ספר פרויקט
044167+044169

Data set reordering processor

VLSI
Project #7078

מנחה:

בניה בן נון

מגישים:

ארז מילרד 313311201

יובל בן חיים 209120864

תוכן עניינים

תקציר

בניסיון לצמצם את זמן האימון והזיכרון הנדרש עבור אימון רשתות נוירונים מצאנו במחקר כי מרבית הפתרונות כיום עוסקים בצמצום גדלי המודל עצמם (משקולות, inputs, outputs וכו׳) לאחר אימונם. החלטנו לגשת לבעיה זו מכיוון אחר על ידי דחיסת מאגרי התמונות (data set) לפני שימושם באימון וכך נוכל לבצע דחיסה של המידע שנשמר ללא פגיעה בביצועי הרשת.

על פי המאמר עליו אנו מתבססים יש לסדר מחדש את התמונות על פי דמיון כדי לדחוס אותן באופן היעיל והמהיר ביותר.

בפרויקט החלטנו להאיץ את אלגוריתם הדחיסה על ידי תיכנון של מודל של מעבד ייעודי עבור סידור התמונות. את מעבד זה נממש בפייתון.

רקע תיאורטי

דחיסת נתונים היא תחום העוסק בהקטנת נפחם של נתונים מבלי לאבד את המשמעות אותה הם מייצגים (או תוך איבוד מינימלי) כך שיתפסו פחות מקום מאשר בצורתם המקורית וכך ניתן לאחסן מידע רב יותר בשטח זיכרון פיזי נתון. דחיסת נתונים אפשרית מפני שבצורתם המקורית של הנתונים קיימת פעמים רבות יתירות גבוהה (או תוצרי לוואי שאינם נדרשים), כלומר קיים מידע החוזר על עצמו או מידע שניתן לייצג בצורה חסכונית יותר.

טכניקות הדחיסה מתחלקות לשתי קבוצות:

- דחיסה משמרת מידע- דחיסה משמרת היא דחיסה המאפשרת לנו להפעיל פונקציה הפוכה לפונקציית הדחיסה, כך שהתוצאה שתתקבל תהיה זהה למקור. טכניקות דחיסה אלו מנצלות את העובדה שבמידע הנבדק יש חוקיות מתמטית.
- דחיסה מאבדת מידע- זוהי דחיסה שלאחר הפעלתה לא נוכל לשחזר את המידע המקורי באופן מושלם. ישנם מקרים רבים בהם נסכים לאבד חלק מהמידע על מנת לקבל דחיסה טובה יותר מזו שניתן היה לקבל ללא איבוד המידע. למשל עבור מקרים בהם יש מידע חושי כמו תמונה, קול או סרט, ההנחה היא שהחושים האנושיים אינם מסוגלים להבחין בווריאציות קטנות ולכן לא נורא אם אובד חלק מהמידע, בתנאי שהמידע שהולך לאיבוד יהיה שולי מבחינת חשיבותו ומבחינת יכולת ההבחנה של התפיסה החושית שלנו.

קידוד האפמן- ישנן אותיות מסוימות באלפבית שהן שכיחות יותר מאותיות אחרות. לכן אם נרצה לייצג טקסט נוכל להקטין את גודלו בעזרת שימוש בקוד קצר לאותיות נפוצות וקוד ארוך יותר לאותיות נדירות. השיטה הנפוצה ביותר לדחיסה כזו קרויה קוד האפמן. רוב פורמטי הדחיסה עושים שימוש באלגוריתם זה.
רוב קבצי הטקסט משתמשים בקידוד ASCII כדי לייצג את התווים המופיעים בהם. בקידוד, לכל תו יש ערך בעל 7 או 8 ביטים כאשר הביט השמיני הוא ביט זוגיות. כלומר, ניתן לייצג את כל התווים בקובץ מסוים בפחות ביטים (כדי לייצג שמונה תווים צריך סה"כ 3 ביטים לכל תו).
בנוסף, ניתן לתת לתווים מסוימים, שמופיעים יותר פעמים, קוד קצר יותר ולתווים שמופיעים לעיתים רחוקות ניתן קודים ארוכים יותר. קידוד זה נקרא "קידוד גודל משתנה", מאחר שלכל תו יכול להיות גודל שונה. וכן, על מנת לפענח את הקוד כאשר לכל תו יש מספר שונה של ביטים המייצגים אותו, אנו נותנים לכל תו קוד משלו שתלוי במספר הפעמים בו הוא מופיע בטקסט וכדי להבדיל בין הקודים השונים לכל קוד יש תחילית שונה.

ביישום למידת מכונה, על מנת לאמן רשת יש להחזיק במאגרי מידע גדולים מאוד על מנת להצליח להביא את הרשת ליכולת חיזוי טובה.
בפרויקט אנו רוצים לדחוס את מאגר הנתונים טרם ביצוע אימון הרשת ולכן בדומה לדחיסת תמונות שיכלנו להרשות איבוד של מידע, כאן נוכל לאפשר איבוד מסוים של המידע (ברמת התמונה הבודדת) אך באופן שלא יפגע באיכות האימון, מכיוון שאנו עובדים על מאגרי מידע גדולים.

הדחיסה בה נעסוק, הקשורה בלמידת מכונה, עוסקת בצמצום נפח המידע המקורי על ידי דמיון בין הפיקסלים של תמונות שונות מפני שמצאנו שכך נשמרת רמת איכות האימון. כלומר, נרצה לסדר מחדש את התמונות במאגר עצמו על פי דמיון פיקסלים ואז נשתמש בדחיסה מסוג הופמן. אנו מסתמכים על מאמר Predictive Coding for Lossless Dataset Compression לפיו נוכל לקבל באופן זה דחיסה טובה יותר של המידע.

מאמרים רלוונטיים

בפרוייקט זה אנו עוסקים בדחיסת מידע עבור מאגרים ללא חשיבות סדר האובייקטים בו (זוהי דחיסה משמרת). במהלכו, מצאנו שני מאמרים שסיפקו פתרונות רלוונטיים לתחום המחקר:

1. המאמר הראשון Data-centric AI workflow based on compressed raw images:

במאמר זה, יש שימוש בשיטת הקידוד jetraw אשר בעזרתה נדחוס את התמונות הנדרשות, מבלי לאבד פרמטרים חשובים, אשר נסתרים מהעין האנושית אך חשובים לאימון של רשת כגון מודל הרעש או אי התלות בין הפיקסלים. בנוסף, המאמר מפרט על היכולת להשתמש באלגוריתם זה על מנת לייצר מידע סינתטי אמין בעל תכונות דומות לתמונות אמיתיות לאימון של רשת יותר מכלילה.

שיטת דחיסת את ה-jetraw מתרחשת בשני שלבים:

1. החלפת רעש רנדומלי ברעש פסודו רנדומלי וקובץ קליברציה בתוך התמונה.
2. דחיסת הקובץ המוחלף באופן יעיל.

שיטת דחיסה זו מביאה להקטנה של גודל הקובץ בפקטור של פי 7 מגודלו המקורי.

לפני החזרת המידע, הוא מושווה לווקטור הפרמטרים שייצרנו וכך נוודא שהוא ייצג תמונה אמיתית מהמאגר.

שיטת העבודה המוצעת במאמר:

1. השגת תמונות ברזולוציה גבוהה ממקור מסויים (במאמר המידע נקלח מצילומים שבוצעו על ידי רחפן).
2. למידה מהם הפרמטרים החשובים ויצירת וקטור הפרמטרים בהתאמה.
3. ייצור תמונות ברזולוציה יותר נמוכה אשר שומרות על אותן תכונות מטרולוגיות (מודל רעש עצמאי לכל פיקסל, כל פיקסל מגיע מפונקציית הסתברות מוגדרת היטב, ואין bias) (תמונות לווין).
4. הגדרת ה-tolerance והערך הרצוי עבור כל פרמטר בוקטור.
5. השוואת כל תמונה שיוצרה לתחומים שנבחרו וקביעה אם להכניסה לאימון המערכת או לא.

לא בחרנו להתבסס על מאמר זה מכיוון שכלל טכנולוגיה קניינית אשר אין לנו גישה לפטנט עבורה ולכן לא יכולנו לממש את הפתרון החומרתי עבורה.

2. המאמר השני שמצאנו, עליו בחרנו להתבסס במהלך הפרויקט שלנו הוא Predictive coding for lossless dataset compression:

מאמר זה מציג אלגוריתם חדש לדחיסה ייעודית של datasets עבור אימון מודלים המתבסס על כך שאין חשיבות לסדר הסיגנלים. באופן זה, נוכל לייצר פרמוטציה חדשה

של המידע מהמאגר, ואיננו מחויבים לסדר בו סודר מלכתחילה. בכך ניתן לקבל דחיסה משמעותית יותר של המידע.

שיטת הדחיסה במאמר מתבססת על predictive coding (שימוש בקידוד קיים לחיזוי קידוד עתידי). הרעיון של predictive coding מתבסס על החזרתיות והדמיון בין הדגימות השונות ולכן הוא מאחסן את השגיאות ביחס למידע המקורי ולא את המידע עצמו, ובכך מתקבל חסכון במספר הביטים הדרושים לשמירה.

אלגוריתם הדחיסה מתחלק לשלושה חלקים:

1. סידור מחדש של איברי המאגר על פי פרמטר דמיון ביניהם אותו נגדיר מראש.
2. דחיסה לפי predictive coding.
3. קידוד לפי האנטרופיה.

עבור השלב הראשון- של הסידור מהחדש, מחברי המאמר פיתחו שיטה חדשה שנקראת KNN-MST ordering אשר מבצעת חיפוש בעץ הפורש של קבוצה ב- KNN שמתחילה בגודל ln(n).

השלב השני של ה-predictive coding מתחיל ביצירת string context לכל פיקסל בתמונה, כך שהוא יכול לקבל עדכונים מפיקסלים שבאו לפניו באותה תמונה/בתמונות קודמות על מנת לשפר את יכולה החיזוי. ה-string context מחזיק את רשימת המאפיינים אשר מתארים את הפיקסל הספציפי.

עבור תמונות צבעוניות, הם מצאו כי עדיף להשתמש בחזאי בודד לשלושת ערוצי הצבע.

לאחר מכן מאמנים את המודל שנבנה על כל הפרמטרים שב-string context של הפיקסלים, ומפעילים את המודל על כל אחד מהפיקסלים למדידת השגיאה. בהמשך, מייצרים מערך שגיאות לתמונה (כאשר 0 מייצג התאמה מושלמת). לאחר מכן משתמשים במקודד האפמן (Huffman) לכיווץ מערך השגיאות.

המודל המוצע הראה שיפור משמעותי מול JPEG-LS ו- PNG, כאשר PNG לא היה בר השוואה בכלל ולכן נשאר מחוץ למאמר, ו-JPEG נוצח ב5-20%.

במסגרת החיפושים של אלגוריתם לצמצום ה-data set, בחנו מספר פתרונות אפשריים:

- הפתרון הראשון באמצעות האלגוריתם הבא: MST-KNN:

האלגוריתם פותח על ידי חיבור של שני אלגוריתמי קירבה (KNN ו- Minimum Spanning Tree) וחיתוך ביניהם. הפעולה שנוצרת בהפעלת האלגוריתם היא יצירת עץ פורש מינימלי של מכל איברי ה-dataset. לאחר מכן, מתבצע חיבור שאר איברי ה-dataset בעזרת KNN לצומת הדומה אליהם ביותר כדי לייצר רשת מינימלית אשר משאירה את הגרף קשיר.

- אלגוריתם נוסף שבחנו הוא: Band reordering based on consecutive continuity breakdown heuristics (BRCCBH):

השיטה עובדת על המנגנון הבא:

1. בונים מטריצת קורלציה בין כל איברי ה-dataset.
2. בוחרים את האיבר הראשון למטריצת ה-reorder למטריצת IN, ומגדירים את כל שאר האיברים כמטריצת OUT.
3. בכל איטרציה לוקחים את האיבר האחרון במטריצת IN, ומחפשים במטריצת OUT את האיבר עם הקורלציה הגבוהה ביותר (לפי מטריצת הקורלציה).

- אלגוריתם אחר שבחנו היה band reordering with CCSDS: האלגוריתם מכיל שני בלוקים פונקציונאליים: Predictor ,Encoder עבור ה-Predictor:

1. חישוב סכום מקומי באופן מלא בעזרת פיקסלים שכנים על פי הנוסחה הבאה:

$$\sigma_{x,y,z} = (s_{z,x-1,y-1} + s_{z,x,y-1} + s_{z,x+1,-1} + s_{z,x-1,y})$$

בנוסף, סכום זה מחושב באופן מצומצם על ידי הנוסחה הבאה:

$$\sigma_{x,y,z} = 4 \cdot s_{z,x,y-1}$$

2. מחושב ההפרש בין שני הסכומים שחושבו ב-1.
3. וקטור ההפרשים שחושב בסעיף 2 מוכפל בוקטור המשקלים ויוצר ערך prediction סקלרי. וקטור משקלים זה מאותחל בערך דיפולטיבי.
4. הערך המנובא מחושב כך:

$$\tilde{s}_{z,x,y} = clip\left[mod_R \cdot \frac{\left[d_{z,x,y} + 2^{\Omega} \cdot \left(\sigma_{z,x,y} - 4 \cdot s_{mid}\right)\right]}{2^{\Omega+1}}\right]$$

$d_{z,x,y}$ הוא תוצאת הכפלת וקטור ההפרשים עם וקטור המשקלים.

R גודל הרגיסטר במצב unsigned

weight resolution parameter – Ω

mid-range sample value -s_{mid}

5. המשקלים מעודכנים על פי שגיאת הניבוי ועל פי ρ.

בהתחלה הערך של ρ הוא $v_{min} + D - \Omega + D$, הוא נקבע על פי הערך של הפרמטר t_{inc} ומוגדל ב-1 עד לערך $v_{max} + D - \Omega + D$. בכל חזרה על הניבוי וקטור המשקלים משתנה כך:

$$W_{i,next} = W_i + [0.5\left(sgn\left[e_{z,x,y}\right] \cdot 2^{\rho(z,x,y)} \cdot U_{z,x,y} + 1\right]$$

ו-$e_{z,x,y}$ מוגדר כך:

$$e_{z,x,y} = 2 \cdot s_{z,x,y} - \tilde{s}_{z,x,y}$$

6. בשלב האחרון, שאר החיזוי ממופה לייצוג של D ביטים. לאחר שלב הניבוי, ה-prediction errors מוצפנים כל ידי entropy coder.

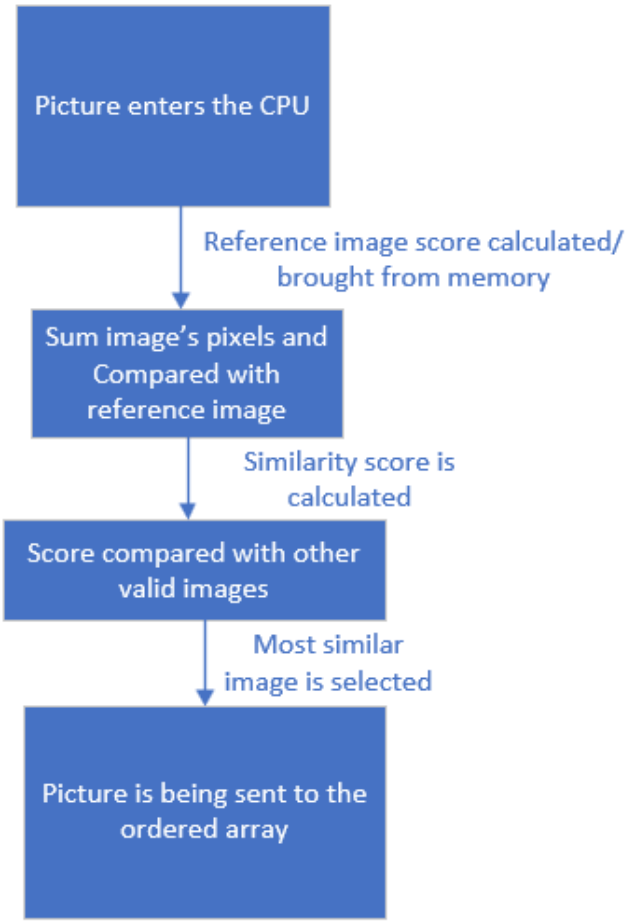
- האלגוריתם האחרון שבחנו, אותו בחרנו לממש בפרויקט הוא סידור על פי דמיון בין פיקסלים בעזרת L2, שהוא המרחק הריבועי בין הפיקסלים.

האלגוריתמים שאינם KNN-MST ו-L2 נפסלו מכיוון שהתבססו על לינאריות בין התמונות, שלא קיימת במקרה שלנו מכיוון שהתמונות במאגר הן בלתי תלויות.
בחרנו באלגוריתם L2 מכיוון שחשבנו שהוא פשוט יותר למימוש וייתן ביצועים טובים יותר מאשר KNN-MST.

אלגוריתם

בנינו סכמה המתארת את המסלול אותו תעבור כל תמונה:

1. תמונה רנדומלית נכנסת לצינור והיא נבחרת להיות תמונה הרפרנס שלנו.
2. באופן מקבילי, כל התמונות נכנסות לצינור לחישוב סכום הפיקסלים בתמונה. הן מושוות מול תמונת הרפרנס ומקבלות ציון למידת הדמיון על פי הפרש בין שני פיקסלים מקבילים בין התמונות השונות. הפרשים אלו נסכמים לציון אחד (התמונה בעלת הסכום הקטן ביותר היא הדומה ביותר).
3. התמונה בעלת ציון הדמיון הנמוך יותר (0 מייצג דמיון מושלם), ממשיכה בצינור ומוגדרת להיות תמונת הרפרנס החדשה ונשלחת החוצה כתמונה הבאה בסידור החדש של התמונות כקלט לאלגוריתם הדחיסה.
4. התמונה שנבחרה מסומנת כתמונה בשימוש ומתחילים מחדש עם שאר התמונות הזמינות.



ארכיטקטורה

בפרויקט, בנינו מעבד ייעודי אשר מבצע סידור מחדש של תמונות ממאגר המידע, כך שנוכל לשלוח אותן כקלט לאלגוריתם הדחיסה.

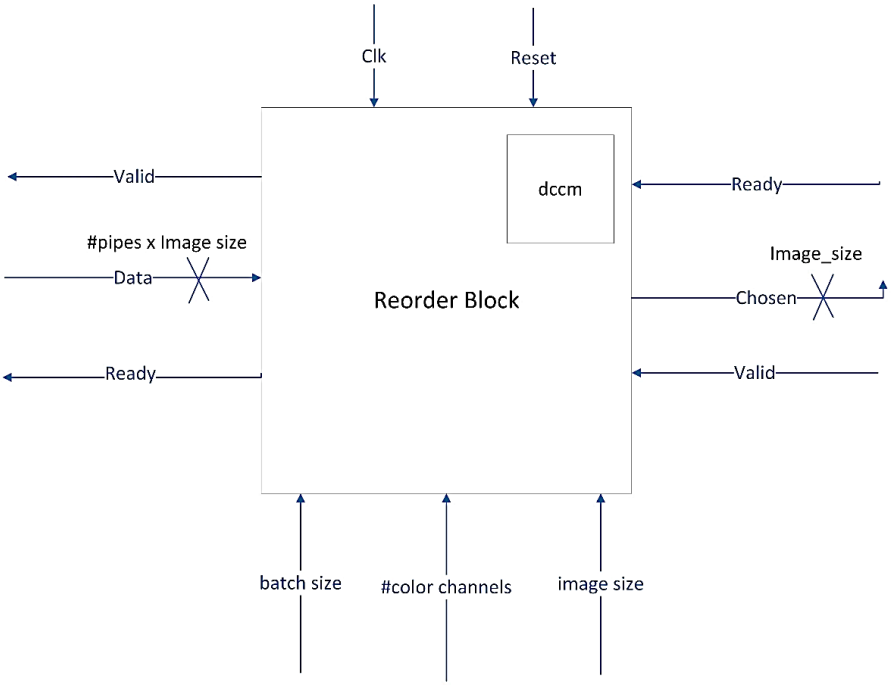
הסיגנלים בבלוק:

1. CLK
2. RST
3. Valid + ready- מהווה את מנגנון התקשורת של הבלוק עם התוכנה, לסנכרון הזנת התמונות לתוכו ושליחת התמונה הנבחרת ממנו.
4. Data- התמונה הנשלחת בביטים למעבד (bus ברוחב: גודל תמונה * מספר הצינורות).
5. Chosen- התמונה שנבחרה להיות התמונה הבאה בתור בסידור החדש, נשלחת לאלגוריתם הדחיסה תוכנתי (bus בגודל רוחב תמונה).

פרמטרים:

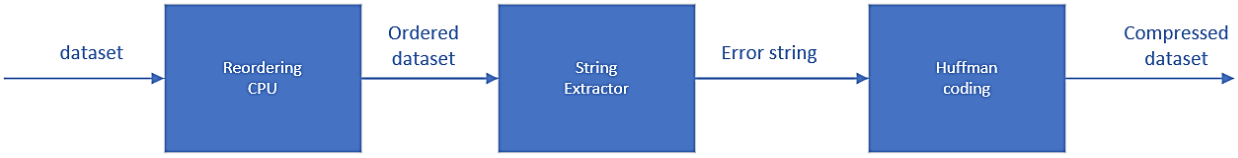
1. Batch size- כמות התמונות עליה מבוצע אלגוריתם ההשוואה בכל איטרציה.
2. Color channels- מספר ערוצי הצבע בהם משתמשים.
3. Image size- גודל התמונה.

DCCM- זיכרון פנימי של הבלוק, אשר מאחסן את התמונות המושוות, תמונת הרפרנס וציוני הדמיון של התמונות ב-batch.



מימוש המעבד

בפרוייקט, שילבנו את המעבד שבנינו בשלב קבלת ה-data. את ה-data המסודר במוצא המעבד אנו מעבירים אל שלב חילוץ ה-error string אשר ממנו הוא מועבר לאלגוריתם הדחיסה הפמן.
ניתן לראות זאת התרשים הבא:



בפרויקט, מימשנו מודל של המעבד בעזרת קוד פייתון. פירוט הקוד מופיע בנספחים.

נסביר את הקוד שמימשנו על ידי חלוקה לפונקציות:

1. הפונקציה הראשונה היא arrange_images:
היא מקבלת images_list, num_of_images, num_of_rows- רשימה של תמונות, אורך הרשימה (מספר התמונות) ומספר השורות בתמונה (אלו הם ממדי התמונה, עסקנו בתמונות ריבועיות).
תפקידה של הפונקציה הוא להביא בכל איטרציה את התמונה הבאה על בסיס ה-id שמצאה הפונקציה bring_image ולהכניסה לתור המסודר. הפונקציה מוודאת כי תמונה לא תיבחר פעמיים על ידי ביט valid המציין אם תמונה נבחרה כבר בעבר. בסוף הפונקציה מתקבלת רשימה חדשה של התמונות המסודרות לפי דמיון.
התמונה הראשונה לרפרנס היא התמונה הראשונה במאגר (נבחרת באופן אקראי ללא חשיבות מהי התמונה).

2. הפונקציה השנייה היא bring_image:
היא מקבלת reference_image_id- תמונת הרפרנס (האחרונה בתור המסודר) לפיה נבחרת התמונה הבאה.
תפקידה של פונקציה זו לחשב את סכומי כלל התמונות לפי דמיון פיקסלים ולמצוא את התמונה בעל הדמיון הרב ביותר מבין התמונות במאגר. הפונקציה מחזירה את ה-id של התמונה המתאימה. כדי לחסוך בזמן ריצה, הפונקציה יודעת לחשב רק את התמונות שעוד לא נבחרו ולכן מועמדות להיות התמונה הבאה בתור.

3. הפונקציה השלישית היא calc_new_diff:
היא מקבלת new_sum, new_id, reference_image_sum- סכום, מזהה תמונה ואת סכום תמונת הרפרנס.
תפקידה לחשב את ההפרש בין כל תמונה לתמונת הרפרנס ובכל איטרציה לעדכן את מזהה התמונה במידת הצורך (אם נמצא הפרש קטן יותר מההפרש הקודם).

4. הפונקציה הרביעית היא sum_image :
היא מקבלת image, image_num- תמונה בגודל המתאים ומזהה התמונה.
תפקידה של הפונקציה הוא לסכום את הפיקסלים של כלל התמונה. היא מבצעת זאת על ידי פריסה של התמונה לשורות והפעלת sum_row עבור כל שורה. את התוצאות היא מאחסנת במערך הגלובלי sums, אשר מאפשר לפונקציות שמעליה לוותר על חישוב חוזר של תמונה לאחר שביצעה את החישוב.

5. הפונקציה האחרונה היא sum_row :
היא מקבלת row_of_image, row, image_number- שורה של פיקסלים מהתמונה, אינדקס השורה ומזהה תמונה.
תפקידה של הפונקציה הוא לסכום את כלל הפיקסלים בשורה הנתונה ולאחסנם במערך ff על מנת שהסכום המחושב יוכל לשמש לחישוב הדמיון עבור הפונקציות מעל.

בריצת הקוד השתמשנו בספרייה MNIST ולכן קבענו את גודל התמונה להיות 28×28 בהתאם לספריה.
בנוסף, הפרמטר MAX_DIFFERENCE נקבע להיות הערך 199920, לפי 255×28×28 (ממדי התמונה × המרחק המקסימלי בין זוג פיקסלים כאשר שניהם אי שליליים).

בדיקות וורפיקציה

על מנת לבדוק את המעבד שבנינו, רצינו לוודא את המקרים הבאים:

1. הזנת מספר מטריצות במרחק שווה אחת מהשניה ובדיקה שאכן סדר הבחירה רנדומלי.

2. הזנת 2 מטריצות זהות, ועוד מטריצה במרחק מקסימלי מהן, ווידוא כי סדר הבחירה נכון(תוך שינוי מטריצת הפתיחה).

3. הזנת מטריצות בעלות ערכים אסורים(פיקסלים שליליים או מימדים לא נכונים) ובדיקה כי המערכת מקפיצה שגיאה.

4. הזנת מטריצות השונות בפיקסל בודד ובדיקה כי הסידור אכן נעשה על בסיסו.

5. הזנת מטריצות משני סוגים לסירוגין. כלומר, לדוגמא מטריצת אפסים, מטריצת אחדים, מטריצת אפסים, מטריצת אחדים וכך הלאה) ובדיקה כי המערך המסודר פיצל אותן לשני מקטעים אחידים (כלומר כל מטריצות האחדים ברצף ואז כל מטריצות האפסים ברצף).

סביבת עבודה

סביבת העבודה שלנו הייתה אנקונדה 3.6, פייתון 3.11
לטובת ההתממשקות בין המודל לקוד מהמאמר והרצת הקוד שלהם עם תוספת המעבד שיצרנו התקנו מספר חבילות, על פי דרישת המאמר אשר מופיעות בנספחים.

תוצאות ומסקנות

על מנת שנוכל להשתמש באלגוריתם שלנו לסידור התמונות מחדש, אחת מהנחות הבסיס בתהליך הסידור היא שהסידור מחדש של התמונות אינו יפגע באחוז הדיוק של אימון הרשת.

רצינו להוכיח הנחה זו, לשם כך השתמשנו ברשת resnet-50, על גבי ה- dataset MNIST ואימנו את הרשת עם ובלי שימוש באלגוריתם שכתבנו.

ניתן לראות בתמונות את הוכחת הנחה זו:

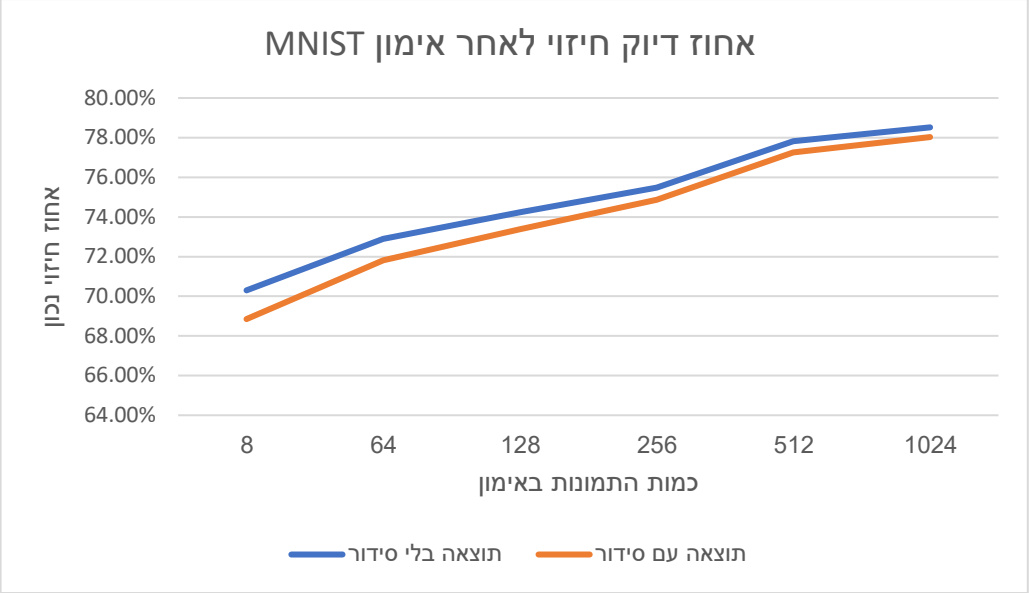
בתמונה הראשונה מתקבל אחוז דיוק אימון הרשת ללא הסידור:

```
emilrad@emilrad-mobl:/mnt/c/Users/emilrad$ python3 resnet-50.py
2024-05-17 00:38:51.084348: I tensorflow/core/util/port.cc:113] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-off errors
from different computation orders. To turn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
2024-05-17 00:38:51.220541: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized to use available CPU instructions in performance-critical operations
.
To enable the following instructions: AVX2 AVX_VNNI FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
2024-05-17 00:38:52.128283: W tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warning: Could not find TensorRT
/home/emilrad/.local/lib/python3.10/site-packages/torchvision/models/_utils.py:208: UserWarning: The parameter 'pretrained' is deprecated since 0.13 and may be removed in the future, pl
ease use 'weights' instead.
  warnings.warn(
/home/emilrad/.local/lib/python3.10/site-packages/torchvision/models/_utils.py:223: UserWarning: Arguments other than a weight enum or `None` for 'weights' are deprecated since 0.13 and
 may be removed in the future. The current behavior is equivalent to passing `weights=None`.
  warnings.warn(msg)
Finished Training
Accuracy of the network on the 1024 test images: 78.52%
emilrad@emilrad-mobl:/mnt/c/Users/emilrad$
```

בתמונה השנייה מתקבל אחוז דיוק אימון הרשת עם הסידור:

```
emilrad@emilrad-mobl:/mnt/c/Users/emilrad$ python3 resnet-50.py
2024-05-16 22:30:08.021796: I tensorflow/core/util/port.cc:113] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-off errors
from different computation orders. To turn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
2024-05-16 22:30:08.939437: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized to use available CPU instructions in performance-critical operations
.
To enable the following instructions: AVX2 AVX_VNNI FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
2024-05-16 22:30:10.163078: W tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warning: Could not find TensorRT
starting reordering
finished reordering
/home/emilrad/.local/lib/python3.10/site-packages/torchvision/models/_utils.py:208: UserWarning: The parameter 'pretrained' is deprecated since 0.13 and may be removed in the future, pl
ease use 'weights' instead.
  warnings.warn(
/home/emilrad/.local/lib/python3.10/site-packages/torchvision/models/_utils.py:223: UserWarning: Arguments other than a weight enum or `None` for 'weights' are deprecated since 0.13 and
 may be removed in the future. The current behavior is equivalent to passing `weights=None`.
  warnings.warn(msg)
Finished Training
Accuracy of the network on the 1024 test images: 78.03%
emilrad@emilrad-mobl:/mnt/c/Users/emilrad$
```

זוהי המחשה מתוך התוכנה, אך לשם חקר הביצועים המלא, ביצענו את הבדיקה על מספר גדלים של מספר תמונות מתוך ה-dataset.



כפי שניתן לראות, עבור כמות קטנה של תמונות, קיימת סטייה קלה של כ-1%, אך היא מתקזזת ככל שגדלה כמות התמונות (ותתקזז לחלוטין כאשר נגיע לגדלים של המאגר האמיתי שמונה כ-60,000 תמונות).

לכן נסיק כי שינוי סדר התמונות אינו פוגע באימון אך מאפשר לנו לדחוס את ה-dataset ובכך להביא לחיסכון במקום הנדרש על מנת לשמור אותו טרם האימון.

על פי תוצאות המאמר, בעזרת שיטה זו אנו משיגים יכולת דחיסה של פי 7.

ביבליוגרפיה

6. Data-centric AI workflow based on compressed raw images
https://arodes.hes-so.ch/record/11774

7. Predictive coding for lossless dataset compression
https://ieeexplore.ieee.org/abstract/document/9413447

8. DataCompression GIT
https://github.com/HsiangHsu/DataCompression

נספחים

בפרויקט, מימשנו מודל של המעבד בעזרת קוד פייתון:

```
# Online Python compiler (interpreter) to run Python online.
# Write Python 3 code in this online editor and run it
from threading import Thread
import numpy as np
import torch
import torchvision
import torchvision.transforms as transforms
import torch.nn as nn
import torch.optim as optim
import matplotlib.pyplot as plt
import tensorflow as tf

# import json

NUM_OF_ROWS = 28
NUM_OF_IMAGES = 5
MAX_DIFFERENCE = 199920
START_VALUE = -1
MATRIX_BIT = 0
CLASSIFY_BIT = 1
VALID_BIT = 2

images_list = []
images = []
ff = []
sums = []
arranged_images = []
last_image_reference_id = -1
min_diff = MAX_DIFFERENCE  # 32X32X255 is the max difference
id = 0

##############################################
def sum_row(row_of_image, row, image_number):
    column = 0
    temp = 0
    while (1):
        if (column < NUM_OF_ROWS):  # it's a square image
            temp = ff[image_number][row] + row_of_image[column]
            ff[image_number][row] = temp
            temp = column + 1
            column = temp
        else:
            return


#######################################################################
###########################
def sum_image(image, image_num):
    global sums
    global ff
    sum = 0
    num_threads = 0
    temp = 0
    threads = []
```

```
    if (sums[image_num] != START_VALUE):
        return sums[image_num]
    while (1):

        # create threads each iteration
        while (1):
            if (num_threads < NUM_OF_ROWS):

                t = Thread(target=sum_row,
args=(image[0][num_threads], num_threads, image_num))
                threads.append(t)
                threads[num_threads].start()
                temp = num_threads + 1
                num_threads = temp
            else:
                break

        # waits for all threads to finish
        i = 0
        while (1):
            if (i < num_threads):
                threads[i].join()
                temp = i + 1
                i = temp
            else:
                break
        break
    # sums all the rows
    i = 0
    while (1):
        if (i < NUM_OF_ROWS):
            temp = sum + ff[image_num][i]
            ff[image_num][i] = 0
            sum = temp
            temp = i + 1
            i = temp
        else:
            break
    sums[image_num] = sum
    return

#####################################################################
##################


def calc_new_diff(new_sum, new_id, reference_image_sum):
    global min_diff
    global id
    temp = 0
    new_diff = new_sum - reference_image_sum
    if ((new_diff < 0)):
        temp = -new_diff
        new_diff = temp

    if ((new_diff < min_diff)):
        min_diff = new_diff
        id = new_id
    else:
        return
```

```
#####################################################################
#########

def bring_image(reference_image_id):
    threads = []
    i = 0
    num_invalid_pictures = 0
    real_thread_index = 0
    reference_image = images[reference_image_id][MATRIX_BIT]
    reference_image_flag = 1
    while (1):
        while (1):
            if (reference_image_flag):
                t = Thread(target=sum_image, args=(reference_image,
id))
                reference_image_flag = 0
            elif (i < NUM_OF_IMAGES):
                if (images[i][VALID_BIT] == 1):
                    t = Thread(target=sum_image,
args=(images[i][MATRIX_BIT], i))
                else:
                    temp = num_invalid_pictures + 1
                    num_invalid_pictures = temp
                    temp = i + 1
                    i = temp
                    continue
            else:
                break
            threads.append(t)
            real_thread_index = i - num_invalid_pictures
            threads[real_thread_index].start()
            temp = i + 1
            i = temp

        i = 0

        while (1):
            if (i < NUM_OF_IMAGES):
                if (images[i][VALID_BIT] == 1):
                    threads[real_thread_index].join()
                if (i == reference_image_id):  # we always calculate
reference picture
                    threads[real_thread_index].join()
                temp = i + 1
                i = temp
            else:
                break

        i = 0
        old_id = id  # so we can update id with causing recursion
        while (1):
            if (i < NUM_OF_IMAGES):
                if (i != old_id):  # not the reference picture
                    if (images[i][VALID_BIT] == 1):
                        calc_new_diff(sums[i], i, sums[old_id])
                temp = i + 1
                i = temp
            else:
                break
        break
    return
```

```
#####################################################################
###########

def arrange_images(images_list, num_of_images, num_of_rows):
    global NUM_OF_ROWS
    global NUM_OF_IMAGES
    global last_image_reference_id
    global min_diff
    global images
    global ff
    global sums

    NUM_OF_ROWS = num_of_rows
    NUM_OF_IMAGES = num_of_images
    # ff = [[[0 for i in range(NUM_OF_ROWS)] for j in
range(NUM_OF_ROWS)] for k in range(NUM_OF_IMAGES)]
    ff = [[0 for i in range(NUM_OF_ROWS)] for k in
range(NUM_OF_IMAGES)]
    sums = [START_VALUE for i in range(NUM_OF_IMAGES)]

    first_image = 1
    images = []
    images_already_chosen = [0 for i in range(NUM_OF_IMAGES)]
  #  print(images_list)
    # adding a valid bit to each image
    i = 0
  #  print("received:")
  #  for labels in images_list:
  #      print(labels)

    while (1):
        if (i < NUM_OF_IMAGES):

images.append([images_list[i][MATRIX_BIT],images_list[i][CLASSIFY_BIT
], 1])
            temp = i + 1
            i = temp
        else:
            break

    i = 0
    while (1):
        if (i < NUM_OF_IMAGES):
            if (first_image):
                arranged_images.append((images[0][MATRIX_BIT],
images[0][CLASSIFY_BIT]))
                images_already_chosen[0] = 1
                last_image_reference_id = 0
                images[0][VALID_BIT] = 0
                first_image = 0
               # print(id)
                # print(arranged_images)
            else:
                bring_image(last_image_reference_id)
               # print(id)
                # print(arranged_images)

                if (images_already_chosen[id] == 0):
                    images_already_chosen[id] = 1
```

```
                else:
                    print("ERROR: image was chosen twice")
                    exit()

                arranged_images.append((images[id][MATRIX_BIT],
images[id][CLASSIFY_BIT]))
                last_image_reference_id = id
                images[id][VALID_BIT] = 0
                min_diff = MAX_DIFFERENCE

            temp = i + 1
            i = temp
        else:
            break
    # print(arranged_images)

    out_arranged_images = []
   # print("arranged:")
   # print(arranged_images)

    for i in range(len(arranged_images)):

out_arranged_images.append((arranged_images[i][MATRIX_BIT],arranged_i
mages[i][CLASSIFY_BIT]))
       i = i + 1

    return (out_arranged_images)

##################################################################
############

def main():
    # np.random.seed(10)
    # Define transformations
    transform = transforms.Compose([
        transforms.Resize((224, 224)),  # Resize to match ResNet-50
input size
        transforms.ToTensor(),
        transforms.Normalize((0.5,), (0.5,))
    ])

    # Load MNIST dataset
    train_dataset = torchvision.datasets.MNIST(root='./data',
train=True, download=True, transform=transform)
    test_dataset = torchvision.datasets.MNIST(root='./data',
train=False, download=True, transform=transform)


    train_dataset_subset =
torch.utils.data.Subset(train_dataset,range(10))
    test_dataset_subset =
torch.utils.data.Subset(test_dataset,range(3))
    torch.set_printoptions(profile="full")
   # for data in train_dataset_subset:
   #     print(data)
   # for i in range(3):
   #     images_list.append(np.random.randint(-255, 255, (28, 28)))
   # arrange_images(images_list, len(images_list),
len(images_list[0]))
    train_dataset_arranged = arrange_images(train_dataset_subset,10,
224)
```

```
'''''
    print("arranged images\n")
    for image in arranged_images:
        print(image)
        print("\n")
'''''
if __name__ == "__main__":
    main()
    #
arrange_images(images_list,len(images_list),len(images_list[0]))
```

על פי דרישות המאמר התקנו את החבילות הבאות:

absl-py

astunparse

attrs

bitstring

cachetools

certifi

cffi

chardet

dill

ffmpeg-python

future

gast

google-auth

google-auth-oauthlib

google-pasta

googleapis-common-protos

grpcio

h5py

idna

idx2numpy

importlib-metadata

joblib

Keras-Preprocessing

Markdown

numpy

oauthlib

opencv-python

opt-einsum

Pillow

promise

protobuf

pyasn1

pyasn1-modules

pycparser

requests

requests-oauthlib

rsa

scikit-learn

scipy

six

scikit-learn

tensorboard

tensorboard-plugin-wit

tensorflow

tensorflow-datasets

tensorflow-estimator

tensorflow-metadata

termcolor

threadpoolctl

tqdm

urllib3

webm

webp

Werkzeug

wrapt

zipp