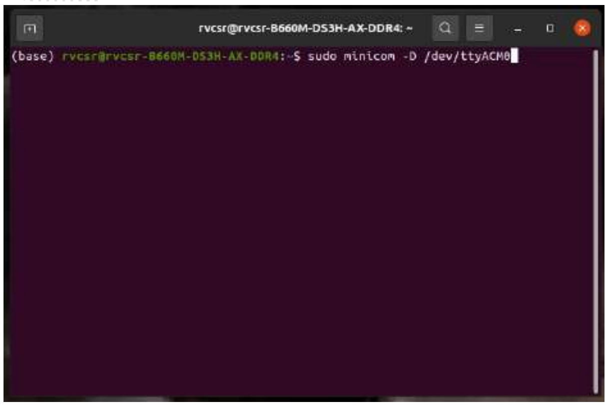
Assignment 3 TinyEngine

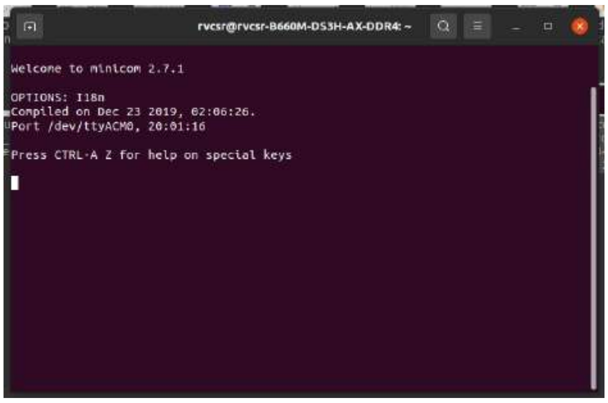
Rahul Vigneswaran K, CS23MTECH02002 Nikhil Kumar Patel, CS23MTECH11013

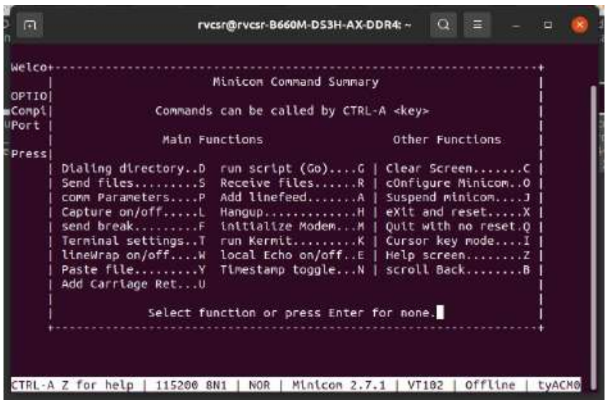
Subject: Hardware Architecture for Deep Learning (CS6490)

Snapshots of steps

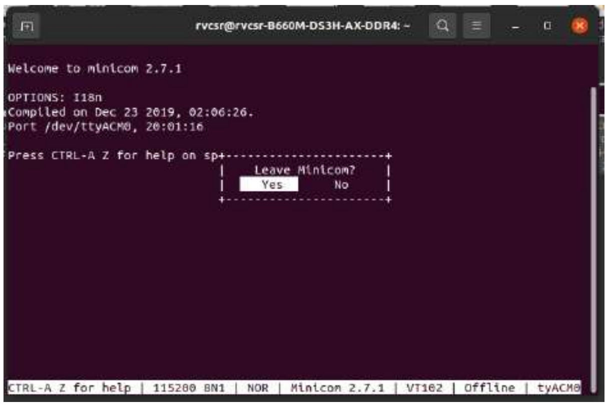
We've closely followed the instructions provided in the TinyEngine GitHub repository, ensuring that our implementation matches it closely. Consequently, we've included snapshots specifically focusing on the UART communication part during on-device training, a section that wasn't covered in the original tutorial.

- Using `minicom`
  - 

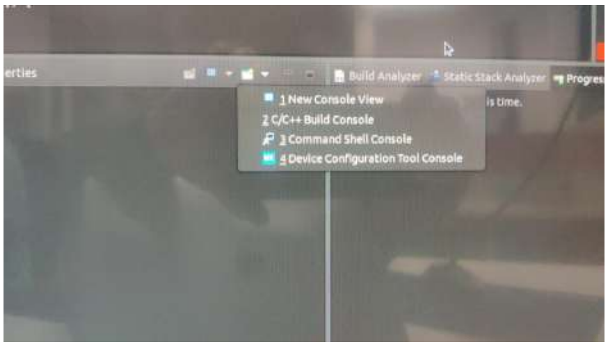
    ```
    rvcsr@rvcsr-B660M-DS3H-AX-DDR4: ~
    (base) rvcsr@rvcsr-B660M-DS3H-AX-DDR4:~$ sudo minicom -D /dev/ttyACM0
    ```
  - 

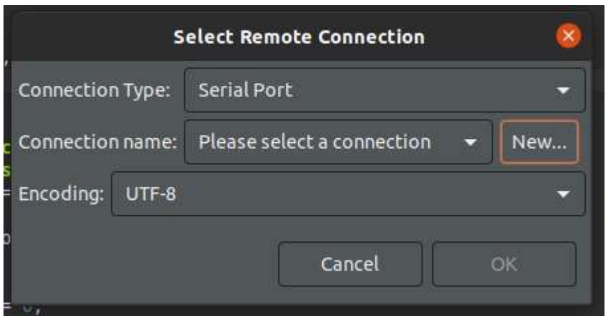
    ```
    rvcsr@rvcsr-B660M-DS3H-AX-DDR4: ~
    Welcome to minicom 2.7.1

    OPTIONS: I18n
    Compiled on Dec 23 2019, 02:06:26.
    Port /dev/ttyACM0, 20:01:16

    Press CTRL-A Z for help on special keys
    ```
    - Now press 3 to go into training mode. Then 1 and 2 for ground truths.
    - Press 4 to go into inference mode.
    - Press CTRL-A Z to go into the menu from which we can exit.
  - 

    ```
    rvcsr@rvcsr-B660M-DS3H-AX-DDR4: ~
    Welco+-----------------------------------------------------------------+
         |                    Minicom Command Summary                      |
    OPTIO|                                                                 |
    Compi|              Commands can be called by CTRL-A <key>             |
    Port |                                                                 |
         |              Main Functions                  Other Functions    |
    Press|                                                                 |
         | Dialing directory..D  run script (Go)....G | Clear Screen.......C |
         | Send files.........S  Receive files......R | cOnfigure Minicom..O |
         | comm Parameters....P  Add linefeed.......A | Suspend minicom....J |
         | Capture on/off.....L  Hangup.............H | eXit and reset.....X |
         | send break.........F  initialize Modem...M | Quit with no reset.Q |
         | Terminal settings..T  run Kermit.........K | Cursor key mode....I |
         | lineWrap on/off....W  local Echo on/off..E | Help screen........Z |
         | Paste file.........Y  Timestamp toggle...N | scroll Back........B |
         | Add Carriage Ret...U                                            |
         |                                                                 |
         |           Select function or press Enter for none.              |
         +-----------------------------------------------------------------+

    CTRL-A Z for help | 115200 8N1 | NOR | Minicom 2.7.1 | VT102 | Offline | tyACM0
    ```
    - Press X to exit.

- 

```
rvcsr@rvcsr-B660M-DS3H-AX-DDR4: ~

Welcome to minicom 2.7.1

OPTIONS: I18n
Compiled on Dec 23 2019, 02:06:26.
Port /dev/ttyACM0, 20:01:16

Press CTRL-A Z for help on sp+---------------------+
                             |    Leave Minicom?   |
                             |   Yes         No    |
                             +---------------------+

CTRL-A Z for help | 115200 8N1 | NOR | Minicom 2.7.1 | VT102 | Offline | tyACM0
```

- Using STM32CubeIDE
  - 

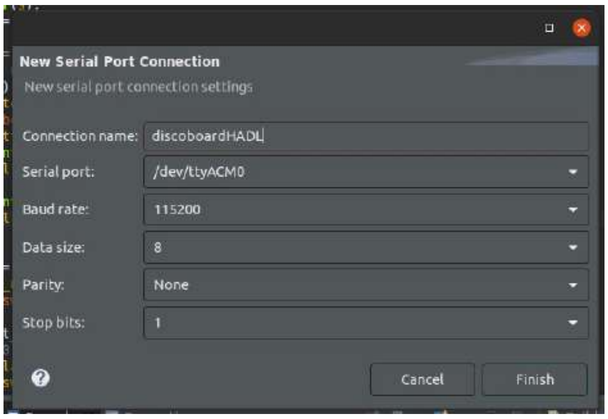
    - Press 3 Command shell console
  - 

    Select Remote Connection

    Connection Type: Serial Port

    Connection name: Please select a connection [New...]

    Encoding: UTF-8

    Cancel OK
    - Set connection type to serial port. Click new for connection name.
  - 

    New Serial Port Connection

    New serial port connection settings

    Connection name: discoboardHADL

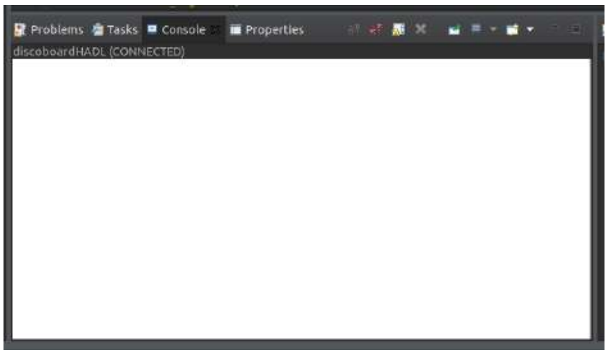
    Serial port: /dev/ttyACM0

    Baud rate: 115200

    Data size: 8

    Parity: None

    Stop bits: 1

    Cancel Finish
    - Give any connection name.
    - Make sure the serial port is set to `/dev/ttyACM0` and Baud rate is set to `115200`.
  - 

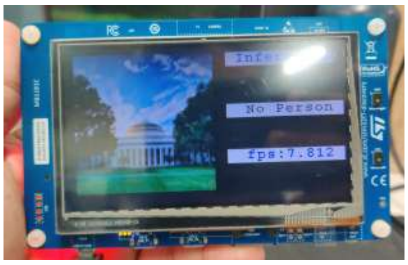
    discoboardHADL (CONNECTED)
    - Now press 3 to go into training mode. Then 1 and 2 for ground truths.
    - Press 4 to go into inference mode.

- Press the "X" symbol on top to close the console.

Snapshots of demo

Inference

- Without Arducam
  - Person | FPS : 7.812
    -

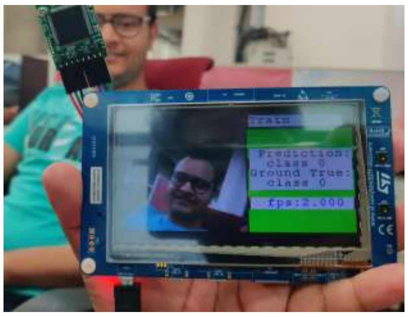
  - No Person | FPS : 7.812
    - 

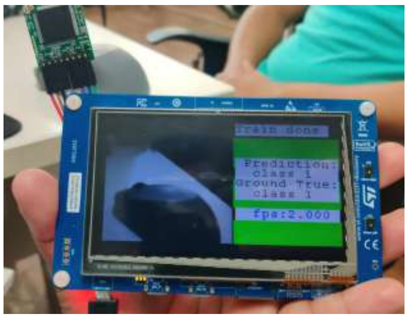
- With Arducam
  - Person | FPS : 6.211
    -

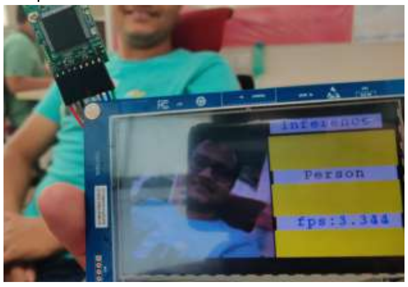
  - No Person | FPS : 6.250
    -

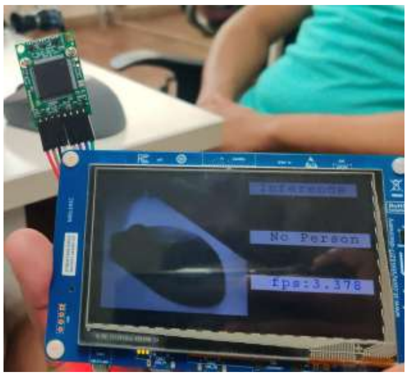
On-device Training

- Training
  - Ground truth for class 0 | FPS : 2.000

    - 

    - Ground truth for class 1 | FPS : 2.000
    - 

- Inference based on on-device training
    - Class 1 | FPS : 3.344
    - 

    - Class 2 | FPS : 3.378
    - 

Issues we encountered

- Ubuntu issue
    - We faced issues with JAVA versions.
        - This was because, initially we were using a non-LTS version of Ubuntu. So we switched to LTS (20.04.6) version of Ubuntu and that fixed it.
- MacOS issue
    - We were not able to install python version 3.6 on M1 mac as its not compatible with arm based chips.
        - Instead we used the latest version and this didn't affect the training in any way.
    - We got a lot of initialisation errors while building on STM32CubeIDE. We were able to fix some of them but not everything. These issues seem to be isolated to MacOS. So we had to switch back to Ubuntu.
- On-device training issue
    - We were not able to figure out how to send commands to the board.
        - After lots of trial and error, we were able to use `minicom` to send commands to the board with the help of this blog.
        - Upon discussing with other groups, we came to know that this can also be done natively inside the IDE itself and was able to replicate it using that method too.

Lessons we learnt based on our interaction with real hardware

- Python 3.6 is not compatible with ARM-based computer chips.
- Initially, we thought we had to SSH into the board to provide the training command. However, after discussing with the course instructor, we understood that the board would not have an IP address, as running a network service would consume unnecessary RAM and was not needed. So we had to use the UART serial communication protocol.
- When we received the board, we discovered a collection of pre-flashed tiny games and apps. But after flashing our custom code onto the board, the previous applications disappeared. This initially caused some concern among us. However, we soon realised that the memory capacity of the chip is limited. Consequently, when new code is flashed onto it, the previous data is overwritten, resulting in the disappearance of the old applications. This realisation underscored the importance of managing memory resources efficiently and highlighted the inherent constraints of working with hardware with limited storage capacity.
- Inference based on models trained on the device often have much lower frames per second (FPS) compared to the model trained separately and uploaded onto the board.
- We learnt that it is necessary to match BAUD rate between the transmitter and the receiver else the data integrity would be affected.
- Before this project, we didn't have much idea about UART (Universal Asynchronous Receiver/Transmitter). During the process, we learned that UART is a set of rules, for exchanging serial data between two devices.
- Open-source code bases don't always work perfectly, and there might be some steps missing in the documentation or instructions.
- It's very easy to plug the wrong wires and potentially damage the board. The components are delicate, so they must be handled with care.
- This experience made us appreciate the capabilities of the mobile phones we have at hand, which are much more compact and powerful compared to the board we were working with.