

שינויים בתרגיל כתובים באדום

מבוא למדעי המחשב 67101 – סמסטר א' 2021/2022

תרגיל 5 – עיבוד תמונה ועבודה עם רשימות רב-מימדיות

להגשה בתאריך **17/11/2021** בשעה 22:00

הקדמה

בתרגיל זה נתרגל שימוש בלולאות ורשימות רב מימדיות וניחשף לכלים בסיסיים בעיבוד תמונה. אנו ממליצים להתחיל לעבוד על התרגיל בשלב מוקדם שכן התרגיל ארוך מקודמיו. עם זאת, אל תיבהלו מאורך מסמך זה, הוא כולל מעט רקע ותמונות רבות. שימו לב, התרגיל מורכב בצורה מובנית ממספר משימות, שבסופו של דבר יתחברו ביחד וירכיבו את המוצר הסופי. לכן 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 (בדגש על סעיף הטיפים שבסוף המסמך). עליכם ליצור קובץ בשם **cartoonify.py** בו תממשו את התרגיל. אתם יכולים לכתוב בקובץ זה פונקציות עזר נוספות מלבד אלה הדרושות בתרגיל, ולהשתמש בהן בקוד שלכם. אבל הפונקציות הדרושות בתרגיל חייבות להיכתב בדיוק על פי הדרישות המפורטות להלן.

הקובץ `ex5_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.

שימו לב: אין לעשות שום שינוי בקובץ זה ואין להגישו.

הספרייה PIL

הקובץ `ex5_helper.py` – הנתון לכם בתרגיל זה עושה שימוש בספרייה PIL של פייתון, ולכן צריך אותה על מנת להריץ את התרגיל. **במעבדת המחשבים של האוניברסיטה (האקווריום) כבר מותקנת ספרייה זו, ואין צורך להתקין שום דבר.** ע"מ להשתמש בגרסה המותקנת באקווריום יש להשתמש ב-`interpreter` שבניתוב:

`/cs/course/2021/intro2cs1/introenv/bin/python3`

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

`python3 -m pip install Pillow`

עבודה עם תמונות צבעוניות

הפרדה לערוצי צבע

כפי שראינו בתרגול, 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** ו**כחול**. בתרגיל, כשנעבוד עם תמונות צבעוניות, נעבוד על כל **ערוץ צבע** בנפרד, ולכן ניצור מטריצה אחת שמכילה את כל הגוונים האדומים, אחת לירוקים ואחת לכחולים, ואז 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 X columns X channels` ומחזירה רשימה תלת-מימדית, שמימדיה `channels X rows X columns`, כלומר רשימה של תמונות דו-מימדיות שכל אחת מייצגת ערוץ צבע בודד.

ניתן להניח שהפונקציה מקבלת תמונה צבעונית חוקית (מלבנית, מכילה רק ערכים חוקיים בעלת יותר מערוץ צבע יחיד וכל הפיקסלים מורכבים מאותו מספר של צבעים ובאותו הסדר).

אין לשנות את תמונת המקור.

לדוגמא:

`separate_channels([[[[1, 2]]]]) → [[[1]], [[2]]]`

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 X columns X channels`.

ניתן להניח שהפונקציה מקבלת תמונה המורכבת לפחות מערוץ צבע יחיד חוקי (מלבני, דו-מימדי, מכיל רק ערכים חוקיים).

אין לשנות את רשימת המקור.

לדוגמא:

`combine_channels([[[[1]], [[2]]]]) → [[[1, 2]]]`

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

2. ממשו את הפונקציה:

`RGB2grayscale(colored_image)`

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

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

אין לשנות את תמונת המקור.

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

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

שימו לב: נוסחה זו כמעט נסכמת ל-1 אך לא במדויק, לכן יש לעגל את התוצאות לשלם הקרוב ביותר.

לדוגמא:

`RGB2grayscale ([[100, 180, 240]]) → [[163]]`

טשטוש

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

3. ממשו את הפונקציה:

`blur_kernel(size)`

המחזירה קרנל החלקה בגודל $\text{size} \times \text{size}$ כרשימה של רשימות.

קרנל ההחלקה בו נשתמש בתרגיל לא יהיה הקרנל הגאוסיאני שראינו בתרגול, אלא קרנל הממצע את כל השכנים בצורה

$$\text{שווה, כלומר כל תא בקרנל מכיל את הערך } \frac{1}{\text{size}^2}.$$

ניתן להניח כי size הינו מספר שלם אי זוגי.

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 הפיקסל `new_image[row][column]` 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`.

ניתן להניח שהפונקציה מקבלת תמונה חוקית בעלת ערוץ צבע יחיד (רשימה דו-מימדית), ושהקרנל בגודל $K \times K$ כאשר

K מספר טבעי אי-זוגי.

אין לשנות את תמונת המקור.

לדוגמא:

`apply_kernel([[0, 128, 255]], blur_kernel(3)) → [[14, 128, 241]]`

שימו לב:

- במידה וסכום זה 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 תמונות המקור כאילו היו בעלי ערך זהה לזה של הפיקסל x .

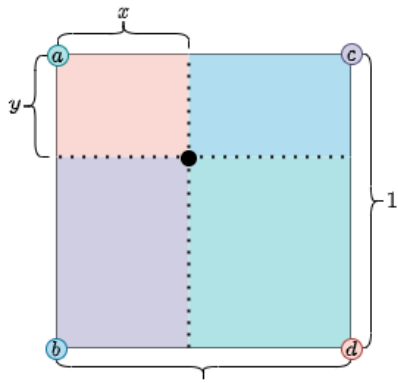
Resize

במקרים רבים נרצה להיות מסוגלים לשנות את גודל התמונה. כפי שראינו בתרגול, 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 בתרגול.

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:



$$\bullet = a(1-x)(1-y) + bxy(1-x) + cx(1-y) + dxy$$

(*שימו לב ש- x ו- y בחישוב אינם ה- x וה- y שברשימת הפרמטרים של הפונקציה)

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
```

שימו לב:

- 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 $\text{new_height} \times \text{new_width}$ כך שערכו של כל פיקסל בתמונה המוחזרת מחושב בהתאם למיקומו היחסי בתמונת המקור. ניתן להניח שהפונקציה מקבלת תמונה דו-מימדית חוקית (ערוץ צבע יחיד) וכי המימדים החדשים הינם מספרים שלמים וחיוביים.

אין לשנות את תמונת המקור.

שימו לב:

- פינות ממופות לפינות.
- במימוש נכון של הפונקציה, לא יתקבלו בעבור התמונה החדשה קוארדינטות מחוץ לגבולות התמונה המקורית.

סיבוב ב-90 מעלות

7. ממשו את הפונקציה:

`rotate_90(image, direction)`

המקבלת תמונה וכיוון (מחרוזת שהיא 'R' או 'L') ומחזירה תמונה דומה, מסובבת ב-90 מעלות לכיוון המבוקש. ניתן להניח שהפונקציה מקבלת תמונה חוקית (תמונה צבעונית או בעלת ערוץ צבע יחיד) ושהקלט בעבור `direction` תקין. 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]]`

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 גווניים. מעבר חד בגווניים מעיד על שינוי בתמונה, כלומר הופעה של אובייקט חדש, ועל כן הם מצביעים על גבול. ראינו בתרגול 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 $block_size \times block_size$) 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 בעלת ערוץ צבע יחיד (רשימה דו-מימדית) ושני 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- $block_size$ הוא מספר שלם, חיובי ואי-זוגי 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:

- כפי שהזכרנו בתרגול, לפני שמחפשים גבולות בתמונה מומלץ לטשטש אותה. תוכלו להשפיע על הטשטוש באמצעות הפרמטר `blur_size`.
- נסמן $r = block_size // 2$, אז הנוסחה ל- $threshold$ היא:
$$threshold[i][j] = avg(image[i - r : i + r + 1][j - r : j + r + 1])$$
- אם $c - threshold[i][j] < image[i][j]$ אז $new_image[i][j]$ יהיה שחור ואחרת לבן.
- בחישוב ערך לפיקסל x הנמצא על גבולות התמונה, יש להתייחס לערכי פיקסלים הנמצאים מחוץ לגבולות תמונות המקור כאילו היו בעלי ערך זהה לזה של הפיקסל x.
- כדי למנוע כפל 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 האופטימליים 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 להשאר. נשתמש בנוסחא:

$$qimg = \text{floor}\left(img \cdot \frac{N}{255}\right) \cdot \frac{255}{N}$$

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 חיובי.

אין לשנות את תמונת המקור.

שימו לב: לקבלת תמונה צבעונית יש להפעיל פעולה זו על כל ערוץ צבע בנפרד, לכן בתמונה הסופית מספר הגוונים יהיה $N^{\#channels}$ (וודאו שאתם מבינים מדוע).

לדוגמא:

`quantize([[0, 50, 100], [150, 200, 250]], 8) →`
`[[0, 32, 96], [128, 191, 223]]`

10. ממשו את הפונקציה:

quantize_colored_image(image, N)

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

חישובו – באלו מהפונקציות שכתבתם עד כה עליכם להשתמש?

ניתן להניח שהפונקציה מקבלת תמונה צבעונית חוקית, וכי N הוא מספר טבעי חיובי.

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 ורשימה דו-מימדית שמימדיה תואמים את המימדים הראשונים של התמונות וערכיה נעים בתחום [0, 1], ומחזירה תמונה חדשה בה כל פיקסל מחושב לפי הנוסחה הבאה:

$$\text{new_image}[i][j] = \text{round}(\text{image1}[i][j] \times \text{mask}[i][j] + \text{image2}[i][j] \times (1 - \text{mask}[i][j]))$$

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

אין לשנות את תמונת המקור.

למעוניינים, נסו ליצור בעצמכם תמונות מעניינות. כיצד תוכלו לעשות זאת בעבור תמונות שמימדיהן אינם זהים? נסו לעשות זאת עם פונקציות שכבר מימשתם.

לדוגמא:

`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 תמונת שחור 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 את התמונה איתה אתם רוצים לעבוד ואת כל הפרמטרים בהם השתמשנו במהלך התרגיל ואיתה 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 נחסר מהערך הממוצע שחישבנו לקבלת ה-threshold הסופי
- **quant_num_shades** – מספר הגוונים בהם נשתמש בשלב הקוונטיזציה

הפונקציה מחזירה את התמונה המאויירת.

ניתן להניח שהפונקציה מקבלת תמונה צבעונית חוקית וערכים חוקיים בעבור כל הפרמטרים.

אין לשנות את תמונת המקור.

שימו לב: פונקציה זו מקבלת תמונה צבעונית מקורית והופכת אותה לתמונה מאויירת. על מנת לעשות זאת היא צריכה לחלץ מהתמונה את קווי המתאר שבה, לבצע קוונטיזציה לערוצי הצבע ולהוסיף את קווי המתאר שמצאנו ע"ג התמונה לאחר קוונטיזציה.

חישבו:

- מתי אנו עובדים על תמונה צבעונית ומתי בגווי 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?

נתבונן בתוצאות

בנוסף לפונקציות שמימשת 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`
- `max_im_size` – הגודל המקסימלי שאנחנו מאפשרים לתמונה, לטובת הקטנתה אם יש צורך.

ושאר הפרמטרים תואמים את התיאור של הפונקציה `cartoonify()`.

יש לוודא את תקינות מספר הארגומנטים.

במידה ומספר הארגומנטים שונה מהמצופה, יש להדפיס הודעת שגיאה ולצאת מהתכנית בצורה מסודרת.

לדוגמא:

```
python3 cartoonify.py ziggy.jpg ziggy_cartoon.jpg 460 5 15 17 8
```

הרצת פקודה זו תוביל לשרשרת האירועים הבאה:

- טעינה של התמונה שבניתוב `image_source` באמצעות הפונקציה `load_image` שבקובץ העזר
- ביצוע `resize` לתמונה כך שגודלה לא יעלה על $m \times \max_im_size$ כאשר $m \leq \max_im_size$ והוא הגודל הנחוץ לשמירה על פרופורציות נכונות 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.

לפני שאתם מתחילים – טיפים והנחיות

- לצורך פתרון התרגיל עליכם להוריד את הקובץ `ex5_helper.py`, מודול זה כבר מומש בשבילכם, והוא מכיל מספר פונקציות הדרושות לתרגיל. אל תעשו שום שינוי בקובץ זה!
- בנוסף, לרשותכם התיקיה `examples.zip`, המכילה שתי דוגמאות עליהן תוכלו לבחון את התוכנית הסופית שלכם. כל דוגמה מורכבת מתמונה מקורית, התמונה המתקבלת בריצת התכנית וקובץ טקסט המפרט את הפרמטרים שיצרו את ה-`cartoon`. שימו לב, אלו לא בהכרח הקבצים שעליהם התרגיל יבדק.
- עליכם ליצור קובץ בשם `cartoonify.py` בו תממשו את התרגיל. אתם יכולים לכתוב בקובץ זה פונקציות עזר נוספות מלבד אלה הדרושות בתרגיל, ולהשתמש בהן בקוד שלכם. אבל הפונקציות הדרושות בתרגיל חייבות להיכתב בדיוק על פי הדרישות המפורטות להלן.
- הקפידו לכתוב תיעוד לקוד שלכם ובפרט לכל פונקציה שאתם כותבים.
- 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`.
- בתהליך הפתרון, מומלץ לקבל חיווי (הדפסת הודעה, או הצגת תמונה, לדוגמה) בשלבים שונים של ריצת התוכנית. כך תוכלו לדעת שהאלגוריתם "מתקדם" ומה מתקבל כתוצאה מהפעלת הפונקציות שכתבתם. שחקו עם הפרמטרים של כל פונקציה ונסו מספר קלטים.
- בתרגיל זה, כל פעולה על תמונה צבעונית תבוצע על כל ערוץ צבע בנפרד. באופן זה ניתן לנצל פונקציות שתומכות 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 אין לבצע שום שינוי בתמונות הקלט, אלא להחזיר תמונות חדשות!
- לצורך פיתרון התרגיל תוכלו גם להשתמש במודולים `sys`, `math`-i `copy`. אין להשתמש במודולים `numpy` או `PIL`.

הוראות הגשה

עליכם להגיש קובץ בשם **ex5.zip** בקישור ההגשה של תרגיל 5 דרך אתר הקורס על ידי לחיצה על "Upload file". הקובץ **ex5.zip** צריך להכיל אך ורק את הקובץ **cartoonify.py**.

בהצלחה!