

# תרגיל בית 2-תכנות מקבילי וմבוזר לуйיבוד נתונים ולמידה חישובית

סמינר חורף 25/26

## הנחיות כלליות לפתרון התרגיל:

- תאריך הגשה: 7/1/26
- מתרגל אחראי על התרגיל: מתן ביתון
- נושא התרגיל: רשתות נוירוניים عمוקות ומקבילות, אוגומנטציה של נתונים.
- פתרון התרגיל בזוגות בלבד (פרט לMKרים חריגים באישור המתרגל האחראי בקורס).
- חובה להגיש את התרגיל על מנת לקבל ציון בקורס.
- שאלות על התרגיל יש לשלו למתרגל האחראי על התרגיל.
- בקשות לקבלת הארכה להגשת התרגיל יש לשלו למתרגל האחראי בקורס.
- בכל קובץ קוד שמוגש, יש לכתוב בהערה בתחילתו את שמות הסטודנטים ואת תאריך המילוי של הקובץ.
- שימושו לב: אתם אחראים על הקוד אשר אתם מגישים. השימוש בכל בינה מלאכותית בתרגיל הוא על אחראיכם ושימוש מופרז עלול להיחשב כהעתקה בתרגיל.

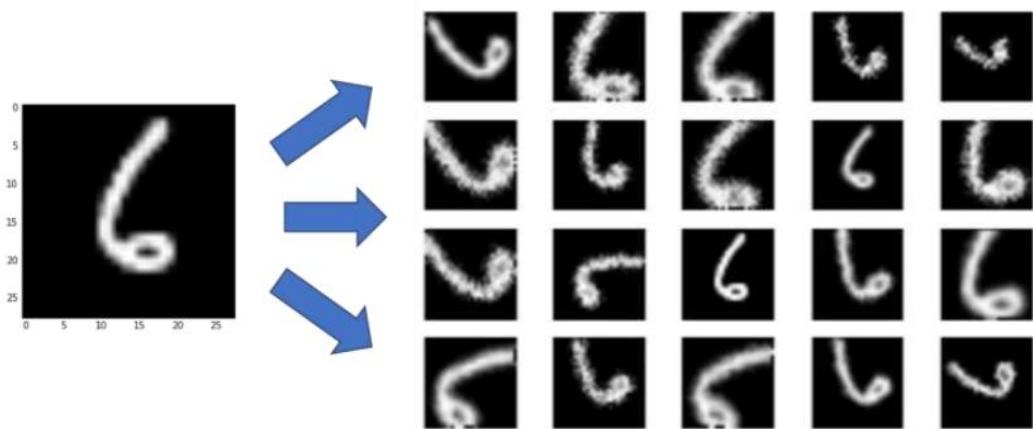
## 1. חלק א' – אוגמנטציה של נתונים אימון מקבילי של רשתות נירוניים

### 1.1. רקע כללי

בהרצאה רأינו שניתן לאמן רשת נירוניים عمוקה (Deep Neural Network) בשימוש בקבוצת אימון  $\{(x, f(x))\}$  כדי לקרב פונקציה לא ידועה  $f(x)$ . אחד האתגרים העיקריים עם רשתות נירוניים عمוקות היא יצירת קבוצת אימון גדולה מספקת כדי שהקרוב יהיה מדויק. בנוסף נתונים לבניית קבוצת האימון איננו פשוט – הוא עשוי לדרוש סוף, מאיץ אנושי, משאבים חישוביים וכמוון זמן. מסיבה זו, נרצה למצוא דרכים להגדלת קבוצת האימון תוך חסוך בעליות הבניה של קבוצה זו.

אגמנטציה של נתונים (Data Augmentation) היא אחת מהשיטות שבהן ניתן להשתמש להגדלת קבוצת האימון באופן מלאכותי מקבוצה קיימת, וזאת על ידי ביצוע שינויים בדוגמאות קלט ופלט קיימות. היתרון בשיטה זו היא שניתן להגדיל את קבוצת האימון שלנו ללא העליות של איסוף נתונים נוסף, וזאת באמצעות הפקת תועלת נוספת מהדוגמאות שכבר יש לנו.

קיימות דרכים רבות לביצוע אוגמנטציה של נתונים, וזאת בתלות במבנה הדוגמאות. אם הדוגמאות הן תמונות, ניתן לסובב אותן, להקטין אותן, לחתוך אותן, להשתמש בפילטרים שונים ועוד. מתמונה אחת ניתן לייצר מספר רב של תמונות חדשות:



בדוגמה המצורפת ניתן לראות כיצד ניתן לבצע אוגמנטציה של תמונות, ובאופן זה להפוך תמונה יחידה של המספר 6 כתוב בכתב יד למספר רב של תמונות שונות.

## 2.1. הנחיות מימוש

המטרה בתרגיל זה היא לאמן רשת נוירונים عمוקה ליזיה ספרות שכתובות בכתב יד, באמצעות קבוצת אימון שתווצר על ידי שיטות לאוגנטציה של תמונות מהמארג MNIST. מאגר זה מכיל תמונות בגונו אפור של ספרות. ניתן לקרוא יותר על מאגר זה בקישור [https://en.wikipedia.org/wiki/MNIST\\_database](https://en.wikipedia.org/wiki/MNIST_database).

המימוש יבוצע בהמשך חלק 1 של תרגיל בית 1, ועליכם יהיה למש בשתי מחלקות: *Worker* ו-*IPNeuralNetwork*.

מחלקה *Worker* (מהקובץ *py.preprocessor*), אשר עליה לרשות המחלקה *Process* שנמצאת בספריית *multiprocessing* שב-*Python*, מכילה מספר פונקציות שבהן אחת מהן שיטה לביצוע אוגנטציה של תמונות. למחלקה נדרש שדה עבור תור העבודות בשם *jobs* (שיכיל את כל התמונות שנדרש לבצע עלייה אוגנטציה) ושדה עבור תור התוצאות בשם *results* (שיכיל את כל התמונות שהתקבלו מהתהליכי האוגנטציה). בנוסף, במחלקה זו יהיו שתי מתודות נוספות – מתודה אשר מפעילה את כל פעולות האוגנטציה על תמונה אחת שמקבלת כפרמטר, בשם *process\_image*; ומתודה אשר מוציאה תמונה מתוך העבודות ומכניסה לתוך התוצאות את התמונה שהתקבלה מהאגנטציה, בשם *out*.

מחלקה *IPNeuralNetwork* (מהקובץ *ip\_network.py*), היורשת מהמחלקה *NeuralNetwork*, דורשת שתי מתודות – מתודה בשם *fit*, שבמקור הייתה אחראית לביצוע תהליכי האימון של רשת הנוירונים העמוקה באמצעות *Stochastic Gradient Descent*, וcutut יש להוסיף אליה את הלוגיקה הדרישה לטיפול במחלקות העיבוד מטיפוס *Worker* על מנת ליצור את התמונות שהתקבלו מהתהליכי האוגנטציה עבור האימון; ומתודה בשם *create\_batches*, שמקבלת *batch* מקבוצת האימון המקורי ויוצרת גרסה של ה-*batch* שעבירה אוגנטציה על ידי עצמים מהמחלקה *Worker*.

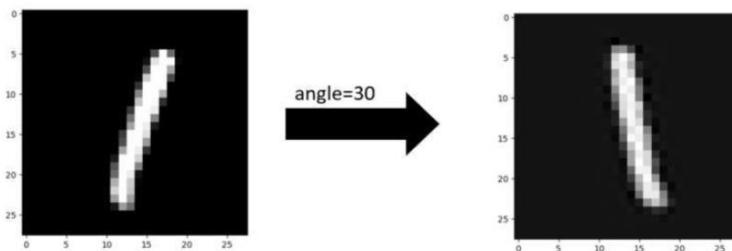
בהמשך, נסקור את פרטי המימוש של שתי מחלקות אלו. לפני כן, חשוב לשים לב להנחיות הבאות:

- על מנת להריץ את המחלקה *NeuralNetwork*, יש להוסיף את הקובץ *py.util* שכתבתם בתרגיל בית 1 לתיקית העבודה. עם זאת, אין להגיש אותו שוב בתרגיל בית זה.
- בתרגיל בית זה מסופק קוד שונה של המחלקה *NeuralNetwork*, בהשוואה לקוד שסופק לכם בתרגיל הבית הקודם, על מנת להקל על המימוש של תרגיל הבית: ניתן בגרסה זו לספק למחלקה נוספת את הפרמטר *number\_of\_batches*; הפונקציה בשם *create\_batches* הינה להיות מתודה של המחלקה בגרסה זו; ומתודה זו מחזירה *batches* אקראים לפי הערך של *number\_of\_batches*.
- מומלץ במהלך מימוש המתודות הסטטיות עבור האוגנטציה להשתמש בפונקציות המבונות בספריות *NumPy* ו-*SciPy*, כדי להקל על המימוש.

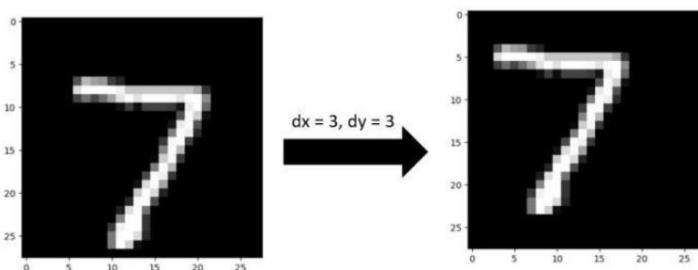
### 3.1. פרט מימוש למחלקה *Worker*

במחלקה *Worker* עליו למש את המתודות והפונקציות הבאות:

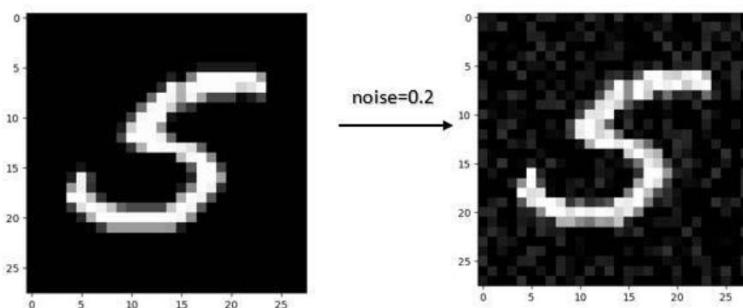
- הפונקציה (*\_\_init\_\_(self, jobs, result)*) תאתחל עצם ממחלקה זו ואת השדות שלו. עליו לבחור את השדות שתרצה לשמר במחלקה זו, ומותר לצורך כל להוציא פרמטרים לפונקציה.
- המתודה הסטטית (*rotate(image, angle)*) תחזיר את התמונה המתקבלת מסיבוב התמונה הנתונה בזויה הנתונה (מומלץ להיעזר בספרייה *SciPy*). לדוגמה:



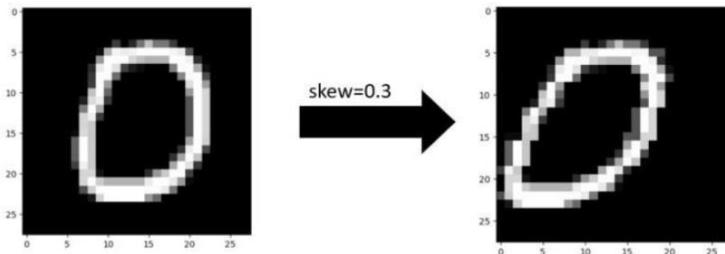
- המתודה הסטטית (*shift(image, dx, dy)*) תחזיר את התמונה המתקבלת מהזזה התמונה הנתונה *dx* תאים שמאלה ו-*dy* תאים למעלה (ערכים אלו יכולים להיות שליליים), כאשר מתיחסים לערכים בקואורדינטות מחוץ לתמונה כל ערכי 0, ככלורן כל תאים שחורים (מומלץ להיעזר בספרייה *SciPy*). לדוגמה:



- המתודה הסטטית (*add\_noise(image, noise)*) תחזיר את התמונה המתקבלת מבחרה של ערך מתפלג יוניפורמי בתוחום [*noise, noise*] לכל תא בתמונה, והוספת ערך זה לערך הבסיסי של התמונה תוך שמירה על תחום המותר לערכי בהירות (מומלץ להיעזר בספרייה *NumPy*). לדוגמה:



- הmethodה הסטטistica (*skew(image, tilt)*) תחזיר את התמונה המתקבלת מהטיהת התמונה, ככלומר על ידי חישוב הערכים בתמונה התוצאה באמצעות הנוסחה  $[result[i][j] = image[i][j + i \cdot tilt]$ , כאשר מתייחסים לערבים בקואורדינטות מחוץ לתמונה כל ערכי 0, ככלומר כאלו תאים שחורים. לדוגמה:



- הmethodה (*process\_image(self, image)*) תריץ את ארבעת המethodות הסטטיות, אחת אחרי השניה בסדר לבחירתכם, כאשר מתחילה מהתמונה שמתקבלת בפרמטר. השתמשו ב-*random* שב-*Python* כדי לקבוע את שאר הארגומנטים למethodות, בתחוםים לבחירתכם. נסו לשפר את סדר הביצוע ואת התחומים, באמצעות ניסויים קצריים, כדי לשפר את הדיקן המתkeletal.
- הmethodה (*self.run*) תיצור את התמונות שנוצרות מהאגמנטציה, לפי ההנחיות שצורפו קודם.

#### 4.1. פרטוי מימוש למחלקה *IPNeuralNetwork*

במחלקה *IPNeuralNetwork* עליו למשם את המethodות הבאות:

- הmethodה (*create\_batches(self, data, labels, batch\_size)*) צריכה לדرس את המethodה *key* מתמחלקה *NeuralNetwork*, וعليה ליצור batches של התמונות לאחר האוגמנטציה (כלומר התמונות שהתקבלו מהתהליך ובנוסף אליהן התמונות המקוריות), ולא להשתמש רק בתמונות המקוריות.
- הmethodה (*fit(self, training\_data, validation\_data=None)*) צריכה לדرس את המethodה *key* מתמחלקה *NeuralNetwork*, וعليה ליצור עצמים מתמחלקה *Worker* בהתאם למספר יחידות העבודה שניתנו בהן (היעזרו בערך [*'SLURM\_CPUS\_PER\_TASK'*, *os.environ*]), לקרוא למmethodה הנדרסת (עם *(super)*) ולאחר מכן לפנות את העצים.

שימוש לב – פתרונות טריוויאליים (עם עצם אחד מטיפוס *Worker* בסך הכל, או עם עצם אחד מטיפוס *Worker* לכל שימוש) לא יקבלו ניקוד מלא.

## 2. חלק ב' – תור מקבילי

בחלק זה, עליכם למשתמש בתור מקבילי פשוט, ולאחר מכן עליכם לשנות את המחלקה *IPNeuralNetwork* כך שתשתמש בעצם במחלקה זו בשדה תור התוצאות (אין צורך לשנות את טיפולו בתור המשימות).

עליכם למשתמש את המחלקה בשימוש בחלוקת Pipe Lock *multiprocessing* שראיתם בתרגולים. מחלקה זו צריכה לתמוך **בכותבים רבים ובקרוא יחיד** (*many-writers, one-reader*), כלומר ניתן להניח שקיים לכל היותר תהיליך אחד שקורא מהתור, וייתכן שקיימים יותר מטהיליך אחד שכותב לתור. השתמשו בהנחה זו כדי להחיליט מתי צריך לסנכרן – ומתי סינכרוניזציה Miyotra.

בקובץ `my_queue.py` ממשו את המתודות הבאות של המחלקה `MyQueue`:

- הפונקציה (`__init__(self)`) תתחילה תור ואת השדות של העצם.
- המתודה (`put(self, msg)`) תשליח את הודעה (שהוא אובייקט כלשהו) דרך התור.
- המתודה (`get(self)`) תקרא הודעה מהתור (כלומר תחזיר את האובייקט שנשלח מהתור).
- המתודה (`empty(self)`) תחזיר האם תור הודעה ריק (כלומר שאין ערכים שימושיים לקרואיה).

ניתן להניח שהמתודה `empty(self)` תיקרא רק על ידי התהיליך הקורא. בנוסף, מותר שהמתודה תחזיר בהתאם ריק גם אם קיימת הודעה שכרגע נשלחת – המתודה צריכה להחזיר שהתור אינו ריק רק אם אכן קיימת הודעה שנשלחה וגם שהטהיליך ששלח אותו סיים לבצע את תוכן המתודה (`put(self, msg)`).

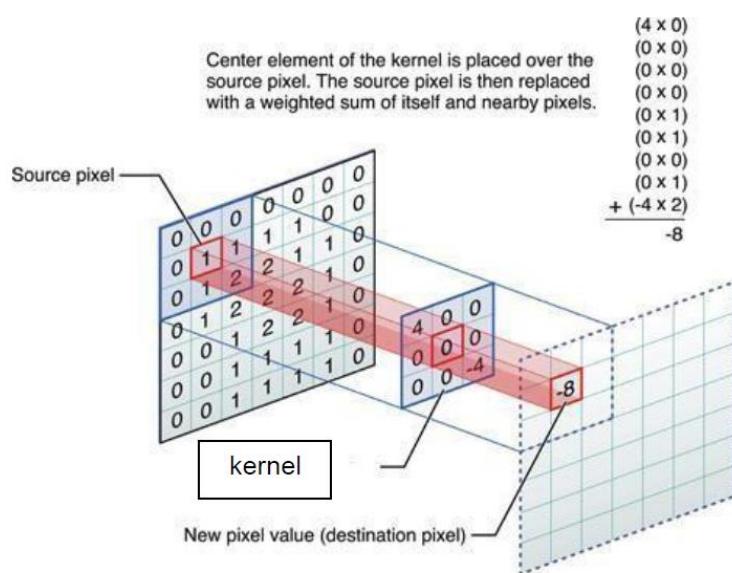
שימוש חלק זה קצר ופשוט יחסית, והשימוש של כל מתודה אמור להיות באורך של כ-3-5 שורות.

### 3. חלק ג' – חישוב מקבילי של kernel

**קורלציה (Correlation)** היא פעולה מתמטית שמקבלת שני סיגנלים כקלט ומחזירה סיגナル שלילי כפלט, והוא מצויה בשימוש רחב במגוון של ענפים ויישומים. בעיבוד תמונה ניתן להשתמש בפעולת זו לטשטוש, לחידוד, להבלטה, ליזיהו קצנות ולשימושים נוספים.

**מטריצת הקורלציה** היא התוצאה שמתתקבל מחישוב ערך הקורלציה בין תמונה נתונה, לכל מקום של פיקסל בה, לבין מטריצה קטנה הקרויה kernel. ניתן לחשב על מטריצת הקורלציה כהכלפה של כל פיקסל בתמונה בסכום משקלל שלו ושל שכניו, כאשר המשקלים מוגדרים על ידי(kernel). בנוסף, אם החישוב מתבצע ליד הקצנות וחלק מהשכנים אינם קיימים ('מצאים' מחוץ לגבולות המטריצה), אז מתייחסים אליהם כבעלי הערך 0 (כלומר אנו 'מרפדים' את התמונה הנתונה באפסים).

לדוגמה, אם kernel הוא מטריצה בגודל  $3 \times 3$  עם ערכי 1, אז בעת חישוב מטריצת הקורלציה של תמונה כלשהי, כל פיקסל מוחלף בסכום שלו ושל כל שכניו. ניתן לסכם באופן כללי את החישוב באמצעות התרשים הבא:



בחלק זה עליכם למשה בקובץ `filters.py` שתי פונקציות שמחسابת את מטריצת הקורלציה – פעם אחת בשימוש ב-`jit` להאצה ופעם אחרת בשימוש ב-GPU:

- בפונקציה (`correlation_numba(kernel, image)`) חשבו את מטריצת הקורלציה, בשימוש ב-`jit`.
- בפונקציה (`correlation_gpu(kernel, image)`) חשבו את מטריצת הקורלציה, בשימוש ב-`cuda`.

וודאו שההתוצאות שמתתקבלות בהינתן `kernel` ותמונה, גם עבור המימוש עם `jit` וגם עבור המימוש עם GPU, זהות לתוצאות שמתתקבלות בשימוש ב-`(scipy.signal.convolve2d(flipped_kernel, image))`.  
כאשר `kernel` הוא `flipped_kernel` לאחר סיבובו ב- $180^\circ$  (היפוך העמודות והשורות).

#### 4. חלק ד' – זיהוי קצוטה בתמונה באמצעות אופרטור Sobel

בחולק זה, נראה שימוש בפונקציות שמיימותם קודם, כדי לזהות קצוטה בתמונה. בחולק זה, עליים להשתמש בפונקציה `correlation_numba(kernel, image)` שמיימתם בחולק הקודם.

עליכם למשבץ בקובץ `filters.sobel_operator()`, אשר מבצע את החישוב הבא:

$$sobel\_filter = \begin{pmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{pmatrix}$$

$$G_x = correlation(sobel\_filter, image)$$

$$G_y = correlation(sobel\_filter^{Transposed}, image)$$

$$result_{i,j} = \sqrt{G_x_{i,j}^2 + G_y_{i,j}^2}$$

הערך המוחזר על ידי הפונקציה הוא המטריצה `result`, במדים של התמונה, אשר כל ערך בו מחושב לפי הנוסחאות שתוארו קודם. צרפו לדוח את התוצאה שמתקבלת עבור `sobel_filter` הנוכחי, וכן עבור מטריצות ה-`kernel` הבאות (המשובץ המוגש של הפונקציה צריך לפעול בשימוש ב-`-`):

$$kernel_1 = \begin{pmatrix} +3 & 0 & -3 \\ +10 & 0 & -10 \\ +3 & 0 & -3 \end{pmatrix}, kernel_2 = \begin{pmatrix} +1 & 0 & -1 \\ +2 & 0 & -1 \\ +1 & 0 & -2 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{pmatrix}, kernel_3 = \begin{pmatrix} +1 & +1 & +1 \\ +1 & 0 & +1 \\ +1 & +1 & +1 \end{pmatrix}$$

לשימוש קל מומלץ להשתמש בפונקציות `sqrt`, `transpose` ו-`pow` מהספרייה NumPy. בנוסף, בקיים שמסופק לכמ, נעשה שימוש בפונקציה `imread` מהספרייה `Imageio`, ובפונקציה `imshow` (`עם cmap='gray'`) מהספרייה `matplotlib-pyplot` להציג התוצאה.

לאחר השימוש, וידאו שההתוצאה המתקבלת זהה בהשוואה בין אחד השימושים שלכם מהחולק הקודם (ניתן להשתמש עבור הבדיקה ב-`numba`, `correlation_numba`, להרצה על המחשב המקומי), לבין השימוש שמתקבל בשימוש ב-`scipy.signal.convolve2d`.

ניתן לקרוא יותר על אופרטור Sobel בקישור [https://www.tutorialspoint.com/dip/sobel\\_operator.htm](https://www.tutorialspoint.com/dip/sobel_operator.htm)

## 5. דוח מסכם והנחיות כלליות

### 1.5. שאלות עבורי הדוח המסכם

השיבו בדוח המסכם, שאורכו לכל היותר 10 עמודים, על השאלות הבאות:

(1) הריצו בחלק א' את `py.main` עם 8, 16 ו-32 יחידות עיבוד (cores) – הדגל להרצה הוא `<cores>c`.

השו את זמן הריצה של `IPNeuralNetwork` (עם תור התוצאות שלכם) בין מספר שונה של יחידות עיבוד. צרפו צילום מסך של התוצאות, וכתבו הסבר קצר שבו כתוב איזה מספר יחידות עיבוד נתן את התוצאות הטובות ביותר, ומדוע.

(2) הריצו שוב את `py.main`, והשו את אחוז הדיק שמתקיים ב-`epochs` שונים, בין `NeuralNetwork` לבין `IPNeuralNetwork`. צרפו טבלה להשוואה והסביר קצר.

(3) לפי מה בחרתם את כמות ה `workers` שבחרתם? מה היה קורה לו היותם בוחרם יותר/פחות מיד?

(4) בחלק א', עשיתם שימוש בחוטים שונים באותו תהליך או בתהליכיים שונים? מדוע?

(5) הציעו שני רעיונות כיצד ניתן להאיץ אף יותר את שלב האימון בחלק א'.

(6) הסבירו את השימוש שלכם בחלק ב', האם השימוש היה שונה לנchnerו קורא אחד לכל היותר מהٿור? מדוע?

(7) הסבירו בפירוט כיצד מימשתם בחלק ג' את `numba.correlation` ואת `gpu.correlation`.

(8) הריצו את `py.filters_test` עם יחידת עיבוד (core) אחת, צרפו צילום מסך שマーאה את ההאצה בין המימושים השונים לבין השימוש של SciPy (`convolve2d`) והסבירו את התוצאות.

(9) לפי תוצאות הסעיף הקודם, מתי כדאי להשתמש ב-`jit` ומתי כדאי להשתמש ב-`cudatools`? הסבירו.

(10) צרפו את התוצאות המתקבלות מהפעלת אופרטור Sobel בשימוש ב-`sobel_filter` ובשימוש בשלוש מטריצות ה-`kernel` הנוספות, בשימוש ב-`matplotlib.pyplot` עם `numba.correlation`. הסבירו מה הבדלים בין התוצאות השונות, והסבירו מה הסיבות להבדלים אלו.

### 2.5. הערות ועצות

- בחלק א' התמונות הן מטריצות `NumPy` בגודל  $1 \times 784$ , אך יש לטפל בתמונות לפי הגודל  $28 \times 28$ .
- הֆונקציות `add_noise`, `shift`, `rotate` ו-`skew` בחלק א' הן פונקציות סטטיות.
- בקובץ `filters_test.py` קיימת פונקציה בשם `show_image(image)` – היעזרו בה כדי להבין מה בדיק מטריצות ה-`kernel` השונות מבוצעות בעת חישוב מטריצת הקורלציה.
- ניתן להוסיף הדפסות ומשתנים לפי רצונכם, אך הקוד חייב להיות נקי ומסודר.
- אין להסיר הדפסות והערות שכבר נמצאות בקוד, ויש להימנע להנחיות שמויפות בהערות.
- יש לטעד את הקוד המוגש.

- מומלץ לפתרו את התרגיל ב-`PyCharm`, אך יש למדוד את הביצועים רק בשרת הקורס. ניתן לדמות GPU במחשב המקומי באמצעות שינוי משתנה הסביבה `NUMBA_ENABLE_CUDASIM` ל-1, אך חשוב לקח בחשבון שהביצוע יהיה איטי באופן משמעותי.
- חשוב לא לשכוח להתקין את הספרייה `imageio` בשרת.
- ההנחיות לחיבור לשרת ולשימוש בו נמצאות בקישור <https://aihpc.cs.technion.ac.il/lambda>.

### 3.5. הגשה

יש להגיש קובץ בשם `hw2.zip`, שבו הקבצים הבאים בלבד:

- הקובץ `preprocessor.py` עם המימוש שלכם
- הקובץ `ip_network.py` עם המימוש שלכם
- הקובץ `my_queue.py` עם המימוש שלכם
- הקובץ `filters.py` עם המימוש שלכם
- הדוח המסכם בשם `hw2.pdf`, בפורמט PDF בלבד

**בהצלחה!**