בית הספר להנדסה ומדעי המחשב ע"ש רחל וסלים בנין

מבוא למדעי המחשב 67101 – סמסטר א' 2024

תרגיל 5 – עיבוד תמונה ועבודה עם רשימות רב-מימדיות

להגשה בתאריך **14/02/2024** בשעה 22:00

הקדמה

בתרגיל זה נכתוב תכנית לעריכת תמונות. במהלך כתיבת התכנית 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.** קראו את התרגיל **כולו** לפני התחלת המימוש, לרבות ההערות בסוף.

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 **image_editor.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, אלא אם צוין אחרת. ניתן יהיה להגיש את התרגיל מחדש עד סוף הסמסטר, בהתאם לכתוב בנהלי הקורס.

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 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 הכחולים, וכך נעבוד על רשימות דו-מימדיות.

שימו לב שמספר הערוצים *(channels)* בפונקציות הבאות אינו בהכרח 3.

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

- 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]]]

image = [[[1, 2, 3], [1, 2, 3], [1, 2, 3]],
         [[1, 2, 3], [1, 2, 3], [1, 2, 3]],
         [[1, 2, 3], [1, 2, 3], [1, 2, 3]],
         [[1, 2, 3], [1, 2, 3], [1, 2, 3]]]
image_lst = [[[1, 1, 1], [1, 1, 1], [1, 1, 1], [1, 1, 1]],
             [[2, 2, 2], [2, 2, 2], [2, 2, 2], [2, 2, 2]],
             [[3, 3, 3], [3, 3, 3], [3, 3, 3], [3, 3, 3]]]
separate_channels(image) → image_lst
```

- 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 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]]]

image_lst = [[[1, 1, 1], [1, 1, 1], [1, 1, 1], [1, 1, 1]],
             [[2, 2, 2], [2, 2, 2], [2, 2, 2], [2, 2, 2]],
             [[3, 3, 3], [3, 3, 3], [3, 3, 3], [3, 3, 3]]]
image = [[[1, 2, 3], [1, 2, 3], [1, 2, 3]],
         [[1, 2, 3], [1, 2, 3], [1, 2, 3]],
         [[1, 2, 3], [1, 2, 3], [1, 2, 3]],
         [[1, 2, 3], [1, 2, 3], [1, 2, 3]]]
separate_channels(image_lst) → image
```

<u>שימו לב</u>: 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: לא רק לתמונות עם 3 ערוצי צבע.

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 **<u>אי-זוגי</u>**.

<u>אין לשנות את תמונת המקור.</u>

לדוגמא:

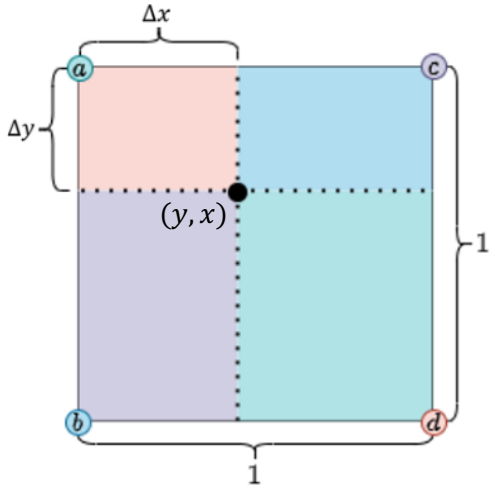
```
apply_kernel([[0, 128, 255]], blur_kernel(3))  →  [[14, 128, 241]]

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

<u>שימו לב:</u>

- **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 בודקים עבור כל פיקסל בתמונת היעד (בגודל החדש) מה הנקודה שבה הוא "נופל" בתמונת המקור, וערכו החדש מחושב כצירוף של 4 הפיקסלים הקרובים לנקודה זו. הערך מחושב באמצעות אינטרפולציה, לפי החישוב הבא:



$$a \cdot (1-\Delta x)\cdot(1-\Delta y) + b\cdot \Delta y\cdot(1-\Delta x) + c\cdot \Delta x\cdot(1-\Delta y) + d\cdot \Delta x\cdot \Delta y$$

כאשר a, b, c, d הם ערכי הפיקסלים בתמונת המקור הקרובים לנקודה שבה הפיקסל מתמונת היעד "נופל".

לדוגמא:

נתבונן בתמונה הבאה, שגודלה 5 x 3 פיקסלים, ונרצה לבצע לה resize לגודל 7 x 6 פיקסלים.

15	30	45	60	75
90	105	120	135	150
165	180	195	210	225

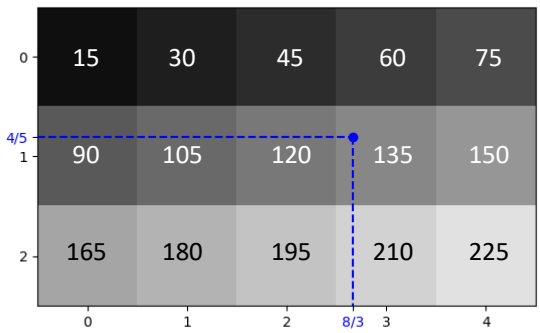
ראשית עלינו להבין מה המקור של כל פיקסל בתמונת היעד. לצורך כך עלינו לחשב איפה הוא נמצא ביחס לאורך ולרוחב, ולהשתמש במיקום היחסי כדי להבין איפה הוא נופל בתמונת המקור.

ניקח לדוגמא את הפיקסל `[y, x] = [2, 4]` בתמונת היעד. ערכי ה-y של תמונת היעד הם בטווח [0,5] ולכן מיקומו היחסי ביחס לציר ה-y הוא $^2/_5$. באופן דומה מיקומו היחסי ביחס לציר ה-x הוא $^4/_6$.

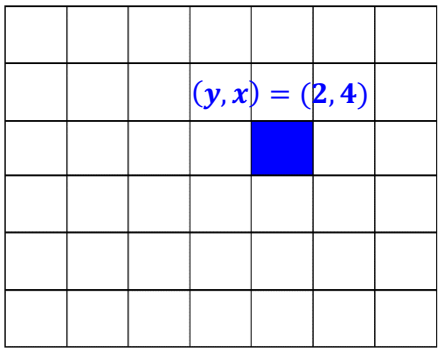
כעת נותר לחשב מה הקואורדינטה בתמונת המקור שמתאימה למיקום היחסי שמצאנו. לצורך כך נכפול את המיקום היחסי של y ב-2 (מפני שערכי ה-y של תמונת המקור הם בטווח [0,2]) ואת המיקום היחסי של x ב-4.

בסה"כ נקבל שהפיקסל נופל בנקודה `[y, x] = [4/5, 8/3]` בתמונת המקור.

Source Image



Destination Image



כעת נמצא את הערכים הדרושים עבור החישוב:

$$\Delta x = \frac{2}{3} \qquad \Delta y = \frac{4}{5} \qquad \underbrace{a = 45}_{image[0,2]} \qquad \underbrace{b = 120}_{image[1,2]} \qquad \underbrace{c = 60}_{image[0,3]} \qquad \underbrace{d = 135}_{image[1,3]}$$

נציב בנוסחא ונקבל:

$$new\_image[2,4] = 45\left(1-\frac{2}{3}\right)\left(1-\frac{4}{5}\right) + 120\cdot\frac{4}{5}\cdot\left(1-\frac{2}{3}\right) + 60\cdot\frac{\mathbf{2}}{3}\cdot\left(1-\frac{4}{5}\right) + 135\cdot\frac{\mathbf{2}}{3}\cdot\frac{4}{5} = 115$$

כלומר ערכו של הפיקסל `[y, x] = [2, 4]` בתמונת היעד יהיה 115.

שימו לב שהערכים של Δx ושל Δy הם בין 0 ל-1 **<u>כולל</u>**.

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 חישוב האינטרפולציה שהצגנו לעיל.

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
bilinear_interpolation([[15, 30, 45, 60, 75], [90, 105, 120, 135, 150],
[165, 180, 195, 210, 225]], 4/5, 8/3) → 115
```

<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:

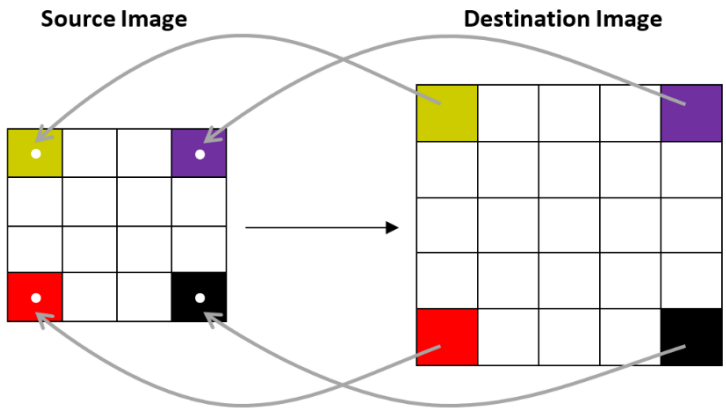
`resize(image , new_height, new_width)`

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:

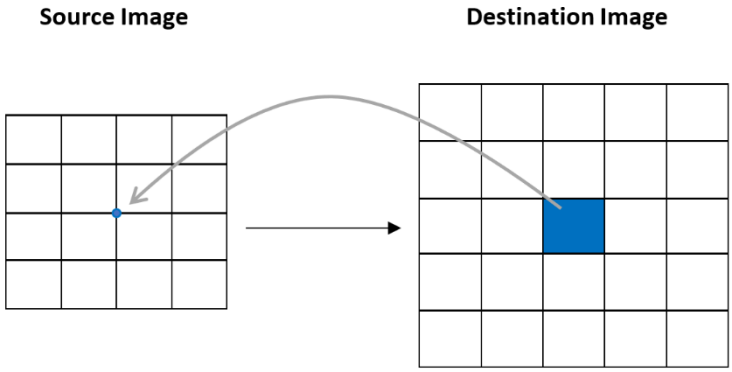
1. פינות ימופו לפינות.

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



2. 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.

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.

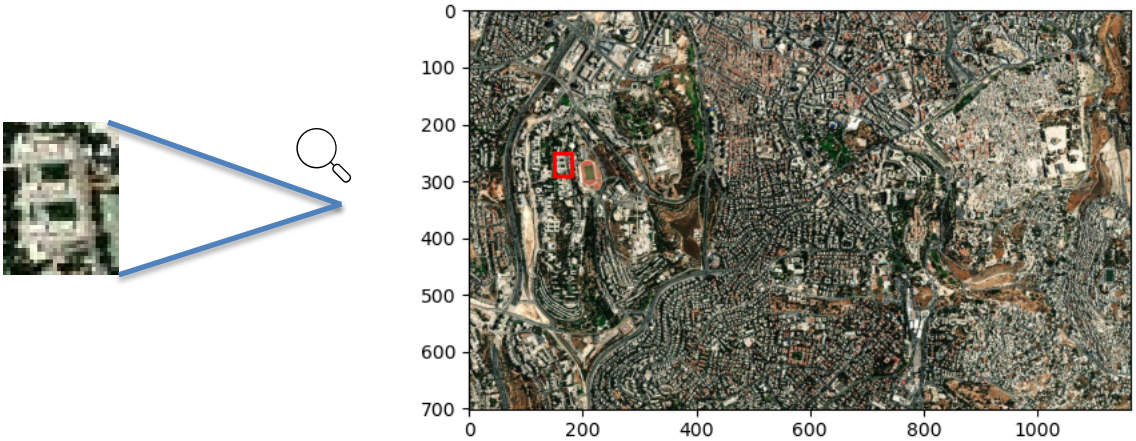
אין לשנות את תמונת המקור.

לדוגמא:

```
rotate_90([[1, 2, 3],          [[4, 1],
           [4, 5, 6]], 'R')  →  [5, 2],
                                [6, 3]]

rotate_90([[1, 2, 3],          [[3, 6],
           [4, 5, 6]], 'L')  →  [2, 5],
                                [1, 4]]

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

חיפוש בתמונה:

בחלק זה נממש חיפוש פא'צים בתוך תמונה. למשל, הפאצ' הבא (בצד שמאל) נמצא בתמונה הימנית במיקום המודגש.



<u>הגדרת מרחק:</u>

אנו נמדוד מרחקים בין שני פאצ'ים (מלבנים בתמונה) באופן הבא: נשתמש במרחק Mean Square Error, שמוגדר ע"י הממוצע של ריבועי הפרשי הפיקסלים באינדקסים תואמים. כלומר, אם נקבל שתי תמונות A,B בגודל NxM, הפונקציה מוגדרת ע"י הנוסחה הבאה:

$$mse(A,B) := \frac{1}{N \cdot M}\sum_{i=1}^{N}\sum_{j=1}^{M}(A[i][j] - B[i][j])^2$$

כאשר N הוא מספר השורות ו M הוא מספר העמודות. כפי שמשתקף בנוסחה, אפשר למדוד מרחק mse בין כל זוג של מלבנים בעלי אותו המימד (תמונות, פאצ'ים, או סתם רשימות דו-מימדיות של מספרים).

8. ממשו את הפונקציה:

`get_best_match(image, patch)`

אשר מקבלת תמונה בעלת ערוץ צבע יחיד image ופאצ' (תמונה קטנה בעלת ערוץ צבע יחיד) patch, ומחזירה טאפל בעל שני ערכים: המיקום והמרחק של הפאצ' "הקרוב ביותר" בimage לפאצ' הקלט patch.

הערך הראשון הוא טאפל בעצמו (x,y), המייצג מיקום של פאצ' ב image. הערך השני הוא מרחק ה mse של אותו פאצ' שמיקומו בתמונה image הוא (x,y), מפאצ' הקלט patch.

<u>ערך המיקום:</u> הקאורדינטות x,y מייצגות את מיקום פינתו השמאלית עליונה של הפאצ' בתמונה, כאשר x הוא מס' שורה ב image, ו y הוא מס' עמודה ב image.

<u>ערך המרחק:</u> יהיה מספר float המייצגת את המרחק בין הפאצ' בתמונה שפינתו השמאלית עליונה היא (x,y), ל patch.

למשל, עבור הקלטים הבאים (image משמאל, patch מימין):

1	5	1	3
7	4	6	2
0	10	2	200
250	9	0	240

7	4	6
0	10	3
249	9	1

הפונקציה תחזיר:

((1, 0), 0.33333334)

כאשר 1,0 זה מיקום הפיקסל השמאלי עליון של הפאצ' שממזער את המרחק מ patch. כלומר הפאצ' מתחיל היכן שמופיע הערך 7 בתמונה: מס' השורה הוא 1, ומס' העמודה הוא 0.

<u>הערות למימוש:</u>

- ניתן להניח שהגודל של `patch` קטן או שווה לגודל של image (כלומר מס' השורות של patch קטן שווה לשל image, וכנ"ל לגבי מס' העמודות)
- ניתן להניח שמס' העמודות והשורות של `patch` הם מספרים חיוביים.
- אין לשנות את הקלט.
- אם יש כמה ערכי `x,y` שממזערים את המרחק, החזירו את הראשון מביניהם: זה עם הערך x הקטן ביותר. אם יש כמה כאלה עם אותו ערך x, אז החזירו מביניהם את זה עם ערך ה y הקטן ביותר.

9. ממשו את הפונקציה:

`find_patch_in_img(image, patch)`

אשר מקבלת תמונה בעלת ערוץ צבע יחיד image, ורשימה דו מימדית patch, ומחפשת את המיקום המשוער של patch ב image בכמה הופעות שונות של סיבובים והגדלות. הפונקציה תחזיר מילון של רשימות, המכילות את כל המיקומים והמרחקים, של הפאצ'ים בתמונה שממזערים את מרחק ה mse ל patch בכל רמת גודל. נסביר:

הפונקציה תחפש את כל אחד מהסיבובים של patch ב 90 מעלות (0,90,180,270) בתוך image, **בחיפוש מקורב בפירמידה**.

<u>הגדרה של חיפוש מקורב בפירמידה:</u>

זהו חיפוש ב 4 צעדים, בהם נקטין את התמונה image והפאצ' patch לפי הסדר הבא: פי 8, פי 4, פי 2 ו פי 1 (הגודל המקורי).

כלומר:

1) תחילה נקטין את התמונה והפאצ' המקוריים פי 2, ונקבל את `image_2, patch_2`. אותם נקטין גם פי 2 כדי לקבל את `image_4, patch_4`, ואת אלה נקטין כדי לקבל את `image_8, patch_8`. שימו לב שההקטנות תלויות אחת בשניה, ואינן הקטנה ישירה של המקור (חוץ מ `image_2, patch_2`).
2) כעת נחפש את המיקום המשוער של `patch_8` ב `image_8` בעזרת הפונקציה הקודמת שתחזיר מיקומים x_8, y_8 בתמונה המוקטנת פי 8 (`image_8`).
3) כעת נחפש את `patch_4` ב `image_4`, אבל נחפש **רק בסביבה הקרובה של הפיקסל** $2x_8, 2y_8$ (הסבר בהמשך), וב `image_4`, ו**לא בכל התמונה**. לפיקסל שיימצא נקרא x_4, y_4.
4) כעת נוכל לבצע חיפוש של `patch_2` ב `image_2` בסביבה הקרובה של $2x_4, 2y_4$, **ולא בכל התמונה**. נקבל x_2, y_2 חדש.
5) כעת נחפש את `patch_1` ב `image_1` (הקלטים המקוריים) בסביבה הקרובה של $2x_2, 2y_2$, **ולא בכל התמונה**.

<u>חיפוש בסביבה קרובה של פיקסל x_0, y_0:</u>

זהו חיפוש בפאצ'ים שמתחילים בפיקסל x_0, y_0 ובשכניו. סה"כ חיפוש ב 9 פא'צים בתוך התמונה, כל הפאצ'ים שקודקודם השמאלי העליון מתחיל ב `(x,y)` הבאים: $\{(x,y)|\ x_0-1 \le x \le x_0+1,\ \ y_0-1 \le y \le y_0+1, and\ (x,y)\ in\ image\}$

הבהרות:

- ברמה הראשונה (התמונה המוקטנת פי 8) יש לחפש את הפאצ' **בכל התמונה**.
- שימו לב שברמה ה i החיפוש מתבצע עם תמונה מוקטנת פי i, וגם עם ה patch שמוקטן פי i.
- הקטנה פי 2 היא הקטנה בעזרת הפונקציה הקודמת שמימשתם, מגודל `(m,n)` לגודל `(m/2,n/2)`.

הפונקציה תחזיר מילון עם ארבעת המפתחות `0, 90, 180, 270` המייצגים כל סיבוב אפשרי של הפאצ'. לכל מפתח תהיה רשימה בת 4 איברים, של הפיקסלים שנמצאו בכל רמה בפירמידה לאותו הפאצ' המסובב.

כל איבר ברשימה **יהיה תואם בדיוק** לפלט של הפונקציה הקודמת שמימשתם: הוא tuple שמכיל שני איברים, כאשר הראשון הוא בעצמו טאפל `x,y` ומרחק (float) של הפאצ' `x,y`, בתמונה המוקטנת פי 8. האיבר השני ברשימה יהיה לפי התמונה המוקטנת פי 4, הבא לפי התמונה המוקטנת פי 2, והאחרון לפי התמונה המוקטנת פי 1 (הגודל המקורי).

<u>דוגמה לחיפוש בסביבה של נקודה:</u>

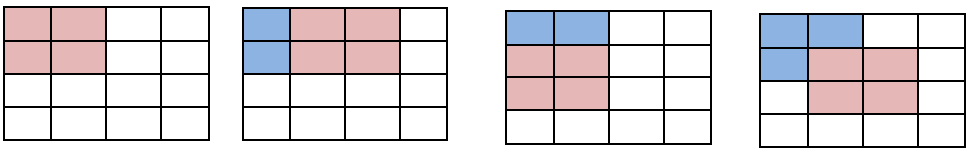
נניח שביצענו את שלבים 1+2 באלגוריתם הנ"ל, וקיבלנו את פיקסל `0,0 == x8, y8` בחיפוש בתמונה המוקטנת פי 8. לפי האלגוריתם, כעת נרצה לחפש את `patch_4` ב `image_4`, רק בסביבה הקרובה של `2*0, 2*0 == 0, 0`.

אם נניח שהטבלאות הבאות מייצגות את `patch_4` (שמאל) ו `image_4` (ימין):

255	255
255	255

0	0	0	0
0	255	255	0
0	255	255	0
0	0	0	0

אז נחפש את patch_4 רק באחד מארבעת המיקומים הכחולים ב image_4 (כל שכניו הקיימים של 0,0 בגבול התמונה):



במקרה זה, נקבל שהבדיקה האחרונה (מימין) תמזער את מרחק ה mse ל 0.0000, ולכן נקבל $x_4, y_4 == 1,1$

הערות למימוש:

- ניתן להניח כי הגודל של patch תקין (מס' השורות והעמודות שלו קטנות שוות לשל image).
- אם קיימים כמה מיקומים x,y המקבלים את ה mse המינימלי בחיפוש כלשהו, החזירו את הראשון מביניהם (מיקום עם מספר x הקטן ביותר. אם קיימים כמה כאלה, אז עם מספר y הקטן ביותר).
- ניתן להניח שמס' השורות והעמודות של image ו patch הם חזקה של 2, ולכל הפחות הינם בגודל 16 (כלומר, התמונה המוקטנת פי 8 תהיה לכל הפחות בגודל 2x2).
- בחיפוש בסביבה של נקודה: אם שכניו של פיקסל x,y הם "מחוץ לתמונה" – לא נבדוק אותם. לכן נבדוק בין 4 ל 9 פיקסלים שכנים (תלוי אם x,y על הגבול של התמונה או לא).
- מומלץ להשתמש בפונקציות עזר.
- יש להשתמש בפונקציה הקודמת שמימשתם (המנעו מכפל קוד).

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 **image_editor.py**.

לפני שאתם מתחילים – טיפים והנחיות

- הקפידו לכתוב תיעוד לקוד שלכם ובפרט לכל פונקציה שאתם כותבים.
- לרשותכם התיקייה examples, המכילה דוגמאות של תמונות איתן אתם יכולים לבדוק את התכנית שלכם.
- 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 (אלא אם צוין אחרת). באופן זה ניתן לנצל פונקציות שתומכת בערוץ צבע יחיד גם בעבור תמונות צבעוניות.
- ניתן להניח תקינות הקלטים לכל אחד מהסעיפים (בהתאם להגדרות הפרטניות 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!

- שימו לב: אין להניח שהרשימות המתקבלות בקלט ים בתרגיל הן רשימות שונות (לא אותה רשימה). כלומר, ניתן לקבל כמה רשימות עם איברים זהים (מחזירות True לאופרטור ==), אך הן **לא בהכרח** רשימות שונות (עלולות להחזיר True או False לאופרטור is).
- לצורך פיתרון התרגיל תוכלו להשתמש במודולים sys , copy ו-math. **אין להשתמש** במודולים numpy או PIL.
- בדוגמה לחיפוש פאצ' מתוך תמונה: מי מזהה מה מוצג בתמונה? ומהו הפאצ'? שתפו אותנו בפורום ☺

בהצלחה!