18, 21]]
```

שימו לב:

- **במידה והסכום אינו שלם, יש לעגלו לשלם הקרוב ביותר.**
- **במידה והסכום קטן מ-0 יש להתייחס אליו כאל 0, ואם הוא גדול מ-255 יש להתייחס אליו כאל 255.**
- **בחישוב ערך לפיקסל x הנמצא על גבולות התמונה, יש להתייחס לערכי פיקסלים הנמצאים מחוץ לגבולות תמונת המקור כאילו היו בעלי ערך זהה לזה של הפיקסל x.**

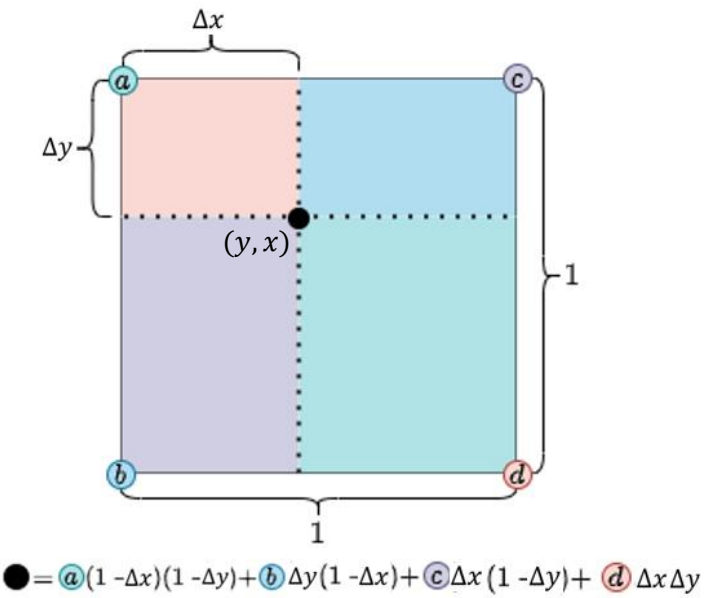
Resize

במקרים רבים נרצה להיות מסוגלים לשנות את גודל התמונה. כפי שראינו בסרטון, בביצוע resize אנחנו מחשבים את הערך של כל פיקסל בתמונת היעד (בגודל החדש) לפי הערכים של הפיקסלים הקרובים ביותר לקואורדינטה בה הוא "נופל" בתמונת המקור. נעשה זאת באמצעות אינטרפולציה כפי שראינו בסרטון.

5. ממשו את הפונקציה:

`bilinear_interpolation(image, y, x)`

המקבלת תמונה בעלת ערוץ צבע יחיד (רשימה דו-מימדית) ואת הקואורדינטות של פיקסל מתמונת היעד כפי שהן "נופלות" בתמונת המקור (y הקואורדינטה לאורך התמונה, כלומר בשורות ו-x לרוחב התמונה, כלומר בעמודות) ומחזירה את ערך אותו הפיקסל (מספר שלם בין 0 ל-255) לפי החישוב:



ניתן להניח שהפונקציה מקבלת תמונה חוקית בעלת ערוץ צבע יחיד (רשימה דו-מימדית).

ניתן להניח שהקואורדינטות x,y הן בתוך גבולות התמונה (ויכולות גם להיות על הגבולות ממש).

לדוגמא:

```
bilinear_interpolation([[0, 64], [128, 255]], 0, 0) → 0
bilinear_interpolation([[0, 64], [128, 255]], 1, 1) → 255
bilinear_interpolation([[0, 64], [128, 255]], 0.5, 0.5) → 112
bilinear_interpolation([[0, 64], [128, 255]], 0.5, 1) → 160
```

<u>שימו לב:</u>

- **הקואורדינטות x,y יכולות להיות מורכבות ממספרים לא שלמים.**
- **במידה והערך המתקבל מהחישוב אינו שלם, יש לעגלו לשלם הקרוב ביותר.**

6. ממשו את הפונקציה:

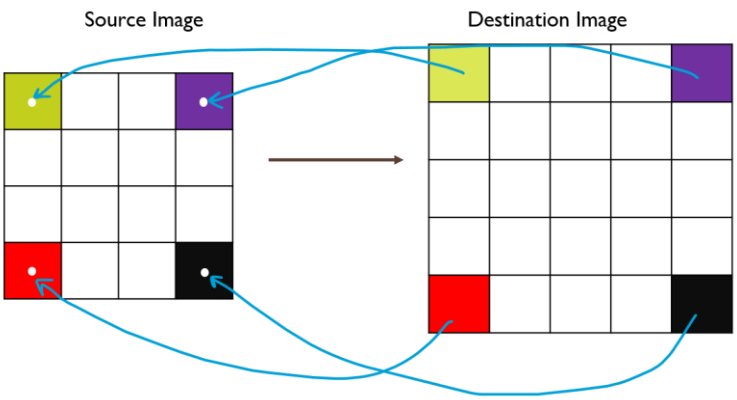
```
resize(image , new_height, new_width)
```

המקבלת תמונה בעלת ערוץ צבע יחיד (רשימה דו-מימדית) ושני מספרים שלמים, ומחזירה תמונה חדשה בגודל new_height × new_width כך שערכו של כל פיקסל בתמונה המוחזרת מחושב בהתאם למיקומו היחסי בתמונת המקור.

מיקומו במקור יחושב באופן הבא:

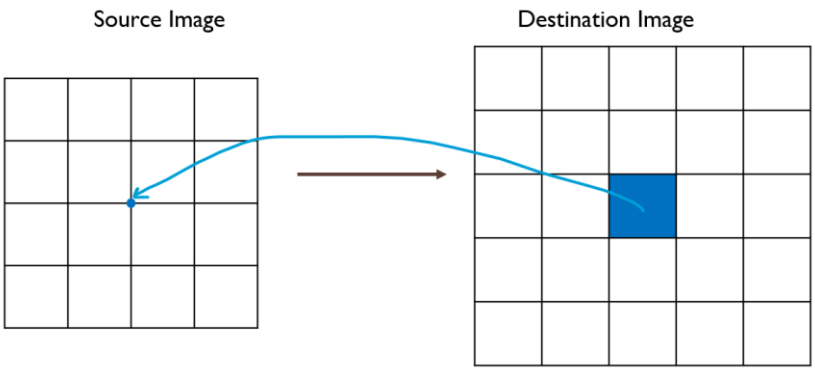
- פינות ימופו לפינות.

לדוגמא הפיקסל בפינה השמאלית העליונה (בשורה האפס ובעמודה האפס) בתמונת היעד, ימופה לפיקסל בפינה השמאלית העליונה (בשורה האפס ובעמודה האפס) בתמונת המקור: $(0,0) \rightarrow (0,0)$.



- כל נקודה אחרת (שאינה פינה), תמופה באופן פרופורציונלי למיקומה ביחס לפינות.

לדוגמא עבור תמונת מקור מגודל 4×4, הפיקסל האמצעי בתמונת יעד מגודל 5×5 (זה שבשורה 2 ועמודה 2) ימופה להיות בדיוק במרכז של תמונת המקור: $(2,2) \rightarrow (1.5,1.5)$.



ניתן להניח שהפונקציה מקבלת תמונה דו-מימדית חוקית (ערוץ צבע יחיד) וכי המימדים החדשים הינם מספרים שלמים וחיוביים הגדולים מ-1.

אין לשנות את תמונת המקור.

ממשו את הפונקציה:

```
scale_down_colored_image(image, max_size)
```

המקבלת תמונה **צבעונית** (רשימה תלת מימדית) ומספר שלם חיובי המייצג את מספר הפיקסלים המקסימלי שאנחנו מעוניינים לאפשר לתמונה בכל כיוון (אורך ורוחב). הפונקציה תבדוק האם התמונה עומדת באילוץ זה (בשני הכיוונים):

- אם כן: הפונקציה תחזיר None.
- אם לא: הפונקציה תחזיר תמונה צבעונית חדשה שהיא הקטנה של תמונת הקלט לגודל המקסימלי שעומד באילוץ תוך כדי שמירת הפרופורציות המקוריות של התמונה.

רמז: לצורך הקטנת תמונה צבעונית השתמשו בפונקציה resize על כל ערוץ בנפרד, אך הקפידו להחזיר תמונה צבעונית (כלומר במימדים rows × columns × channels).

אין לשנות את תמונת המקור.

סיבוב ב-90 מעלות

7. ממשו את הפונקציה:

`rotate_90(image, direction)`

המקבלת תמונה (צבעונית או בעלת ערוץ יחיד) וכיוון (מחרוזת שהיא 'R' או 'L') ומחזירה תמונה דומה, מסובבת ב-90 מעלות לכיוון המבוקש.

ניתן להניח שהפונקציה מקבלת תמונה חוקית ושהקלט direction תקין. שימו לב שהתמונה יכולה להיות צבעונית – תלת מימדית, או בעלת ערוץ יחיד – דו-מימדית, על הקוד שלכם לתמוך בשני המקרים.

אין לשנות את תמונת המקור.

לדוגמא:

```
rotate_90([[1, 2, 3], [4, 5, 6]], 'R') → [[4, 1], [5, 2], [6, 3]]
rotate_90([[1, 2, 3], [4, 5, 6]], 'L') → [[3, 6], [2, 5], [1, 4]]
rotate_90([[[1, 2, 3], [4, 5, 6]], [[0, 5, 9], [255, 200, 7]]], 'L') →
 [[[4, 5, 6], [255, 200, 7]], [[1, 2, 3], [0, 5, 9]]]
```

זיהוי גבולות

גבול בתמונה הוא קו מתאר של אובייקט מסויים בה. בהרבה בעיות שונות בעיבוד תמונה נרצה לדעת לזהות ולבודד את קווי המתאר הללו, וישנן מספר שיטות לזיהוי גבולות. שיטות אלה מתבססות על מעברים חדים בין גוונים. מעבר חד בגוונים מעיד על שינוי בתמונה, כלומר הופעה של אובייקט חדש, ועל כן הם מצביעים על גבול. ראינו בסרטון שיטה לזיהוי גבולות הנקראת adaptive threshold אותה נממש כעת. נזכיר בקצרה, ב-adaptive threshold אנו מחשבים לכל פיקסל ערך סף שהוא ממוצע השכנים שלו בסביבה מסויימת (בגודל block_size × block_size) פחות קבוע כלשהו c. אם פיקסל מסויים כהה מהסביבה שלו, סימן שסביבו יש פיקסלים שערכם גבוה משלו (שכן שחור זה 0 ולבן זה 255) ולכן ערכו יהיה קטן מהממוצע בסביבתו. הקבוע c אותו אנו מפחיתים מהערך הממוצע מאפשר לנו לדרוש שהפיקסל יהיה כהה משמעותית מהסביבה שכן ערכו צריך להיות אפילו קטן מהממוצע פחות אותו קבוע.

8. ממשו את הפונקציה:

`get_edges(image, blur_size, block_size, c)`

המקבלת תמונה בגווני שחור לבן (רשימה דו-מימדית) ושלושה מספרים, ומחזירה תמונה חדשה, בעלת אותם מימדים, המורכבת משני ערכים בלבד (שחור ולבן) כאשר פיקסלים שחורים מסמנים גבולות בתמונה.

ניתן להניח שהפונקציה מקבלת תמונה דו-מימדית חוקית (ערוץ צבע יחיד), ש-block_size ו-blur_size הם מספרים שלמים, חיוביים **ואי-זוגיים** וש-c הוא מספר אי-שלילי.

אין לשנות את תמונת המקור.

לדוגמא:

```
get_edges([[200, 50, 200]], 3, 3, 10) → [[255, 0, 255]]
```

שימו לב:

- **בשביל למצוא גבולות בתמונה עליכם לטשטש תחילה את התמונה באמצעות הפרמטר `blur_size`.**
- **נסמן $r = block\_size // 2$, אז הנוסחא ל-$threshold$ היא:**

$$threshold[i][j] = avg(blurred\_image[i-r:i+r+1][j-r:j+r+1]) - c$$

שימו לב, זהו ביטוי מופשט – לא קוד בפייתון, הביטוי מתאר ממוצע של ערכי התאים הנמצאים בריבוע עם צלע מאורך block_size תאים סביב התא ה i,j בתמונה, פחות הערך c.

(בחישוב הממוצע, יש להתייחס לערכי פיקסלים הנמצאים מחוץ לגבולות התמונה כאילו היו בעלי ערך זהה לזה של הפיקסל אותו אתם בודקים בתמונה המטושטשת).

- **אם $blurred\_image[i][j] < threshold[i][j]$ אז $new\_image[i][j]$ יהיה שחור ואחרת לבן.**
- **כדי להימנע מכפל קוד, היעזרו בפונקציות קודמות שמימשתם ככל הניתן.**

צמצום מספר הצבעים בתמונה (קוונטיזציה)

בתהליך הקוונטיזציה (Quantization), אנחנו מצמצמים מגוון של ערכים לכדי ערך בודד – למשל בוחרים גוון ספציפי של אדום שיחליף 10 גוונים שונים. למעשה אנחנו מחלקים את 256 הגוונים שלנו למספר מסויים של גוונים.

קיימים מספר אלגוריתמים מתחום למידת המכונה שמטרתם לבחור את הגוונים האופטימלים שנרצה לשמור, ואת הדרך הנכונה לשייך כל גוון מקורי לגוון החדש, את אלגוריתמים אלה עוד תראו בקורסים עתידיים. אנחנו נשתמש בחישוב פשוט יותר אשר בוחר N גוונים במרחקים שווים, מבלי לבחור גוונים אופטימליים, ומעביר כל גוון בתמונה המקורית לגוון הכי קרוב אליו מבין N הגוונים שבחרנו להשאיר. נשתמש בנוסחא:

$$qimg[i][j] = round\left(floor\left(img[i][j] \cdot \frac{N}{256}\right) \cdot \frac{255}{N-1}\right)$$

9. ממשו את הפונקציה:

```
quantize(image, N)
```

שמקבלת תמונה כרשימה דו-מימדית (בעלת ערוץ צבע יחיד) ומחזירה תמונה בעלת מימדים זהים, בה הערכים של הפיקסלים מחושבים לפי הנוסחא לעיל.

ניתן להניח שהפונקציה מקבלת תמונה דו-מימדית חוקית (ערוץ צבע יחיד) וכי N הוא מספר טבעי גדול מ-1.

אין לשנות את תמונת המקור.

לדוגמא:

```
quantize([[0, 50, 100], [150, 200, 250]], 8) → [[0, 36, 109], [146, 219, 255]]
```

לקוונטיזציה של תמונה צבעונית יש להפעיל את הקוונטיזציה שמימשנו עבור תמונה בעלת צבע יחיד על כל ערוץ צבע בנפרד, לכן בתמונה הסופית מספר הגוונים יהיה $N^{\#channels}$ (וודאו שאתם מבינים מדוע).

10. ממשו את הפונקציה:

`quantize_colored_image(image, N)`

שמקבלת תמונה צבעונית (רשימה תלת-מימדית) ומחזירה תמונה דומה לאחר קוונטיזציה ל-$N^{\#channels}$ גוונים.

חישבו – באלו מהפונקציות שכתבתם עד כה עליכם להשתמש?

ניתן להניח שהפונקציה מקבלת תמונה צבעונית חוקית, וכי N הוא מספר טבעי גדול מ-1.

אין לשנות את תמונת המקור.

חיבור תמונות באמצעות Mask

ננסה ליצור שילובים מעניינים בין זוג תמונות בעזרת תמונת mask שקובעת כמה לקחת מכל תמונה.

11. ממשו את הפונקציה:

`add_mask(image1, image2, mask)`

המקבלת 2 תמונות במימדים זהים (rows × columns × channels), ורשימה דו-מימדית שמימדיה תואמים את מימדי ערוץ יחיד בתמונות (rows × columns) ועריכה נעים בתחום [0, 1], ומחזירה תמונה חדשה בה כל פיקסל מחושב לפי הנוסחה הבאה:

new_image[i][j] = round(image1[i][j] x mask[i][j] + image2[i][j] x (1 - mask[i][j]))

ניתן להניח ש-image1, image2 הן תמונות חוקיות בנות אותם מימדים (שתיהן צבעוניות או שתיהן בעלות ערוץ צבע יחיד) וש-mask חוקית כמתואר.

אין לשנות את תמונת המקור.

לדוגמא:

```
add_mask([[[1,2,3], [4,5,6]],[[7,8,9],[10,11,12]]], [[[250,250,250],
[0,0,0]],[[250,250,100],[1,11,13]]],  [[0, 0.5]]*2) → [[[250, 250, 250], [2,
2, 3]], [[250, 250, 100], [6, 11, 12]]]
add_mask([[50, 50, 50]], [[200, 200, 200]], [[0, 0.5, 1]]) → [[200, 125, 50]]
```

שימו לב: בתמונה בתחילת סעיף זה הצגנו את ה-mask כתמונה, כשלמעשה ערכי ה-mask הם בין 0 ל-1 ולא בין 0 ל-255. ניתן לחשוב על כל mask כעל תמונת שחור לבן מנורמלת לתחום [0, 1] (פשוט נחלק את כל ערכיה ב-255). כך אתם יכולים ליצור בעצמכם mask מכל תמונה בשחור לבן.

Cartoonify

ניעזר בכל הפונקציות שמימשנו עד כה לכתיבת תוכנית אשר מקבלת כקלט תמונה צבעונית ומחזירה את התמונה עם אפקט של איור:

כפי שאתם רואים, אפקט זה מתקבל מהדגשת הגבולות בתמונה יחד עם צמצום מספר הצבעים בה.

12. ממשו את הפונקציה:

`cartoonify(image, blur_size, th_block_size, th_c, quant_num_shades)`

המקבלת תמונה צבעונית ואת כל הפרמטרים בהם השתמשנו במהלך התרגיל ואיתם ניתן לשחק:

- **`blur_size`** – גודל קרנל הטשטוש, מספר טבעי **אי-זוגי**
- **`th_block_size`** – גודל הסביבה אותה נמצע לקביעת ה-threshold של כל פיקסל, מספר טבעי אי-זוגי
- **`th_c`** – הקבוע אותו נחסיר מהערך הממוצע שחישבנו לקבלת ה-threshold הסופי
- **`quant_num_shades`** – מספר הגוונים בהם נשתמש בשלב הקוונטיזציה

ומחזירה תמונה צבעונית שהיא הגרסה המאויירת של תמונת המקור.

על מנת לעשות זאת עליכם:

1. **לחלץ מהתמונה את קווי המתאר שבה.**
2. **לבצע קוונטיזציה לערוצי הצבע.**

3. לחבר את 2 הקודמים, כלומר להוסיף את קווי המתאר שמצאנו ע"ג התמונה לאחר קוונטיזציה.

חישבו:

- **מתי אנו עובדים על תמונה צבעונית ומתי בגווני שחור לבן?**
- **איך add_mask יכולה להועיל בהוספה של קווי המתאר? מי תהיה תמונת ה-mask? כיצד נתחום אותה בתחום [0,1]?**
- **כאשר אנו עובדים על תמונה צבעונית – מתי נרצה להפריד את התמונה לערוצי צבע? אילו פעולות נבצע על כל ערוץ בנפרד? באיזה שלב נאחד הכל חזרה?**

ניתן להניח שהפונקציה מקבלת תמונת RGB חוקית וערכים חוקיים בעבור כל הפרמטרים.

אין לשנות את תמונת המקור.

נתבונן בתוצאות

בנוסף לפונקציות שמימשתם עד כה, עליכם לכתוב מקטע קוד שמריץ את התוכנית ושומר את התוצאה (תמונת ה-cartoon) לקובץ. מקטע זה יכתב תחת

```
if __name__==__"main__":
```

ויקבל ארגומנטים משורת הפקודה. הרצת התכנית תתבצע ע"י הפקודה:

```
python3 cartoonify.py <image_source> <cartoon_dest> <max_im_size>
 <blur_size> <th_block_size> <th_c> <quant_num_shades>
```

כאשר:

- **image_source** – ניתוב תמונת המקור
- **cartoon_dest** – הניתוב בו תישמר תמונת ה-cartoon. הקפידו לשמור את התמונה עם סיומת png.
- **max_im_size** – הגודל המקסימלי שאנחנו מאפשרים לתמונה, לטובת הקטנתה אם יש צורך.

שאר הפרמטרים תואמים את התיאור בפונקציה `cartoonify()`.

יש לוודא את תקינות מספר הארגומנטים.

במידה ומספר הארגומנטים שונה מהמצופה, יש להדפיס הודעת שגיאה ולסיים את הריצה.

דוגמא להרצה:

```
python3 cartoonify.py ziggy.png ziggy_cartoon.png 460 5 13 11 8
```

הרצת שורה זו תוביל לשרשרת האירועים הבאה:

- טעינה של התמונה שבניתוב `image_source` באמצעות הפונקציה **load_image** שבקובץ העזר.
- בדיקה שגודל התמונה לא עולה על `max_im_size` בכל כיוון (אורך ורוחב) , ובמידת הצורך הקטנת התמונה כדי שתעמוד באילוץ הגודל תוך שמירה על פרופורציות המקור של התמונה.
- הרצת **cartoonify** על התמונה המוקטנת עם הפרמטרים הנתונים.
- שמירת התוצאה בניתוב `cartoon_dest` בעזרת הפונקציה **save_image** שבקובץ העזר.

נסו לשחק מעט עם הפרמטרים וראו כיצד הם משפיעים על התוצאה הסופית. חשבו אילו פרמטרים מתאימים לתמונה עם יותר פרטים ולכזו עם פחות. נסו לנסח לעצמכם על מה משפיע כל פרמטר ומה יקרה אם תגדילו / תקטינו אותו.

הוראות הגשה

עליכם להגיש קובץ בשם **ex6.zip** בקישור ההגשה של תרגיל 6 דרך אתר הקורס על ידי לחיצה על "Upload file". הקובץ **ex6.zip** צריך להכיל אך ורק את הקובץ **cartoonify.py**.

לפני שאתם מתחילים – טיפים והנחיות

- הקפידו לכתוב תיעוד לקוד שלכם ובפרט לכל פונקציה שאתם כותבים.
- לרשותכם התיקייה examples.zip, המכילה שתי דוגמאות של תמונות לפני ואחרי הרצת התכנית. כל דוגמה מורכבת מתמונה מקורית, תמונת ה-edges ותמונת ה-cartoon שלה. שימו לב, אלו לא בהכרח הקבצים שעליהם התרגיל יבדק.
- אנו מעודדים אתכם לבחון את התרגיל גם עם תמונות שלכם! מכיוון שזמן הריצה תלוי במספר הפיקסלים אנו ממליצים לעבוד בתחילת התרגיל עם תמונות קטנות, או להקטין את התמונות באמצעות הפונקציה resize.
- בתרגיל זה, כל פעולה על תמונה צבעונית תבוצע על כל ערוץ צבע בנפרד. באופן זה ניתן לנצל פונקציות שתומכת בערוץ צבע יחיד גם בעבור תמונות צבעוניות.
- ניתן להניח תקינות הקלטים לכל אחד מהסעיפים (בהתאם להגדרת הפרטנית של כל סעיף). בפרט, ניתן להניח כי כל התמונות ניתנות בפורמט תקין (רשימה של רשימות, שבכל אחת מהן אותו מספר פיקסלים), וכי כל הרשימות הן אכן רשימות לא ריקות.
- בכל הפונקציות בתרגיל זה <u>אין לבצע שום שינוי</u> בתמונות הקלט, אלא להחזיר תמונות חדשות!
- לצורך פיתרון התרגיל תוכלו להשתמש במודולים sys, copy ו-math. **אין להשתמש** במודולים numpy או PIL.

בהצלחה!