שלום רב,

להלן סיכום מפגש שלישי וחומרים נלווים.

מטרת המפגש הייתה לספק לעומק ולפרטי פרטים את המידע והדוגמאות המלמדות על ה-TRT אשר מהוות את הבסיס לשם פיתוח הפרויקט שהוצג במפגש הראשון.

מטה תמצאו תיאור מפורט המסכם את הצעדים והחומר אותם סקרנו במהלך המפגש כולל דוגמא מלאה שיכולה להוות עבורכם נקודת פתיחה לפרויקט.

לאור סטאטוס התשתיות שהביא למצב שלחלקנו אין גישה לתשתית NVIDIA העליתי את האפשרויות הבאות:

·        אתם יכולים להצטוות לשלישיות בהגשת הפרויקט – על פי שיקול דעתכם

·        אפשר יהיה להגיש את הפרויקט לכל המאוחר עד לסוף חודש 08

·        שיתפתי אתכם עם דוגמא מלאה של מה שנדרש להגשה כי:

o       כל מה שעומד מול עיני זה שאתם תצאו עם בסיס יציב ואיכותי של הבנה וחומר ממנו תוכלו לצאת לכל עשייה עצמאית בעתיד בתחום ה-AI DL deployment

o       מניסיוני לא נתקלתי בסיטואציה שפיתוח היום מתחיל מאפס אלא תמיד מתחיל מדוגמא עליה מבצעים את ההתאמות הנדרשות

o       מי שבאמת רוצה למנף ולהצליח בהשגת מטרת שלב זה של הקורס יעשה הכל בעצמו תוך כדי קבלת עזרה ממני או מהדוגמא שסיפקתי

מזכיר שוב שאני בחופש בתאריכים 24-28\7 אז זמינותי מוגבלת אך באופן כללי תרגישו חופשי לשאול שאלות באמצעי המדיה ואשתדל לחזור לכולם במהרה.

כמו כן, נעשה מאמץ בתחילת חודש 08 להשמיש שוב פעם את המכונות בענן ואם זה יצלח כמובן נעדכן.

במפגש הבא:

·        נלמד על TRT Quantization ל-Int8 – 3 שעות

·        נלמד על מודול OpenCV DNN - שעה

·        חצי יום פתוח לליווי, מענה לשאלות, תמיכה, חזרה וכדומה לטובת קידום פיתוח הפרויקט

·        אם יתאפשר נציג עוד מספר יכולות אשר נלוות ל-TRT SDK

בברכה

להלן פירוט של החומר:

1.      מצורפת מצגת שהוצגה במהלך המפגש - 9\_NVIDIA\_TensorRT.pdf.  
מצגת זו מסכמת את ה-Highlights של יכולות ה-TRT.  
לינק למדריך השלם המכיל את כל התיאוריה, יכולות וקוד דוגמא ניתן למצוא כאן:  
[TensorRT SDK | NVIDIA Developer](https://developer.nvidia.com/tensorrt)  
[TensorRT - Get Started | NVIDIA Developer](https://developer.nvidia.com/tensorrt-getting-started)

2.     סקרנו דוגמאות של ה-TRT מבוססות Python שהן חלק מה-TRT SDK:

o       onnx\_resnet50.py:

1.     קבלת onnx מוכן לאחר אימון,

2.     טעינה שלו,

3.     הכנת אובייקטים של TRT הנדרשים לבצע את הצעדים הכוללים,

4.     המרה של ה-Onnx למודל מאופטם,

5.     טעינת תמונה לדוגמא,

6.     ביצוע pre processing,

7.     הקצאת זיכרונות על ה-Device באמצעותם ה-TRT יבצע את ה-Inference,

8.     ביצוע ה-Inference

9.     ניתוח התוצאות.

o       yolov3\_onnx.py:

1.     מדגים טעינת מודל מאומן של Torch מבוסס cfg אשר מחזיק את הטופולוגיה של המודל כלומר רק שכבות ללא משקלים המגיעים מקובץ אחר בדרך כלל pth

2.     יש להוריד את התכנים המוצגים בקובץ download.yml שלא מגיעים כחלק מה-SDK

3.     טוען את כל שמות השכבות של המודל ל-Container מסוג List

4.     טוען את המשקלים מקובץ נפרד (בד"כ סיומת קובץ זה תהיה \*.pth או \*.weights)

5.     מרכיבים ידנית את ה-Onnx model לפי המידע שנטען על הטופולוגיה והמשקלים).  
זו דוגמא להמחשה של יכולות מתקדמות, בפועל איש Deployment  יקבל Onnx מוכן הכול כבר את כל המידע שנדרש לקראת המעבר ל-TRT.  
  
המטרה כאן היא בנוסף להמחיש כמה חופש קיים בשלב ה-Deployment ועד כמה ניתן לאלתר, לערוך "להמציא" מחדש עד כמה שנדרש במידה ויש צורך בכך.

6.     ביצוע בדיקת היתכנות על ידי check model שמחזיר Runtime exception שמידה ויש תקלה כך שאם היא לא עלתה זו אינדיקציה שהכל תקין

7.     שמירה לדיסק של ה-Onnx בשם yolov3.onnx

o       onnx\_to\_tensorrt.py:

1.     טעינת ה-Onnx

2.     ביצוע Pre processing הכולל:

1.     Resize לפי מה שהמודל מצפה לקבל כ-Tensor input size

2.     המרה ל-fp 32 bits

3.     קיבוע ל-Row major memory layout

4.     המרה מ-NHWC ל-NCHW

5.     הוספת ממד ה-Batch size

3.     ביצוע Inference

4.     Post processing:

1.     המרה מ-NCHW ל-NHWC

2.     פירוק של מערך רציף של כל שלושת ה-Tiles שנותנים לכל Tile מערך בגודל של 255 ל-3 גדלים נפרדים לכל Anchor שלו וכך לכל אחיד יש 85 ערכים שמורכבים מ:

1.     Box top, left, right, bottom

2.     Objectness

3.     80 confidences values כי יש 80 labels

3.     ביצוע פעולות מתמטיות שנכון לגרסת YoloV3 ישנה היו מחוץ למודל, כיום הן חלק ממנו.  
מטרת פעולות אלו להביא למצב שפורמט הפלט הגולמי של הרשת יהיה אנושי כלומר ניתן להבנה של אפיוני ה-Detections  
לאחר פעולות אלו פר Tile, פר כל Anchor שלו אנו מקבלים את שלושת הנתונים המוזכרים מעלה בפורמט אנושי

4.     NMS – סינון Detections לפי פילטרים כגון ערך סף ל-Confidence, חישוב IOU בין שני Detections עם אותו Label

5.     יצירת תמונה עם ה-BB

שתי דוגמאות שדיברנו עליהם בעל פה בלבד על מנת לתאר את היכולות שחבילת ה-TRT מספקת לטובת הכירות בלבד

o       onnx\_packnet

1.     המחשה של יכולות Onnx Graph surgeon – כלי שהוא חלק מה-TRT SDK ונדרש להתקנה נפרדת על ידי whl ובאמצעות pip install

2.     כלי המאפשר עריכה של כל Onnx קיים ואפילו יצירה שלו מ-Scratch

3.     בדוגמא הנוכחית השתמשו ברשת שנקראת Packnet

4.     ביצעו שתי עריכות עיקריות:

1.     קיבוץ של מספר שכבות שמבצעות Instance Normalization, Reshape, mul & Add לשכבה אחת שבמבצעת הכל בפעולה אחת מורכבת יעילה יותר שנקראת GroupNormalizationPlugin שבעצם הופכת למה שנקרא TRT plugin שגם משמש לטיפול ב-Custom layers עבור Unsupported layer

2.      במודל המקורי היה קבוצת שכבות שתוצאתן הייתה קלט נוסף לפעולת Pad.  
פעולות היו קבועות ופעלו תמיד על מאפיינים קבועים שקיבעו את תוצאתן ולכן ניתן להיפטר מהן ולקבע את התוצאה כמאפיין קבוע בערכו של פעולת ה-Pad וכך לחסוך חישוב שלהן כל פעם מחדש ב-Runtime.

o       engine\_refit\_onnx\_bidaf:

1.     המחשה שניתן לערוך מחדש מודל מאופטם במידת הצורך ולא חייב לבנות אותו מחדש מה-Onnx

2.     העריכה היא כירורגית במקומות שרק בהן זה נדרש.

3.     במקרה של דוגמא זו מראים ביצוע של שינוי שמות של שכבות וגרמו בכוונה תחילה לכך שלא נשמרו להם המשקלים.

4.     לאחר מכן מראים שלאחר יצירת ה-Engine באמצעות refitter ניתן לטעון להם משקלים ישירות למודל המאופטם

3.     כדי להיות מסוגלים להריץ מודל שאומן על ידי Framework טיפוסי כגון TF או Torch באמצעות TRT נדרש תחילה להמיר לOnnx.  
עבור מודל שאומן על ידי TF מבצעים זאת באמצעות כלי שנקרא tf2onnx שזה Python package שניתן להתקנה באמצעות pip.  
עבור Torch היכולת להמיר ל-Onnx היא חלק מהחבילה המקורית של Torch.  
לינקים להתקנות של TF וגם של Torch מתועדות בסיכום המפגש הראשון.  
קוד דוגמא להמרה מ-Torch אל Onnx ניתן למצוא בלינק זה:  
[(optional) Exporting a Model from PyTorch to ONNX and Running it using ONNX Runtime — PyTorch Tutorials 1.12.0+cu102 documentation](https://pytorch.org/tutorials/advanced/super_resolution_with_onnxruntime.html)

o       הוצג תרחיש Simple ללא תקלות כיצד מייצרים Onnx עבור TF & Torch VGG19 וגם עבור Torch Resnet50.  
בשני המקרים הקלטים למודלים היו Static size.

o       הוצג דוח ה-Torch Onnx export verbose שממנו אפשר ללמוד כיצד למפות Onnx layer אל ה-Torch layer שממנה הוא נוצר

o       כדי לדמות תרחיש תקלה ביצירת Onnx בחרתי לייצר את ה-Onnx באופן דינאמי מאפיים שהתנגש עם שכבה שנקראת adaptive\_avg\_pool2d

o       הצגה של תרחישי תקלות:

1.     תקלת Onnx export –   
דוח Verbose וכיצד ניתן להשתמש בו כדי להבין איפה מקור התקלה במודל המקורי ומה ניתן לעשות כדי להתגבר עליה:

2.     "ניסור" – חילוץ לוגיקה שהיא חלק מהמודל ללוגיקה חיצונית שאו תמומש באמצעות PlugIn\Custom layer או ממש חיצונית מפעולת ה-Inference

3.     בכל מקרה עבור כל שינוי צריך לוודא שהשינוי לא דורש אימון מחדש

4.     חיפוש אחר אלטרנטיבות ברמת Python\Tortch למשל Reshape במקום squeeze למשל

5.     Data Dependency – Dynamic Shape (DD-DS) לא נתמך למשל NonZero operator

4.     סקרנו את הקוד גם מהיבט של Code review וגם מהיבט של תפעול של הפרויקטים הבאים:

o       TensorRT\_Playground.zip:

1.     OptimizeYoloV5WithTensorRT.py  -

1.     בעיקר שוחחנו על הבדלים של שלב ה-Post processing בין V5 ל-V3 (ראו תיאור מעלה)

2.     הצגנו שבדוגמא זו לא מופעלת הלוגיקה שנקראת NMS המשתמשת בשיטת חישוב IOU לפילטור BB חופפים לטובת ניקוי BB הנחשבים כרגע.

3.     הזכרתי שכיום לפחות ברמת ה-Python ה-Frameworks כגון Torchvision מספקים מתודת NMS מובנית ולא צריך ליישמן עצמאית.  
במעבר ל-C++\CUDA בהחלט נדרש להכיר את הלוגיקה וליישם.

4.     <https://github.com/ultralytics/yolov5>

2.     OptimizeONNXModelWithTensorRT.py – לא הספקנו לסקור לפרטים אבל זו עוד דוגמא למסלול End2End משלב יצירת מודל מאומן באמצעות TF, המרה ל-Onnx, המרה ל-TRT והרצת Inference.  
חומר נלווה לדוגמא זו נקרא fashion-mnist וצריך להוריד אותו מהלינק:  
[GitHub - zalandoresearch/fashion-mnist: A MNIST-like fashion product database. Benchmark](https://github.com/zalandoresearch/fashion-mnist)

3.     מצורף קובץ עם דוגמאות הקוד TensorRT\_Playground.zip.  
יש לשנות את סוג הקובץ מ-.py.txt ל-.py

5.     סקרנו את הפרויקט Deployment\_PreTrained\_Playgroud – מצורף קובץ Deployment\_PreTrained\_Playgroud.zip.

o       גרסת הקבצים מבוססת על הפרויקט שאני עצמי הגשתי במחזור שלי.

o       הלוגיקה מכילה דוגמאות של 3 Classifiers:

1.     Alexnet - Torch

2.     Resnet50 - Torch

3.     VGG19 –ל-TF & Torch

o       כמו כן יש לוגיקה המפעילה Inference באמצעות חבילת Onnxsim לטובת השוואת תוצאות אל מול Torch\TF predict וכך ניתן לוודא תקינות ה-Onnx טרם שילוב ה-TRT

o       הפעלת ה-TRT ב-3 מודים:

1.     FP 32

2.     FP 16

3.     Int 8 – עוד לא נלמד

שלום,

כחלק ממה שאני מתכנן למפגש הבא בנוגע ל-OpenCV DNN אני חושב שיעזור לכם מאוד לקרוא את המידע שמתואר בלינק הבא:

[Demystifying GPU Architectures For Deep Learning – Part 1 | (learnopencv.com)](https://learnopencv.com/demystifying-gpu-architectures-for-deep-learning/?ck_subscriber_id=272192348)

בלינק זה תמצאו את כל המידע בנוגע ל-CUDA basic & GPGPU arch עליהם הצגתי במהלך המפגשים שהיו עד כה.

במהלך המפגש הבא אני אתאר בין השאר מה זה LearnOpenCV ואיך זה יכול לעזור לכם:

[AI Education For All, From Your First Steps to Mastery! (learnopencv.com)](https://learnopencv.com/)

בברכה