

תרגיל בית 2 - מקבוצר

.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.p
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 העיבוד עולה, זאת מכיוון ששכל שיש יותר יחידות עיבוד, קיימים יותר תהליכים במערכת שמריצים את עיבוד התמונות (היות שאצלנו במערכת מספר ה-workers = number of threads). בגלל זה, תור התוצאות מתמלא מהר יותר וזה מה שמאיץ את פעולת ה-create batches בתוך ipnueralnetwork. אף על פי כן, קיבלנו שאחוז הדיוק test accuracy כמעט זהה בין שלוש ההרצות, ולכן אנחנו מסיקים כי הדיוק אינו תלוי במספר יחידות העיבוד.

2. נתבונן בהרצה של 16 ליבות מהסעיף הקודם:

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

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

תחילה נשים לב כי neural network מבצעת קפיצה גדולה ומהירה בגודל ה-accuracy שלה ב-epoch הראשונים, בפרט היא מגיעה ל-accuracy גבוה יותר מאשר ip nueral network בין epoch 6-1. ככל שמספר האפוקים גדל נשים לב כי גודל ה-accuracy ב-NeuralNetwork ip גדל וכי הוא גדל ממש בין אפוק לאפוק, זאת לעומת ה-accuracy של neural netowrk שכן הוא מתחיל להתקבע סביב 83.5% וישנה לפעמים מגמת ירידה (לדוגמה בין אפוקים 12-13). מהתוצאות האלה ניתן להסיק כי האוגמנטציה שביצענו לתמונות ובחירת הפרמטרים בצורה רנדומלית שבה אנחנו משתמשים משפרת את הדיוק בזמן האימון וגם בזמן ה-test, שכן יש הפרש משמעותי בדיוק של שתי הרשתות (כ-5.5%).

3. בחרנו את כמות ה-workers באופן דינמי כך שתתאים בדיוק למשאבי המערכת הזמינים עליה רץ הקוד, כאשר העדיפות היא לקרוא את המשתנה SLURM_CPUS_PER_TASK המייצג את ההקצאה האמיתית בשרת, ובהיעדרו להשתמש במספר יחידות העיבוד במכונה (cpu_count), תוך אכיפת סף תחתון של לפחות 2 תהליכים. המטרה הייתה למקסם את המקביליות מבלי לחרוג ממשאבי החומרה. לו היינו בוחרים מספר קטן מדי של workers ביחס לליבות הזמינות, היינו גורמים underutilization של המעבד, מצב בו תהליך האימון הראשי מחכה לנתונים מה-workers וזמן הריצה מתארך. מנגד, אם היינו בוחרים מספר גדול מדי של workers החורג ממספר הליבות, היינו גורמים ל-overhead על מערכת ההפעלה עקב ריבוי החלפות הקשר וניהול תהליכים, וזה היה פוגע ביעילות החישובית ומאט את הביצועים הכוללים.

4. השתמשנו בתהליכים שונים ולא בחוטים שונים של אותו תהליך. ב-python משתמשים במנגנון ה-GIL, המונע משני חוטים לרוץ בו זמנית על המעבד בתוך אותו תהליך. מכיוון שמשימת עיבוד התמונה היא משימה אינטנסיבית מבחינת מעבד (CPU-bound) הכוללת חישובים מתמטיים (לסיבוב ולשינוי התמונה) שימוש בחוטים לא היה מאפשר מקביליות אמיתית והביצועים לא היו משתפרים משמעותית. שימוש בתהליכים שונים עוקף את ה-GIL כיוון שלכל תהליך יש מרחב זיכרון ומפרש משלו, מה שמאפשר להם לרוץ במקביל על ליבות שונות באמת.

5. תחילה נשים לב כי כרגע, הקוד עובד בצורה סדרתית ברמת ה-epoch: הפונקציה fit קוראת ל-create_batches, והמעבד הראשי ממתין עד שהעובדים יסיימו לייצר את כל ה-batches לאותו Epoch, ורק אז מתחיל האימון. זה גורם לכך שבזמן שהנתונים נוצרים, הרשת לא לומדת, ובזמן שהרשת לומדת, ה-Workers נחים. היינו משנות את המבנה הזה למבנה של Generator או תור אסינכרוני. ה-Workers

ייצירו נתונים באופן רציף לתוך תור בעל גודל מוגבל. תהליך האימון ישלף batch אחד בכל פעם מתוך התור ויאמן עליו מיד, מבלי לחכות שכל ה-batches יהיו מוכנים. וכך נקבל כי ה-Workers מייצרים את ה-batch הבא במקביל לכך שהרשת מתאמנת על ה-batch הנוכחי. בנוסף על כך, כרגע ה-Workers מחזירים את התמונות המעובדות דרך התור שכתבנו. בפיתון, העברת אובייקטים כבדים כמו תמונות או מטריצות גדולות בין תהליכים דורשת סריאליזציה פעולה זו צורכת זמן מעבד רב ורוחב פס של הזיכרון. לפיכך, נרצה לעבור למודל של shared memory, כל ה-Workers והתהליך הראשי יכולים לגשת אליו. ה-Workers יכתבו את תוצאות העיבוד ישירות לתוך הזיכרון המשותף, וישלחו דרך התור רק הודעה קטנה המכילה את ה-Index (המיקום במערך) שבו נמצא המידע המוכן. זה יחסוך את עלות ההעתקה והסריאליזציה של המידע.

6. מימשנו את התור המקבילי בצורה הבא:

לתור שלושה שדות, מנעול לכתובים שכן מותר לכתוב רק לכותב אחד בזמנית, מונה למספר האיברים שנמצאים כרגע ב-pipe ו-pipe עם צד לכתובה וצד לקריאה. מכיוון שהמידע שעובר ב-pipe נמצא באותו סדר כמו בתור אפשר להתייחס אליו ככזה. בפעולה get, נחזיר את האיבר הראשון שנמצא ב-pipe, אין צורך בסנכרון הקוראים כי נתון לנו שיש רק אחד כזה ונחסיר 1 מהמונה. בפעולה put נצטרך לטפל בכך שכותבים מרובים יכולים לגשת ל-pipe ולנסות לכתוב בזמנית, לכן כפי שלמדנו נשתמש במנעול לעטוף את פעולת הכתיבה ל-pipe ושינוי המונה ב-1 ונשחרר אותו בסיום הכתיבה. בפעולת empty נשאל האם המונה שווה ל-0 ונחזיר את התשובה. אם לא היה קורא יחיד היינו צריכים להוסיף מנעול נוסף לקריאה ולסנכרן בין הקוראים את הגישה ל-get ו-empty.

7.

Correlation_gpu:

תחילה אנחנו מעבירים את התמונה ואת מטריצת הגרעין מהזיכרון של ה-RAM ל-gpu. נאתחל מערך חדש של תוצאות ונעביר גם אותו ל-gpu. כעת, נגדיר את כמות הבלוקים להיות מספר השורות ואת מספר החוטים לכל בלוק כמספר העמודות- זה ידאג לכך שכל חוט יעבוד על משבצת אחת במטריצה. ואז נקרא לטיפול הקרנל. ב-correlation_kernel אנחנו מבצעים טיפול של משבצת אחת בתוך המטריצה, נזהה מי אנחנו בעזרת ה-block id וה-thread id, נחשב את מרכז הגרעין בעזרת $kernel_height // 2$ ו- $kernel_width // 2$, נבדוק שאנחנו לא בורחים מגודל המטריצה ונחשב את הקונבולוציה בעזרת $rr = row - pad_w + j$ ו- $cc = col + pad_h$. אם השכן לא קיים אנחנו נוסיף 0 במקום. לבסוף, נשמור את התוצאה במשבצת הנכונה במטריצת התוצאה ונחזיר.

correlation_numba:

פונקציה זו עושה בדיוק את אותו חישוב אך בצורה סדרתית, נעבור בלולאה מקוננת על מטריצת התמונה ונעשה בדיוק את אותו חישוב כמו קודם. נשמור את התוצאה 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: 0.001449752599009308
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
=====
```

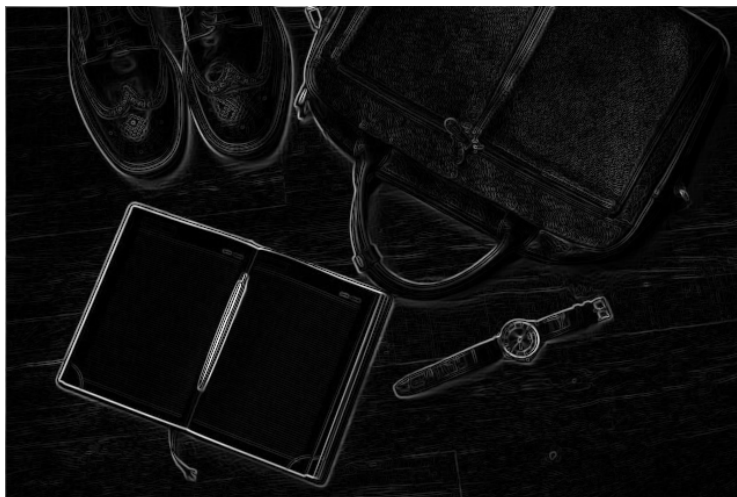
נשים לב כי לאורך כל ההרצות משך זמן הריצה של ה-cuda אינו משתנה מאוד אך ישנה עליה קטנה ככל שמטריצת הקרנל עולה, זאת מפני שאנחנו מקצים חוט לכל משבצת בתוך המטריצה ויש יותר משבצות ויותר עבודה ככל שהמטריצה גדלה ולכן זאת תוצאה צפויה. בנוסף לכך, נשים לב כי cuda מניב את התוצאות הטובות ביותר, בפרט הוא מבצע את הפעולות 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

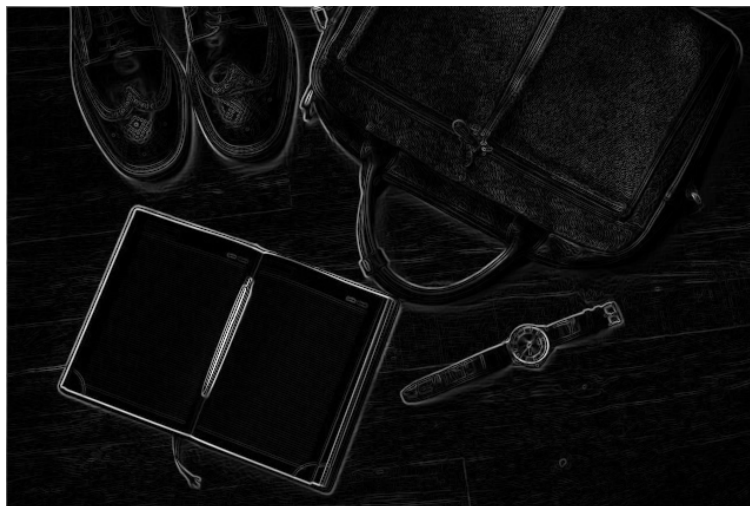
בטבלה לעיל ניתן לראות את ה-speedup המתקבל על ידי ההרצה המצורפת, מתקבל כי cuda כמעט זהה ל-scipy ול-numba במטריצות קטנות 3x3. לאחר מכן כי הוא מהיר יותר כמעט פי שלושה עבור מטריצות בינוניות 5x5 ולאחר מכן הוא מהיר כמעט פי שישה עבור מטריצות גדולות 7x7.

9. לפי התוצאות לעיל, ניתן לראות כי נעדיף להשתמש ב-cuda ככל שהמטריצה גדולה יותר ויש בה יותר מידע. מתקבל ספידאפ משמעותי וגדול ככל שממקבלים יותר בעזרת cuda. עבור מטריצות קטנות עם מעט מידע סביר כי היינו מקבלים ביצועים טובים יותר אם היינו משתמשים ב-numba.

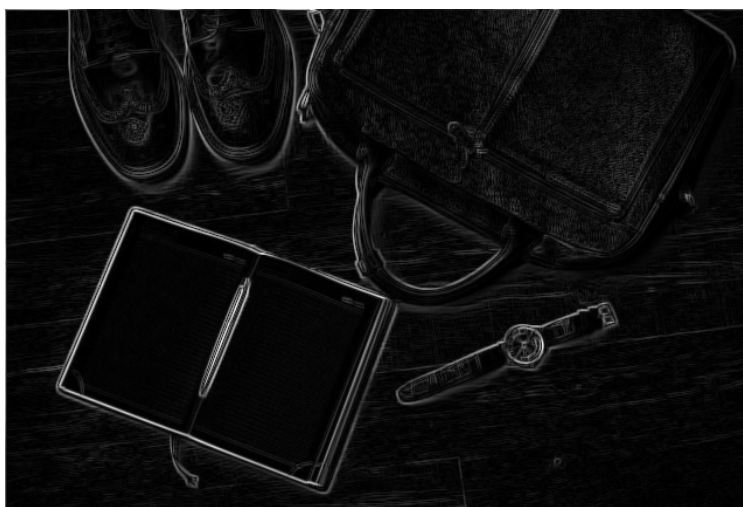
10. התמונה המתקבלת עם המטריצת סובל הרגילה:



התמונה המתקבלת עם המטריצה kernel1:



התמונה המתקבלת עם kernel2:



התמונה המתקבלת עם kernel3:



נתחיל בהשוואת שלושת הקרנלים kernel1, sobel ו-kernel2, תוך התבוננות במבנה המטריצות המייצגות אותם. בשלושת הקרנלים ניתן להבחין במבנה משותף: העמודה האמצעית מכילה אפסים בלבד. העמודה השמאלית והעמודה הימנית מכילות ערכים זהים בגודל אך בעלי סימן הפוך. המשקלים בעמודות הקיצוניות מדגישים הבדלים אופקיים בין אזורים בתמונה.

מבנה זה גורם לכך שבחישוב הקורלציה עבור פיקסל מסוים, התרומה של פיקסלים משמאל ומימין לפיקסל המרכזי נבחנת בצורה מנוגדת. בפועל, מתבצע חישוב של הפרש משוקלל בין ערכי הפיקסלים בשני צידי הפיקסל הנבדק. כאשר ערכי הפיקסלים בעמודה השמאלית ובעמודה הימנית דומים או קרובים בערכם, הסכימה של הערכים החיוביים והשליליים מבטלת את עצמה בקירוב, והתוצאה תהיה ערך קרוב ל-0. פיקסל כזה יוצג ככהה 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, והסכימה תניב ערך גבוה במוחלט. פיקסל כזה יוצג כבהיר או בולט בתמונה.

מכאן נובעת המסקנה כי שלושת הקרנלים מתאימים במיוחד לזיהוי קווי מתאר ומעברים חדים בצבע. אזורים אחידים יחסית בתמונה יישארו כהים, בעוד שקצוות של אובייקטים או גבולות בין אזורים בצבע שונה יודגשו בצורה ברורה. זהו מקור הדמיון המרכזי בין kernel1, sobel ו-kernel2.

למרות הדמיון המבני ואופן הפעולה העקרוני, קיימים הבדלים בין הקרנלים. ההבדלים נובעים מהמשקלים השונים הניתנים לפיקסלים בסביבה הקרובה. ב-kernel1, לדוגמה, המשקלים גדולים יותר במוחלט, ובעיקר במרכז השורה האמצעית, ולכן התרומה של פיקסלים מסוימים חזקה יותר. משמעות הדבר היא רגישות גבוהה יותר לשינויים בעוצמת הצבע, ולעיתים קווי מתאר מודגשים יותר. לעומת זאת, kernel2 ו-sobel משתמשים במשקלים קטנים יותר או בחלוקה שונה שלהם, מה שמוביל לתוצאה מעט שונה בעוצמת הקצוות המודגשים. בפועל, ניתן לראות הבדלים בעוצמת הבהירות של הקצוות: אזורים מסוימים יופיעו לבנים יותר או פחות, בהתאם לקרנל שבו משתמשים.

כלומר, בעוד שכל הקרנלים מזהים קצוות, כל אחד מהם מדגיש שינויים בעוצמה מעט שונה, בהתאם למשקלים ולמרחב הסביבה שעליו הוא מסתכל.

kernel3 שונה באופן מהותי משלושת הקרנלים הקודמים. בניגוד לקרנלים שמטרתם להדגיש הבדלים בין פיקסלים, kernel3 בנוי כך שכל הפיקסלים בסביבה תורמים באופן חיובי לערך הפיקסל המרכזי. בחישוב הקורלציה, ערכו של כל פיקסל נקבע כסכום משוקלל של הפיקסלים השכנים שלו, ללא ביטול בין ערכים חיוביים ושליילים. פעולה זו גורמת לערבוב ערכי הצבעים באזור המקומי.

כתוצאה מכך מתקבל אפקט של טשטוש בתמונה: פרטים חדים וקווי מתאר מיטשטשים, ואזורי צבע הופכים אחידים ורכים יותר. קרנל מסוג זה אינו מיועד לזיהוי קצוות, אלא לעיבוד תמונה שמטרתו החלקה והפחתת רעש.