

תרגיל בית 2 - מקבוץ

.1

```
(tf23-gpu) dashay@lambda:~/CDPwinter26/HW2$ srun --gres=gpu:1 -c 8 --pty python3 main.py
Epoch 1, accuracy 19.54 %.
Epoch 2, accuracy 53.02 %.
Epoch 3, accuracy 73.39 %.
Epoch 4, accuracy 76.6 %.
Epoch 5, accuracy 81.59 %.
Epoch 6, accuracy 82.55 %.
Epoch 7, accuracy 83.24 %.
Epoch 8, accuracy 82.93 %.
Epoch 9, accuracy 82.99 %.
Epoch 10, accuracy 82.88 %.
Epoch 11, accuracy 82.8 %.
Epoch 12, accuracy 82.81 %.
Epoch 13, accuracy 82.9 %.
Epoch 14, accuracy 82.9 %.
Epoch 15, accuracy 82.97 %.
Time regular: 10.117957592010498
Test Accuracy: 80.98151260504201%
Starting 8 workers...
Epoch 1, accuracy 9.9 %.
Epoch 2, accuracy 33.19 %.
Epoch 3, accuracy 57.82 %.
Epoch 4, accuracy 70.01 %.
Epoch 5, accuracy 73.59 %.
Epoch 6, accuracy 75.44 %.
Epoch 7, accuracy 83.34 %.
Epoch 8, accuracy 84.81 %.
Epoch 9, accuracy 84.63 %.
Epoch 10, accuracy 86.72 %.
Epoch 11, accuracy 88.16 %.
Epoch 12, accuracy 87.19 %.
Epoch 13, accuracy 87.79 %.
Epoch 14, accuracy 88.72 %.
Epoch 15, accuracy 89.13 %.
Time with image processing: 26.484982013702393
Test Accuracy: 87.9109243697479%
```

```
(tf23-gpu) dashay@lambda:~/CDPwinter26/HW2$ srun --gres=gpu:1 -c 16 --pty python3 main.py
Epoch 1, accuracy 22.51 %.
Epoch 2, accuracy 63.99 %.
Epoch 3, accuracy 75.38 %.
Epoch 4, accuracy 78.47 %.
Epoch 5, accuracy 82.74 %.
Epoch 6, accuracy 82.9 %.
Epoch 7, accuracy 83.21 %.
Epoch 8, accuracy 83.28 %.
Epoch 9, accuracy 83.39 %.
Epoch 10, accuracy 83.53 %.
Epoch 11, accuracy 83.62 %.
Epoch 12, accuracy 83.72 %.
Epoch 13, accuracy 83.56 %.
Epoch 14, accuracy 83.79 %.
Epoch 15, accuracy 83.7 %.
Time regular: 9.678184032440186
Test Accuracy: 82.04873949579832%
Starting 16 workers...
Epoch 1, accuracy 10.67 %.
Epoch 2, accuracy 28.32 %.
Epoch 3, accuracy 63.03 %.
Epoch 4, accuracy 73.03 %.
Epoch 5, accuracy 72.5 %.
Epoch 6, accuracy 79.17 %.
Epoch 7, accuracy 83.42 %.
Epoch 8, accuracy 84.02 %.
Epoch 9, accuracy 86.46 %.
Epoch 10, accuracy 86.0 %.
Epoch 11, accuracy 86.82 %.
Epoch 12, accuracy 87.5 %.
Epoch 13, accuracy 87.82 %.
Epoch 14, accuracy 88.69 %.
Epoch 15, accuracy 89.1 %.
Time with image processing: 19.985740423202515
Test Accuracy: 87.509243697479%
```

```
(tf23-gpu) dashay@lambda:~/CDPwinter26/HW2$ srun --gres=gpu:1 -c 32 --pty python3 main.py
Epoch 1, accuracy 19.77 %.
Epoch 2, accuracy 63.28 %.
Epoch 3, accuracy 73.69 %.
Epoch 4, accuracy 79.14 %.
Epoch 5, accuracy 82.39 %.
Epoch 6, accuracy 83.15 %.
Epoch 7, accuracy 83.33 %.
Epoch 8, accuracy 83.78 %.
Epoch 9, accuracy 83.68 %.
Epoch 10, accuracy 83.6 %.
Epoch 11, accuracy 83.96 %.
Epoch 12, accuracy 83.86 %.
Epoch 13, accuracy 83.73 %.
Epoch 14, accuracy 83.78 %.
Epoch 15, accuracy 83.81 %.
Time regular: 6.760045766830444
Test Accuracy: 81.8689075630252%
Starting 32 workers...
Epoch 1, accuracy 10.9 %.
Epoch 2, accuracy 23.99 %.
Epoch 3, accuracy 57.13 %.
Epoch 4, accuracy 73.35 %.
Epoch 5, accuracy 76.46 %.
Epoch 6, accuracy 79.44 %.
Epoch 7, accuracy 81.86 %.
Epoch 8, accuracy 83.19 %.
Epoch 9, accuracy 86.08 %.
Epoch 10, accuracy 86.65 %.
Epoch 11, accuracy 85.85 %.
Epoch 12, accuracy 88.76 %.
Epoch 13, accuracy 87.83 %.
Epoch 14, accuracy 89.2 %.
Epoch 15, accuracy 89.11 %.
Time with image processing: 15.985614776611328
Test Accuracy: 87.50252100840336%
```

עבור 8 יחידות עיבוד קיבלנו זמן של 26.48 שניות, עבור 16 יחידות עיבוד קיבלנו זמן של 19.98 שניות ועבור 32 יחידות עיבוד קיבלנו זמן של 15.98 שניות. זמן החישוב יורד ככל שמספר יחידות היעבוד עולה, זאת מכיוון שהקל שיש יותר יחידות עיבוד, קיימים יותר תהליכיים במערכת שMRI'ים את עיבוד התמונות (היות שאצלנו במערכת מספר threads = number of threads). בכלל זה, תוך התוצאות מתמלא מהר יותר וזה מה שמאיצץ את פעולה ה-create batches בטור ipknueralnetwork.ip. אף על פי כן, קיבלנו שאחד הדיקוקים accuracy כמעט זהה בין שלוש ההרצאות, ולכן מסבירים כי הדיקוק אינו תלוי במספר יחידות היעבוד.

2. נתבונן בהרצה של 16 ליבות מהסעיף הקודם:

IP nueral network	Neural network	epoch
0.67§	22.51	1
28.32	63.99	2
63.03	75.38	3
73.03	78.47	4

72.5	82.74	5
79.17	82.9	6
83.42	83.21	7
84.02	83.28	8
86.46	83.39	9
86.0	83.53	10
86.82	83.62	11
87.5	83.72	12
87.82	83.56	13
88.69	83.79	14
89.1	83.7	15

תחליה נשים לב כי neural network מבצעת קפיצה גדולה ומהירה בגודל ה-epochים הראשונים, בפרט היא מגיעה ל-accuracy יותר מאשר neural networkipo בין epochים 1-6. ככל שמספר האפוקים גדל נשים לב כי גודל ה-accuracy של neural networkipo נזקן לאפוק, זאת לעומת accuracy של neural network שכך הוא מתחילה להתקבע סביבה 83.5% וישנה פעמים מגמת ירידה (לדוגמה בין אפוקים 12-13). המתוואות האלה ניתן להסביר כי האוגומנטציה שביצעונו לתמונות ובחרית הפרמטרים בצורה רנדומלית שבה אנחנו משתמשים משפרת את הדיק בזמן האימון וגם בזמן ה-test, שכן יש הפרש משמעותי בדיק של שתי הרשותות (-5.5%).

3. בחרנו את כמות ה-workers באופן DINAMI כך שתתאים בדיק למשאבי המערכת הזמינים עליה רץ הקוד, כאשר העדיפות היא לקרוא את המשטנה SLURM_CPUS_PER_TASK המציג את ההקצתה האמיתית בשרת, ובהיעדרו להשתמש במספר יחידות העיבוד במכונה (cpu_count), תוך אפשרות סף תחתון של לפחות 2 תהליכי. המטרה הייתה למסס את המקבילות מוביל לחזור ממשאבי החומרה. לו היינו בוחרים מספר קטן מדי של workers ביחס לライブות הזמן, היינו גורמים underutilization של המעבד, כלומר אם גודל מדי של האימון הראשי מחייב לנזונים מה-workers וזמן הריצה מתארך. מנגד, אם היינו בוחרים מספר גדול מדי של workers החזור מספק הליבות, היינו גורמים overhead על מערכת הפעלה עקב ריבוי החלפות הקשר וניהול תהליכי, וזה היה פוגע ביעילות החישובית ומאט את הביצועים הכלליים.

4. השתמשנו בתהליכי שונים ולא בחוטים שונים של אותו תהליך. ב-*python* משתמשים במנגנון ה-GIL, המונע משני חוטים לרוץ בו זמן על המעבד בטור אותו תהליך. מכיוון שימושם עיבוד התמונה היא משימה אינטנסיבית מבחינה מעבד (CPU-bound) הכוללת חישובים מתמטיים (לטבוב ולשינוי התמונה) שימוש בחוטים לא יהיה אפשר מקבילות אמיתית והביצועים לא יהיו מושפעות. שימוש בתהליכי שונים יעוף את ה-GIL כיוון שלכל תהליך יש מרחב זיכרון ומפרש משלו, מה שמאפשר להם לרוץ במקביל על ליבות שונות באמת.

5. תחליה נשים לב כי כרגע, הקוד עובד בצורה סדרתית ברמת epoch: הפונקציה fit קוראת create_batches, והעבד הראשי ממתין עד שהעבדים יסיפו ליצור את כל ה-*batches* לאותו Epoch, ורק אז מתחילה האימון. זה גורם לכך שזמן שהנתונים נוצרים, הרשות לא לומדת, ובזמן שהרשות לומדת, ה-workers נחים. היינו משנות את המבנה הזה למבנה של Generator או תוך אסינכרוני. ה-workers

יצירנו נתונים באופן רציף לתוך תור בעל גודל מוגבל. תהילך האימון ישלוּף אחד בכל פעם מתוך התור ויאמן עליו מיד, מבלי להחות שכל ה-batches יהי מוכנים. וכך נקבל כי-helpers מייצרים את ה-batch-ים הבא במקביל לכך שהרשת מתאימה על ה-batch הנוכח. בנוסף על כך, כרגע helpers מוחזירים את התמונות המעובדות דרך התור שכטבנו. בפייתון, העברת אובייקטים כבדים כמו תמונות או מטריצות גדולות בין תהליכי דוחשת סריאלייזציה פועלה זו צורכת זמן מעבד רב ורחב פס של הזיכרון. לפיך, נרצה לעבור למודל של shared memory, כל ה-Workers יכולים לגשת אליו. ה-Workers יכתבו את תוצאות העבודה ישירות לתוך הזיכרון המשותף, וישלחו דרך התור רק הودעה קטנה המכילה את ה-index (המיקום במערך) שבו נמצא המידע המקורי. זה יחסור את עלות העתקה והסריאלייזציה של המידע.

6. מישנו את התור המקביל בצורה הבא:

لتור שלושה שדות, מנעל לכתובים שכן מותר לכתוב רק לכותב אחד בזמןית, מונה למספר האיברים שנמצאים כרגע ב-block-andread עם צד לכתיבה הצד לקריאה. מכיוון שהמידע שעובר בסקום נמצא באותו סדר כמו בתור אפשר להתייחס אליו כזה. בפעולה `get`, נחזיר את האיבר הראשון שנמצא ב-block, אין צורך בסכנון הקוראים כי נתן לנו שיש רק אחד זהה ונחסיר 1 מהמונה. בפעולה `set` נctrיך לטפל בכר שכותבים רבים יכולים לגשת ל-block ולנסות לכתוב בזמןית, לכן כדי שלמדנו להשתמש במנעל לעתוף את פעולה הכתיבה ל-block ושינוי המונה ב-1 ונסחרר אותו בסיום הכתיבה. בפעולה `empty` נשאל האם המונה שווה ל-0 ונחזיר את התשובה. אם לא היה קורא יחיד היינו צריכים להוסיף מנגול נוסף לקריאה ולסנן בין הקוראים את הגישה ל-`get`-`empty`.

.7

Correlation_gpu:

תחילה אנחנו מעבירים את את התמונה ואת מטריצת הגרען מהזיכרון של ה-RAM ל-gpu. נתחיל מערך חדש של תוצאות ונעביר גם אותו ל-gpu.ikut, נגדיר את כמות הבלוקים להיות מספר השורות ואת מספר החוטים לכל בלוק כמספר העמודות. זה יdag לך שכל חוט יעבד על משכבה אחת במטריצה. ואז נקרה לטיפול הkernel. ב-correlation_kernel אנחנו מבצעים טיפול של משכבת אחת בתוך המטריצה, נזהה מי אנחנו בעזרת ה-thread-id וה-block-id, נחשב את מרכז הגרען בעזרת $\text{row} = \text{row} // 2$ ו- $\text{kernel_height} = \text{kernel_width} // 2$, נבדוק שאנו לא בורחים מגודל המטריצה ונחשב את הקונבולוציה בעזרת $\text{z} = \text{z} + \text{w_pad_h} + \text{i} * \text{cc} - \text{col} - \text{pad_h}$. אם השcn לא קיים אנחנו נוסיף 0 למקום. לבסוף, נשמר את התוצאה במשכבת הנכונה במטריצת התוצאה ונחזיר.

correlation_numba:

פונקציה זו עשויה לבדוק את אותו חישוב אך בצורה סדרתית, נעבור בלולאה מקוננת על מטריצת התמונה הונעשה בבדיקה את אותו חישוב כמו קודם. נשמר את התואנה במטריצה חדשה ונחזיר.

.8

```
(tf23-gpu) dashay@lambda:~/CDPwinter26/HW2$ srun --gres=gpu:1 -c 1 --pty python3 filters_test.py
CPU 3X3 kernel: 0.0015325061976909637
Numba 3X3 kernel: 0.0014497525990009308
CUDA 3X3 kernel: 0.0010697096586227417
-----
CPU 5X5 kernel: 0.0032722987234592438
Numba 5X5 kernel: 0.0033623650670051575
CUDA 5X5 kernel: 0.0010733455419540405
-----
CPU 7X7 kernel: 0.0063840486109256744
Numba 7X7 kernel: 0.006138764321804047
CUDA 7X7 kernel: 0.0010749846696853638
-----
```

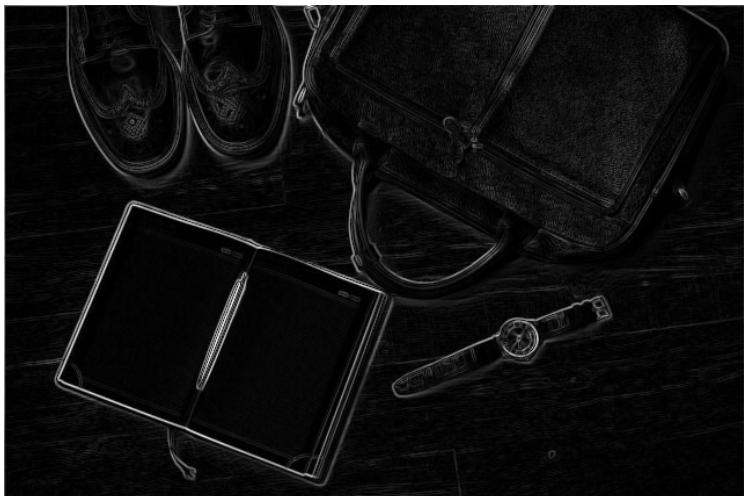
נשים לב כי לאורך כל ההרצאות משך זמן הריצה של cuda אינו משתנה מאוד אך ישנה עליה קטנה ככל שמטריצת הקרןל עולה, זאת מפני שאנו מזמנים מקטים חוט לכל משਬצת בתוך המטריצה ויש יותר משובצות וייתר עבדה ככל שהמטריצה גדולה וכן זאת תוצאה צפויה. בנוסף לכך, נשים לב כי cuda מביב את התוצאות הטובות ביותר, בפרט הוא מבצע את הפעולות מהר יותר מה-numba.

Kernal size	Cuda-numba speedup	Cuda-scipy speedup
3x3	1.355	1.4312
5x5	3.1314	3.0579
7x7	5.7364	5.966

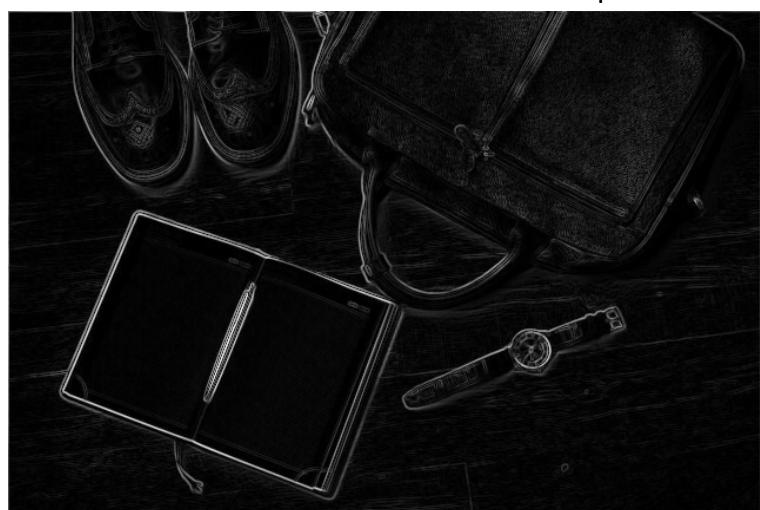
בטבלה לעיל ניתן לראות את ה-speedups המתקבל על ידי ההרצה המצורפת, מתקובל כי cuda כמעט זהה ל-scipy ול-numba במטריצות קטנות 3x3. לאחר מכן כי הוא מהיר יותר כמעט פי שלושה עבור מטריצות ביגנויות 5x5 ולאחר מכן הוא מהיר כמעט פי שישה עבור מטריצות גדולות 7x7.

9. לפ' התוצאות לעיל, ניתן לראות כי גודף להשתמש ב-cuda ככל שהמטריצה גדולה יותר ויש בה יותר מידע. מתקובל ספידאף משמעותית וגדול ככל שמקבלים יותר בעזרת cuda. עבור מטריצות קטנות עם מעט מידע סביר כי הינו מקבלים ביצועים טובים יותר אם הינו משתמשים ב-numba.

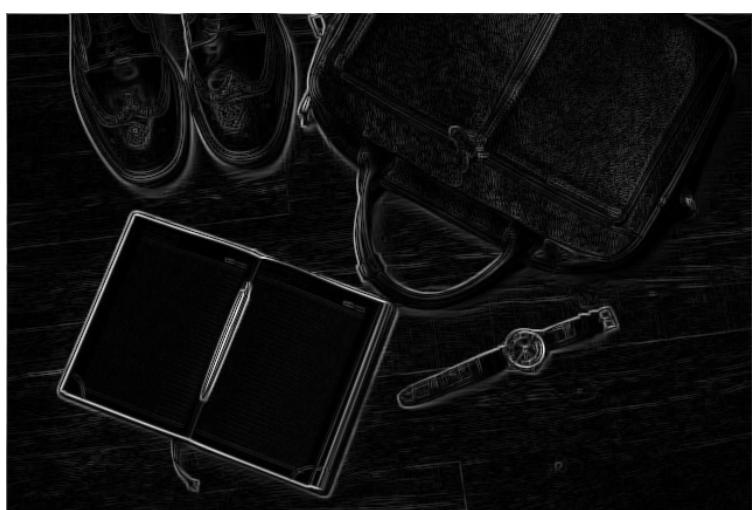
10. התמונה המתקבלת עם המטריצה סובל הרגילה:



התמונה המתקבלת עם המטריצה kernel1



התמונה המתקבלת עם kernel2



התמונה המתבקשת עם 3 kernel:



נתחיל בהשוואת שלושת ה kernels sobel, kernel1 ו-kernel2, תוך התבוננות במבנה המטריצות המייצגות אותם. בשלושת ה kernels ניתן להבחין במבנה משותף:

- העמודה האמצעית מכילה אפסים בלבד.
- העמודה השמאלית והעמודה הימנית מכילות ערכים זהים בגודל אך בעלי סימן הפוך.
- המשקלים בעמודות הקיצונית מציגים הבדלים אופקיים בין אזוריים בתמונה.

מבנה זה גורם לכך שביחסוב הקורלציה עבור פיקסל מסוים, התרומה של פיקסלים משמאלי ומימין לפיקסל המרכז נבחנת בצורה מנוגדת. בפועל, מתבצע חישוב של הפרש משוקל בין ערכי הפיקסלים בשני צידי הפיקסל הנבדק. כאשר ערכי הפיקסלים בעמודה השמאלית ובעמודה הימנית דומים או קרובים בערכם, הסכימה של הערכים החוביים והשליליים מבטלת את עצמה בקירוב, והتوزואה תהיה ערך קרוב ל-0. פיקסל זה יציג כהה או שחור בתמונה. לעומת זאת, כאשר קיימים הבדלים משמעותיים בין ערכי הפיקסלים בשני הצדדים (למשל מעבר חד בין אזור בהיר לאזור כהה), הביטול אינו מתקיים, והסכום ת寧יב ערך גבוה במוחלט. פיקסל זה יציג כבahir או בולט בתמונה.

מכאן נובעת המסקנה כי שלושת ה kernels מתאימים במיוחד לציהוי קווי מתאר וمبرורים חדשים בצלב. אזוריים אחידים יחסית בתמונה יישארו כהים, בעוד שקצבות של אובייקטים או גבולות בין אזוריים בצלב שונה יודגשו בצורה ברורה. זהו מקור הדמיון המרכזי בין kernel1 ו-kernel2.

למרות הדמיון המבני ואופן הפעולה העיקרי, קיימים הבדלים בין ה kernels. ההבדלים נובעים מהמשקלים השונים הניתנים לפיקסלים בסביבה הקרובה. ב-kernel1, למשל, המשקלים גדולים יותר במוחלט, ובעקב בכך במרכז השורה האמצעית, וכך התרומה של פיקסלים מסוימים חזקה יותר. משמעות הדבר היא רגשות גבואה יותר לשינויים בעוצמת הצלב, ולעתים קווים מתאר מודגשים יותר. לעומת זאת, sobel ו-kernel2 משתמשים במשקלים קטינים יותר או בחלוקת שונה שלהם, מה שmobail לתוצאה מעט שונה בעוצמת הקצאות המודגשים. בפועל, ניתן לראות הבדלים בעוצמת הבחרות של הקצאות: אזוריים מסוימים יופיעו לבנים יותר או פחות, בהתאם לקרnell שbow משתמשים.

כלומר, בעוד שכל ה kernels מזהים קצאות, כל אחד מהם מציג שינויים בעוצמה מעט שונה, בהתאם למשקלים ולמרחב הסביבה שעליו הוא מסתכל.

kernel 3 שונה באופן מהותי משלשות הקרים הקודמים. בניגוד לקרים שמטרתם להציג הבדלים בין פיקסלים, kernel 3 בנייתו כרך שכל הפיקסלים בסביבה תורמים באופן חיוני לערך הפיקסל המרכזי. בחישוב הקורלציה, ערכו של כל פיקסל נקבע כסכום משוקל של הפיקסלים השכנים שלו, ללא ביטול בין ערכים חיוביים ושליליים. פועלה זו גורמת לערבותם ערכי הצבעים באזור המוקומי.

כיצדאה מכך מתקיים אפקט של טשטוש בתמונה: פרטים חדים וקווים מתאר מיטשטשים, ואזורי צבע הופכים אחידים ורכים יותר. קרבן מסווג זה אינו מיועד לזיהוי קצוט, אלא לעיבוד תמונה שמטרתו חילקה והפחתת רעש.