תרגיל בית 2 - תכנות מקבילי ומבוזר לעיבוד נתונים ולמידה חישובית

סמסטר חורף 2025

0. הנחיות כלליות לתרגיל

- מועד אחרון לפתרון התרגיל: **יום שני ה - 20.1.2025 בשעה 23:59**
- מתרגל אחראי על תרגיל הבית: **יואב סהר**
- נושא התרגיל: **רשתות נוירונים עמוקות מקביליות, אוגמנטציה של נתונים**

- פתרון התרגיל בזוגות בלבד.
- חובה להגיש את התרגיל על מנת לקבל ציון בקורס.
- שאלות על התרגיל יש לשלוח למתרגל שאחראי על התרגיל.
- פניות בנוגע לקשיים בחיבור לשרתי הקורס יש להפנות למתרגל האחראי בקורס.
- בקשות לקבלת הארכה להגשת התרגיל יש להפנות למתרגל האחראי בקורס.
- בכל קובץ קוד שמוגש, יש לכתוב בהערה בתחילתו את שמות הסטודנטים ואת תאריך המילוי של הקובץ.

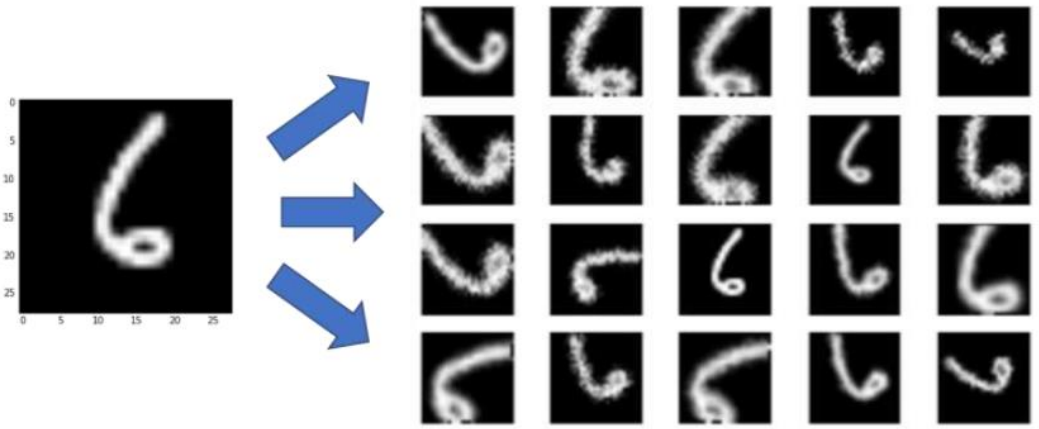
1. חלק א' – אוגמנטציה של נתונים ואימון מקבילי של רשתות נוירונים

1.1. רקע כללי

בהרצאה ראינו שניתן לאמן רשת נוירונים עמוקה (Deep Neural Network) בשימוש בקבוצת אימון $\{(x, f(x))\}$ כדי לקרב פונקציה לא ידועה $f(x)$. אחד האתגרים העיקריים עם רשתות נוירונים עמוקות היא יצירה של קבוצת אימון גדולה מספיק כדי שהקירוב יהיה מדויק. איסוף נתונים לבניית קבוצת האימון אינו פשוט – הוא עשוי לדרוש כסף, מאמץ אנושי, משאבים חישוביים וכמובן זמן. מסיבה זו, נרצה למצוא דרכים להגדלת קבוצת האימון תוך חסכון בעלויות הבנייה של קבוצה זו.

אוגמנטציה 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 העלויות 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, להגדיל אותן, לחתוך אותן, להשתמש בפילטרים שונים ועוד. מתמונה אחת ניתן ליצור 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 עמוקה לזיהוי ספרות שכתובות בכתב יד, באמצעות קבוצת אימון שתיווצר על ידי שיטות לאוגמנטציה של תמונות מהמאגר MNIST. מאגר זה מכיל תמונות בגווני אפור של ספרות. וניתן לקרוא יותר על מאגר זה בקישור https://en.wikipedia.org/wiki/MNIST_database .

המימוש יתבצע בהמשך לחלק 1 של תרגיל בית 1, ועליכם יהיה לממש שתי מחלקות: *Worker* ו־*IPNeuralNetwork*.

המחלקה *Worker* (מהקובץ *preprocessor.py*), אשר עליה לרשת מהמחלקה *Process* שנמצאת בספריית *multiprocessing* שב־Python, מכילה מספר פונקציות שבכל אחת מהן שיטה לביצוע אוגמנטציה של תמונות. למחלקה נדרש שדה עבור תור העבודות בשם *jobs* (שיכיל את כל התמונות שנדרש לבצע עליהן אוגמנטציה) ושדה עבור תור התוצאות בשם *results* (שיכיל את כל התמונות שהתקבלו מתהליך האוגמנטציה). בנוסף, במחלקה זו יהיו שתי מתודות נוספות – מתודה אשר מפעילה את כל פעולות האוגמנטציה על תמונה אחת שמתקבלת כפרמטר, בשם *process_image*; ומתודה אשר מוציאה תמונה מתור העבודות ומכניסה לתור התוצאות את התמונה שהתקבלה מהאוגמנטציה, בשם *run*.

המחלקה *IPNeuralNetwork* (מהקובץ *ip_network.py*), היורשת מהמחלקה *NeuralNetwork*, דורסת שתי מתודות – מתודה בשם *fit*, שבמקור הייתה אחראית לביצוע תהליך האימון של רשת הנוירונים העמוקה באמצעות Stochastic Gradient Descent, וכעת יש להוסיף אליה את הלוגיקה הדרושה לטיפול במחלקות העיבוד מטיפוס *Worker* על מנת ליצור את התמונות שהתקבלו מתהליך האוגמנטציה עבור האימון; ומתודה בשם *create_batches*, שמקבלת batch מקבוצת האימון המקורית ויוצרת גרסה של ה־batch שעברה אוגמנטציה על ידי עצמים מהמחלקה *Worker*.

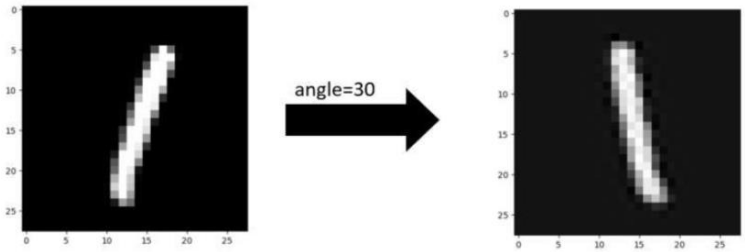
בהמשך, נסקור את פרטי המימוש של שתי מחלקות אלו. לפני כן, חשוב לשים לב להנחיות הבאות:

- על מנת להריץ את המחלקה *NeuralNetwork*, יש להוסיף את הקובץ *util.py* שכתבתם בתרגיל בית 1 לתיקיית 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 בנוסף את הפרמטר *number_of_batches*; הפונקציה בשם *create_batches* הפכה להיות מתודה של המחלקה בגרסה זו; ומתודה זו מחזירה batches אקראיים לפי הערך של *number_of_batches*.
- מומלץ במהלך מימוש המתודות הסטטיות עבור האוגמנטציה להשתמש בפונקציות המובנות שבספריות NumPy ו־SciPy, כדי להקל על המימוש.

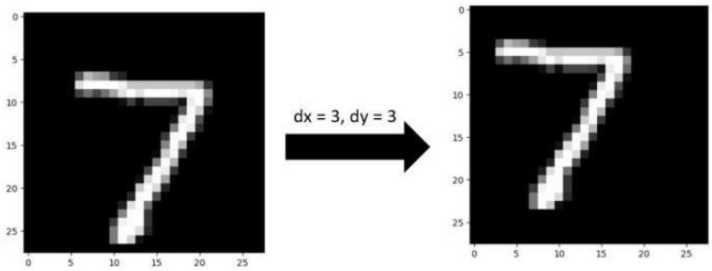
3.1. פרטי מימוש למחלקה *Worker*

במחלקה *Worker* עליכם לממש את המתודות והפונקציות הבאות:

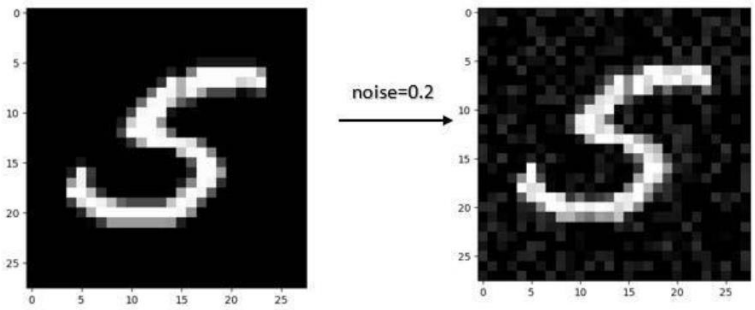
- הפונקציה *__init__(self, jobs, result)* תאתחל עצם ממחלקה זו ואת השדות שלו. עליכם לבחור את השדות שתרצו לשמור במחלקה זו, ומותר לצורך כל להוסיף פרמטרים לפונקציה.
- המתודה הסטטית *rotate(image, angle)* תחזיר את התמונה המתקבלת מסיבוב התמונה הנתונה בזווית הנתונה (מומלץ להיעזר בספרייה SciPy). לדוגמה:



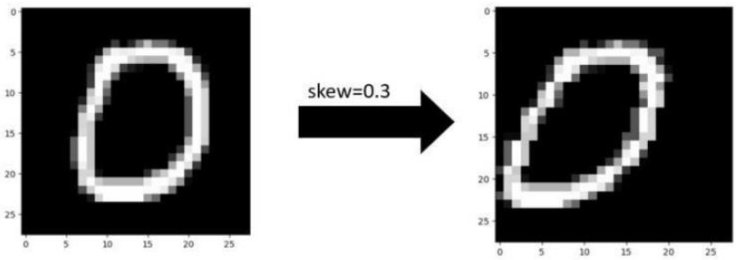
- המתודה הסטטית *shift(image, dx, dy)* תחזיר את התמונה המתקבלת מהזזת התמונה הנתונה *dx* תאים שמאלה ו־*dy* תאים למעלה (ערכים אלו יכולים להיות שליליים), כאשר מתייחסים לערכים בקואורדינטות מחוץ לתמונה כאל ערכי 0, כלומר כאל תאים שחורים (מומלץ להיעזר בספרייה SciPy). לדוגמה:



- המתודה הסטטית *add_noise(image, noise)* תחזיר את התמונה המתקבלת מבחירה של ערך מתפלג יוניפורמית בתחום $[-noise, noise]$ לכל תא בתמונה, והוספת ערך זה לערך הבהירות של התמונה תוך שמירה על תחום המותר לערכי בהירות (מומלץ להיעזר בספרייה NumPy). לדוגמה:



- המתודה הסטטית *skew(image, tilt)* תחזיר את התמונה המתקבלת מהטיית התמונה, כלומר על ידי חישוב הערכים בתמונת התוצאה באמצעות הנוסחה $result[i][j] = image[i][j + i \cdot tilt]$, כאשר מתייחסים לערכים בקואורדינטות מחוץ לתמונה כאל ערכי 0, כלומר כאל תאים שחורים. לדוגמה:



- המתודה *process_image(self, image)* תריץ את ארבעת המתודות הסטטיות, אחת אחרי השנייה בסדר לבחירתכם, כאשר מתחילים מהתמונה שמתקבלת בפרמטר. השתמשו ב־*random* שב־Python כדי לקבוע את שאר הארגומנטים למתודות, בתחומים לבחירתכם. נסו לשפר את סדר הביצוע ואת התחומים, באמצעות ניסויים קצרים, כדי לשפר את הדיוק המתקבל.
- המתודה run (self) תיצור את התמונות שנוצרות מהאוגמנטציה, לפי ההנחיות שצורפו קודם.

4.1. פרטי מימוש למחלקה *IPNeuralNetwork*

במחלקה *IPNeuralNetwork* עליכם לממש את המתודות הבאות:

- המתודה *create_batches(self, data, labels, batch_size)* צריכה לדרוס את המתודה הקיימת מהמחלקה *NeuralNetwork*, ועליה ליצור batches של התמונות לאחר האוגמנטציה (כלומר התמונות שהתקבלו מהתהליך ובנוסף אליהן התמונות המקוריות), ולא להשתמש רק בתמונות המקוריות.
- המתודה *fit(self, training_data, validation_data=None)* צריכה לדרוס את המתודה הקיימת מהמחלקה *NeuralNetwork*, ועליה ליצור עצמים מהמחלקה *Worker* בהתאם למספר יחידות העיבוד שניתן להשתמש בהן (היעזרו בערך *os.environ['SLURM_CPUS_PER_TASK']*), לקרוא למתודה הנדרסת (עם *super()*) ולאחר מכן לפנות את העצמים.

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

2. חלק ב' – תור מקבילי

בחלק זה, עליכם לממש תור מקבילי פשוט, ולאחר מכן עליכם לשנות את המחלקה *IPNeuralNetwork* כך שתשתמש בעצם ממחלקה זו בשדה תור התוצאות (אין צורך לשנות את טיפוס תור המשימות).

עליכם לממש את המחלקה בשימוש במחלקות Pipe ו־Lock מ־multiprocessing שראיתם בתרגולים. מחלקה זו צריכה לתמוך **בכותבים מרובים ובקורא יחיד** (many-writers, one-reader), כלומר ניתן להניח שקיים לכל היותר תהליך יחיד שקורא מהתור, וייתכן שקיים יותר מתהליך אחד שכותב לתור. השתמשו בהנחה זו כדי להחליט מתי צריך לסנכרן – ומתי סינכרוניזציה מיותרת.

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

- הפונקציה *__init__(self)* תאתחל תור ואת השדות של העצם.
- המתודה *put(self, msg)* תשלח את ההודעה (שהוא אובייקט כלשהו) דרך התור.
- המתודה *get(self)* תקרא הודעה מהתור (כלומר תחזיר את האובייקט שנשלח מהתור).
- המתודה *empty(self)* תחזיר האם תור ההודעות ריק (כלומר שאין ערכים שממתינים לקריאה).

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

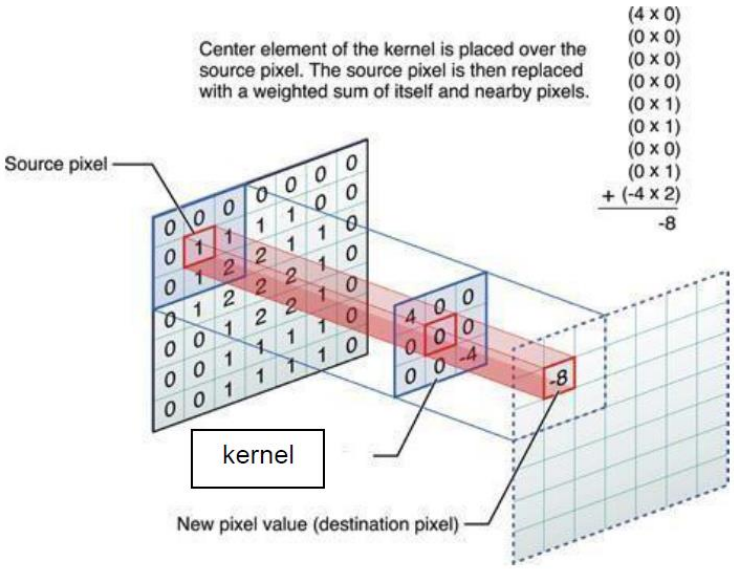
מימוש חלק זה קצר ופשוט יחסית, והמימוש של כל מתודה אמור להיות באורך של כ־3-5 שורות.

3. חלק ג' – חישוב מקבילי של kernel

קורלציה (Correlation) היא פעולה מתמטית שמקבלת שני סיגנלים כקלט ומחזירה סיגנל שלילי כפלט, והיא מצויה בשימוש רחב במגוון של ענפים ויישומים. בעיבוד תמונה ניתן להשתמש בפעולה זו לטשטוש, לחידוד, להבלטה, לזיהוי קצוות ולשימושים נוספים.

מטריצת הקורלציה היא התוצאה שמתקבלת מחישוב ערך הקורלציה בין תמונה נתונה, לכל מיקום של פיקסל בה, לבין מטריצה קטנה הקרויה kernel. ניתן לחשוב על מטריצת הקורלציה כהחלפה של כל פיקסל בתמונה בסכום משוקלל שלו ושל שכניו, כאשר המשקלים מוגדרים על ידי ה־kernel. בנוסף, אם החישוב מתבצע ליד הקצוות וחלק מהשכנים אינם קיימים ('נמצאים' מחוץ לגבולות המטריצה), אנו מתייחסים אליהם כבעלי הערך 0 (כלומר אנו 'מרפדים' את התמונה הנתונה באפסים).

לדוגמה, אם ה־kernel הוא מטריצה בגודל 3×3 שכל ערכיו הם 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 לממש בקובץ `filters.py` שתי פונקציות שמחשבת את מטריצת הקורלציה – פעם אחת בשימוש ב־njit להאצה ופעם אחת בשימוש ב־GPU:

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

וודאו שהתוצאות שמתקבלות בהינתן *`kernel`* ותמונה, גם עבור המימוש עם njit וגם עבור המימוש עם GPU, זהות לתוצאות שמתקבלות בשימוש ב־*`scipy.signal.convolve2d(flipped_kernel, image)`*, כאשר *`flipped_kernel`* הוא ה־*`kernel`* לאחר סיבובו ב־$180°$ (היפוך העמודות והשורות).

4. חלק ד' – זיהוי קצוות בתמונה בעזרת אופרטור Sobel

בחלק זה, נראה שימוש בפונקציות שמימשתם קודם, כדי לזהות קצוות בתמונה. בחלק זה, עליכם להשתמש בפונקציה *correlation_numba(kernel, image)* שמימשתם בחלק הקודם.

עליכם לממש בקובץ filters.py את הפונקציה *sobel_operator()*, אשר מבצע את החישוב הבא:

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

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

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

$$result_{i,j} = \sqrt{{G_x}_{i,j}^2 + {G_y}_{i,j}^2}$$

הערך המוחזר על ידי הפונקציה הוא המטריצה $result$, בממדים של התמונה, אשר כל ערך בו מחושב לפי הנוסחאות שתוארו קודם. צרפו לדוח את התוצאה שמתקבלת עבור $sobel\_filter$ הנתון, וכן גם עבור מטריצות ה־kernel הבאות (המימוש המוגש של הפונקציה צריך לפעול בשימוש ב־$sobel\_filter$):

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

למימוש קל מומלץ להשתמש בפונקציות *transpose*, *sqrt* ו־*pow* מהספרייה NumPy. בנוסף, בקוד שמסופק לכם, נעשה שימוש בפונקציה *imread* מהספרייה Imageio, ובפונקציה *imshow* (עם *cmap='gray'*) מהספרייה Matplotlib-pyplot להצגת התוצאה.

לאחר המימוש, וודאו שהתוצאה המתקבלת זהה בהשוואה בין אחד המימושים שלכם מהחלק הקודם (ניתן להשתמש עבור הבדיקה ב־*correlation_numba*, להרצה על המחשב המקומי), לבין המימוש שמתקבל בשימוש ב־*scipy.signal.convolve2d* עם מטריצת ה־kernel המהופכת.

ניתן לקרוא יותר על אופרטור Sobel בקישור https://www.tutorialspoint.com/dip/sobel_operator.htm .

5. דוח מסכם והנחיות כלליות

1.5. שאלות עבור הדוח המסכם

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

(1) הריצו בחלק א' את `main.py` עם 8, 16 ו־32 יחידות עיבוד (cores) – הדגל להרצה הוא `-c <cores>`. השוו את זמני הריצה של *IPNerualNetwork* (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 `main.py`, והשוו את אחוז הדיוק שמתקבל ב־epochs שונים, בין *NeuralNetwork* לבין *IPNeuralNetwork*. צרפו טבלה להשוואה והסבר קצר.

(3) לפי מה בחרתם את כמות ה workers שבחרתם? מה היה קורה לו הייתם בוחרים יותר/פחות מידי workers?

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

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

(6) הסבירו את המימוש שלכם לחלק ב', האם המימוש היה שונה לולא הנחנו קורא אחד לכל היותר מהתור? מדוע?

(7) הסבירו בפירוט כיצד מימשתם בחלק ג' את *correlation_numba* ואת *correlation_gpu*.

(8) הריצו את `filters_test.py` עם יחידת עיבוד (core) אחת, וצרפו צילום מסך שמראה את ההאצה בין המימושים השונים לבין המימוש של SciPy (עם *convolve2d*). הסבירו את התוצאות.

(9) לפי תוצאות הסעיף הקודם, מתי נעדיף להשתמש ב njit ומתי נעדיף להשתמש בcuda? 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 ההבדלים בין התוצאות השונות, והסבירו מה הסיבות להבדלים אלו.

2.5. הערות ועצות

- בחלק א' התמונות הן מטריצות NumPy בגודל 1×784, אך יש לטפל בתמונות לפי הגודל 28×28.
- הפונקציות *rotate*, *shift*, *add_noise* ו־*skew* בחלק א' הן פונקציות סטטיות.
- בקובץ filters_test.py קיימת פונקציה בשם *show_image(image)* – היעזרו בה כדי להבין מה בדיוק מטריצות ה־kernel השונות מבצעות בעת חישוב מטריצת הקורלציה.
- ניתן להוסיף הדפסות ומשתנים לפי רצונכם, אך הקוד חייב להיות נקי ומסודר.
- אין להסיר הדפסות והערות שכבר נמצאות בקוד, ויש להיצמד להנחיות שמופיעות בהערות.
- יש לתעד את הקוד המוגש.

- מומלץ לפתור 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 לקחת 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**, ובו 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!