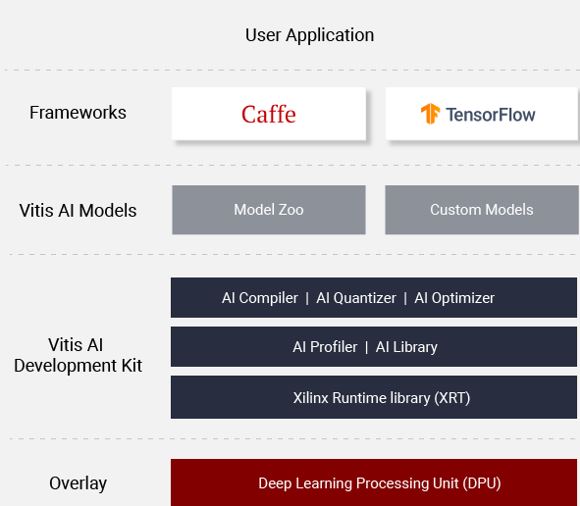
## 

## **Guide Book**

## Deep Neural Network Inference on FPGA using TensorFlow



**מבצעים**: אלון נמירובסקי ועמית שטאובר  
**מנחים**: אינה ריבקין, עוז שמואלי

אביב תשפ"א / אפריל 2021

תוכן עניינים

[מבוא ...........................................................................................................................3](#_Toc69473980)

[הקמת עמדה 4](#_Toc69473981)

[צריבת SD-Card 8](#_Toc69473982)

[Setup של כרטיס FPGA 9](#_Toc69473983)

[הקמת רשת, קומפילציה והרצת הרשת Partial Flow For Resent-50 : 15](#_Toc69473984)

[ שלב הקוונטיזציה 15](#_Toc69473985)

[ שלב הקומפילציה 19](#_Toc69473986)

[הקמת רשת, קומפילציה, אופטימיזציה וסיווג: Full Flow For Mnist 22](#_Toc69473987)

[ שלב הקוונטיזציה 23](#_Toc69473988)

[ שלב הקומפילציה 24](#_Toc69473989)

[ שלב הסיווג/הרצה על הboard 27](#_Toc69473990)

[ אופטימיזציה וביצועים 28](#_Toc69473991)

[טיפים, בעיות ופתרון בעיות 31](#_Toc69473992)

[תובנות כלליות 32](#_Toc69473993)

מבוא   
  
**רקע כללי**

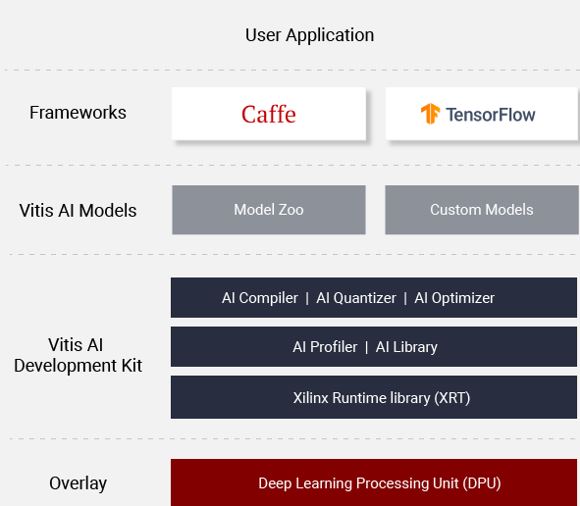
* תחום המערכות הלומדות המבוסס על רשתות נוירונים צובר תאוצה בשנים האחרונות, בעקבות ההצלחה של מערכות אלו בפתרון בעיות מורכבות.
* יישום של רשתות מתבצע ב-2 שלבים – שלב האימון ושלב הסיווג.
* שלב האימון הוא שלב מקדים המתבצע פעם אחת לכל רשת נתונה ולרוב מבוצע על מערכת בעלת עוצמת חישוב גבוהה כגון GPU. כיוון ששלב זה מבוצע פעם אחת לכל מערכת אין חשיבות עקרונית לזמן וכוח חישוב ששלב זה דורש.
* לעומת זאת בהרבה אפליקציות כגון רובוטים, מל"טים, רשתות חיישנים, טלפונים סלולאריים, רכבים אוטונומיים וכו' שלב הסיווג הוא שלב המתבצע בשטח ע"י רכיבים המוגבלים בכוח החישוב, בהספק, ולעתים גם יש דרישה לעבודה בזמן אמת ולמחיר זול.
* כיוון שאחד המאפיינים הבולטים של שלב הסיווג ברשתות נוירונים הוא מקביליות, מימוש שלב זה דווקא על חומרה כגון FPGA מאפשר להגיע לביצועים גבוהים מאוד תוך שמירה על צריכת הספק נמוכה ומחיר זול. כמו-כן, מימוש החומרה על FPGA נותן גמישות ויכולת התאמה של הרשתות עבור בעיות שונות.
* כדי לקדם מימוש רשתות נוירונים בחומרה על רכיבי FPGA יצרני רכיבים אלו פתחו סט כלים חדש שמאפשר מימוש שלב הסיווג ואופטימיזציה של רשת החל משלב ההגדרה עד שלב המימוש בפועל, תוך שימוש בכלים סטנדרטיים.

**מטרת הפרויקט**

לחקור ולהעריך סט כלים חדש **Vitis AI** עבור  DNN inference(סיווג ע"י רשת נוירונים) על רכיבי  FPGAשל חברת Xilinx.

בנית הרשת תעשה באמצעות TensorFlow

**דרישות הפרויקט:**

1. הקמת המערכת על מחשב ייעודי.
2. בחירה והרצה של מודל שאומן מראש תוך שימוש בכל כלי התוכנה של חברת Xilinx.
3. מימוש סביבת הבדיקה.
4. ביצוע שינויים והתאמות במודלים במטרה לבדוק את גמישותם ויכולתם של הכלים.

לדוגמא שינוי מבנה הרשת (model zoo vs Custom), טיפוסי המשתנים (FP vs Fixed point) וכו'.

1. השוואה בין המימושים השונים תוך בדיקה של דיוק, throughput, גודל ואתגרים במימוש ושימוש בכלים.
2. הסקת מסקנות ודיווח על מגבלות, יכולות ובעיות של הכלים השונים בהם נעשה שימוש.

# הקמת עמדה

חברת Xilinx מספקת לנו תשתית שלמה להרצת מערך הVitis-AI הדרוש להרצת מערכות בינה מלאכותית על הלוח ZCU104. נתאר כאן את תהליך הקמת התשתית על גבי המחשב האישי הכולל התקנת Virtual Machine ותוכנות שונות.

**סדר פעולות:**

**1. הקמת VM על גבי המחשב האישי**

1.1 ) הורדת קובץ ubuntu-18.04-desktop-amd64.iso : https://releases.ubuntu.com/18.04

1.2 ) יצירת VM באמצעות VMware player workstation -   
התקנה סטנדרטית (כלומר להתקדם באשף כרגיל) הכוללת: הקצאת 4GB ראם, 350GB הארד-דיסק ו4 ליבות.

הערה - אין לעבוד עם VirtualBox כי אנחנו קיבלנו הודעת שגיאה שיש פקודות שלא נתמכות בVirtualBox.

**2. התקנת Docker:**

Docker היא פלטפורמה ייחודית שמאפשרת למשתמשים להריץ Containers שונים. כל Container מכיל סביבה שונה בעלת הגדרות וקונפיגורציה שהוגדרו מראש. בפרויקט שלנו אנחנו נעבוד עם Container מוכן אשר Xilinx מספקים לנו.

יש לפעול לפי המדריך Install Docker Engine on Ubuntu - Program/References/PDFs\_Archive/1))

המדריך כולל את כל השלבים שיש לבצע על מנת להתקין Docker על גבי המכונה הוירטואלית.

**3. התקנת Pre-built container :**

המדריך **Setting Up the Host** (Program/References/PDFs\_Archive/21) מפרט כיצד יש להעתיק מקומית לסביבה שלנו על גבי הלינוקס את הקוד הרלוונטי ואת הפקודות של ההרצה:

git clone --recurse-submodules <https://github.com/Xilinx/Vitis-AI>

cd Vitis-AI

cd docker

./docker\_build\_cpu.sh

./docker\_run.sh xilinx/vitis-ai-cpu:latest

הערה: במדריך מצוינת גם אופציה של שימוש בBuild של GPU. כאשר ניסינו להשתמש בו נתקלנו בשגיאה, אך היא אינה רלוונטית עבור המשך המסמך.

גרסת הדוקר:



לאחר ההתקנה הראשונית, מדי פעם נרצה לצאת מהdocker ולהיכנס אליו חזרה. על כן, נרשום את הפקודות הרלוונטיות לרצת הdocker שוב.

פקודות להרצת הdocker:

./docker\_run.sh xilinx/vitis-ai-cpu:latest

conda activate vitis-ai-tensorflow

נציין כי אנו נתקלנו Docker Permission denied -

תמונה שמכילה טקסט

התיאור נוצר באופן אוטומטי

ניתן לראות כפי שמופיע בתמונה לעיל, הפתרון הינו להריץ:

sudo chmod 666 /var/run/docker.sock

באופן כללי, כפי שמובא מהאתר הרשמי של Xilinx, יש ארבעה שלבים שיש לוודא שביצעתם:

תמונה שמכילה טקסט

התיאור נוצר באופן אוטומטי

**4. הורדת qBitorrent (לא הכרחי):**

במכונה הוירטואלית יש להריץ -

sudo apt install qbittorrent

נציין שאין הכרח לבצע התקנה של qBitorrent על מנת להתקדם במדריך. זה הכרחי במידה והData שלכם לאימון נדרש להורדה דרך פלטפורמה זאת.

**5. התקנת הPetalinux:**

כחלק מהפרויקט בדקנו היתכנות של קימפול קוד הRTL של רכיב הDPU וצריבתו על הZCU-104. נעזרנו במדריך הבא:

Program/References/PDFs\_Archive/17

**הערה:** יש לשים לב שבעת ביצוע השלבים, אתם פועלים בהתאם לדרישות המקדימות בתחילת המדריך (גרסת petalinux, גרסת מכשיר וכו').

ראשית יש לבצע קימפול של קוד הdesign של חומרת ה-DPU בעזרת כלי Xilinx.

בהמשך, מורידים מפה את שני הקבצים:

<https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/embedded-design-tools.html>

תמונה שמכילה טקסט

התיאור נוצר באופן אוטומטי

תמונה שמכילה טקסט

התיאור נוצר באופן אוטומטי

יש לפעול לפי הוראות ההדרכה. נציין כי בכמה מקרים ביצענו התאמות:

פקודות בהן נעזרנו (בוצעו התאמות לכרטיס שלנו בהתאם לסוגו):

petalinux-config --get-hw-description=~/Vitis-AI/BootExp/VivadoOutput/prj –-silentconfig

petalinux-build

נדרשנו לשנות את הקונפיגורציה בקובץ הבא:

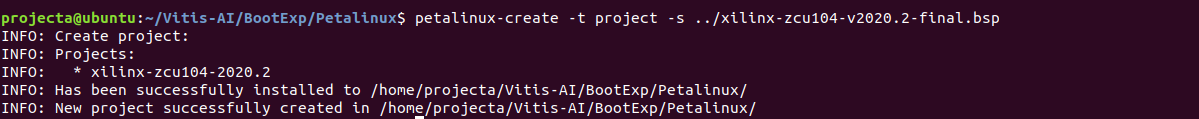
project-spec/meta-user/conf/petalinuxbsp.conf

לערכים הבאים:

תמונה שמכילה טקסט

התיאור נוצר באופן אוטומטי

באופן כללי לאחר יצירת הפרויקט תקבלו את ההודעה הבאה:



**בעיות נפוצות:**

* במקרה של השגיאה הבאה:

petalinux-create: command not found

יש להיעזר בדיון הבא:

Program/References/PDFs\_Archive/18

* במידה ויש כישלון בתהליך הBuild, ניתן להיעזר במדריך הבא:

Program/References/PDFs\_Archive/19

* העברה לכרטיס זיכרון יש לבצע על פי המדריך הבא, עמוד 61:

Program/References/PDFs\_Archive/20

**קישורים חשובים:**

* העמוד הרשמי של Vitis-Ai בGitHub. כולל הסברי התקנה שונים וכולל הסבר כללי על מה אנחנו מקבלים בVitis-AI. מומלץ לעיין:

Program/References/PDFs\_Archive/3

* **במידה ויש צורך לבצע התקנת VART (Vitis AI Runtime),** ניתן להיעזר במדריך הבא מGitHub:

Program/References/PDFs\_Archive/5

או במדריך הבא מהאתר של Xilinx:

Program/References/PDFs\_Archive/2

* פרויקט נוסף בGitHub של שימוש בVitis-Ai:

Program/References/PDFs\_Archive/4

# צריבת SD-Card

צריבת הכרטיס הינה פעולה הכרחית לטעינה קוד התכן אל הלוח. אנו נעזרנו בתוכנת צריבה אשר נראית etcher ובקובץ מוכן שניתן על ידי Xilinx. יש להיעזר באתחול במדריך הבא:

Program/References/PDFs\_Archive/14

שלבים (מופיעים גם במדריך):

1) יש להוריד את הImg עבור ZCU-104:

<https://www.xilinx.com/bin/public/openDownload?filename=xilinx-zcu104-dpu-v2020.1-v1.2.0.img.gz>

2) הורדת תוכנת Etcher והרצתה כמנהל

3) הכנסת כרטיס זיכרון מסוג sd-card למחשב

4) במסך שקיבלתם יש ללחוץ על select image ולבחור את הקובץ שהורדנו בסעיף 1 –



5) כעת התוכנה תזהה את כרטיס הזיכרון וכל מה שנדרש הוא לבצע Flash.

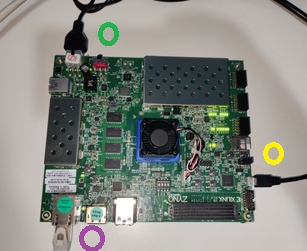
6) כעת כרטיס הזיכרון שלכם צרוב וטעון עם קובץ BOOT שמכיל שני חלקים עיקריים:

* קוד הDPU שייצרב אל הFPGA (קוד Verilog)
* מערכת ההפעלה שתרוץ על מעבדי הARM

# Setup של כרטיס FPGA

בשלב הזה נרצה לבצע אתחול של ZCU-104. ראשית נציין מהם החיבורים הרלוונטים:

* כבל חשמל (עיגול ירוק)
* כבל Micro-USB – ישמש לתקשורת עם המחשב האישי בפרוטוקול UART (עיגול צהוב)
* כבל Ethernet – ישמש לתקשורת עם העולם החיצוני דרך רשת האינטרנט (עיגול סגול)



ZCU-104 Bring Up:

1) יש לחבר את הכבלים כפי שצילמנו בתמונה לעיל.

2) יש לסדר את המתגים כמתואר למטה בתמונה (נקודה כחולה מעיד על למעלה ואדומה על למטה).

תמונה שמכילה אלקטרוניקה, מעגל חשמלי

התיאור נוצר באופן אוטומטי

3) יש לבצע הדלקה של הלוח ולחכות כדקה.

תמונה כללית של הלוח לאחר הדלקה (ZCU104):

תמונה שמכילה אלקטרוניקה, מעגל חשמלי

התיאור נוצר באופן אוטומטי

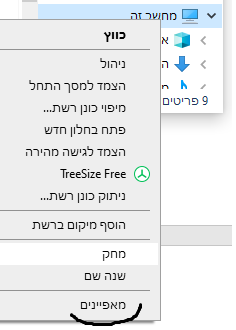
יש לוודא כי הנורה המסומנת דלוקה:

תמונה שמכילה אלקטרוניקה, מעגל חשמלי

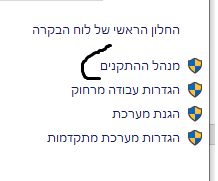
התיאור נוצר באופן אוטומטי

4) כעת נחבר את הכרטיס למחשב האישי באמצעות כבל micro-usb.

חיבור זה יוצר שלושה חיבורי COM: יש לבחור את הCOM בעל הערך המינימלי . ניתן למצוא את זה תחת מנהל ההתקנים. כדי להגיע למנהל ההתקנים יש ללחץ למחשב שלי ולבחור מאפיינים:



בעמוד שייפתח יש לבחור במנהל ההתקנים:

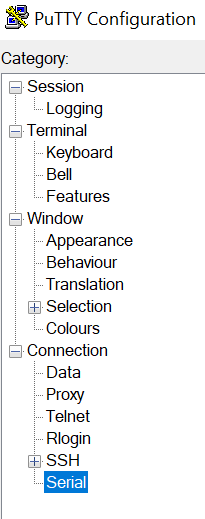


ובדף ההגדרות שייפתח לפתוח את חיבורי הCom –

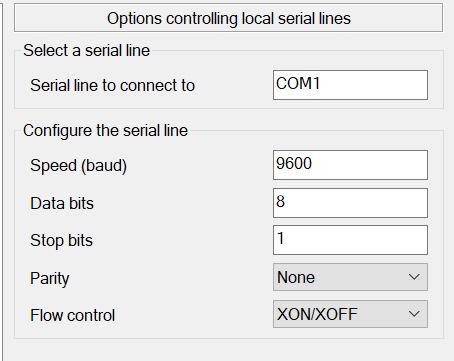


כדי להתחבר לכרטיס ניעזר בתוכנת Putty שמאפשרת חיבור במגוון פרוטוקולים.

בתוכנה נבחר בחיבור טורי (סריאלי):



ובמסך ההגדרות:



5) יש לשנות את ה-Serial line המתאים שהופיע במחשב שלך ולעדכן את מהירות העברת הנתונים:

baud rate: 115200 bps

המסך המתקבל בחיבור הטורי:תמונה שמכילה טקסט

התיאור נוצר באופן אוטומטי

לאחר ההתחברות אפשר לרשום ifconfig ולמצוא את כתובת הIP של הלוח.

6) במידה ויש צורך, ניתן להתחבר ללוח עם כבל רשת ב ssh(שם משתמש וסיסמא – (root:

תמונה שמכילה צילום מסך, צג, מחשב, מחשב נישא

התיאור נוצר באופן אוטומטי

**העברת קבצים מול הלוח**

ניתן להוריד WinSCP – תוכנה אשר משמשת להעברת קבצים מהמחשב הלוקלי למחשב המרוחק עם תצוגה למשתמש.

לחלופין, ניתן להשתמש בפקודות הבאות:

פקודה להעביר תיקייה מהמחשב הלוקאלי למחשב המרוחק:

scp -r c:\move2fpga\ root@192.168.1.26:~/Vitis-AI/VART/samples/images

במקרה שמתקבלת תשובה לתמונה אחת בלבד יש לפעול לפי האתר הבא:

Program/References/PDFs\_Archive/15

פקודת להעברת קובץ:

scp c:\move2fpga\dpu\_resnet50\_tf\_0.elf root@192.168.1.26:~/Vitis-AI/VART/samples/images

תמונה שמכילה טקסט

התיאור נוצר באופן אוטומטילהלן דוגמה להרצה של מודל resnet-50 אשר מגיע "מוכן" על גבי הכרטיס:

כאשר הרצנו:

root@xilinx-zcu104-2020\_1:~/Vitis-AI/VART/samples/amitalon\_resnet50# ./resnet50 dpu\_resnet50\_tf\_0.elf

קובץ הC שקומפל לresnet50 נלקח מפה:

Program/References/PDFs\_Archive/16

# הקמת רשת, קומפילציה והרצת הרשת Partial Flow For Resent-50 :

## שלב הקוונטיזציה

בשלב זה אנו מריצים מודל שהוכן על ידי Xilinx והיה קיים בZoo. מדובר באחת העוצמות של Xilinx אשר מספקת ללקוחותיה מודלי AI שהוכנו מראש. כמובן, שניתן היה להשתמש גם במוצר הגולמי שכבר מגיע מוכן לריצה על הלוח. עם זאת, אנחנו לקחנו אך ורק את המודל בתצורתו הגולמית (כפי שיצא מTF) כדי לנסות ולהעביר חוטר שכולל את כל הכלים של Xilinx לביצוע קוונטיזציה, קימפול והרצה.

לשם הרצה בחומרה, אנחנו מעוניינים לעבור לייצוג המשקלים במודל בצורה של מספרים שלמים. כאן בדיוק שלב הקוונטיזציה נכנס לפועל. מטרתו ראשית לבצע קוונטוט למשקלים כדי למודל לרוץ ביעילות על הFPGA. בנוסף הוא מאפשר לבצע מיני אופטימזציות – עליהן נדבר בהמשך.

לשימושכם, מספר מדריכים שבהם נעזרנו לביצוע התהליכים שנפרט עליהם:

הדרכה כללית מאתר Xilinx:

Program/References/PDFs\_Archive/6

הורדת validation set (שימושי באופן כללי בשלב זה):

Program/References/PDFs\_Archive/7

הורדת מודל:

Program/References/PDFs\_Archive/8

1) אנו בחרנו להשתמש במודל הבא:



Resnet-50 הינו מודל AI לסיווג עצמים בתמונות. זה והמודל שהונחינו להשתמש בהחלטה משותפת עם המנחים.

2) נעבוד כעת על סביבת ה-Linux (אלא אם נאמר אחרת). יש ליצור תיקייה חדשה בתוך התיקיות שנעצרו לאחר התקנת הImg של הDocker.

3) כדי לבצע את שלב הקוונטיזציה, נדרשים שלושה אמצעים:

תמונה שמכילה שולחן

התיאור נוצר באופן אוטומטי

על כן, יצרנו תיקייה בשם imagenet\_images עם אלף תמונות מהvalidation set (קישור לעיל). היא תשמש לכיול.

בנוסף, יצרנו קובץ פייתון input\_fn כפי שמוגדר במדריך 6:

תמונה שמכילה טקסט

התיאור נוצר באופן אוטומטי

כמו כן יצרנו קובץ imagenet\_calib.txt. הוא מכיל אלף שורות מן הצורה:

תמונה שמכילה טקסט

התיאור נוצר באופן אוטומטי

להעברה לכרטיס, ניתן להיעזר בפקודה להעתקת אלף הקבצים הראשונים:

cp `ls | head -1000` ../../Vitis-AI/projecta/imagenet\_images

לאחר כל שלבים אלה ניתן להריץ שלב הקוונטיזציה.

הפקודה שהרצנו לבצע את תהליך הקוונטיזציה:

vai\_q\_tensorflow quantize --input\_frozen\_graph resnet\_v1\_50\_inference.pb --input\_nodes input --input\_shapes ?,224,224,3 --output\_nodes resnet\_v1\_50/predictions/Reshape\_1 --input\_fn input\_fn.calib\_input --method 1 --gpu 0 --calib\_iter 20 --output\_dir ./quantize\_results

ההודעה בסיום ההרצה:

תמונה שמכילה טקסט

התיאור נוצר באופן אוטומטי

**תקלות נפוצות**

1. תקלת core dumped – הסיבה לתקלה זו נבעה אצלנו כתוצאה מהקצאה של מקום בכונן שאין בו מקום, לכן חשוב להקצות מקום בכונן עם הרבה מקום, במקרה של המחשב הנוכחי זה היה כונן D.
2. תקלת FMA Instructions not supported – תקלה אשר התקבלה כתוצאה מעבודה עם virtual box, חשוב לעבוד בVMARE אחרת לא ניתן לעבוד.

## שלב הקומפילציה

מדריך כללי מאתר Xilinx:

Program/References/PDFs\_Archive/9

שלב הקומפילציה הוא שלב שבו אנו לוקחים את הפלט של שלב הקוונטיזציה ומעבירים אותו למעשה לתוצר שהרכיב שלנו מצפה לקבל. בסופו של דבר, זה יהיה ייצוג של המודל עבור הDPU (קוד החומרה).

הפקודה לביצוע קומפילציה (הרצנו אותה מתוך אותה תיקייה שהרצנו את הפקודה של הquantization):

vai\_c\_tensorflow --frozen\_pb ./quantize\_results/deploy\_model.pb --arch /opt/vitis\_ai/compiler/arch/dpuv2/ZCU104/ZCU104.json --output\_dir model --net\_name resnet50\_tf

הערה:שימו לב כי הפקודה באתר איננה מדויקת, הפקודה שרשמנו כאן היא המדויקת.

**הפלט שקיבלנו:**

projecta@ubuntu:/workspace/projecta$ vai\_c\_tensorflow --frozen\_pb ./quantize\_results/deploy\_model.pb --arch /opt/vitis\_ai/compiler/arch/dpuv2/ZCU104/ZCU104.json --output\_dir model --net\_name resnet50\_tf

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* VITIS\_AI Compilation - Xilinx Inc.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

[VAI\_C][Warning] layer [resnet\_v1\_50\_SpatialSqueeze] (type: Squeeze) is not supported in DPU, deploy it in CPU instead.

[VAI\_C][Warning] layer [resnet\_v1\_50\_predictions\_Softmax] (type: Softmax) is not supported in DPU, deploy it in CPU instead.

Kernel topology "resnet50\_tf\_kernel\_graph.jpg" for network "resnet50\_tf"

kernel list info for network "resnet50\_tf"

Kernel ID : Name

0 : resnet50\_tf\_0

1 : resnet50\_tf\_1

Kernel Name : resnet50\_tf\_0

--------------------------------------------------------------------------------

Kernel Type : DPUKernel

Code Size : 0.59MB

Param Size : 24.35MB

Workload MACs : 6964.51MOPS

IO Memory Space : 2.25MB

Mean Value : 0, 0, 0,

Total Tensor Count : 59

Boundary Input Tensor(s) (H\*W\*C)

input:0(0) : 224\*224\*3

Boundary Output Tensor(s) (H\*W\*C)

resnet\_v1\_50\_logits\_Conv2D:0(0) : 1\*1\*1000

Total Node Count : 58

Input Node(s) (H\*W\*C)

resnet\_v1\_50\_conv1\_Conv2D(0) : 224\*224\*3

Output Node(s) (H\*W\*C)

resnet\_v1\_50\_logits\_Conv2D(0) : 1\*1\*1000

Kernel Name : resnet50\_tf\_1

--------------------------------------------------------------------------------

Kernel Type : CPUKernel

Boundary Input Tensor(s) (H\*W\*C)

resnet\_v1\_50\_SpatialSqueeze:0(0) : 1\*1\*1000

Boundary Output Tensor(s) (H\*W\*C)

resnet\_v1\_50\_predictions\_Softmax:0(0) : 1\*1\*1000

Input Node(s) (H\*W\*C)

resnet\_v1\_50\_SpatialSqueeze : 1\*1\*1000

Output Node(s) (H\*W\*C)

resnet\_v1\_50\_predictions\_Softmax : 1\*1\*1000

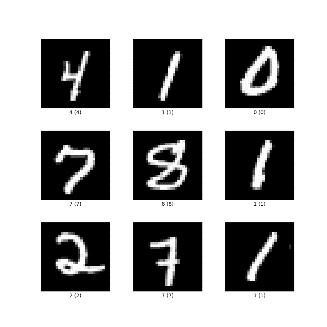
נציין כי קיבלנו תיקיית מודל ולהלן הפירוט שלה:

תמונה שמכילה טקסט

התיאור נוצר באופן אוטומטי

# הקמת רשת, קומפילציה, אופטימיזציה וסיווג: Full Flow For Mnist

בשלב זה אנו יוצרים מודל בצורה מלאה, החל מכתיבתו בTF ועד הרצת שלב הסיווג על גבי הכרטיס.  
בחרנו לעבוד על מימוש מודל פשוט לזיהוי ספרות בתמונות על מאגר ידוע שנקרא mnist:



התבססנו על המדריך הבא:

Program/References/PDFs\_Archive/10

מיקום בגיט: References/Vitis-AI-Tutorials-MNIST-Classification-TensorFlow.zip

1) השלב הראשון היה באופן טבעי יצירת המודל. נעזרנו בפלטפורמת הColab בשפת python בעזרת ספריות TensorFlow ,keras. אנו מצרפים את הקוד הסופי כנספח.  
באופן כללי הקוד מחולק לשני חלקים עיקריים: יצירת המודל בפני עצמו ואימונו כדי לקבל את הפרמטרים המעודכנים. בשלב הזה קיבלנו את הגרף של המודל (graph). בחלק השני ביצענו "הקפאה" של הגרף וזאת כדי שנוכל לייצא אותו כקובץ PB. זהו סוג הקובץ שנדרש בשלב הקוונטיזציה.

הקוד הסופי מצורף: מיקום בגיט: References/Start.ipynb

**הערות ובעיות נפוצות שנתקלנו בהן:**

* לא ניתן להעזר בגרסה tensorflow 2 ומעלה (דיון בפורום בנוגע להתאמה בין גרסאות של טנזור):

Program/References/PDFs\_Archive/12

* לא ניתן לעבוד עם שכבות FC, יש לעבוד עם שכבות קונבולוציה.

Program/References/PDFs\_Archive/13

* נוצרה בעיה כאשר נעזרנו בשכבת dense-softmax, המרנו את השכבה לשכבת סיגמואיד והגענו לתוצאה מעולה.
* את תהליך שמירת הגרף כתבנו ונעזרנו גם בקוד שמופיע בדיון הזה:

Program/References/PDFs\_Archive/11

2) השלב הבא היה יצירת קבצי קליברציה:

מיקום בגיט: References/trainimages\_calib\_original.zip

סקריפטים ליצירת התמונות:

מיקום בגיט: References/4\_quant.zip

## שלב הקוונטיזציה

תזכורת:

כדי לבצע את שלב הקוונטיזציה אנו זקוקים לשלושה דברים:

תמונה שמכילה שולחן

התיאור נוצר באופן אוטומטי

את שתי השורות הראשונות יש לנו מסעיף קודם.

1) כעת ניעזר בקוד מהמדריך שציינו בתחילה: image\_input\_fn.py

מיקום בגיט: References/ image\_input\_fn.ipynb

2) יש להגדיר בסקריפט מחדש את השם של ה) Input nodeנלקח מהלוגים של הקפאת המודל בCollab). השורה מסומנת בכתום.

תמונה שמכילה טקסט

התיאור נוצר באופן אוטומטי

3) פקודת להרצת תהליך הקוונטיזציה:



## שלב הקומפילציה

1) פקודה להרצת תהליך הקומפילציה:



הפלט לאחר קומפילציה:

INFO: Output:

quantize\_eval\_model: ./quantize\_results/quantize\_eval\_model.pb

deploy\_model: ./quantize\_results/deploy\_model.pb

(vitis-ai-tensorflow) projecta@ubuntu:/workspace/projecta-mnist$ vai\_c\_tensorflow --frozen\_pb ./quantize\_results/deploy\_model.pb --arch /opt/vitis\_ai/compiler/arch/dpuv2/ZCU104/ZCU104.json --output\_dir model --net\_name mnist\_tf

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* VITIS\_AI Compilation - Xilinx Inc.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

[VAI\_C][Warning] layer [activation\_1\_Sigmoid] (type: Sigmoid) is not supported in DPU, deploy it in CPU instead.

Kernel topology "mnist\_tf\_kernel\_graph.jpg" for network "mnist\_tf"

kernel list info for network "mnist\_tf"

Kernel ID : Name

0 : mnist\_tf\_0

1 : mnist\_tf\_1

Kernel Name : mnist\_tf\_0

--------------------------------------------------------------------------------

Kernel Type : DPUKernel

Code Size : 3.46KB

Param Size : 0.08MB

Workload MACs : 3.08MOPS

IO Memory Space : 5.86KB

Mean Value : 0, 0, 0,

Total Tensor Count : 5

Boundary Input Tensor(s) (H\*W\*C)

conv2d\_input\_1:0(0) : 28\*28\*1

Boundary Output Tensor(s) (H\*W\*C)

conv2d\_3\_1\_Conv2D:0(0) : 1\*1\*10

Total Node Count : 4

Input Node(s) (H\*W\*C)

conv2d\_4\_Conv2D(0) : 28\*28\*1

Output Node(s) (H\*W\*C)

conv2d\_3\_1\_Conv2D(0) : 1\*1\*10

Kernel Name : mnist\_tf\_1

--------------------------------------------------------------------------------

Kernel Type : CPUKernel

Boundary Input Tensor(s) (H\*W\*C)

activation\_1\_Sigmoid:0(0) : 1\*1\*10

Boundary Output Tensor(s) (H\*W\*C)

activation\_1\_Sigmoid:0(0) : 1\*1\*10

Input Node(s) (H\*W\*C)

activation\_1\_Sigmoid : 1\*1\*10

Output Node(s) (H\*W\*C)

activation\_1\_Sigmoid : 1\*1\*10

## שלב הסיווג/הרצה על הBoard

שלב הסיווג הוא השלב שבו אנו בסופו של דבר מריצים את המודל על data אמיתי על גבי הלוח. בסופו של דבר, זאת המטרה הסופית בשימוש כל הכלים של VitisAI ובלוח.

1) ראשית, יש ליצור תיקייה חדשה על גבי הרכיב (שם כללי)

2) יש להיעזר בקובץ app\_mt.py מהמדריך לתוך תיקייה זאת

מיקום בגיט: References/ app\_mt.py

3) בנוסף העברנו תמונות לסיווג ואת קובץ הELF של הקומפילציה (תוצר שלב הקומפילציה)

תמונה שמכילה שולחן

התיאור נוצר באופן אוטומטי

4) כעת ביצענו את הרצת הtest set:

python3 app\_mt.py -m dpu\_mnist\_tf\_0.elf

לשינוי threads להוסיף -t

תמונה שמכילה טקסט

התיאור נוצר באופן אוטומטי

בוצע על מס' threads שונה, 1 ו-6.

**כפי שניתן לראות הצלחנו לייצר מסלול מקיף:**  
התחלנו במודל בפייתון, העברנו אותו אימון ולאחר מכן ביצענו קוונטיזציה. המשכנו לקימפול המודל שיתאים לDPU וללוח שלנו. בהמשך, יצרנו קובץ פייתון שיודע להריץ את שלב הסיווג והרצנו את הסיווג עצמו על הרכיב וקיבלנו דיוק מרשים תוך latency נמוך.

## אופטימיזציה וביצועים

בדיקת ביצועים לאחר שינוי פרמטרים**:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **מבוצע על ידי** | **שלב ארכיטקטוני** | **פרמטר שינוי** | **תיקייה** | **ביצועים** |
| Colab | - | - | - | **Accuracy 98.25%** |
| Board | - | - | **Modifications/Quantization/**  **Original** | **Accuracy 98.19%** |
| Board | Quantization – קוונטיזציה | --weight\_bit  Bit width for quantized weight and bias.  הערך הדיפולטי: 8.  שינוי שביצענו: מעבר ל16. | **Modifications/Quantization/**  **Weight\_bit\_16**  INT8 is only supported for DPU deployment. You can use 16 and 32 bits for simulation, but you won't be able to deploy this on DPU hw. | **ביצועים ירודים כתוצאה מחוסר תמיכה של המערכת**  **הערה – תהליך הקוונטיזציה/קומפילציה עבדו, אך בפועל בזמן הסיווג קיבלנו ביצועים ירודים** |
| Board | Quantization – קוונטיזציה | --activation\_bit  Bit width for quantized activation.  הערך הדיפולטי: 8.  שינוי שביצענו: מעבר ל16. | **Modifications/Quantization/**  **Activation\_bit\_16**  INT8 is only supported for DPU deployment. You can use 16 and 32 bits for simulation, but you won't be able to deploy this on DPU hw. | **ביצועים ירודים כתוצאה מחוסר תמיכה של המערכת**  **הערה – תהליך הקוונטיזציה/קומפילציה עבדו, אך בפועל בזמן הסיווג קיבלנו ביצועים ירודים** |
| Board | Quantization – קוונטיזציה | --calib\_iter  The iterations of calibration. Total number of images for calibration = calib\_iter \* batch\_size. | **Modifications/Quantization/**  **Calib\_Iter\_100** | **Accuracy 98.22%** |
| Board | Quantization – קוונטיזציה | שתי שיטות על פיהן ניתן לקוונטט. שיטה 0 יוצרת טווח רחב של ערכים קוונטיזציה ושיטה 1 מאפשרת טווח צר יותר. | **Modifications/Quantization/**  **Method\_0** | **Accuracy 98.33%** |
| Board | Inference – סיווג | --threads  Number of threads. Default is 1 | **-** | **Accuracy 98.19%** |

ללא שינויים:

תמונה שמכילה טקסט

התיאור נוצר באופן אוטומטי

Method 0

תמונה שמכילה טקסט

התיאור נוצר באופן אוטומטי

Calib:

תמונה שמכילה טקסט

התיאור נוצר באופן אוטומטי

Activation:

תמונה שמכילה טקסט, צילום מסך, צג, מסך

התיאור נוצר באופן אוטומטי

Weight:

תמונה שמכילה טקסט, צילום מסך, צג, מסך

התיאור נוצר באופן אוטומטי

Threads:

תמונה שמכילה טקסט

התיאור נוצר באופן אוטומטי

# טיפים, בעיות ופתרון בעיות

|  |  |  |
| --- | --- | --- |
| **הבעיה** | **מה היא אומרת** | **איך ניתן לתקן** |
| שגיאת core dumped | בתחילה ניסינו ליצור מכונה עם גודל אחסון של כמה עשרות ג'יגה, הגענו במהירות לאחסון מלא. | נתנו למכונה הוירטואלית מרחב אחסון גדול בהרבה של 350 ג'יגה. |
| Virtual box לא תומך בפקודות FMA |  | יצרנו מכונה וירטואלית בVMARE. |
| המכשור לא מסוגל לעבוד עם גרסה tensorflow 2 ומעלה |  | ניתן לבצע המרה בתחילת הקוד בcolab לגרסה ישנה יותר כפי שהראנו בקוד. |
| המכשור לא מסוגל לעבוד עם שכבות FC אחת אחרי השניה. | באופן כללי מתבצעת המרה משכבת FC לשכבת קונבולוציה. כאשר משתמשים בשתי שכבות כאלו אחת אחרי השניה נוצר מצב בו גודל הכניסה לשכבת הקונבולוציה גדול מהגודל המקסימלי ((16x16. | מעבר לארכיטקטורה של שכבות קונבולוציה בלבד כפי שהראנו בקוד שהכנו. |
| שכבת dense softmax נכשלת בשלב הקוונטיזציה | הקוונטייזר לא יודע איך לעבוד עם סוג המשתנה שיוצא משכבה זו (int32) | עברנו לשכבת אקטיביזציה אחרת בשם סיגמואיד. |
| שכבת סיגמואיד לא עובדת על הDPU | מכיוון שהמשתנה היוצא משכבה זו הינו int16 הDPU לא יכול להריץ אותו ולכן הוא מורץ על הCPU. | זוהי אינה בעיה. הDPU לא יכול להריץ את כל סוגי השכבות. אין בעיה שחלק מהקוד יורץ על הCPU. |
| הDPU לא תומך ברוחב ביט שאינו 8 |  | הפתרון הינו כמובן לעבוד ברוחב ביט זה בלבד. |

**תמיכה בסוגי שכבות על גבי הDPU**

תמונה שמכילה שולחן

התיאור נוצר באופן אוטומטי

במהלך בניית הרשת נתקלנו בבעיה עם שכבת Dense Softmax : לא הצלחנו לבצע את שלב הקוונטיזציה. כתוצאה מכך עברנו לעבוד עם שכבת Sigmoid. שכבה זאת יכולה לעבוד רק על גבי הCPU ואינה נתמכת על גבי הDPU. מתוך הטבלה ניתן לראות סוגים שונים של שכבות ולפי העמודה הימנית ביותר (DPU-Support) ניתן לזהות אם היא נתמכת על גבי הDPU או לא.

# תובנות כלליות

**מסקנות שעלו בעקבות תהליך הביצועים והאופטימיזציה**

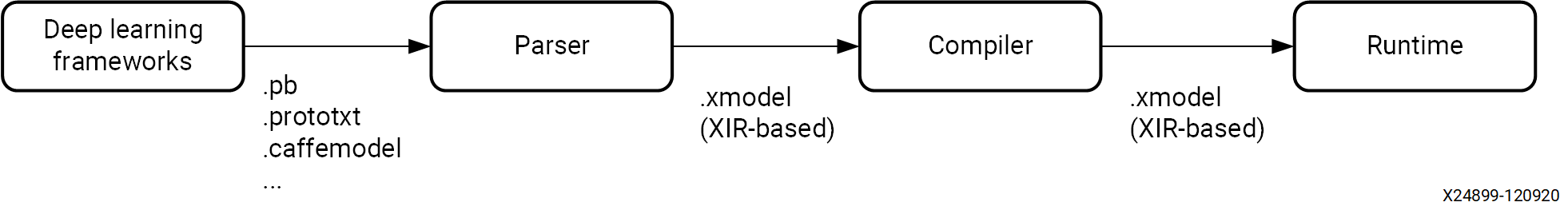
* ביצועים אופטימליים מבחינת סיווג – יש לבצע קוונטיזציה על פי שיטה 0 אשר עובדת עם טווח רחב של ערכים.
* ביצועים אופטימליים מבחינת זמנים – במקרה של עבודה עם הלוח יש לבצע חלוקה למספר חוטים אשר נותן תוצאה אופטימלית. במקרה שלנו זה התקבל עבור 8 חוטים.
* יש לעבוד עם רוחב ביט של 8.
* יש לבצע שינויים בפרמטרים בשלב הקוונטיזציה, בשלב הקומפילציה לא נמצאו פרמטרים רלוונטיים.

**הרצת שכבות על גבי הDPU ועל גבי הCPU**

כפי שניתן לראות בתוצאות הקומפילציה, ישנן שכבות אשר מיועדות לריצה על גבי המעבד (ARM) וחלקן מיועדות לריצה על גבי הDPU. נרצה טיפה להסביר על המניע וכיצד המנגנון הפנימי עובד.

התשתיות בפרויקט מתבססות על פרוטוקול ייצוג XIR - Xilinx Intermediate Representation. זהו פרוטוקול מבית XILINIX של ייצוג גרף לאלגוריתמי AI. הוא נועד להצגה איכותית של רשתות נוירונים ולאחר מכן גם לפרישה נוחה ומהירה על גבי פלטפורמות חומרה שונות (במקרה שלנו DPU על גבי הFPGA).

באופן כללי התהליך נראה כך:



כאשר המודל (רשת הנוירונים הסופית) מכילה פעולות שהDPU אינו יכול לבצע ברמת החומרה, נוצרים Sub-graph לשכבות הספציפיות הללו והם ממופים לביצוע על גבי ה-CPU.   
המידע לגבי איזה שכבות ניתן להריץ על גבי הCPU ואילו לא, מוחלט בשלב הקומפילציה על ידי הקומפיילר. בסופו של דבר, כאשר נעזרים ב- Vitis AI unified high-level APIהמודל שלנו ינסה לרוץ על ה-DPU וכאשר לא יוכל יבצע חלק מהפעולות על גבי הCPU. באופן כללי, הData Flow הוא מן הצורה: ...CPU--DDR--DPU--DDR...

**קימפול ה-RTL לכרטיס (בדיקת היתכנות)**  
  
מתוך המחקר שביצענו, אנו ראינו כי אכן יש אפשרות להצליח לקמפל את קוד הDPU ולהורידו לכרטיס. עם זאת, מדובר בתהליך מורכב, הכולל שלבים רבים של התאמות וכולל בעיות רבות של סנכרון בין גרסאות שונות של תוכנות שונות ושימוש בקבצי תלויות מסוימים. כפי שניתן לראות, המדריך המקורי (מדריך 17) יועד לכרטיס אחר- ZCU-102. אנו מאמינים שבמידה וניתן יהיה להשיג כרטיס זה, עבודת הקימפול והצריבה לכרטיס לאחר מודיפיקציות שונות תתבצע בקלות יותר (אך עדיין מורכבת לאור התלויות המשתנות).

**אחרית דבר**

* במקרים מסוימים, אכן הכרטיס בשילוב סביבת Vitis-AI מצליחים להשיג אחוזי דיוק טובים יותר מאשר מאשר סיווג על גבי מחשב ממוצע וזאת הודות לחומרה הייעודית שלו.
* חברת Xilinx הצליחה להעניק בצורה יחסית פשוטה יכולות של בינה מלאכותית לכל החפץ בכך - גם כאלה בעלי מעט מאוד ניסיון בעולם זה.
* יצירת מודל חדש הינו תהליך מורכב מאוד. לאורך התהליך מהרגע שבו יצרנו מודל חדש ועד להרצתו על גבי הכרטיס, נדרשו מספר רב של התאמות לאורך הדרך.
* לצערנו המידע על תשתיות Xilinx ברשת הינו מוגבל ולעיתים קרובות קשה מאוד להתמודד עם בעיות שצפות במהלך תהליך הפיתוח של מודל חדש.