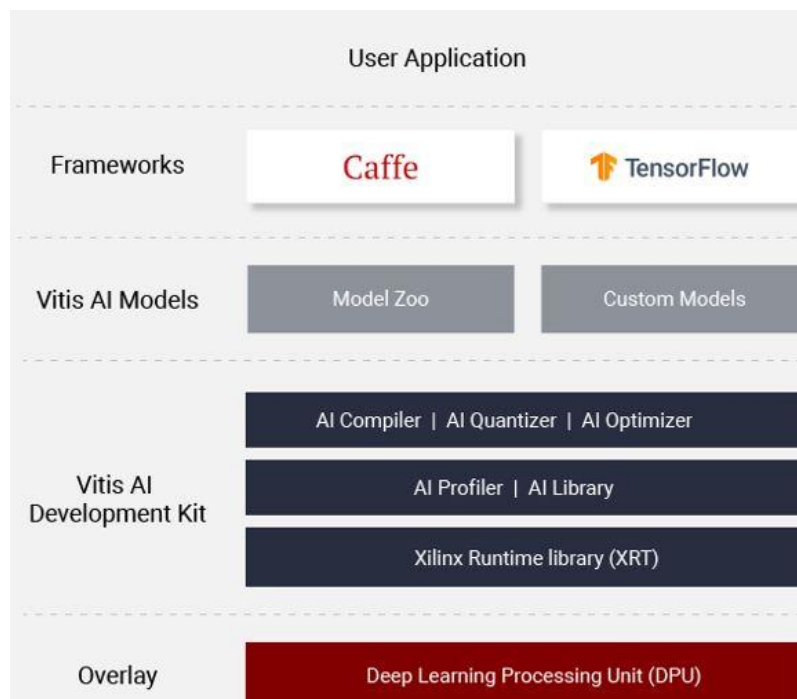


# Guide Book

## Deep Neural Network Inference on FPGA using TensorFlow



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

אביב תשפ"א / מאי 2021

# תוכן עניינים

3.....	מבוא
4.....	רקע תיאורטי
6.....	הקמת עמדה
10.....	צריבת SD-Card
11.....	Setup של כרטיס FPGA
17.....	הקמת רשת, קומפילציה והרצת הרשת: Partial Flow For Resnet-50
17 .....	• תיאור כללי
18 .....	• שלב הקוונטיזציה
21 .....	• שלב הקומפילציה
24.....	הקמת רשת, קומפילציה, אופטימיזציה וסיווג: Full Flow For Mnist
24 .....	• תיאור כללי
25 .....	• שלב הקוונטיזציה
26 .....	• שלב הקומפילציה
29 .....	• שלב הסיווג / הרצה על Boardn
30 .....	• אופטימיזציה וביצועים
33.....	טיפים, בעיות ופתרון בעיות
34.....	תובנות כלליות

## רקע כללי

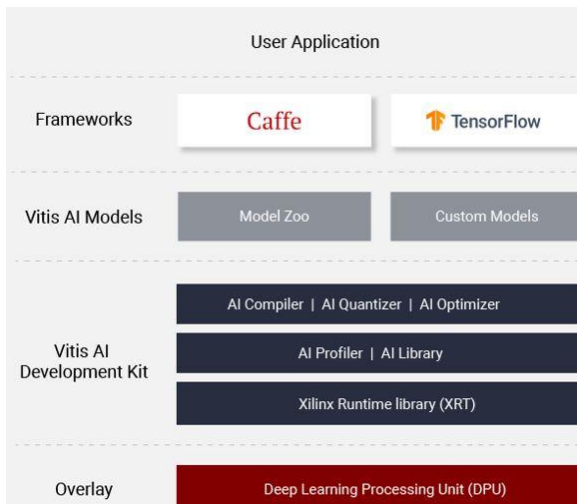
- תחום המערכות הלומדות המבוסס על רשתות נוירונים צובר תאוצה בשנים האחרונות, בעקבות ההצלחה של מערכות אלו בפתרון בעיות מורכבות.
- יישום של רשתות מתבצע ב-2 שלבים – שלב האימון ושלב הסיווג.
- שלב האימון הוא שלב מקדים המתבצע פעם אחת לכל רשת נתונה ולרוב מבוצע על מערכת בעלת עוצמת חישוב גבוהה כגון GPU. כיוון ששלב 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 FPGA של חברת Xilinx.

בניית הרשת תעשה באמצעות TensorFlow

## דרישות הפרויקט:

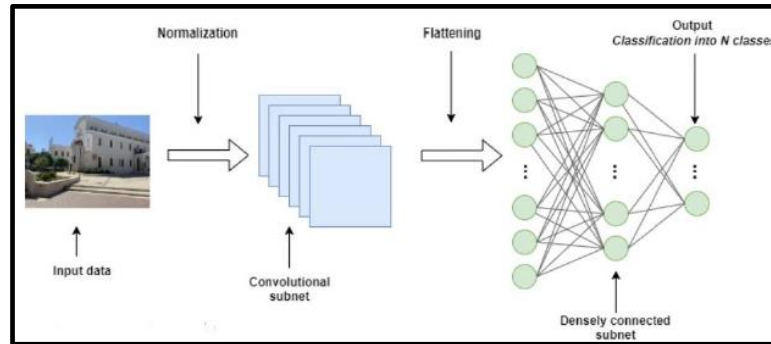


1. הקמת המערכת על מחשב ייעודי.
  2. בחירה והרצה של מודל שאומן מראש תוך שימוש בכל כלי התוכנה של חברת Xilinx.
  3. מימוש סביבת הבדיקה.
  4. ביצוע שינויים והתאמות במודלים במטרה לבדוק את גמישותם ויכולתם של הכלים.
- לדוגמא שינוי מבנה הרשת ( model zoo vs Custom ), טיפוס המשתנים ( FP vs Fixed point ) וכו'.
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.

## רקע תיאורטי

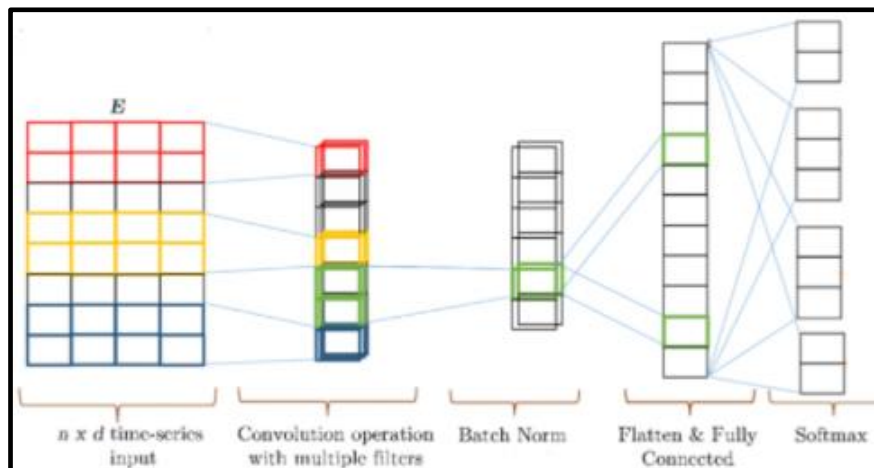
רשת נוירונים או רשת קשרית הוא מודל מתמטי חישובי, שפותח בהשראת תהליכים מוחיים או קוגניטיביים המתרחשים ברשת עצבית טבעית ומשמש במסגרת למידת מכונה. רשת מסוג זה מכילה בדרך כלל מספר רב של יחידות מידע (קלט ופלט) המקושרות זו לזו, קשרים שלעיתים קרובות עוברים דרך יחידות מידע חבויות. שיטה זאת מאוד פופולרית ושימושית בתחומים רבים.

דוגמה לארכיטקטורה של רשת נוירונים:



ארכיטקטורה שימושית בתחומים של עיבוד תמונה היא ארכיטקטורה אשר נקראת convolutional neural network (CNN). ארכיטקטורה זו מבוססת על שכבות הנקראות שכבות קונבולוציה. כל נוירון בשכבה זו מוזן רק מכמות מוגבלת של ערכים הנמצאים בסביבתו הקרובה. כל הנוירונים בשכבה מסוימת זהים, זאת אומרת שהם משתמשים באותם המשקלים (תכונה המכונה weight sharing).

דוגמה לארכיטקטורה של רשת קונבולוציה:



באופן כללי, תהליך של יצירת מודל באמצעות רשת נוירונים מורכב ממספר שלבים:

- יצירת המודל – הגדרת הארכיטקטורה והשכבות
- אימון המודל באמצעות CPU/GPU
- ביצוע אופטימיזציות שונות על המודל המאומן
- ביצוע סיווג של דאטה אמיתי – יכול להתבצע בפלטפורמות שונות כגון: מעבד רגיל, גרפי או חומרה ייעודית. במידה ובוחרים באופציה השלישית (חומרה ייעודית), יש לתכנן וליצור אותה

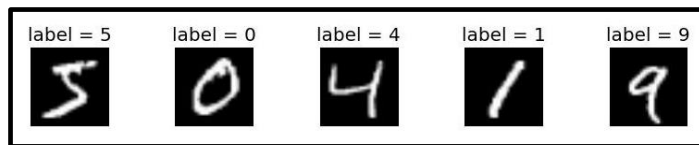
בפרויקט שלנו אנו נרצה להתמקד בשלב האחרון של התהליך.

מחברת Xilinx או מקבלים את סביבת Vitis-AI. סביבה זאת מכילה כלים ומוצרים רבים:

- IP ייעודי שנקרא Deep Learning Processing Unit - DPU. חומרה ייעודית המשמשת להרצת שכבות רלוונטיות לרשתות נוירונים בצורה מהירה ואופטימלית
- AI Zoo - מאגר של מודלים מאומנים בסביבה
- AI Quantizer – מודלים יכולים לעבוד עם משקלים Bias עם ערך ממשי (כלומר double/float). רכיב זה לוקח מודל שכזה וממיר את כל הפרמטרים לייצוג uint (ניתן להגדיר בדיוק כמה ביטים בהרצה)
- AI Compiler – ממדל את רשת הנוירונים לסט פקודות DPU וCPU על גבי הכרטיס ZCU-104

באופן כללי אנו נעבוד עם שני מודלים נפוצים ומוכרים היום בתעשייה.

- המודל הראשון משמש לזיהוי ספרות מתוך מאגר תמונות שנקרא MNIST המכיל 70,000 אלף תמונות בשחור-לבן בגודל  $28 \times 28$ . להלן דוגמה מהמאגר:



- המודל השני נקרא Resnet-50. מודל זה משמש לזיהוי 1000 קטגוריות שונות בתמונות לאחר אימון בהתאם לdataset שהבאנו לו

### מטרות הפרויקט:

הערכת הכלים אשר קיימים בסביבה של Vitis-AI המשמשים להרצת מודלים חישוביים המבוססים רשתות קונבולוציה על גבי חומרה ייעודית. כל אלה כוללים:

- הבנת הסביבה וה-Ecosystem של Vitis-AI
- יצירת תהליך מלא של מעבר מ-TF ל-FPGA
  - שלב 1 – שימוש במודל מאומן מוכן מה-ZOO (מאגר של מודלים מאומנים בסביבה). בחרנו ב-resnet-50
  - שלב 2 – יצירת מודל ואימונו בעצמנו ב-TF והעברתו ל-FPGA
- במהלך כל שלב בדרך נשנה פרמטרים ונתעד את ההשפעות על התהליך מבחינת ביצועים ונכונות
- יצירת מדריך מלא לביצוע כל התרחישים שאנו נבצע הכוללים פירוט נרחב של הבעיות שעולות בדרך ואופן פתרון

## הקמת עמדה

חברת Xilinx מספקת לנו תשתית שלמה להרצת מערך Vitis-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 בBuilda 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 docker שוב.

### פקודות להרצת docker :

```
./docker_run.sh xilinx/vitis-ai-cpu:latest
```

```
conda activate vitis-ai-tensorflow
```

נציין כי אנו נתקלנו ב-Docker Permission denied

```
projecta@projecta-VirtualBox: ~$ docker run hello-world
docker: Got permission denied while trying to connect to the Docker daemon sock
et at unix:///var/run/docker.sock: Post http://%2Fvar%2Frun%2Fdocker.sock/v1.40
/containers/create: dial unix /var/run/docker.sock: connect: permission denied.
See 'docker run --help'.
projecta@projecta-VirtualBox:~$ sudo systemctl restart docker
[sudo] password for projecta:
projecta@projecta-VirtualBox:~$ docker run hello-world
docker: Got permission denied while trying to connect to the Docker daemon sock
et at unix:///var/run/docker.sock: Post http://%2Fvar%2Frun%2Fdocker.sock/v1.40
/containers/create: dial unix /var/run/docker.sock: connect: permission denied.
See 'docker run --help'.
projecta@projecta-VirtualBox:~$ sudo chmod 666 /var/run/docker.sock
projecta@projecta-VirtualBox:~$ docker run hello-world
Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
   (amd64)
3. The Docker daemon created a new container from that image which runs the
   executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it
   to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
$ docker run -it ubuntu bash
```

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:

- Four steps to set up and verify the Vitis AI environment on a host machine
  - Step 1: Install the Docker engine
  - Step 2: Ensure that the Linux user is in the group Docker
  - Step 3: Clone the Vitis AI repository
  - Step 4: Run the Docker container

#### 4. הורדת qBittorrent (לא הכרחי):

במכונה הווירטואלית יש להריץ -

`sudo apt install qbittorrent`

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

## 5. התקנת Petalinux :

כחלק מהפרויקט בדקנו היתכנות של קימפול קוד ה RTL של רכיב ה DPU וצריבתו על ה ZCU-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 ה design 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>

 PetaLinux 2020.2 Installer (TAR/GZIP - 1.85 GB)

MD5 SUM Value : 687b018f7502a4258bd633dc483bde79

 ZCU104 BSP (BSP - 1.35 GB)

MD5 SUM Value : d5d1b32de55f9d67f59e9d5593238884

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 :

```
#User Configuration
#OE_TERMINAL = "tmux"

IMAGE_BOOT_FILES_zynqmp = "BOOT.BIN boot.scr Image rootfs.cpio.gz.u-boot"
IMAGE_INSTALL_remove = "gststreamer-vcu-examples"
CONFIG_PRE_MIRROR_URL="http://petalinux.xilinx.com/sswreleases/rel-v2020/downloads"
CONFIG_YOCTO_NETWORK_SSTATE_FEEDS_URL="http://petalinux.xilinx.com/sswreleases/rel-v2020/aarch64/sstate-cache"
IMAGE_INSTALL_append = "opencv"
```



באופן כללי לאחר יצירת הפרויקט תקבלו את ההודעה הבאה :

```
projecta@ubuntu:~/Vitis-AI/BootExp/Petalinux$ petalinux-create -t project -s ../xilinx-zcu104-v2020.2-final.bsp
INFO: Create project:
INFO: Projects:
INFO: * xilinx-zcu104-2020.2
INFO: Has been successfully installed to /home/projecta/Vitis-AI/BootExp/Petalinux/
INFO: New project successfully created in /home/projecta/Vitis-AI/BootExp/Petalinux/
```

#### בעיות נפוצות:

- במקרה של השגיאה הבאה :

petalinux-create: command not found

יש להיעזר בדיון הבא :

Program/References/PDFs\_Archive/18

- במידה ויש כישלון בתהליך Buildn, 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 Imgn עבור ZCU-104 :

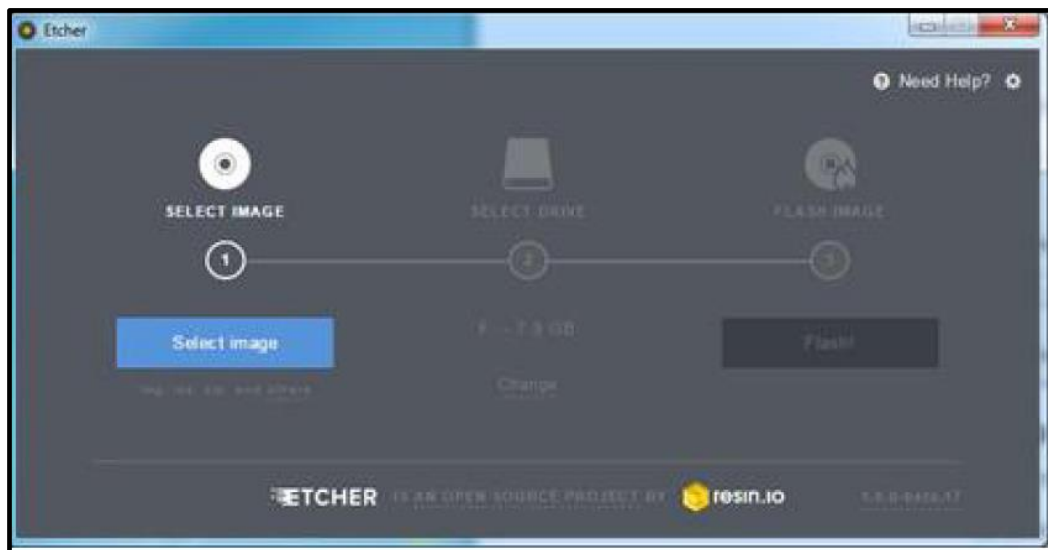
<https://www.xilinx.com/bin/public/openDownload?filename=xilinx-zcu104-dpu-v2020.1-v1.2.0.img.gz>

שימו לב כי מדובר בגרסה : xilinx-zcu104-dpu-v2020.1-v1.2.0.

(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 ה DPU שייצר 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:

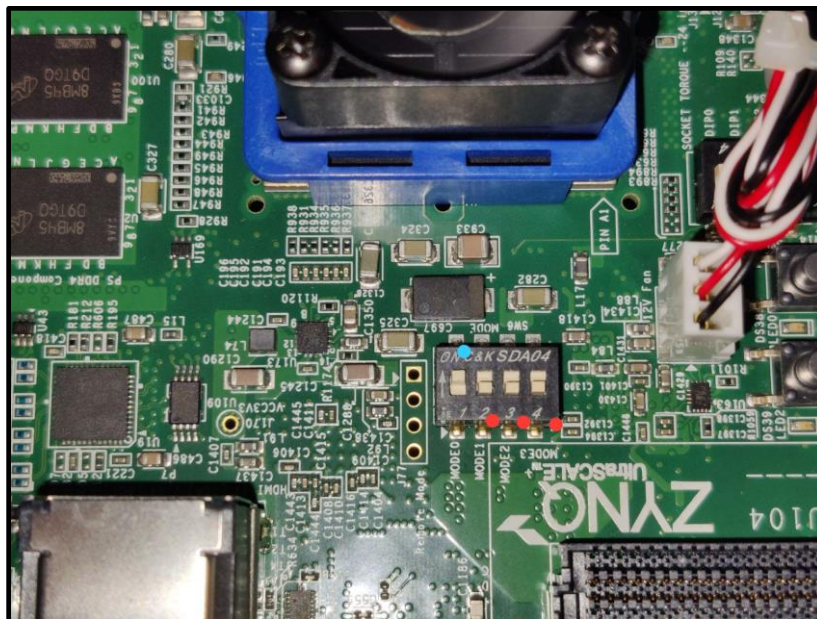
- כבל חשמל (עיגול ירוק)
- כבל Micro-USB – ישמש לתקשורת עם המחשב האישי בפרוטוקול UART (עיגול צהוב)
- כבל Ethernet – ישמש לתקשורת עם העולם החיצוני דרך רשת האינטרנט (עיגול סגול)



### :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).

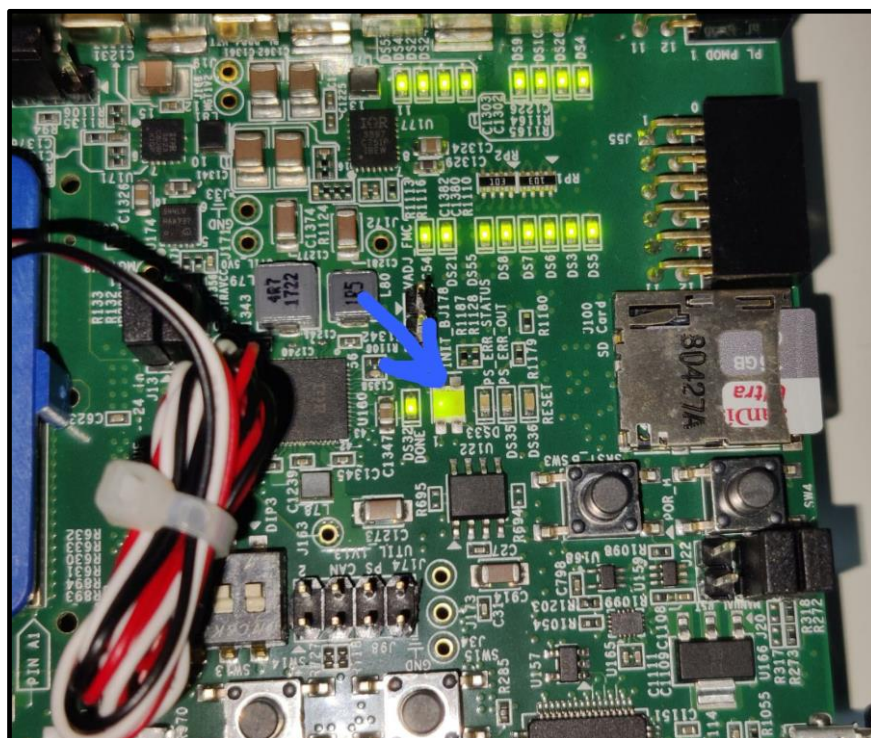


(3) יש לבצע הדלקה של הלוח ולחכות כדקה.

תמונה כללית של הלוח לאחר הדלקה (ZCU104):

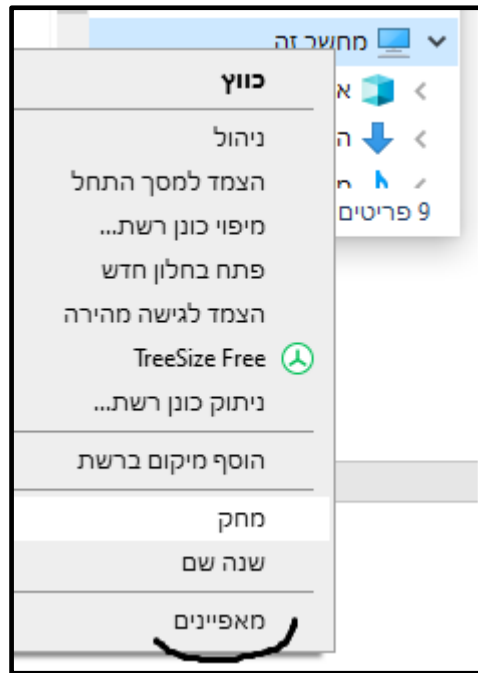


יש לוודא כי הנורה המסומנת דלוקה:



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

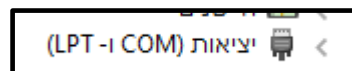
חיבור זה יוצר שלושה חיבורי COM: יש לבחור את הCOM בעל הערך המינימלי. ניתן למצוא את זה תחת מנהל ההתקנים. כדי להגיע למנהל ההתקנים יש ללחץ למחשב שלי ולבחור מאפיינים:



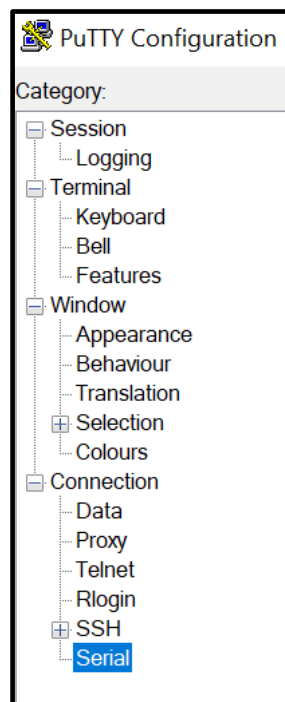
בעמוד שייפתח יש לבחור במנהל ההתקנים :



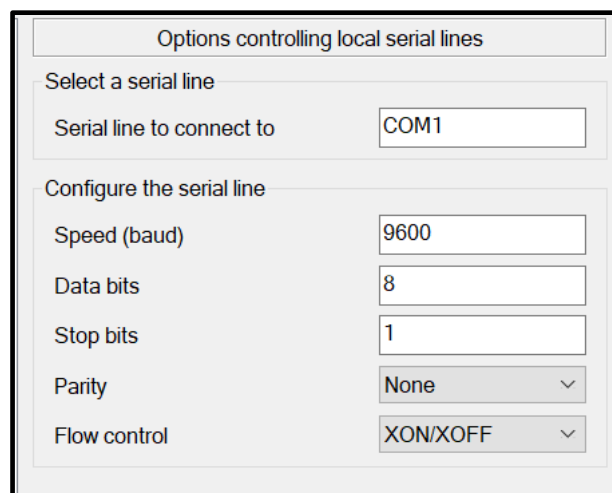
ובדף ההגדרות שייפתח לפתוח את חיבורי COM –



כדי להתחבר לכרטיס נייער 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:

```
COM7 - PuTTY
Xilinx Zynq MP First Stage Boot Loader
Release 2020.1 Jun 22 2020 - 11:05:38
NOTICE: ATF running on XCZU7EV/silicon v4/RTL5.1 at 0xfffea000
NOTICE: BL31: v2.2 (release):v1.1-5588-g5918e656e
NOTICE: BL31: Built : 10:53:46, Jun 22 2020

U-Boot 2020.01 (Jun 22 2020 - 10:56:22 +0000)

Model: ZynqMP ZCU104 RevC
Board: Xilinx ZynqMP
DRAM: 2 GiB
PMUFW: v1.1
EL Level: EL2
Chip ID: zu7ev
NAND: 0 MiB
MMC: mmc@ff170000: 0
In: serial@ff000000
Out: serial@ff000000
Err: serial@ff000000
Bootmode: LVL_SHFT_SD_MODEL
Reset reason: EXTERNAL
Net:
ZYNQ GEM: ff0e0000, mdio bus ff0e0000, phyaddr 12, interface rgmii-id
```

לאחר ההתחברות אפשר לרשום ifconfig ולמצוא את כתובת ה IP של הלוח.

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

```
192.168.1.26 - PuTTY
login as: root
root@192.168.1.26's password:
root@xilinx-zcu104-2020_1:~#
```

### העברת קבצים מול הלוח

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

לחלופין, ניתן להשתמש בפקודות הבאות:

פקודה להעביר תיקייה מהמחשב הלוכלי 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:



פקודת להעברת קובץ:

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

להלן דוגמה להרצה של מודל resnet-50 אשר מגיע "מוכן" על גבי הכרטיס:

```

192.168.1.26 - PuTTY
root@xilinx-zcu104-2020_1:~/Vitis-AI/VART/samples/amitalon_resnet50# clear
root@xilinx-zcu104-2020_1:~/Vitis-AI/VART/samples/amitalon_resnet50# ./resnet50_dpu_resnet50_tf_0.elf
WARNING: Logging before InitGoogleLogging() is written to STDERR
I0701 12:39:33.563982 1134 main.cc:288] create running for subgraph: resnet50_tf_0

Image : ILSVRC2012_val_00000992.JPEG
top[0] prob = 0.866095 name = basset, basset hound
top[1] prob = 0.117213 name = bloodhound, sleuthhound
top[2] prob = 0.003540 name = English setter
top[3] prob = 0.002757 name = German short-haired pointer
top[4] prob = 0.002147 name = Sussex spaniel

Image : ILSVRC2012_val_00001000.JPEG
top[0] prob = 0.302144 name = bakery, bakeshop, bakehouse
top[1] prob = 0.183260 name = china cabinet, china closet
top[2] prob = 0.142723 name = vending machine
top[3] prob = 0.086366 name = refrigerator, icebox
top[4] prob = 0.067418 name = pretzel

Image : ILSVRC2012_val_00000998.JPEG
top[0] prob = 0.786279 name = tank, army tank, armored combat vehicle, armoured combat vehicle
top[1] prob = 0.136635 name = amphibian, amphibious vehicle
top[2] prob = 0.050265 name = aircraft carrier, carrier, flattop, attack aircraft carrier
top[3] prob = 0.023744 name = hair track
top[4] prob = 0.000921 name = warplane, military plane

Image : ILSVRC2012_val_00000989.JPEG
top[0] prob = 0.557833 name = fountain
top[1] prob = 0.263502 name = paintbrush
top[2] prob = 0.094937 name = ear, spike, capitulum
top[3] prob = 0.013119 name = corn
top[4] prob = 0.013119 name = broom

Image : ILSVRC2012_val_00000985.JPEG
top[0] prob = 0.998485 name = street sign
top[1] prob = 0.001501 name = traffic light, traffic signal, stoplight
top[2] prob = 0.000005 name = pole
top[3] prob = 0.000004 name = bakery, bakeshop, bakehouse
top[4] prob = 0.000001 name = book jacket, dust cover, dust jacket, dust wrapper

```

כאשר הרצנו:

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

קובץ ה- C שקומפל resnet50 נלקח מפה:



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

### • תיאור כללי

בשלב זה אנו מריצים מודל שהוכן על ידי Xilinx והיה קיים ב-Zoo. 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 אף ורק את המודל בתצורתו הגולמית (כפי שיצא מ-TF) 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 ה-FPGA. בנוסף הוא מאפשר לבצע מיני אופטימוציות – 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 :

47	tf_resnetv1_50_imagenet_224_224_6.97G_1.2	207MB	55576ff6afdc700ee00664642e19a6fa
----	---	-------	----------------------------------

Resnet-50 הינו מודל AI לסיווג עצמים בתמונות. זהו המודל 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 ה-Img של Docker.

## • שלב הקוונטיזציה

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

No.	Name	Description
1	frozen_graph	Frozen Resnet-50 model.
2	calib_images	Before launching quantization for ResNet-50, prepare the calibration dataset. You can download 100 to 1000 images of ImageNet dataset from <a href="http://academictorrents.com/collection/imagenet-2012">http://academictorrents.com/collection/imagenet-2012</a> or <a href="http://www.image-net.org/download.php">http://www.image-net.org/download.php</a> and then change the calibration dataset path in the input_fn.
3	input_fn	A Python function to read images in the calibration dataset and perform pre-processing (e.g. resize, normalization).

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

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

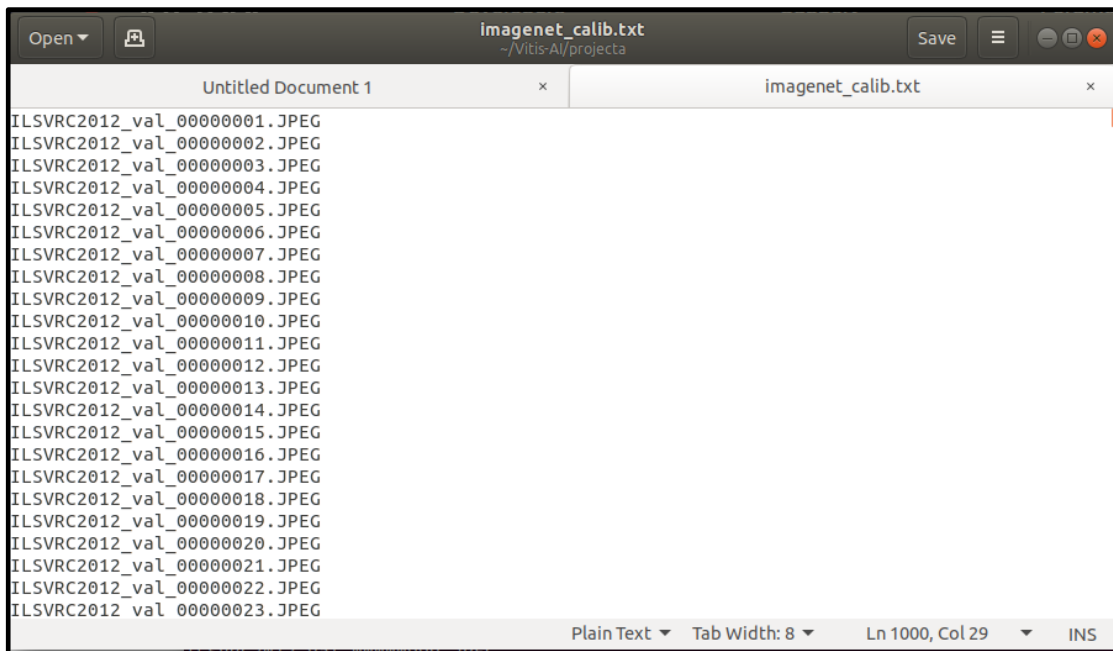
```
import tensorflow as tf
import os

_R_MEAN = 123.68
_G_MEAN = 116.78
_B_MEAN = 103.94

class Data_loader(object):
    def __init__(self, out_height, out_width, smallest_side=256):
        self._sess = tf.Session()
        self._out_height = out_height
        self._out_width = out_width
        self._smallest_side = smallest_side

        self._decode_inpg_data = tf.placeholder(dtype=tf.string)
```

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



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:

```
projecta@ubuntu: /workspace/projecta
File Edit View Search Terminal Help
2020-11-07 02:50:50.280900: W tensorflow/core/framework/cpu_allocator_impl.cc:81] Allocation of 160563
200 exceeds 10% of system memory.
2020-11-07 02:50:50.623561: W tensorflow/core/framework/cpu_allocator_impl.cc:81] Allocation of 160563
200 exceeds 10% of system memory.
2020-11-07 02:50:50.818927: W tensorflow/core/framework/cpu_allocator_impl.cc:81] Allocation of 160563
200 exceeds 10% of system memory.
2020-11-07 02:50:51.604160: W tensorflow/core/framework/cpu_allocator_impl.cc:81] Allocation of 160563
200 exceeds 10% of system memory.
2020-11-07 02:50:51.864837: W tensorflow/core/framework/cpu_allocator_impl.cc:81] Allocation of 802816
00 exceeds 10% of system memory.
INFO: Float Graph Check Done.
2020-11-07 02:50:54.387672: W tensorflow/contrib/decent_q/utils/quantize_utils.cc:538] Convert mean no
de resnet_v1_50/pool5 to AvgPool
2020-11-07 02:50:54.392173: W tensorflow/contrib/decent_q/utils/quantize_utils.cc:628] Scale output of
avg_pool node resnet_v1_50/pool5 to simulate DPU.
INFO: Calibrating for 20 iterations...
100% (20 of 20) |#####| Elapsed Time: 0:22:59 Time: 0:22:59
INFO: Calibration Done.
INFO: Generating Deploy Model...
[DEPLOY WARNING] Node resnet_v1_50/predictions/Reshape_1(Type: Reshape) is not quantized and cannot be
deployed to DPU, because it has unquantized input node: resnet_v1_50/predictions/Softmax. Please deplo
y it on CPU.
INFO: Deploy Model Generated.
***** Quantization Summary *****
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$ ll
```

## 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 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 ה DPU (קוד החומרה).

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

```
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

resnet50\_tf\_0 : 0

resnet50\_tf\_1 : 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 : CPUPKernel

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

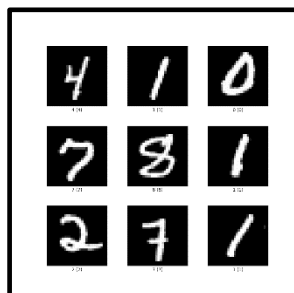
נציין כי קיבלנו תיקיית מודל ולהלן הפירוט שלה:

```
(vitis-ai-tensorflow) projecta@ubuntu:/workspace/projecta/model$ ll
total 25604
drwx----- 2 projecta vitis-ai-users    4096 Nov  7 04:07 ./
drwxr-xr-x  7 projecta          1000    4096 Nov  7 04:06 ../
-rw-r--r--  1 projecta vitis-ai-users 26204736 Nov  7 04:07 dpu_resnet50_tf_0.elf
-rw-r--r--  1 projecta vitis-ai-users    325 Nov  7 04:07 resnet50_tf_kernel_graph.gv
```

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

### • תיאור כללי

בשלב זה אנו יוצרים מודל בצורה מלאה, החל מכתיבתו ב-TF ועד הרצתו 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 Colab 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:

No.	Name
1	frozen_graph
2	calib_images
3	input_fn

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 Input noden (נלקח מהלוגים של הקפאת המודל בCollab).  
השורה מסומנת בכתום.

```
calib_image_list = './calib.txt'
calib_batch_size = 10

def calib_input(iter):
    images = []
    line = open(calib_image_list).readlines()
    for index in range(0, calib_batch_size):
        curline = line[iter * calib_batch_size + index]
        calib_image_name = curline.strip()

        # open image as grayscale
        image = cv2.imread(calib_image_name, cv2.IMREAD_GRAYSCALE)

        # reshape
        image = image.reshape(28,28,1)

        # normalize
        image = image/255.0

        images.append(image)
    return {"flatten input 1": images}
```

(3) פקודת להרצת תהליך הקוונטיזציה :

```
vai_q_tensorflow quantize --input_frozen_graph frozen_graph.pb --input_nodes  
conv2d_input_1 --input_shapes ?,28,28,1 --output_nodes flatten_1/Reshape --input_fn  
image_input_fn.calib_input --method 1 --gpu 0 --calib_iter 10 --output_dir ./quantize_results
```

## • שלב הקומפילציה

(1) פקודה להרצת תהליך הקומפילציה :

```
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
```

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

mnist\_tf\_0 : 0

mnist\_tf\_1 : 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 : CPUIKernel

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 Boardn

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 ELF של הקומפילציה (תוצר שלב הקומפילציה)

/home/root/Vitis-AI/VART/samples/projecta-mnist/					
Name	Size	Changed	Rights	Owner	
..		12/07/2020 07:21:33	rwxr-xr-x	root	
images		12/07/2020 08:17:31	rwxr-xr-x	root	
dpu_mnist_tf_0.elf	87 KB	19/12/2020 17:00:49	rw-r--r--	root	
app_mt.py	6 KB	20/12/2020 12:04:07	rw-r--r--	root	

(4) כעת ביצענו את הרצת test set :

`python3 app_mt.py -m dpu_mnist_tf_0.elf`

לשינוי threads לחוסף -t

```
root@xilinx-zcul04-2020_1:~/Vitis-AI/VART/samples/projecta-mnist# python3 app_mt.py -m dpu_mnist_tf_0.elf
Command line options:
--image_dir : images
--threads   : 1
--model     : dpu_mnist_tf_0.elf
Pre-processing 10000 images...
Starting 1 threads...
AmitAlon log: dpu function
FPS=2898.11, total frames = 10000 , time=3.4505 seconds
Correct: 9819 Wrong: 181 Accuracy: 0.9819
root@xilinx-zcul04-2020_1:~/Vitis-AI/VART/samples/projecta-mnist# python3 app_mt.py -m dpu_mnist_tf_0.elf -t 6
Command line options:
--image_dir : images
--threads   : 6
--model     : dpu_mnist_tf_0.elf
Pre-processing 10000 images...
Starting 6 threads...
AmitAlon log: dpu function
AmitAlon log: dpu function
AmitAlon log: dpu function
AmitAlon log: dpu function
AmitAlon log: dpu function
AmitAlon log: dpu function
FPS=3700.52, total frames = 10000 , time=2.7023 seconds
Correct: 9819 Wrong: 181 Accuracy: 0.9819
root@xilinx-zcul04-2020_1:~/Vitis-AI/VART/samples/projecta-mnist#
```

בוצע על מס' threads שונה, 1 ו-6.

**כפי שניתן לראות הצלחנו לייצר מסלול מקיף:**

התחלנו במודל בפייטון, העברנו אותו אימון ולאחר מכן ביצענו קוונטיזציה. המשכנו לקימפול המודל שיתאים לDPU וללוח שלנו. בהמשך, יצרנו קובץ פייטון שיועד להריץ את שלב הסיווג והרצנו את הסיווג עצמו על הרכיב וקיבלנו דיוק מרשים תוך latency נמוך.

בדיקת ביצועים לאחר שינוי פרמטרים:

מבצע על ידי	שלב ארכיטקטוני	פרמטר שינוי	תיקיה	ביצועים
Colab	-	-	-	Accuracy 98.25% 2.221[Sec] 0.02221 $\left[\frac{\mu Sec}{Pic}\right]$
Board	-	-	Modifications/Quantization/Original	Accuracy 98.19% 3.4864[Sec] 0.034864 $\left[\frac{\mu Sec}{Pic}\right]$
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% 3.4645[Sec] 0.034645 $\left[\frac{\mu Sec}{Pic}\right]$
Board	Quantization – קוונטיזציה	שתי שיטות על פיהן ניתן לקוונט. שיטה 0 יוצרת טווח רחב של ערכים קוונטיזציה ושיטה 1 מאפשרת טווח צר יותר.	Modifications/Quantization/Method_0	Accuracy 98.33% 3.4648[Sec] 0.034648 $\left[\frac{\mu Sec}{Pic}\right]$
Board	– Inference סיווג	--threads  Number of threads. Default is 1	-	Accuracy 98.19% 2.5925[Sec] 0.025925 $\left[\frac{\mu Sec}{Pic}\right]$

ללא שינויים:

```
root@xilinx-zcu104-2020_1:~/Vitis-AI/VART/samples/projecta-mnist# python3 app_mt.py -m dpu_mnist_tf_0.elf
Command line options:
  --image_dir : images
  --threads   : 1
  --model     : dpu_mnist_tf_0.elf
Pre-processing 10000 images...
Starting 1 threads...
AmitAlon log: dpu function
FPS=2868.33, total frames = 10000 , time=3.4864 seconds
Correct: 9819 Wrong: 181 Accuracy: 0.9819
root@xilinx-zcu104-2020_1:~/Vitis-AI/VART/samples/projecta-mnist#
```

Method 0

```
root@xilinx-zcu104-2020_1:~/Vitis-AI/VART/samples/projecta-mnist# python3 app_mt.py -m Method_0/model/dpu_mnist_tf_0.elf
Command line options:
  --image_dir : images
  --threads   : 1
  --model     : Method_0/model/dpu_mnist_tf_0.elf
Pre-processing 10000 images...
Starting 1 threads...
AmitAlon log: dpu function
FPS=2886.14, total frames = 10000 , time=3.4648 seconds
Correct: 9833 Wrong: 167 Accuracy: 0.9833
root@xilinx-zcu104-2020_1:~/Vitis-AI/VART/samples/projecta-mnist#
```

Calib:

```
root@xilinx-zcu104-2020_1:~/Vitis-AI/VART/samples/projecta-mnist# python3 app_mt.py -m Calib_Iter_100/model/dpu_mnist_tf_0.elf
Command line options:
  --image_dir : images
  --threads   : 1
  --model     : Calib_Iter_100/model/dpu_mnist_tf_0.elf
Pre-processing 10000 images...
Starting 1 threads...
AmitAlon log: dpu function
FPS=2885.50, total frames = 10000 , time=3.4656 seconds
Correct: 9822 Wrong: 178 Accuracy: 0.9822
root@xilinx-zcu104-2020_1:~/Vitis-AI/VART/samples/projecta-mnist#
```

: Activation

```
root@xilinx-zcu104-2020_1:~/Vitis-AI/VART/samples/projecta-mnist# python3 app_mt.py -m Activation_bit_16/model/dpu_mnist_tf_0.elf
Command line options:
  --image_dir : images
  --threads   : 1
  --model     : Activation_bit_16/model/dpu_mnist_tf_0.elf
Pre-processing 10000 images...
Starting 1 threads...
AmitAlon log: dpu function
FPS=2841.37, total frames = 10000 , time=3.5194 seconds
Correct: 980 Wrong: 9020 Accuracy: 0.098
root@xilinx-zcu104-2020_1:~/Vitis-AI/VART/samples/projecta-mnist#
```

: Weight

```
root@xilinx-zcu104-2020_1:~/Vitis-AI/VART/samples/projecta-mnist# python3 app_mt.py -m Weight_bit_16/model/dpu_mnist_tf_0.elf
Command line options:
  --image_dir : images
  --threads   : 1
  --model     : Weight_bit_16/model/dpu_mnist_tf_0.elf
Pre-processing 10000 images...
Starting 1 threads...
AmitAlon log: dpu function
FPS=2885.55, total frames = 10000 , time=3.4655 seconds
Correct: 980 Wrong: 9020 Accuracy: 0.098
root@xilinx-zcu104-2020_1:~/Vitis-AI/VART/samples/projecta-mnist#
```

```

root@xilinx-zcu104-2020_1:~/Vitis-AI/VART/samples/projecta-mnist# python3 app_mt.py -m dpu_mnist_tf_0.elf
Command line options:
--image_dir : images
--threads   : 1
--model     : dpu_mnist_tf_0.elf
Pre-processing 10000 images...
Starting 1 threads...
FPS=2884.90, total frames = 10000 , time=3.4663 seconds
Correct: 9819 Wrong: 181 Accuracy: 0.9819
root@xilinx-zcu104-2020_1:~/Vitis-AI/VART/samples/projecta-mnist# python3 app_mt.py -m dpu_mnist_tf_0.elf -t 3
Command line options:
--image_dir : images
--threads   : 3
--model     : dpu_mnist_tf_0.elf
Pre-processing 10000 images...
Starting 3 threads...
FPS=3803.81, total frames = 10000 , time=2.6289 seconds
Correct: 9819 Wrong: 181 Accuracy: 0.9819
root@xilinx-zcu104-2020_1:~/Vitis-AI/VART/samples/projecta-mnist# python3 app_mt.py -m dpu_mnist_tf_0.elf -t 6
Command line options:
--image_dir : images
--threads   : 6
--model     : dpu_mnist_tf_0.elf
Pre-processing 10000 images...
Starting 6 threads...
FPS=3709.51, total frames = 10000 , time=2.6958 seconds
Correct: 9819 Wrong: 181 Accuracy: 0.9819
root@xilinx-zcu104-2020_1:~/Vitis-AI/VART/samples/projecta-mnist# python3 app_mt.py -m dpu_mnist_tf_0.elf -t 7
Command line options:
--image_dir : images
--threads   : 7
--model     : dpu_mnist_tf_0.elf
Pre-processing 10000 images...
Starting 7 threads...
FPS=3816.50, total frames = 10000 , time=2.6202 seconds
Correct: 9819 Wrong: 181 Accuracy: 0.9819
root@xilinx-zcu104-2020_1:~/Vitis-AI/VART/samples/projecta-mnist# python3 app_mt.py -m dpu_mnist_tf_0.elf -t 8
Command line options:
--image_dir : images
--threads   : 8
--model     : dpu_mnist_tf_0.elf
Pre-processing 10000 images...
Starting 8 threads...
FPS=3857.26, total frames = 10000 , time=2.5925 seconds
Correct: 9819 Wrong: 181 Accuracy: 0.9819
root@xilinx-zcu104-2020_1:~/Vitis-AI/VART/samples/projecta-mnist# python3 app_mt.py -m dpu_mnist_tf_0.elf -t 9
Command line options:
--image_dir : images
--threads   : 9
--model     : dpu_mnist_tf_0.elf
Pre-processing 10000 images...
Starting 9 threads...
FPS=3809.07, total frames = 10000 , time=2.6253 seconds
Correct: 9819 Wrong: 181 Accuracy: 0.9819
root@xilinx-zcu104-2020_1:~/Vitis-AI/VART/samples/projecta-mnist#

```



## טיפים, בעיות ופתרון בעיות

הבעיה	מה היא אומרת	איך ניתן לתקן
שגיאת core dumped	בתחילה ניסינו ליצור מכונה עם גודל אחסון של כמה עשרות גיגה, הגענו במהירות לאחסון מלא.	נתנו למכונה הוירטואלית מרחב אחסון גדול בהרבה של 350 גיגה.
Virtual box לא תומך בפקודות FMA		יצרנו מכונה וירטואלית ב-VMARE.
המכשור לא מסוגל לעבוד עם גרסה 2 tensorflow ומעלה		ניתן לבצע המרה בתחילת הקוד ב colab 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	הקוונטיזציה לא יודע איך לעבוד עם סוג המשתנה שיוצא משכבת זו (int32)	עברנו לשכבת אקטיביזציה אחרת בשם סיגמואיד.
שכבת סיגמואיד לא עובדת על ה DPU	מכיוון שהמשתנה היוצא משכבת זו הינו int16 ה DPU לא יכול להריץ אותו ולכן הוא מורץ על ה CPU.	זוהי אינה בעיה. ה DPU לא יכול להריץ את כל סוגי השכבות. אין בעיה שחלק מהקוד יורץ על ה CPU.
ה DPU לא תומך ברוחב ביט שאינו 8		הפתרון הינו כמובן לעבוד ברוחב ביט זה בלבד.

## תמיכה בסוגי שכבות על גבי ה DPU

Operators	Framework	Parameters	DPU Support
Const	Tensorflow	-	Arbitrary
Shape	Tensorflow	-	Arbitrary
Identity	Tensorflow	-	Arbitrary
Batchnorm+	Caffe	-	Arbitrary
Neg*	Tensorflow	-	Partially
Mul*	Tensorflow	-	Partially
Sub*	Tensorflow	-	Partially
Gettiling*	Caffe	reverse, stride	Partially
Permute*	Caffe	order	Partially
Flatten*	Caffe/TensorFlow	start_dim, end_dim	Partially
Squeeze*	Tensorflow	dims	Partially
Reshape*	Tensorflow	shape	Partially
Stack*	Tensorflow	axis	Partially
Matmul*	Tensorflow	transpose_a, transpose_b	Partially
Strided_Slice*	Tensorflow	begin, end, strides, begin_mask, end_mask, ellipsis_mask, new_axis_mask, shrink_axis_mask	Partially
Mean*	Tensorflow	dims, keep_dims	Avgpool-like configurations
Resize*	Tensorflow	scale, align_corners, mode	scale = 2, false, NEAREST
Pad*	Tensorflow	pad, pad_mode, constant_value	"Constant" and pad with 0, "SYMMETRIC"
Resize_nearest*	Tensorflow	align_corners	False
DeepPhiResize*	Caffe	scale, mode	Scale = 2, NEAREST
Upsample2D**	Tensorflow	align_corners	-
Resize_bilinear**	Tensorflow	align_corners	-
Space_to_batch**	Tensorflow	block_shape, Paddings	-
Batch_to_space**	Tensorflow	block_shape, Paddings	-
Prior_box**	Caffe	-	-
Softmax**	Tensorflow	axis	-

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 ה CPU ואינה נתמכת על גבי ה DPU. מתוך הטבלה ניתן לראות סוגים שונים של שכבות ולפי העמודה הימנית ביותר (DPU-Support) ניתן לזהות אם היא נתמכת על גבי ה DPU או לא.

## תובנות כלליות

### מסקנות שעלו בעקבות תהליך הביצועים והאופטימיזציה

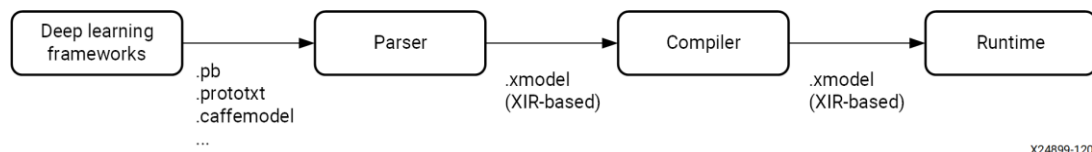
- ביצועים אופטימליים מבחינת סיווג – יש לבצע קוונטיזציה על פי שיטה 0 אשר עובדת עם טווח רחב של ערכים.
- ביצועים אופטימליים מבחינת זמנים – במקרה של עבודה עם הלוח יש לבצע חלוקה למספר חוטים אשר נותן תוצאה אופטימלית. במקרה שלנו זה התקבל עבור 8 חוטים.
- יש לעבוד עם רוחב ביט של 8.
- יש לבצע שינויים בפרמטרים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

### הרצת שכבות על גבי DPU ועל גבי CPU

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 API המודל 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.