Assignment 4 TinyML follow-up

Rahul Vigneswaran K, CS23MTECH02002

Nikhil Kumar Patel, CS23MTECH11013

Subject: Hardware Architecture for Deep Learning (CS6490)

Reader's Notes:

- The actual assignment comprises only the first 5 pages. The subsequent pages contain the **appendix** section, which provides additional details for reference.

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 only the loading of different pre-trained models.

- For non-patchwise models, we utilized the `example/vww.py` code and simply modified the `net_id` parameter.

```
import os
from tqdm import tqdm
import sys
from code_generator.CodegenUtilTFlite import GenerateSourceFilesFr
from mcunet.mcunet.model_zoo import download_tflite, net_id_list
print(net_id_list)

# net_id_list = ["mcunet-vww1"]
for model in tqdm(net_id_list):
    print(f"Running for {model}:")
    print("="*20)
    # 1: Let's first build our pretrained VWW model
    # 2: To deploy the model on MCU, we need to first convert the
    # get the weight parameters and scale parameters.
    tflite_path = download_tflite(net_id=model)
```

- Similarly, for patchwise models, we employed the `example/vww_patchbased.py` code and adjusted only the `net_id` parameter.

```
from code_generator.CodeGenerator import CodeGenerator
from code_generator.GeneralMemoryScheduler import GeneralMemoryScheduler
from code_generator.InputResizer import PatchResizer
from code_generator.PatchBasedUtil import getPatchParams
from code_generator.TfliteConvertor import TfliteConvertor
from mcunet.mcunet.model_zoo import download_tflite, net_id_list

for model in tqdm(net_id_list):
    print(f"Running for {model}:")
    print("="*20)
    # 1: Let's first build our pretrained VWW model
    # 2: To deploy the model on MCU, we need to first convert the model to
    # get the weight parameters and scale parameters.
    tflite_path = download_tflite(net_id=model)
```

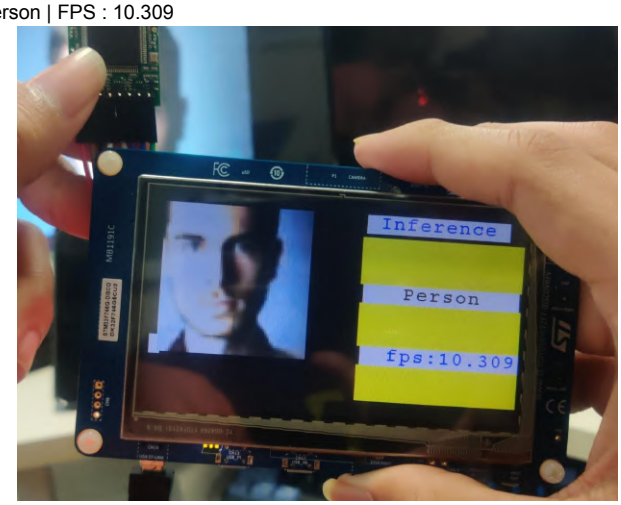
  - Additionally, we made several other modifications to the code to accommodate patchwise implementation, which we elaborated on in the **Issues we encountered** section.
- Furthermore, we automated the repetitive task of downloading pre-trained models and organizing files for building. Once the submission deadline has passed, we will upload the code to this repo.

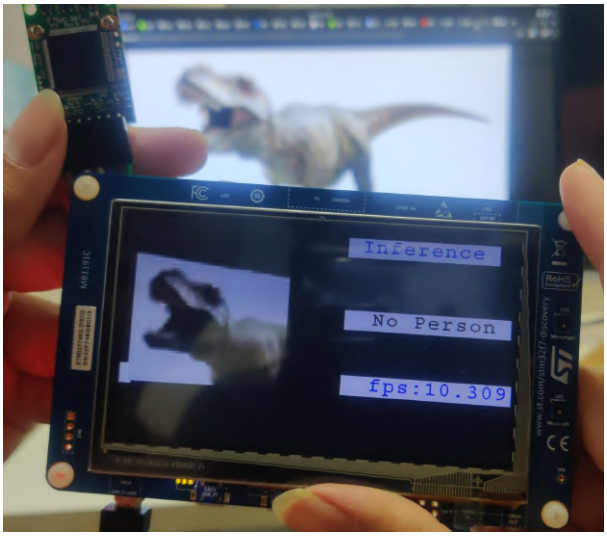
Snapshots of demo

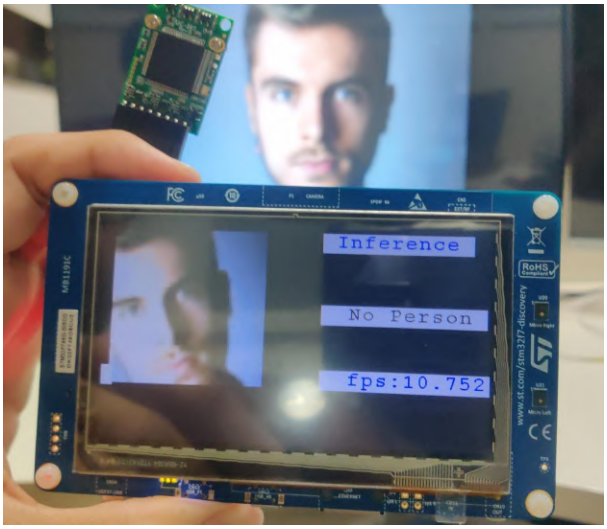
In this section, we have only shared snapshots for `mcunet-vww0` for both patchwise and non-patchwise as an example. For snapshots of all the other setups, check the **Appendix** section.

Credits : We have used the following images as test cases for all our setups.

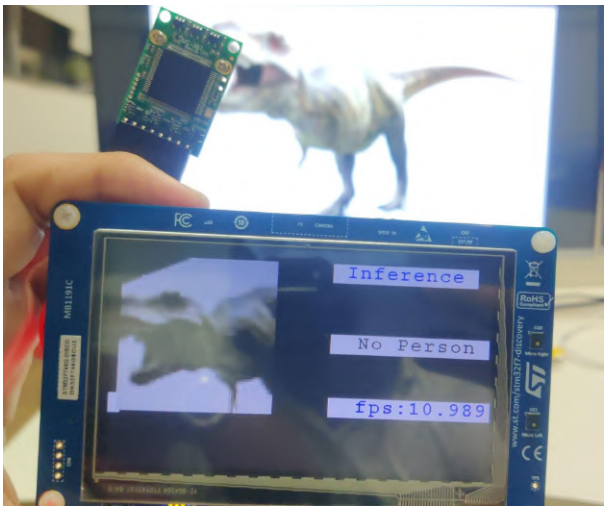
- Person

- No Person
- Non-Patchwise
  - mcunet-vww0
    - Person | FPS : 10.309
      - 

    - No Person | FPS : 10.309
      - 

- Patchwise
  - mcunet-vww0
    - Person | FPS : 10.752
      - 

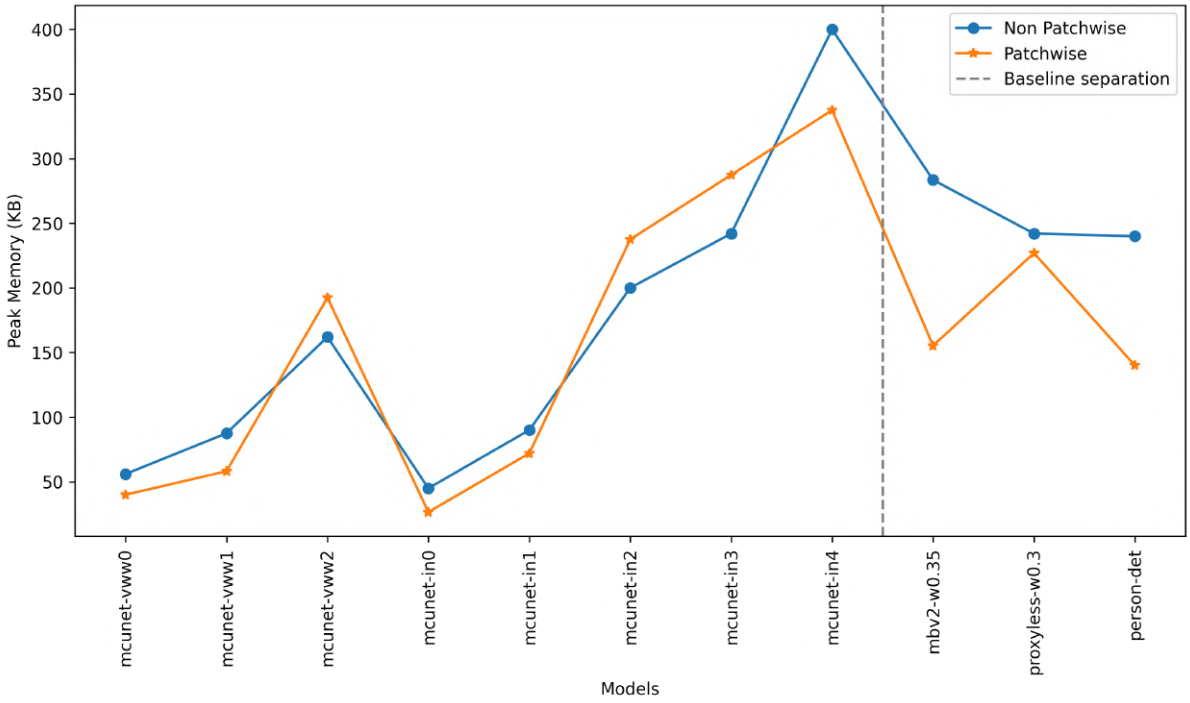
    - No Person | FPS : 10.989

- 

Issues we encountered

For a much more detailed log and observations check the **Appendix** section.

- `mcunet-in4` fails to build because of flash overflow.
  - Upon discussing with our course instructor, we learnt that this might be due to the memory limitation of the STM32F746 Discovery board we are using and a higher variant would fix this build fail issue.
- For models with original resolution > 100, we encounter a black screen instead of the prediction result.
  - This could be happening because the `loadRGB565LCD` function in the `main.cpp` is trying to fit all these pixels in the display.
  - We tried to reduce the `resize` factor in the `loadRGB565LCD` function but the issue persists.
  - As a temporary workaround, we have currently used half the resolution but it doesn't look like the correct way to do it.
  - ⭐ We have raised an issue in the official repo - #103
- All the ImageNet variants of MCUNet were either spotty in their prediction or straight up gave the same prediction for everything.
  - After several discussions with the course instructor and other groups, we concluded that this might be because these models were pre-trained on objects like dog, cat, apple, etc. and not for person prediction. But the official repo doesn't provide any `.cpp` code for detecting such objects. So forcing a model to predict person while it is not trained to do so is the reason for such irregular predictions.
  - ⭐ We have raised an issue in the official repo - #105.
- For patchwise inference we faced the following build errors :
  - "`undefined reference to arm_nn_mat_mult_kernel_s8_s16_reordered`"
    - We found an existing issue in the repo that address this by moving some files around but we found that a much simpler workaround is renaming the function `arm_nn_mat_mult_kernel_s8_s16_reordered` to `arm_nn_mat_mult_kernel_s8_s16_reordered_8mul`.
    - ⭐ We have updated this much simpler workaround in that issue for others to benefit from - #102 .
  - "`undefined reference to arm_nn_requantize`"
    - With the help of the workaround mentioned in an existing issue, we added the imports `#include "arm_nnfunctions.h"` and `#include "arm_nnsupportfunctions.h"` to the file `TinyEngine/include/tinyengine_function.h`.
- For patchwise inference, the predicted output of both the person and no person remains the same. That is, some configs predict both as person and some configs predict both as no person. According to the Tiny Engine, the accuracy of the model should increase when doing patchwise inference but here we see a degradation in performance.
  - We found an existing issue in the repo that points out the same. We built upon their insights and explored additional strategies like below, yet encountered persistent issues.
    - Experimented with diverse patch grid configurations.
    - Set the 'inplace' parameter to False.
  - While these adjustments occasionally impact prediction outcomes, the issue of classifying 'person' and 'no person' as the same remains consistent.
  - ⭐ We have updated these new explorations in that issue for others to benefit from - #102.
- While the Tiny Engine suggests that patchwise inference generally decreases peak memory usage across various models, our experimentation reveals a more nuanced reality. Illustrated in the figure below are the peak memory comparisons between patchwise and non-patchwise inference across different models. Notably, models such as `mcunet-vww2`, `mcunet-in2`, and `mcunet-in3` exhibit higher peak memory usage with patchwise inference.



- Despite thorough examination and consultation with our course instructor, we have yet to identify a discernible trend or explanation for this inconsistency.

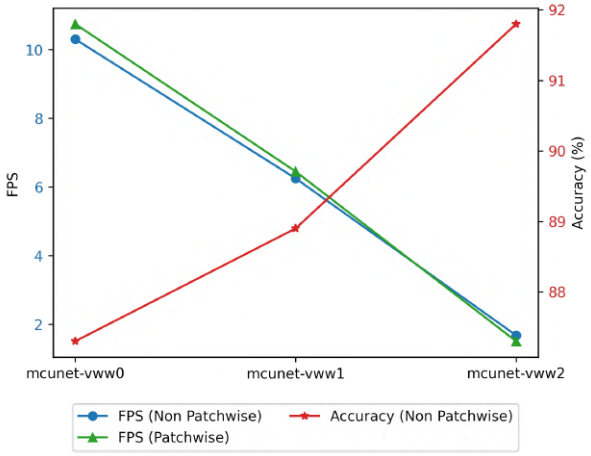
⭐ We have raised an issue in the official repo - #104

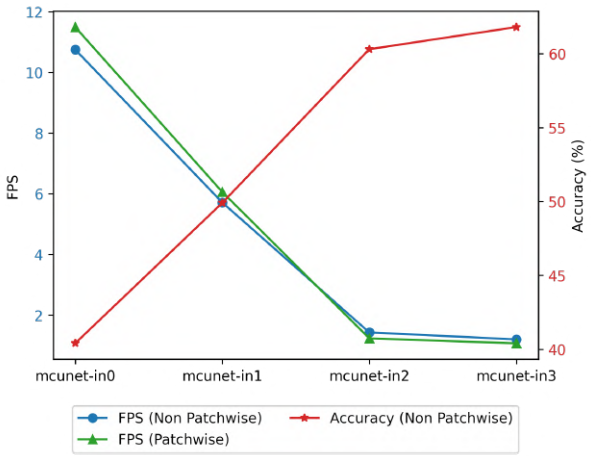
- We encountered a significant amount of repetitive and time-consuming steps due to the numerous models being run for inference.
  - We revamped the Python code and bash scripts to automate repetitive steps and streamline model inference. Now, running a single file generates code for all models, organizes folders, and creates tutorial-like directories for each models.

Lessons we learnt based on our interaction with real hardware

- FPS Vs Accuracy

  Credits : We borrowed the accuracy directly from the Tiny engine repo as we were directly using their pre-trained models and the accuracy was calculated locally by them and not on the board.





    - We discovered an inverse relationship between FPS (frames per second) and accuracy in model inference. As the accuracy of a model increases, so does its complexity and size, leading to slower performance on hardware with limited resources, consequently reducing the FPS.
    - Interestingly, when employing patchwise inference, we generally observed slightly higher FPS, with the exception of extremely heavy models.
- Furthermore, our observations highlighted that the VWW variants of MCUNet models are notably lighter and exhibit higher FPS compared to their ImageNet counterparts, with only a few exceptions.
- We've recognized that reproducing results from research papers may not always yield identical outcomes. This underscores the importance of thorough experimentation and validation, as well as an understanding of potential variables that could impact results.
- We learnt the critical importance of handling boards properly to prevent static damage. Our discussions with the course instructor revealed that large companies often mandate certification programs to ensure safe practices with static-sensitive components. In the event of any mishap, the certificate is revoked, requiring the employee to undergo the certification process again.
- Certain groups observed irregular behavior in specific boards. During our discussions, our course instructor clarified that while temperature changes may not be the primary cause of this erratic behavior in this scenario, they can occasionally play a contributing role. This was quite surprising to us.
- In our previous Tiny Engine assignment, we didn't delve deeply into the hardware components themselves, but this time around, we seized the opportunity to explore further.
    - During our interactions with our course instructor, we learnt that each board is furnished with a crystal oscillator comprising quartz. When voltage is applied, this oscillator generates a signal utilized for time calculation purposes.
    - Additionally, we delved into the significance of Arducam board pins.
        - MOSI (Master Out Slave In): This pin is used for data transmission from the master device (in this case, the board) to the slave device (the Arducam).
        - MISO (Master In Slave Out): This pin facilitates data transmission from the slave device (Arducam) to the master device (board).
        - SCK (Serial Clock): This pin generates the clock signal that synchronizes data transmission between the master and slave devices.
        - CS (NSS) (Chip Select or Slave Select): This pin is used to select the Arducam as the target device for communication.
        - VCC: This pin provides a power supply voltage of 3.3V to the Arducam.
        - GND: This pin is connected to the ground, providing a reference voltage for the Arducam.
            - We learnt that, sometimes it is required to connect to multiple GND pins to reduce the noise.
    - While unrelated to this assignment, we additionally learnt about the PWM (Pulse Width Modulation) pins which apparently can be used to control the speeds of the motor and create those blinking light effects we see on a daily basis. This might be interesting to dig deeper later.

Acknowledgement

We are grateful for the invaluable discussions with fellow groups and the guidance of our course instructor, which greatly enhanced our learning experience amidst the challenges encountered during this assignment.

Appendix

Raw observations

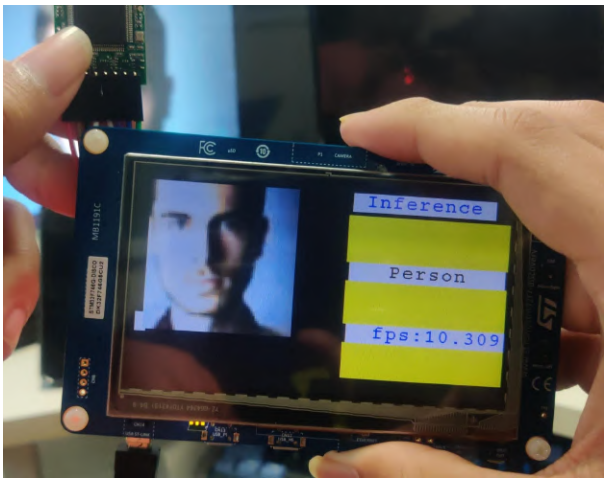
In this section, we have showed all the raw observations and informed decisions we made during our experimentation in tabular format.

Models	Resolution (Non Patchwise)	Build successful	FPS	Notes	Resolution (Patchwise)	Build successful	FPS	Notes
mcunet-vww0	64	YES	10.309		64	YES	10.752	1. Prediction is always "No Person"
mcunet-vww1	80	YES	6.250		80	YES	6.451	1. Prediction is always "No Person"
mcunet-vww2	72	YES	1.680	1. Original resolution was 144 but we encountered black screen. So we used 144/2.	72	YES	1.503	1. Prediction is always "No Person"
mcunet-in0	48	YES	10.869	1. The prediction was very spotty but top1 accuracy is only 40.4%. So its okay.	48	YES	11.494	1. Prediction is always "Person"
mcunet-in1	96	YES	5.747	1. The prediction was very spotty but top1	96	YES	6.060	1. Prediction is always "Person"

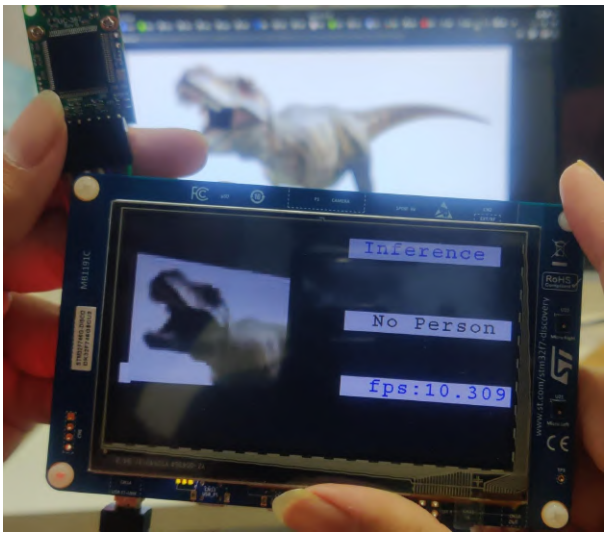
Models	Resolution (Non Patchwise)	Build successful	FPS	Notes	Resolution (Patchwise)	Build successful	FPS	Notes
				accuracy is only 49.9%. So its okay.				
mcunet-in2	80	YES	1.438	1. Prediction is always "Person" irrespective of higher top1 accuracy. 2. Original resolution : 160. Black screen issue. Used 160/2.	80	YES	1.234	1. Prediction is always "Person"
mcunet-in3	88	YES	1.200	1. Prediction is spotty irrespective of higher top1 accuracy. 2. Original resolution : 176. Black screen issue. Used 176/2.	88	YES	1.070	1. Prediction is always "Person"
mcunet-in4	80	NO	-	1. Build failed. Flash overflow. 2. Original resolution : 160. Used 160/2.	80	NO	-	1. Build failed. Flash overflow.
mbv2-w0.35	72	YES	3.125	1. Original resolution : 144. Black screen issue. Used 144/2.	72	YES	3.690	1. Prediction is always "Person"
proxyless-w0.3	88	YES	2.169	1. Prediction is always "Person". 2. Original resolution : 176. Black screen issue. Used 176/2.	88	YES	2.336	1. Prediction is always "Person"
person-det	80	YES	4.237	1. Prediction is always "Person". 2. Original resolution was unknown. So we used 80.	80	YES	5.405	1. Prediction is always "Person"

Snapshots

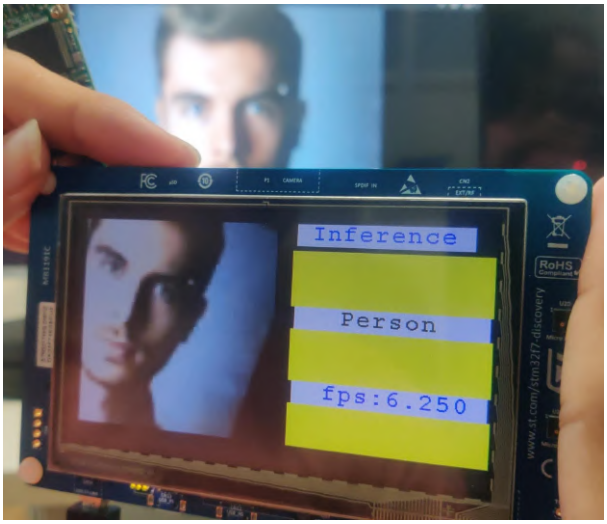
In this section we have add snapshots of inference of all the configurations.

- Non patchwise inference
  - mcunet-vww0
    - Person | FPS : 10.309
      - 

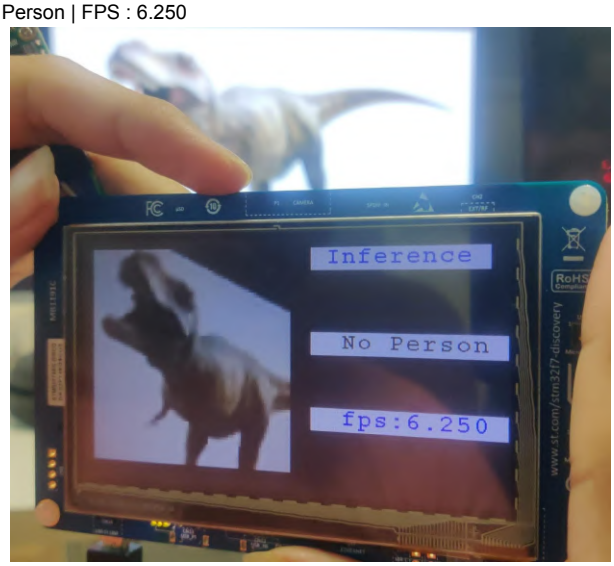
    - No Person | FPS : 10.309



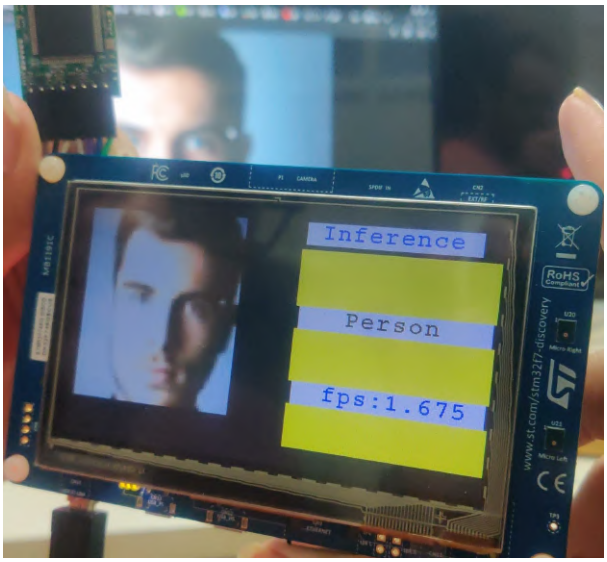
- mcunet-vww1
  - Person | FPS : 6.250

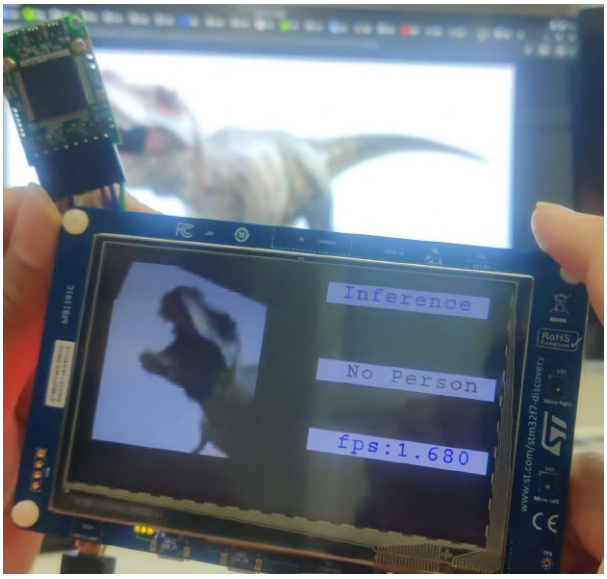


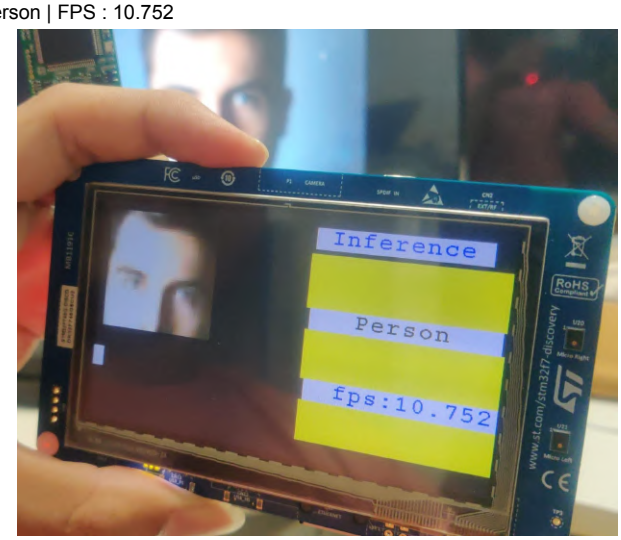
  - No Person | FPS : 6.250



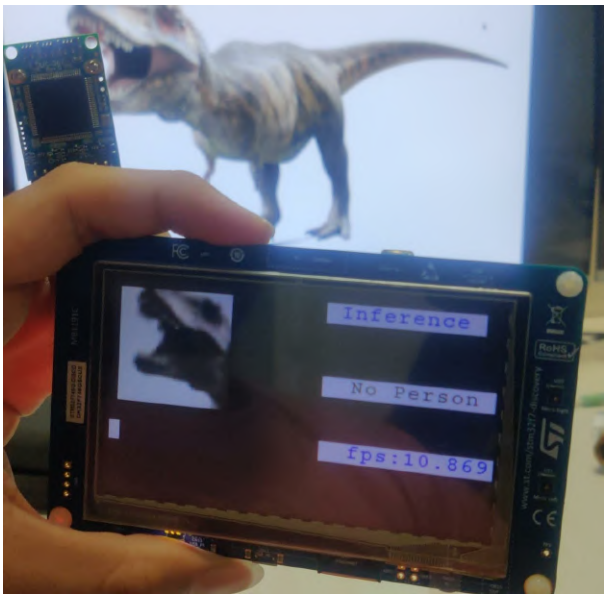
- mcunet-vww2
  - Person | FPS : 1.675

    - 

  - No Person | FPS : 1.680
    - 

- mcunet-in0
  - Person | FPS : 10.752
    - 

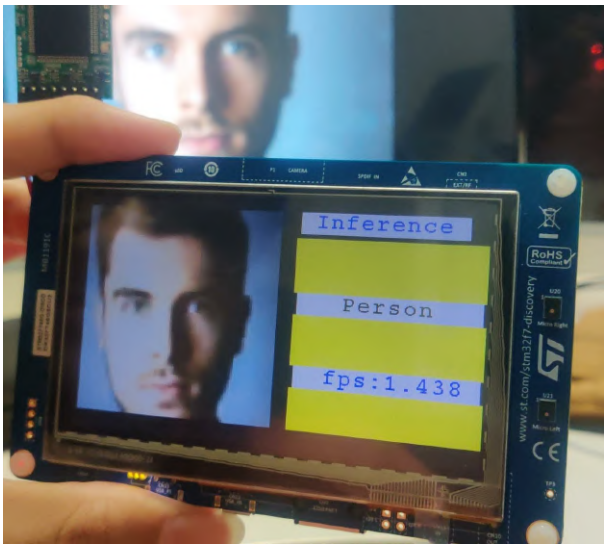
  - No Person | FPS : 10.869

- 

- mcunet-in1
  - Person | FPS : 5.714
    - 

  - No Person | FPS : 5.747
    - 

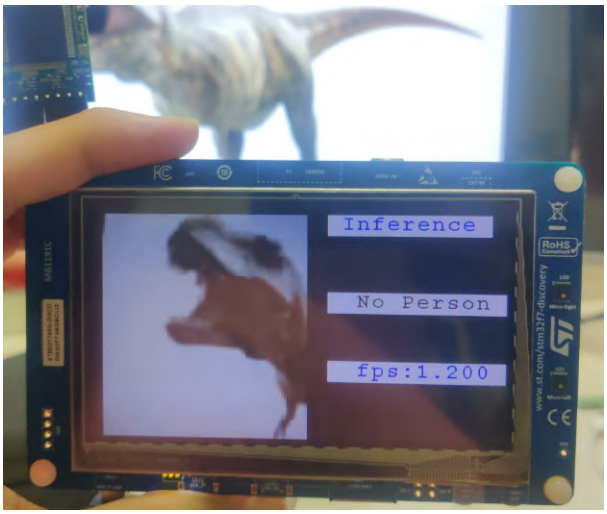
- mcunet-in2
  - Person | FPS : 1.438

    - 

  - No Person | FPS : 1.430
    - 

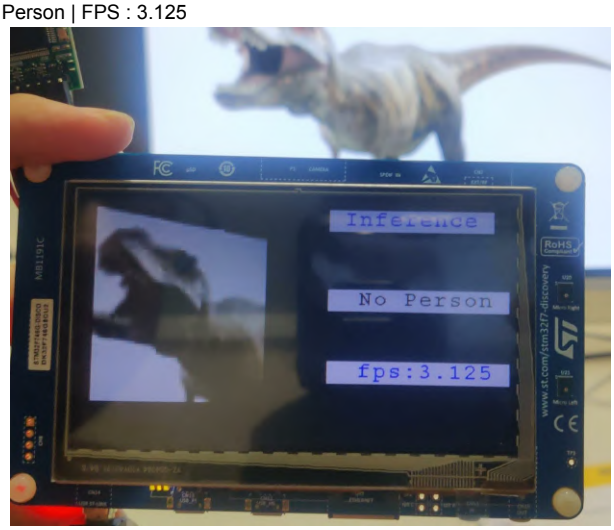
- mcunet-in3
  - Person | FPS : 1.200
    - 

  - No Person | FPS : 1.200



- mcunet-in4
  - Build failed due to flash overflow.
- mbv2-w0.35
  - Person | FPS : 3.125

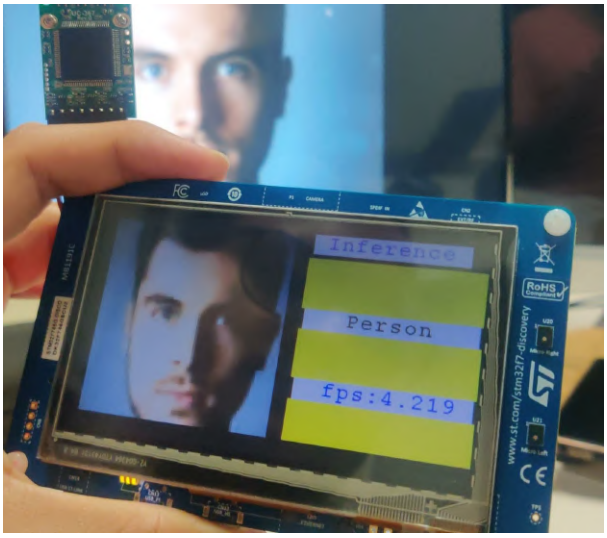
  - No Person | FPS : 3.125



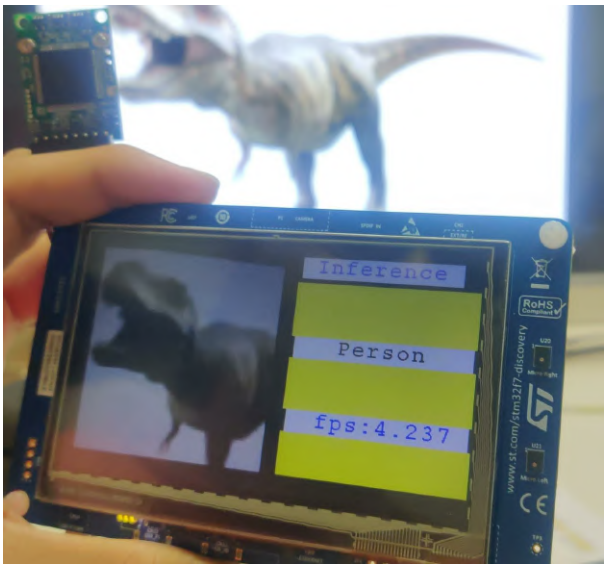
- proxyless-w0.3
  - Person | FPS : 2.164

    - 

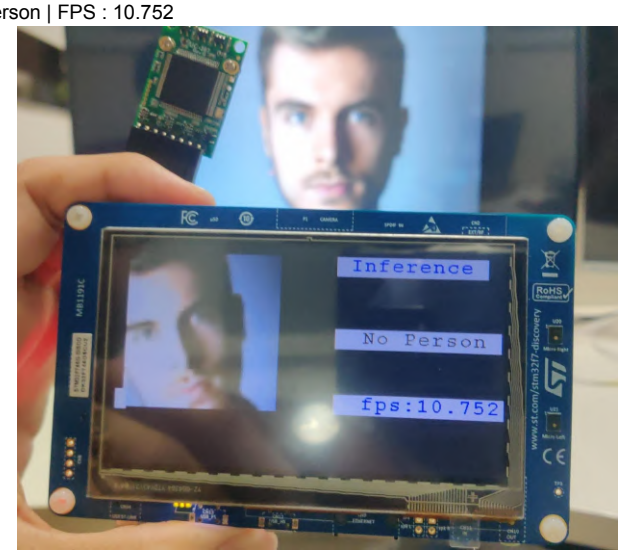
  - No Person | FPS : 2.169
    - 

- person-det
  - Person | FPS 4.219
    - 

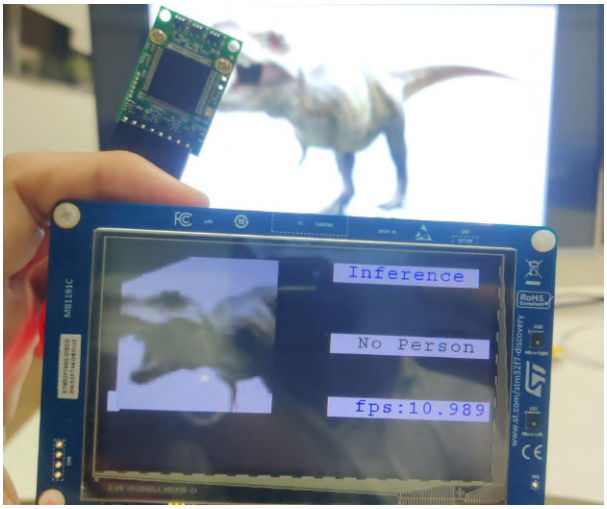
  - No Person | FPS 4.237



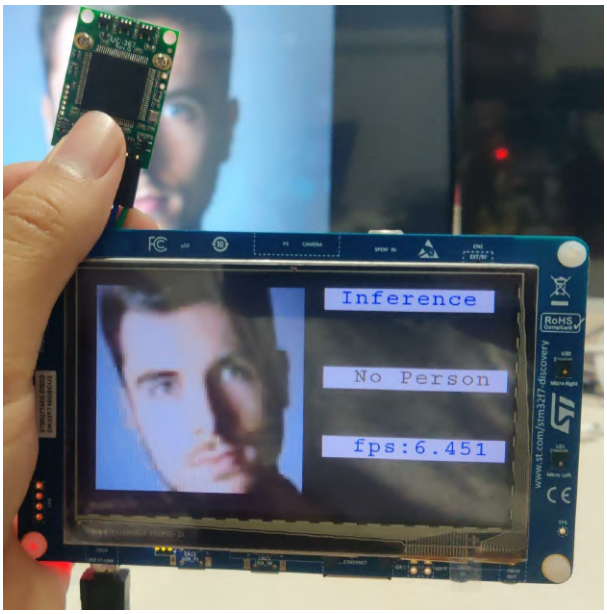
- Patchwise inference
  - mcunet-vww0
    - Person | FPS : 10.752



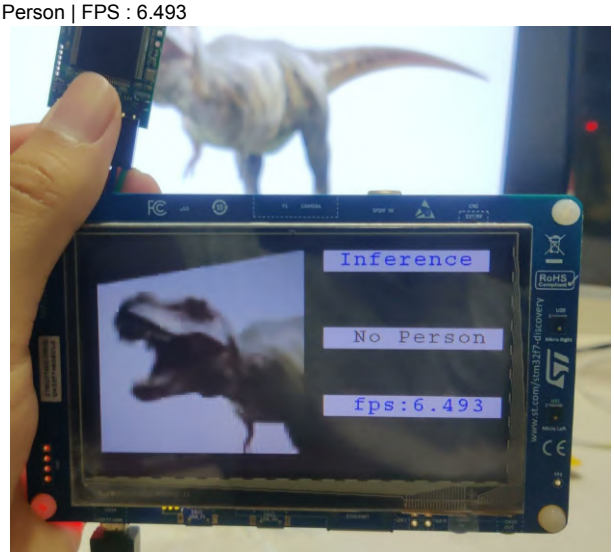
    - No Person | FPS : 10.989



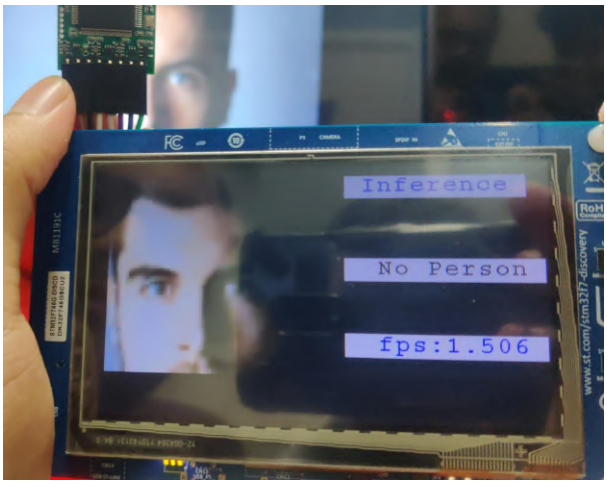
  - mcunet-vww1
    - Person | FPS : 6.451



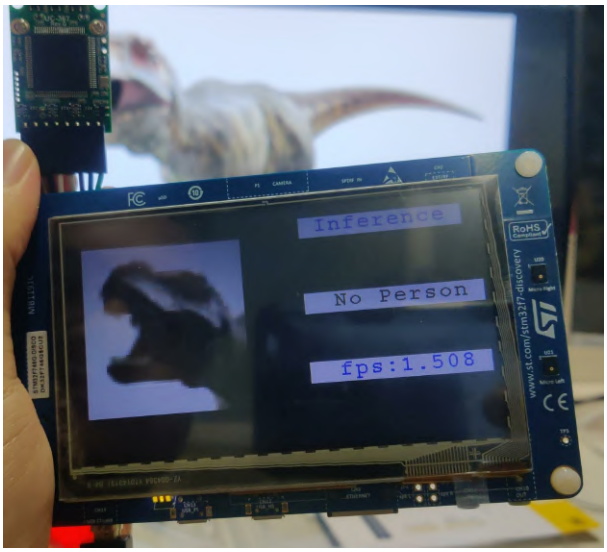
- No Person | FPS : 6.493



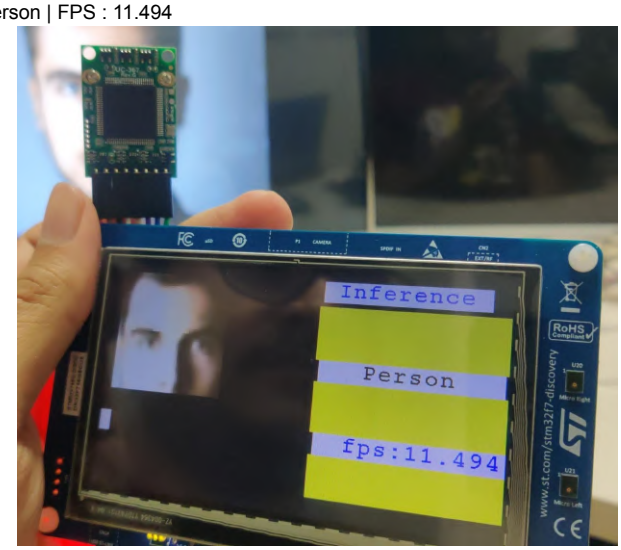
- mcunet-vww2
  - Person | FPS : 1.506



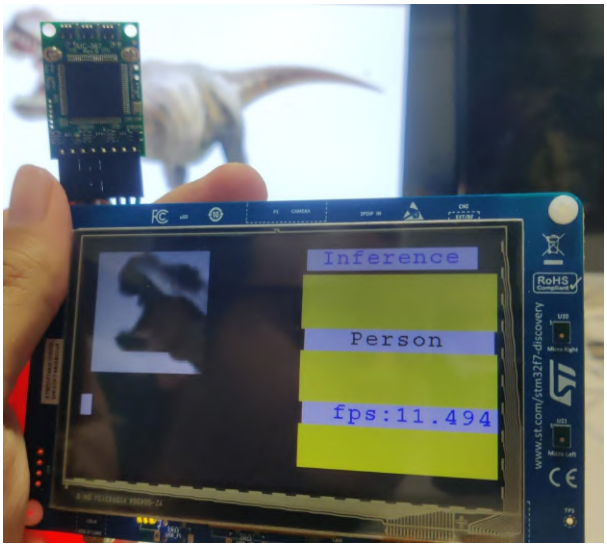
  - No Person | FPS : 1.503



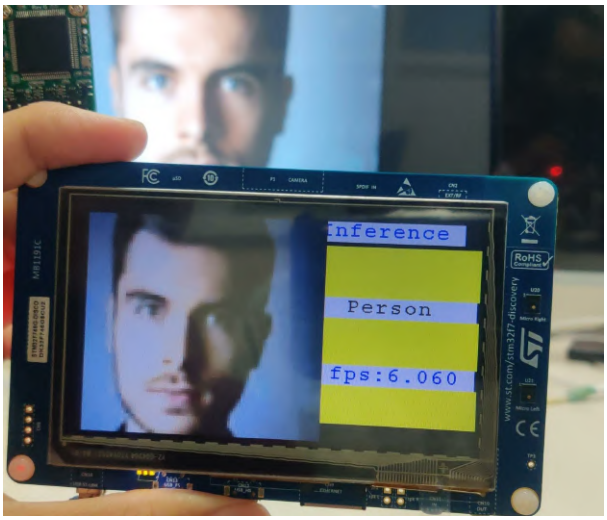
- mcunet-in0
  - Person | FPS : 11.494



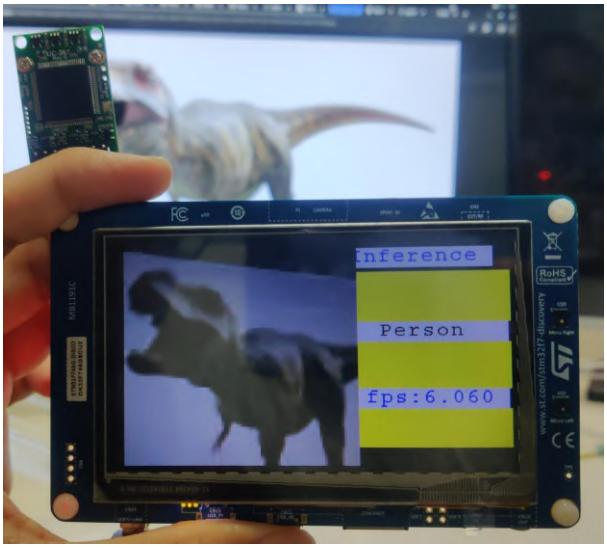
  - No Person | FPS : 11.494



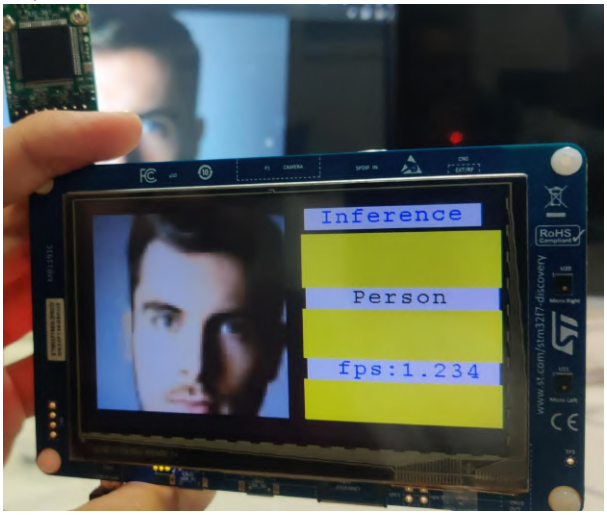
- mcunet-in1
  - Person | FPS : 6.060



- No Person | FPS : 6.060

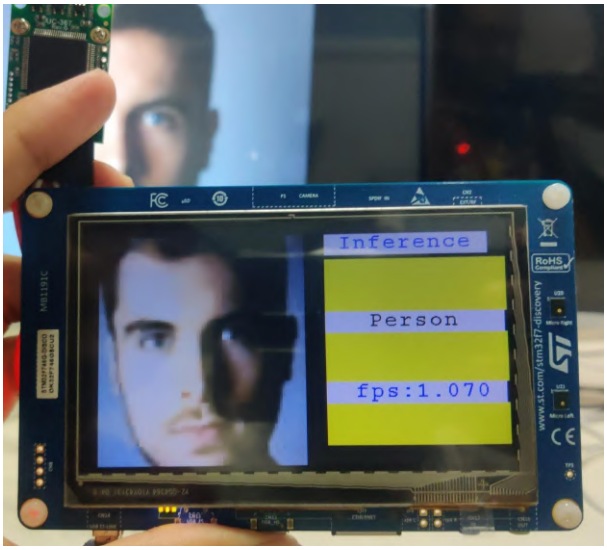


- mcunet-in2
  - Person | FPS : 1.234



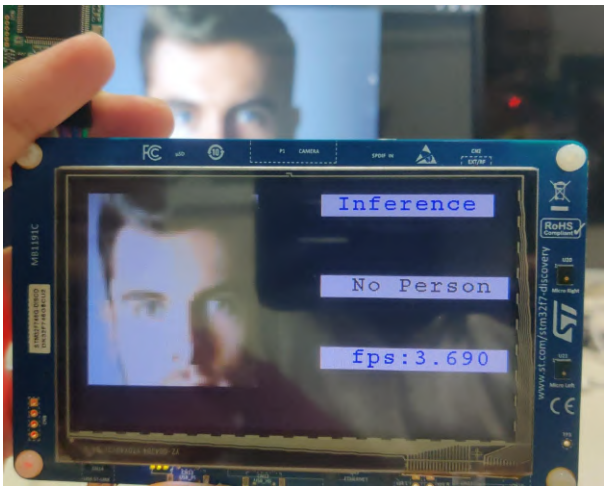
  - No Person | FPS : 1.234

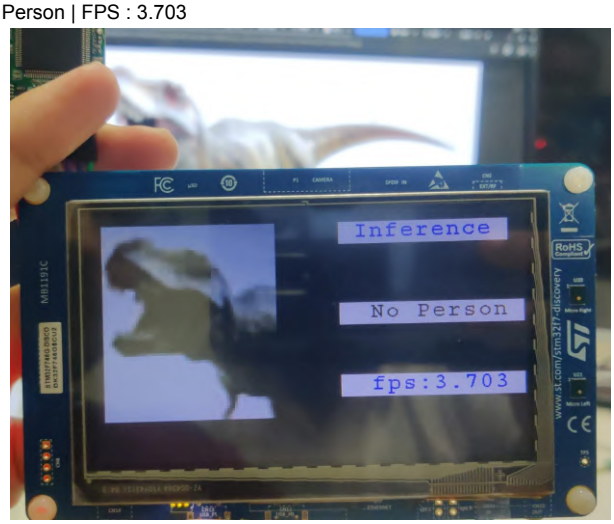
- mcunet-in3
  - Person | FPS : 1.070

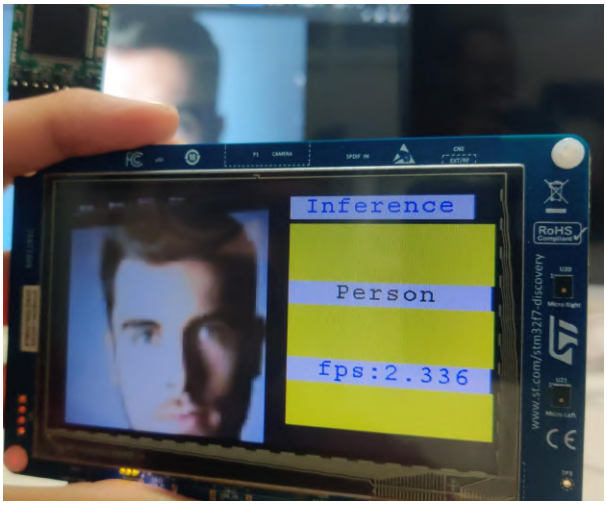


  - No Person | FPS : 1.072

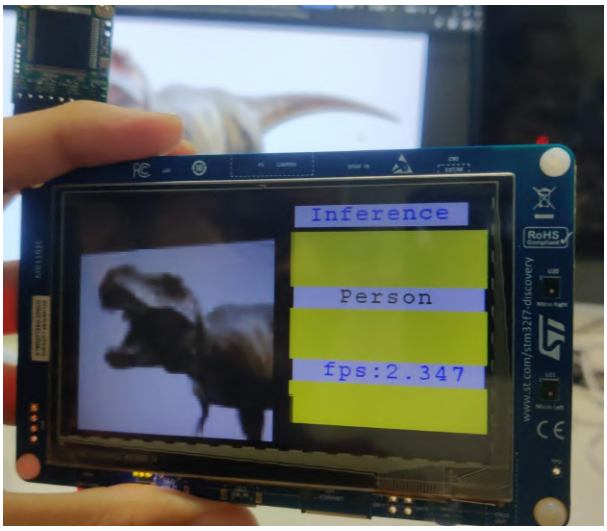
- mcunet-in4
  - Build failed due to flash overflow.
- mbv2-w0.35
  - Person | FPS : 3.690

    - 

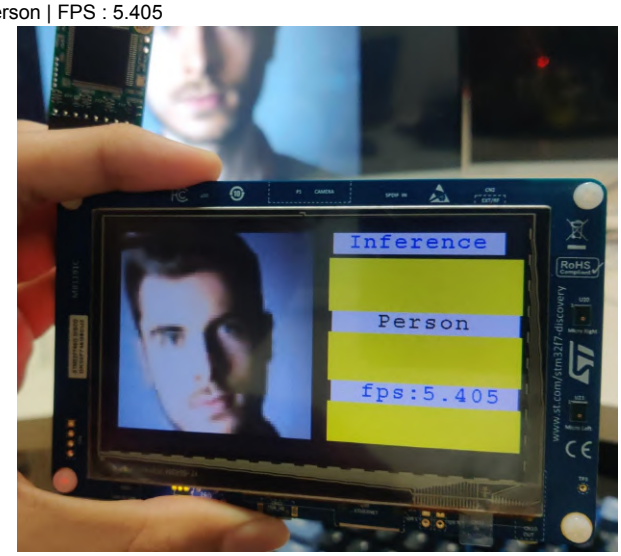
  - No Person | FPS : 3.703
    - 

- proxyless-w0.3
  - Person | FPS : 2.336
    - 

  - No Person | FPS : 2.347



- person-det
  - Person | FPS : 5.405



  - No Person | FPS : 5.434