

236370 - תכנות מקבילי וմבוזר לעיבוד נתונים ולמידת מכונה

דוח תרגיל בית 2

טל בן עמי 212525257

קורן מעברי 207987314

1. לא הצלחתי להריץ ביותר מ 4 ליבוט את התכנית על השרת.
 הרצתי לוקלית لكن התוצאות עלולות לשינוי
 הריצות שביצעת הן על 4,8,12 ליבוט:

```
Epoch 1, accuracy 9.67 %.
Epoch 2, accuracy 64.8 %.
Epoch 3, accuracy 70.9 %.
Epoch 4, accuracy 80.77 %.
Epoch 5, accuracy 82.59 %.
Epoch 6, accuracy 83.13 %.
Epoch 7, accuracy 83.2 %.
Epoch 8, accuracy 83.09 %.
Epoch 9, accuracy 83.01 %.
Epoch 10, accuracy 83.19 %.
Epoch 11, accuracy 83.01 %.
Epoch 12, accuracy 82.99 %.
Epoch 13, accuracy 83.35 %.
Epoch 14, accuracy 83.24 %.
Epoch 15, accuracy 83.18 %.
Time regular: 2.5879921913146973
Test Accuracy: 81.34957983193277%
Creating 8 workers
Epoch 1, accuracy 22.08 %.
Epoch 2, accuracy 62.57 %.
Epoch 3, accuracy 73.67 %.
Epoch 4, accuracy 83.61 %.
Epoch 5, accuracy 84.39 %.
Epoch 6, accuracy 84.1 %.
Epoch 7, accuracy 86.29 %.
Epoch 8, accuracy 88.18 %.
Epoch 9, accuracy 89.07 %.
Epoch 10, accuracy 89.53 %.
Epoch 11, accuracy 90.17 %.
Epoch 12, accuracy 90.15 %.
Epoch 13, accuracy 90.34 %.
Epoch 14, accuracy 90.7 %.
Epoch 15, accuracy 89.83 %.
Time with image processing: 6.563018798828125
Test Accuracy: 88.70420168067227%
```

```
Epoch 1, accuracy 26.05 %.
Epoch 2, accuracy 63.81 %.
Epoch 3, accuracy 75.35 %.
Epoch 4, accuracy 80.08 %.
Epoch 5, accuracy 82.33 %.
Epoch 6, accuracy 83.21 %.
Epoch 7, accuracy 83.63 %.
Epoch 8, accuracy 83.8 %.
Epoch 9, accuracy 83.63 %.
Epoch 10, accuracy 83.78 %.
Epoch 11, accuracy 83.78 %.
Epoch 12, accuracy 83.85 %.
Epoch 13, accuracy 83.83 %.
Epoch 14, accuracy 83.89 %.
Epoch 15, accuracy 83.81 %.
Time regular: 2.719339370727539
Test Accuracy: 81.77310924369749%
Creating 4 workers
Epoch 1, accuracy 21.78 %.
Epoch 2, accuracy 61.95 %.
Epoch 3, accuracy 72.94 %.
Epoch 4, accuracy 81.27 %.
Epoch 5, accuracy 85.03 %.
Epoch 6, accuracy 86.58 %.
Epoch 7, accuracy 87.37 %.
Epoch 8, accuracy 87.8 %.
Epoch 9, accuracy 86.68 %.
Epoch 10, accuracy 89.82 %.
Epoch 11, accuracy 89.08 %.
Epoch 12, accuracy 90.33 %.
Epoch 13, accuracy 89.7 %.
Epoch 14, accuracy 90.51 %.
Epoch 15, accuracy 91.07 %.
Time with image processing: 6.109336853027344
Test Accuracy: 89.89243697478992%
```

```
(hw2) C:\programming\parallel\hw2_cdp>python main.py
Epoch 1, accuracy 21.98 %.
Epoch 2, accuracy 60.68 %.
Epoch 3, accuracy 74.91 %.
Epoch 4, accuracy 82.51 %.
Epoch 5, accuracy 83.69 %.
Epoch 6, accuracy 84.42 %.
Epoch 7, accuracy 84.36 %.
Epoch 8, accuracy 84.04 %.
Epoch 9, accuracy 83.97 %.
Epoch 10, accuracy 83.9 %.
Epoch 11, accuracy 83.79 %.
Epoch 12, accuracy 83.87 %.
Epoch 13, accuracy 83.79 %.
Epoch 14, accuracy 83.84 %.
Epoch 15, accuracy 83.86 %.
Time regular: 2.6298906803131104
Test Accuracy: 82.0%
Creating 12 workers
Epoch 1, accuracy 32.47 %.
Epoch 2, accuracy 61.43 %.
Epoch 3, accuracy 73.91 %.
Epoch 4, accuracy 85.41 %.
Epoch 5, accuracy 87.29 %.
Epoch 6, accuracy 86.21 %.
Epoch 7, accuracy 87.29 %.
Epoch 8, accuracy 89.19 %.
Epoch 9, accuracy 88.35 %.
Epoch 10, accuracy 89.51 %.
Epoch 11, accuracy 88.28 %.
Epoch 12, accuracy 90.5 %.
Epoch 13, accuracy 91.07 %.
Epoch 14, accuracy 89.41 %.
Epoch 15, accuracy 91.15 %.
Time with image processing: 7.716958284378052
Test Accuracy: 90.33445378151261%
```

מספר ייחידות העיבוד שנותן את התוצאה הטובה ביותר: בהתבסס על זמן הריצה, השימוש ב-4 ייחידות עיבוד (Workers) הניב את הזמן המהיר ביותר (6.109 שניות). ככל שהגדלנו את מספר ה-Workers (ל-8 ול-12), זמן הריצה הכלול לא השתרף אלא דוקא התקאר.

זה נבע מ:

- 1) תקורת הנעלויות שבמצעים על התורמים דרך תורים דורשת זמן יקר של סyncron. כאשר פעולה העיבוד עצמה (האגמנטציה) היא יחסית קצרה, הזמן שלוקח לניהל את התקשרות עולה על הזמן שנחסך בחישוב המקביל.
- 2) יצירה וניהול של מטפים רב של תהליכיים צורכים משאבי מערכת וזמן שימושו ב-context switching.
- 3) חוק אמדל (Amdahl's Law): ישנו חלק סדרתי בתוכנית שאינו ניתן למקבול. הוספה מעבדים משפרת רק את החלק המקביל, אך התקורה הנוסף גורמת בסופו של דבר להאטה (chattiness) (Parallel Slowdown) כאשר עוברים את נקודת האיזון האופטימלית עברו גודל המשימה הספציפי זהה.

```
● (tf23-gpu) tal-ben@lambda:~/hw2_cdp$ srun --gres=gpu:1 -c 2 --pty python3 main.py
Epoch 1, accuracy 20.3 %.
Epoch 2, accuracy 49.49 %.
Epoch 3, accuracy 73.72 %.
Epoch 4, accuracy 78.96 %.
Epoch 5, accuracy 82.77 %.
Epoch 6, accuracy 83.21 %.
Epoch 7, accuracy 83.55 %.
Epoch 8, accuracy 83.9 %.
Epoch 9, accuracy 83.9 %.
Epoch 10, accuracy 83.88 %.
Epoch 11, accuracy 84.03 %.
Epoch 12, accuracy 84.03 %.
Epoch 13, accuracy 84.01 %.
Epoch 14, accuracy 83.95 %.
Epoch 15, accuracy 83.61 %.
Time regular: 9.45073676109314
Test Accuracy: 82.0%
Creating 2 workers
Epoch 1, accuracy 20.55 %.
Epoch 2, accuracy 64.35 %.
Epoch 3, accuracy 76.09 %.
Epoch 4, accuracy 83.32 %.
Epoch 5, accuracy 86.89 %.
Epoch 6, accuracy 85.99 %.
Epoch 7, accuracy 88.43 %.
Epoch 8, accuracy 89.13 %.
Epoch 9, accuracy 88.26 %.
Epoch 10, accuracy 88.63 %.
Epoch 11, accuracy 89.6 %.
Epoch 12, accuracy 90.61 %.
Epoch 13, accuracy 90.45 %.
Epoch 14, accuracy 91.18 %.
Epoch 15, accuracy 90.55 %.
Time with image processing: 45.93725109100342
Test Accuracy: 89.26218487394958%
```

2. ניתן לראות שאחד הדיק ב-IPNeuralNetwork (הכוללת אוגמנטציות) גבוהים יותר הן ב-chain והן ב-Test לעומת הרשות הרגילה. הסיבה לכך היא שהאגמנטציות מייצרות דוגמאות אימון מגוונות (סיבובים, רעש, חזוזות) מתוך המידע המקורי. גיון זה מנסה על המודל לשנן את המידע (Overfitting) ומאלץ אותו ללמידה מאפיינים כליליים יותר של הספרות, מה שmobilit ליכולת הכללה טובה יותר על נתונים שלא ראה (Test Set).

3. בחרנו את מספר ה-workers כמספר הליבות בהן התהילה מוגדר לרוץ בשרת. אך אם היינו בוחרים יותר מדי workers אז הזמן overhead של אתחולם וונכראן יהיה דומיננטי מזמן העבודה שייקח ללבות לסימן באופן כזה שסכום הזמן יהיה גדול מהזמן העבודה שיקח ללבוה אחת לסימן ולכן נקבל יתרה ביצועים. אם היינו בוחרים יותר coresworkers מcoresworkers אז היינו גם מקבלים תקורת context switching אם היינו בוחרים מעט מדי workers אז overhead האתחול וונכראן יהיה יותר קטן וזמן

העבודה יהיה יותר גדול כך שסכום הזמןנים גם כן יהיה יותר גדול מזמן עבודה שיקח ליליבת אחת לשיטם.

4. python משתמש במנוע גלובלי (GIL) שנועל את כל משאבי ה interpreter runtime של ה runtime שבמהר חוט יכול לגעת בכך למנוע data races בזמן אינטראקטיבית של תכנית python. (לאחרונה יש דיבור על החלפת המימוש כך שתאפשר ההסרה של המנוע). מכאן שאנו משתמשים בספרית multiprocessing שבעצם עושה שימוש בתהליכים שונים שלא משתפים את אותו מרחב הדיכרון ב HEAP כך שאין חשש מ data races פנימיים של ה interpreter (אך כן יש צורך לדאוג בכך עצמו שאין גישות בעייתיות למשאבי משותפים של התהליכים כפי שראינו בתרגיל)

5. ניתן לבצע את האוגננטציות ב GPU ולהעביר את כל משקליו המודול והדוגמאות המקוריות גם כן ל GPU כך שכל החישובים שמתבצעים בתהליך האימון יתבצעו ב GPU במקומם במעבד. ניתן ליצור ולשמור את האוגננטציות כך שייהי ניתן ישר להשתמש בהן בritchאות הבאות במקומם ליצור אותן במהלך ה-.run time.

6. ממשנו את התור באמצעות multiprocessing.Pipe. השפעת הנחת קורא יחיד: המימוש שלנו משתמש ב- Lock אך ורק בפונקציית the-recv (כדי לסנן בין כתבים רבים שעולמים לנוסות לכתוב ל-Pipe בו זמן). בפונקציית the-get, אנו קוראים שירות (self.recv_conn.recv()) ללא מניעול. LOLA הנחה זו: אם היי מספר תהליכי קוראים (Readers), היינו חייבים להויסף מניעול (Lock) גם בפונקציית the-get. ללא סנכרון בקריאה, שני תהליכי היו עלולים לנוסות לקרוא מה-Pipe בו-זמן, מה שהיא מוביל למצב שבו הודעה 'נכ儒家' או שגיאות בגיןה למשאב המשותף.

7. מימוש correlation_numba :

השתמשנו בדקורטור @njit(parallel=True). המימוש נועד להציג את חישוב הקורלציה על המעבד באמצעות מהדר njit של ספריית Numba. השתמשנו בדקורטור זה כדי ל�מפל את פונקציית הפיתון לקוד מכונהיעיל בזמן ריצה. הfrmטטर parallel=True מאפשר ל-Numba לבצע אופטימיזציה של LOLAOות באופן מקבילי אוטומטי (Auto-parallelization) על גבי ליבות המעבד השונות.

הכנות הנתונים (Padding):

לפני ביצוע הקורלציה, ביצענו ריפורד (Padding) של התמונה באפסים מסביב לשוליה. גודל הריפורד (w_pad, pad_h) מחושב כחצי מגודל הקרמל. שלב זה הכרחי כדי שנוכל לחשב את הקורלציה גם עבור הפיקסלים הנמצאים בקצוות התמונה (אנו לא משפיעים על התוצאות בגל האפסים) המקורית מבלתי לחרוג מגבולות המערך.

lolola החישוב (prange):

lolola החישונית משתמשת ב-prange (במקום range רגיל). זהה פקודה ספציפית ל-Numba המורה למادر לחלק את האיטרציות של lolola (במקרה זה, השורות של התמונה) בין התהליכים (Threads) של המעבד ולבצע אותן במקביל.

בתוך lolola הפנימית, עבור כל פיקסל (j,i), חילכנו את האזורי הרלוונטי מהתמונה המרופדת (region) בגודל הקרמל וביצענו כפל איבר-איבר (Element-wise multiplication) בין האזורי לבין הקרמל, וככמנו את התוצאה (sum.sum). הערך המתkeletal נשמר במטריצת התוצאה.

חלק ב': מימוש correlation_cuda (ריצה על ה-GPU)

המימוש מחולק לשתי פונקציות: פונקציית מעתפת ("Host") המכינה את הנתונים, ופונקציית הליבה ("Kernel") הרצה על הכרטיסי הגרפי.

1. הפונקציה `cuda_gemm` (Host Code correlation) מפעילה פונקציה זו אחראית על ניהול הזיכרון והזנת התהיליכים ב-GPU.

הכנת נתונים והמרת טיפוסים: בדומה למקודם מבצעים ריפורט (Padding) של התמונה.

העברה ל-Device: שימוש ב-`cuda.to_device` כדי להעתיק את התמונה המרופדת ואת הקERNEL מזיכרון המחשב (Host) לזיכרון ה-(Device) GPU.

чисוב גRID החוטים (Grid & Block Dimensions):

נקבע גודל בלוק של 16×16 תהיליכונים (threads per block).

מספר הבלוקים (blocks per grid) מחושב כך שיכסה את כל פיקסל התמונה (גובה ורוחב התמונה חלקו גודל הבלוק, עם עיגול כלפי מעלה).

הפעלת הקERNEL: קרייה ל-`kernel.correlation` עם תצורת הגRID והפרמטרים הדרושים.

קבלת התוצאה: העתקת התוצאה חזרה מה-GPU ל-CPU באמצעות `copy_to_host`.

2. הפונקציה `correlation_kernel` (Device Code):
זהה הפונקציה שרצה במקביל על אלפי תהיליכונים ב-GPU (డקורטור @`cuda.jit`).

זהוי הפיקסל (`cuda.grid`): כל תהיליכון ("Thread") מזזה על איזה פיקסל (j,i) הוא אחראי במאזעות (2). `cuda.grid`: מכוון שמספר הבלוקים הוא כפולת של 16, יתכונו תהיליכונים שחורגים מגודל התמונה האמתי. התנאי `w_i < j and h_i < i` אם מבטיח שרק תהיליכונים רלוונטיים יבצעו חישוב.

чисוב הקורלציה:

כל תהיליכון מבצע לולאה כפולה קטנה על מימדי ה-(`ki, kj`). `kernel`:

הוא ניגש לפיקסלים המתאימים בתמונה המרופדת (`padded_image`) ובמצע כפל-סכום (`Multiply-Add`) עם ערכי הקERNEL.

מכיוון שכל תהיליכון כותב לתא זיכרון נפרד במטריצת התוצאה (`result[i,j]`), אין צורך במנועלים או סנסרין מורכב (Race Condition) בכתיבתיה.

```

● (tf23-gpu) tal-ben@lambda:~/hw2_cdp$ srun --gres=gpu:1 -c 1 --pty python3 filters_test.py
CPU 3X3 kernel: 0.0017057880759239197
Numba 3X3 kernel: 0.0015553124248981476
CUDA 3X3 kernel: 0.0012050382792949677
-----
CPU 5X5 kernel: 0.0037094131112098694
Numba 5X5 kernel: 0.0025892816483974457
CUDA 5X5 kernel: 0.0011866390705108643
-----
CPU 7X7 kernel: 0.0071222372353076935
Numba 7X7 kernel: 0.004271987825632095
CUDA 7X7 kernel: 0.0012408047914505005
-----
○ (tf23-gpu) tal-ben@lambda:~/hw2_cdp$ █

```

.8

ניתן לראות שיפור בביצועים במעבר מ-CPU ל-Numba, ושיפור נוסף במעבר ל-CUDA.

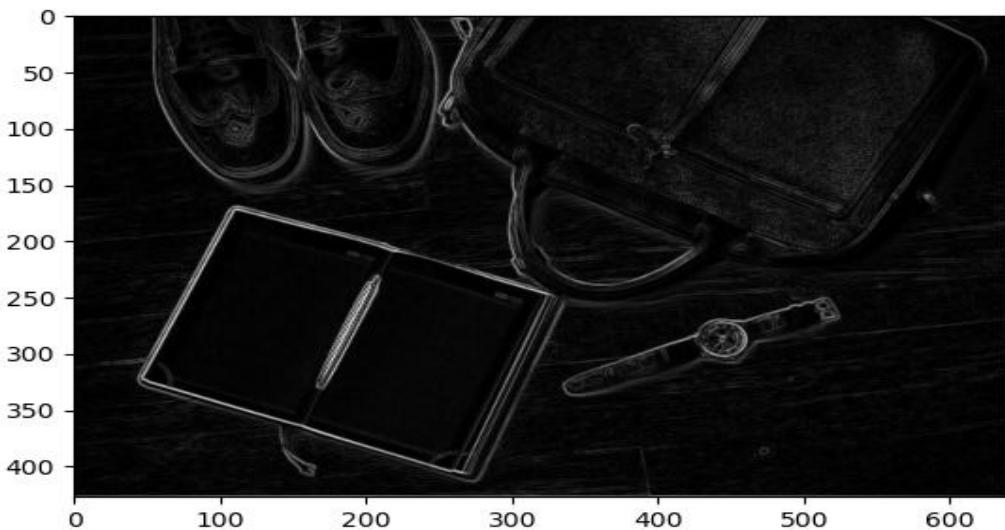
מימוש scipy: תהילך אחד שמריץ את הקוד ווסבל מ"תקורת המפרש" (Interpreter Overhead) ואינה מנצלת היטב את המעבד לחישובים מקבילים ברמת הפייקסל.

Numba: מהיר יותר מה-CPU הרגיל למراتות שגם כן רץ על CPU עם core בודד. המהדר (DJIT Compiler) הופר את קוד הפיתון לקוד מכונה אופטימלי בזמן ריצה. זה חוסר את התקורת של המפרש ומאפשר אופטימיזציות של המעבד (כגון וקטורייזציה).

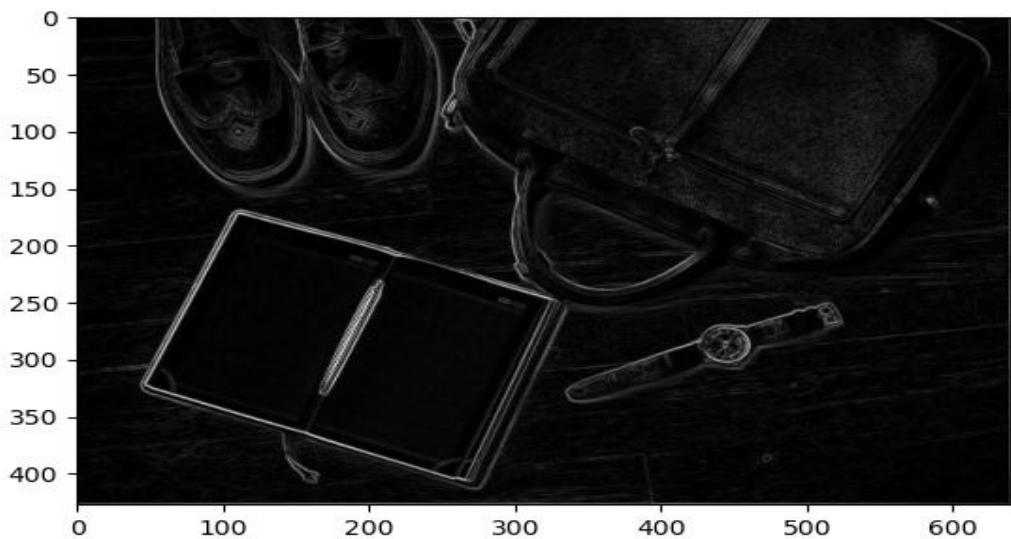
CUDA: מהיר ביותר (במקרה זה). GPU הוא מעבד המיעוד לעיבוד מקבילי מסיבי. בפעולות קונפלואנטיה, כל פיקסל מחושב באופן בלתי תלוי, מה שמאפשר לאלפי ליבוט ה-GPU לעבוד במקביל. החישוב כולל את הפעולה הבנויה fused multiply-add (FMA) שרצה מאוד מהר על gpu

9. נרצה להשתמש בnumba עבור חישוב קורולציה של kernels קטנים ונרצה כموנן להריץ את numba עם יותר מעבדים. זה מכיוון שנראה מההתוצאות שיכלשה kernel יותר קטן הזמן חישוב של numba קטן מכיוון שימושה החישוב קטנה ככל שמקטיניהם את ה kernel אז המעבד מסיים את עבודתו יותר מהר. נשים לב שזמן cuda לא משתנים, ככל הנראה זה נובע מכך שימושה החישוב קטנות מיד' כך ש cuda הוא memory bound עבור הקרנלים הקטנים.

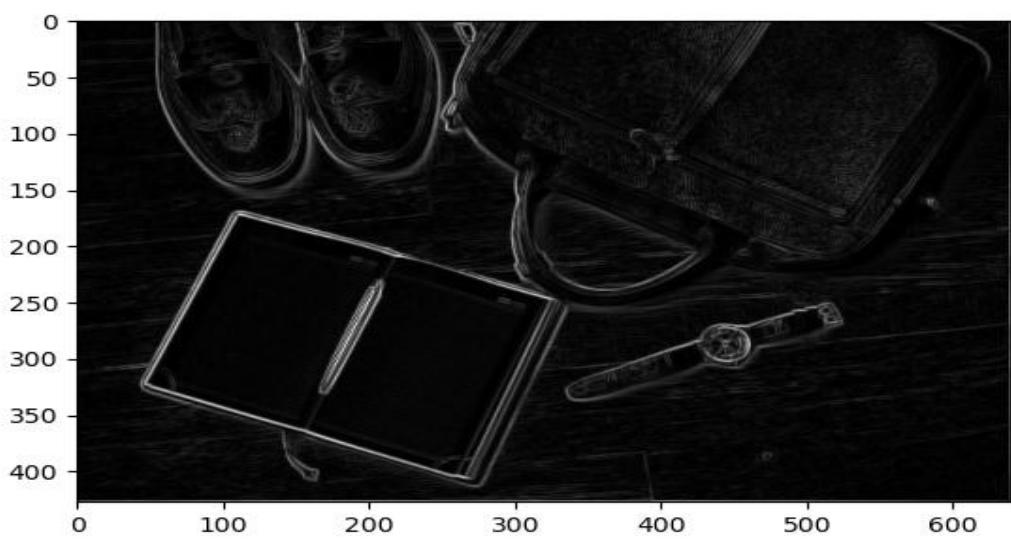
sobel_filter .10

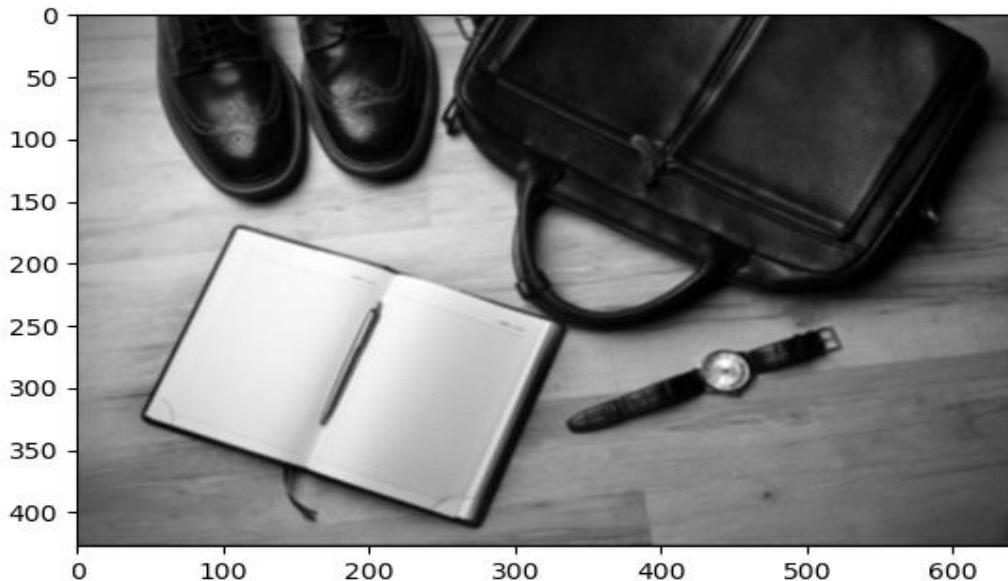


kernel1:



kernel2:





:sobel

התוצאה: **זיהוי קצוט סטנדרטי**

הסבר: המטריצה מכילה ערכים חיוביים בצד שמאל ושליליים בצד ימין כך שיחסוב קורולציה עם מטריצה זו גורם לҚקצוט שמאליים בתמונה להיות יותר יתר בהירים וקצוטים ימניים להיות יותר כהים.

: Kernel 1

התוצאה: האפקט זהה למקודם רק שהוא יותר חזק יותר (בעל ניגודיות יותר גדולה)

הסבר: המטריצה נותנת משקל גבוה יותר למרכז (10 במקום 2) ולערכים הקיצוניים (3 במקום 1), מה שմגביר את הרגישות לҚקצוט.

Kernel 2 (מטריצה מוארכת 3x3):

התוצאה: **זיהוי קצוט "חלק" יותר.**

הסבר: המטריצה הגבואה מבצעת החלקה (Smoothing) על פני 5 פיקסלים אנכיאים, מה שמחזית רעשים אנכיאים אך שומר על הקצוות הראשיים.

: Kernel 3

התוצאה: **תמונה מטושטשת ובהירה (לא זיהוי קצוט).**

הסבר: המטריצה מכילה רק ערכים חיוביים. היא מבצעת סכימה/מוצע של פיקסלים ולא חיסור. לכן, במקום לקבל הפרשיים (קצוט), מקבלים אפקט של טשטוש (Blur) בהיר על התמונה המקורית.