CHALLENGES OF INFERENCING ON THE EDGE DEVICES :

Machine Learning is a broad domain in itself. An end to end process requires proper model building,evaluation and accurate deployment of those models.Time constraints have led us use the state of the art models and run them in our development environment but the deployment of these models is even harder than building a model pipeline. Here are some few challenges we faced while inference on the edge devices:

The various edge devices we used for inference are :
1.Edge TPU
2.Raspberry PI
3.Jetson Nano

1.**Edge TPU inference** : Inference on edge TPU requires our model to be fully quantized for weights and other parameters. Full quantization meaning converting to the unsigned integer 8 bits . For some of our processes we had already provided pretrained models by tensorflow model zoo. Object Detection, Pose Estimation,Face Detection, Image Classification and some others too
So it was quite easy using pre existing models trained on the dataset and using them for the inference. The results were quite astonishing.And deployment of the models on those devices were no pain within few lines of code using PYCORAL API and EDGERUNTIME Library provided by the Google.

However when we tried to convert these state of the art models to the smaller versions like the tensorflow lite and the edge TPU version we faced a lot of challenges while doing so :

1. Converting to tflite : The tensorflow documentation seems well defined and the examples show only few steps for the conversion of the model from .pb to .tflite but we faced a lot of issues while doing so :
    a. The documentation did not well defined the conversion steps from .pb to tflite and hence it took quite a while converting from tf to tflite.

The model required some additional information or the metadata or The metagraph definition from the model. Which we were not sure whether it had been provided in the model or not. Since that model was not trained by us. And graph definition was also unknown. So we weren't quite sure what the input and output tensors were.

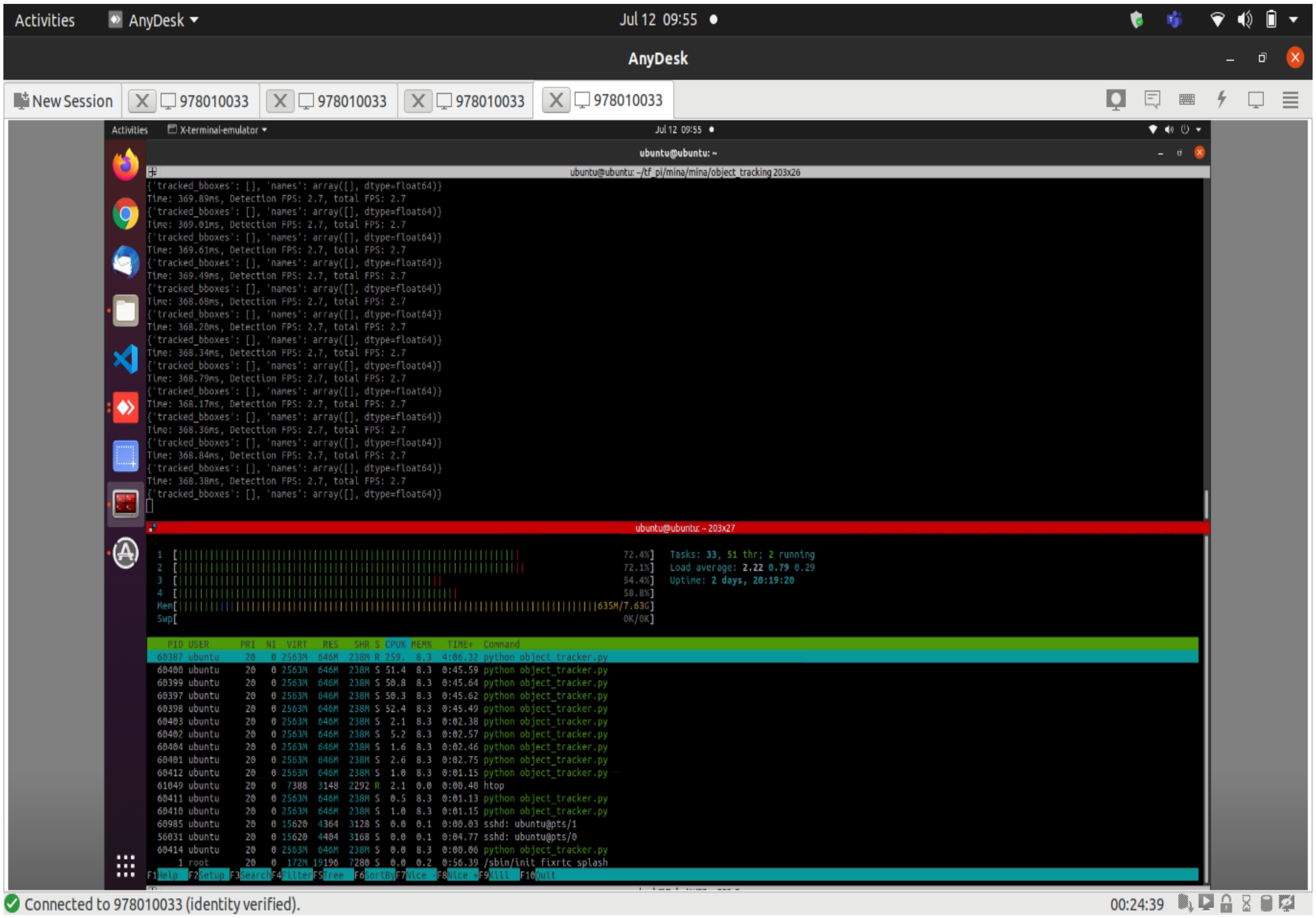
b. The conversion path was also not well defined in official documentation we faced a lot of time solving that issue.

c. The conversion from the tflite model to the quant process model required some dataset that were utilized during training .So it was not possible for us to train the quant model unless we went for the fine tuning and transfer learning.

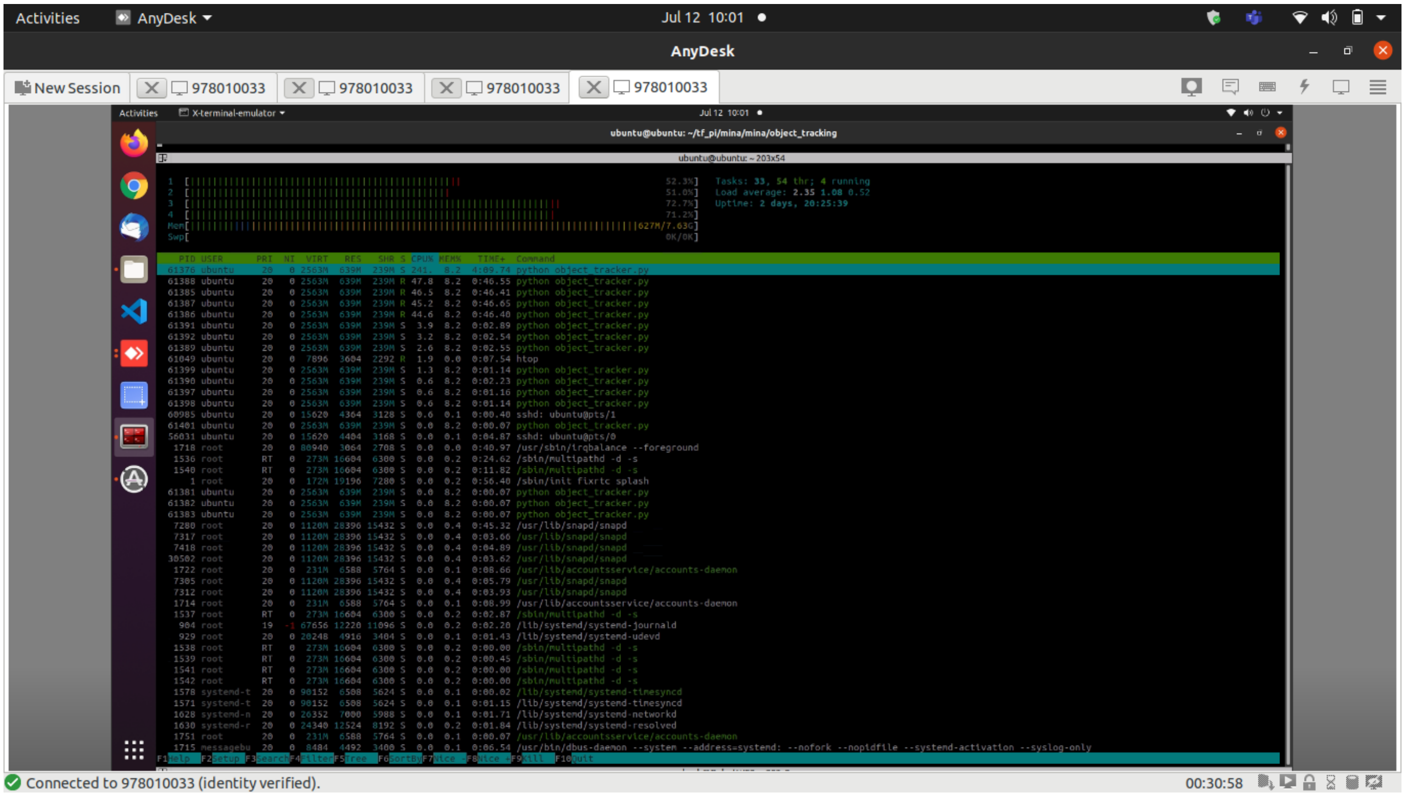
2.Raspberry PI : Inference on raspberry PI was not easy either because Installing tensorflow on Raspberry PI took 3 days . And that also we weren't sure whether It will work or not . Tensorflow on rasberryPI required to be built from the source using bazelisk .
1.Building bazel was a very hard task at first.
2.The binary file execution error was consistent.
3.Once build later the file vanished
4. It took 3 days to finally install Tensorflow on our PI system. Later we directly build the tensorflow from the wheel files found on the github.

We were still not sure whether it would work on our system or whether inference could be performed with such limited space and memory limitations.It wasn't so easy. However we managed to perform our experiment on PI reducing our Input size to 320 from 416
And changing batch size from 4 to 8

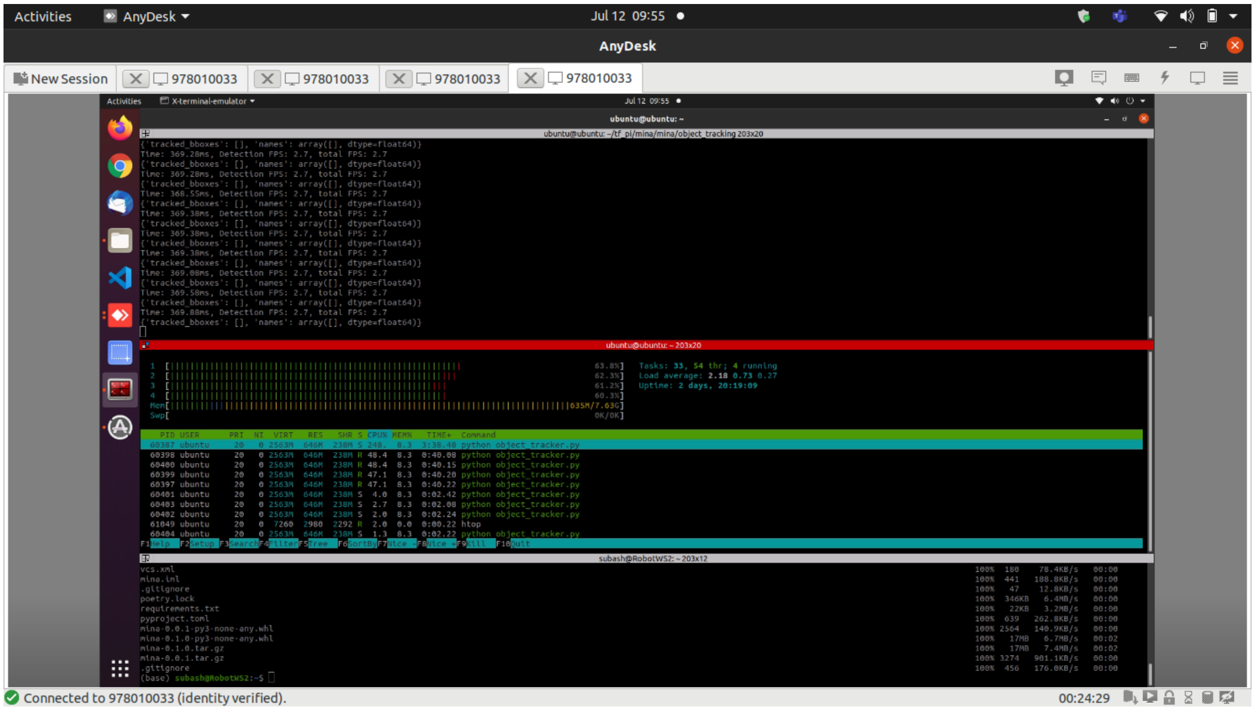
Some of the results of the process are screenshot and kept here :



We obtained an inference time of about 2.7 FPS .The above picture explains the use of the Memory which is also all consumed by our process.



3.JETSON NANO : The most challenging task was installing the dependencies and facing the challenge of installing the correct version . We faced a lot of version issues while installing tensorflow and inferencing on Jetson Nano. One of the main issue was OOM(out of memory).Our Nano couldn't accept tensor of large sizes which we fixed through resizing the input size and increasing the batch dimensions. The following outputs were obtained while inferencing on Jetson Nano .



3 Frames per second.

```
Traceback (most recent call last):
  File "object_tracker.py", line 146, in <module>
    yolo = Load_Yolo_model()
  File "/home/robot/Documents/mina/mina/object_tracking/yolov3/utils.py", line 76, in Load_Yolo_model
    yolo = Create_Yolo(input_size=YOLO_INPUT_SIZE, CLASSES=YOLO_COCO_CLASSES)
  File "/home/robot/Documents/mina/mina/object_tracking/yolov3/yolov3.py", line 375, in Create_Yolo
    conv_tensors = YOLOv3(input_layer, NUM_CLASS)
  File "/home/robot/Documents/mina/mina/object_tracking/yolov3/yolov3.py", line 113, in YOLOv3
    route_1, route_2, conv = darknet53(input_layer)
  File "/home/robot/Documents/mina/mina/object_tracking/yolov3/yolov3.py", line 86, in darknet53
    input_data = convolutional(input_data, (3, 3, 512, 1024), downsample=True)
  File "/home/robot/Documents/mina/mina/object_tracking/yolov3/yolov3.py", line 42, in convolutional
    bias_initializer=tf.constant_initializer(0.))(input_layer)
  File "/home/robot/miniforge3/envs/jupyter/lib/python3.6/site-packages/tensorflow/python/keras/engine/base_layer.py", line 926, in __call__
    input_list)
  File "/home/robot/miniforge3/envs/jupyter/lib/python3.6/site-packages/tensorflow/python/keras/engine/base_layer.py", line 1098, in _functional_construction_call
    self._maybe_build(inputs)
  File "/home/robot/miniforge3/envs/jupyter/lib/python3.6/site-packages/tensorflow/python/keras/engine/base_layer.py", line 2643, in _maybe_build
    self.build(input_shapes)  # pylint:disable=not-callable
  File "/home/robot/miniforge3/envs/jupyter/lib/python3.6/site-packages/tensorflow/python/keras/layers/convolutional.py", line 204, in build
    dtype=self.dtype)
  File "/home/robot/miniforge3/envs/jupyter/lib/python3.6/site-packages/tensorflow/python/keras/engine/base_layer.py", line 614, in add_weight
    caching_device=caching_device)
  File "/home/robot/miniforge3/envs/jupyter/lib/python3.6/site-packages/tensorflow/python/training/tracking/base.py", line 750, in _add_variable_with_custom_getter
    **kwargs_for_getter)
  File "/home/robot/miniforge3/envs/jupyter/lib/python3.6/site-packages/tensorflow/python/keras/engine/base_layer_utils.py", line 145, in make_variable
    shape=variable_shape if variable_shape else None)
  File "/home/robot/miniforge3/envs/jupyter/lib/python3.6/site-packages/tensorflow/python/ops/variables.py", line 260, in __call__
    return cls._variable_v1_call(*args, **kwargs)
  File "/home/robot/miniforge3/envs/jupyter/lib/python3.6/site-packages/tensorflow/python/ops/variables.py", line 221, in _variable_v1_call
    shape=shape)
  File "/home/robot/miniforge3/envs/jupyter/lib/python3.6/site-packages/tensorflow/python/ops/variables.py", line 199, in <lambda>
    previous_getter = lambda **kwargs: default_variable_creator(None, **kwargs)
  File "/home/robot/miniforge3/envs/jupyter/lib/python3.6/site-packages/tensorflow/python/ops/variable_scope.py", line 2597, in default_variable_creator
    shape=shape)
  File "/home/robot/miniforge3/envs/jupyter/lib/python3.6/site-packages/tensorflow/python/ops/variables.py", line 264, in __call__
    return super(VariableMetaclass, cls).__call__(*args, **kwargs)
  File "/home/robot/miniforge3/envs/jupyter/lib/python3.6/site-packages/tensorflow/python/ops/resource_variable_ops.py", line 1518, in __init__
    distribute_strategy=distribute_strategy)
  File "/home/robot/miniforge3/envs/jupyter/lib/python3.6/site-packages/tensorflow/python/ops/resource_variable_ops.py", line 1651, in _init_from_args
    initial_value() if init_from_fn else initial_value,
  File "/home/robot/miniforge3/envs/jupyter/lib/python3.6/site-packages/tensorflow/python/ops/init_ops_v2.py", line 385, in __call__
    dtype)
  File "/home/robot/miniforge3/envs/jupyter/lib/python3.6/site-packages/tensorflow/python/ops/init_ops_v2.py", line 1035, in random_normal
    shape=shape, mean=mean, stddev=stddev, dtype=dtype, seed=self.seed)
  File "/home/robot/miniforge3/envs/jupyter/lib/python3.6/site-packages/tensorflow/python/util/dispatch.py", line 201, in wrapper
    return target(*args, **kwargs)
  File "/home/robot/miniforge3/envs/jupyter/lib/python3.6/site-packages/tensorflow/python/ops/random_ops.py", line 94, in random_normal
    shape_tensor, dtype, seed=seed1, seed2=seed2)
  File "/home/robot/miniforge3/envs/jupyter/lib/python3.6/site-packages/tensorflow/python/ops/gen_random_ops.py", line 642, in random_standard_normal
    _ops.raise_from_not_ok_status(e, name)
  File "/home/robot/miniforge3/envs/jupyter/lib/python3.6/site-packages/tensorflow/python/framework/ops.py", line 6843, in raise_from_not_ok_status
    six.raise_from(core._status_to_exception(e.code, message), None)
  File "<string>", line 3, in raise_from
tensorflow.python.framework.errors_impl.ResourceExhaustedError: OOM when allocating tensor with shape[3,3,512,1024] and type float on /job:localhost/replica:0/task:0/device:GPU:0 by allocator GPU_0_b
fc [Op:RandomStandardNormal]
(jupyter) robot@robot-desktop:~/Documents/mina/mina/object_tracking$
```

OUT OF MEMORY ERROR

We also tried running sort model in jetson Nano but some issues with that as well.Recently we tried inferencing from our mobilenet ssd Detections to our deep sort algorithm but that requires the frame captured from the edge device to be compatible to the Frames captured by the nano device.
We experienced the same graph definition issues in jetson nano from conversing from the heavy model to the light models.

We are still exploring the alternate algorithms that could be used for the object Tracking. Optical flow,mean shift, Open Cv library also includes many of such algorithms but those have various issues regarding:
1.Occlusion
2. Id switching

Till date the best algorithm in accordance to our subject is the SOTA Deep Sort algorithm . However we are exploring more on the context.