EEL3090 Embedded Systems

Project Report

GroupMembers : Suyash Bansal(B20CS076)
Sudip Kumar(B20CS070)

- ## Objective

  Our Project (ESC - 14) , main aim is to save memory at the inference for Neural Networks via operator reordering so as to make them work efficiently on MCU's (MicroController units)

- ## Explanation

  Neural network execution:

  - ➢ By combining neighboring operators and integrating batch normalization layers into earlier linear operations, for example, modern deep learning frameworks optimize the network's computation graph for inference in advance. One operator is evaluated at a time in the graph's nodes according to their topological order as the execution moves forward.
  - ➢ The computation graph of more modern architectures, such as ResNet, Inception, and NasNet, no longer follows a linear path and now has branches. These architectures allow for the processing of the same tensor by several layers through the use of divergent processing paths. As a result, the inference software may have many operators ready for use at any time.

  Resource constraints in microcontroller:

  - ➢ The two different types of on-chip memory can be used to directly map the memory needs of a neural network. Network parameters can be saved in NOR-Flash as static data that can be incorporated as immutable parameters in the executable code. In order to be stored in SRAM, any intermediate tensors that depend on input are generated at runtime.
  - ➢ As a result, the NOR-Flash and SRAM memories' capacities, respectively, set a limit on the model size and peak memory utilization.

Method of implementation:

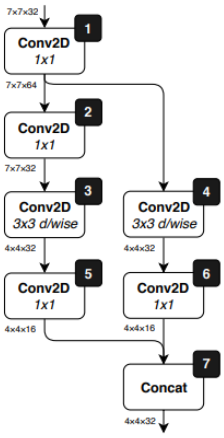
- ➢ **Optimizations:**
  - ★ Operator reordering:
    - Reordering the operator so that we can use the system memory effectively.
    - Modify the model to minimize peak memory usage by reordering operators in the model file
    - Below is the attached table of the tensor information for the neural network to be run on the RAM and the operator execution schedule for the tensors.

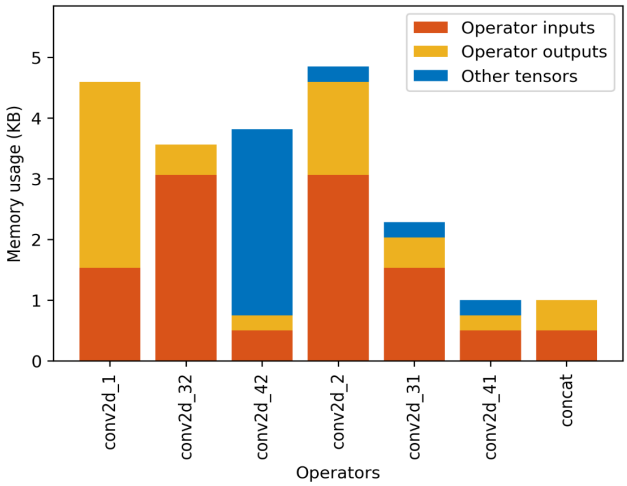
  - ★ Information about the operator/tensors and memory usage for execution of these operators. This information is given to show the importance of operator reordering.

```
Tensor information (weights excluded):
+----+-----------------------+-----------------+-----------------+
| Id |         Tensor        |      Shape      | Size in RAM (B) |
+----+-----------------------+-----------------+-----------------+
|  1 |       Conv1/Relu      | (1, 30, 30, 16) |          14,400 |
|  2 |      Conv1_input      |  (1, 32, 32, 3) |           3,072 |
|  4 |       Conv2/Relu      | (1, 28, 28, 16) |          12,544 |
|  5 |      FC2/BiasAdd      |     (1, 10)     |              10 |
|  7 |      FC2/Softmax      |     (1, 10)     |              10 |
|  8 |    activation/Relu    |     (1, 128)    |             128 |
|  9 | max_pooling2d/MaxPool | (1, 14, 14, 16) |           3,136 |
+----+-----------------------+-----------------+-----------------+

Operator execution schedule:
+------------------------+-------------------------+----------------+
| Operator (output name) | Tensors in memory (IDs) | Memory use (B) |
+------------------------+-------------------------+----------------+
|       Conv1/Relu       |          [1, 2]         |         17,472 |
|       Conv2/Relu       |          [1, 4]         |         26,944 |
| max_pooling2d/MaxPool  |          [4, 9]         |         15,680 |
|    activation/Relu     |          [8, 9]         |          3,264 |
|      FC2/BiasAdd       |          [8, 5]         |            138 |
|      FC2/Softmax       |          [5, 7]         |             20 |
+------------------------+-------------------------+----------------+
Current peak memory usage: 26,944 B
```

The below image shows the operator reordering mechanism via parallel operator execution example





➢ Implementation

- ★ We have firstly used a large dataset [mnist_dataset] from the tensorflow python library where there are nearly about 87 K instances of data samples of handwritten digits . So basically it is a multiclass classification problem where we need to classify each digit image to its corresponding value.
- ★ Then , we have developed multiple layers of neural-networks to make the dataset trained on it where I have used multiple operations of 2D convolution and Pooling in order to demonstrate the Operator reordering mechanism.
- ★ Further, model trained and fitted on the dataset is saved as *my_model.h5 file*. Then in the file *onnx_operator_reordering.py* we have built a tensorRT Engine() and using the ONNX parser we have converted the trained the keras model to ONNX format *my_model.onnx* which is then built onto the trt_engine.
- ★ The parsed trt_engine is then used for the plotting the memory usage curves for the dataset trained via tensorrt- nvidia-driver GPU. parsed trt_engine on ONNX is used for optimizing the model on operator sequencing which is then improved via changing the order of operator reorder.
- ★ The graphs built are displayed to represent the inference of the model memory usage on the operator reordering and without operator reordering.

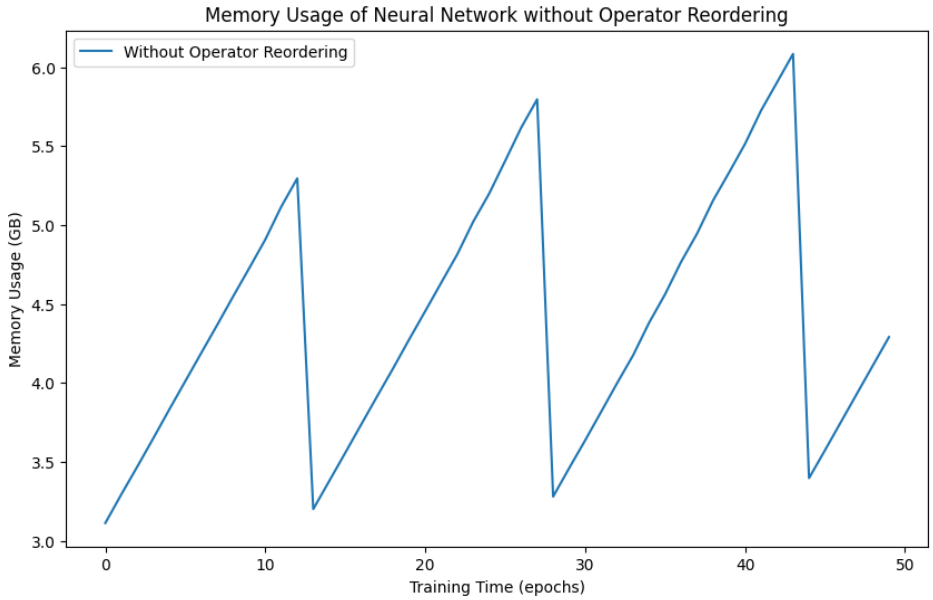
➢ Result and Conclusion:

- ★ On running the model for the epochs = 50 we have found out that the model memory usage without the reordering of the operators is varying with the layer operator precedence.

★ But when we consider the case of the operator execution reordering we can see a graph with the nearly constant memory usage that is memory usage does on hike on evaluation of operator on each epoch of neural network.

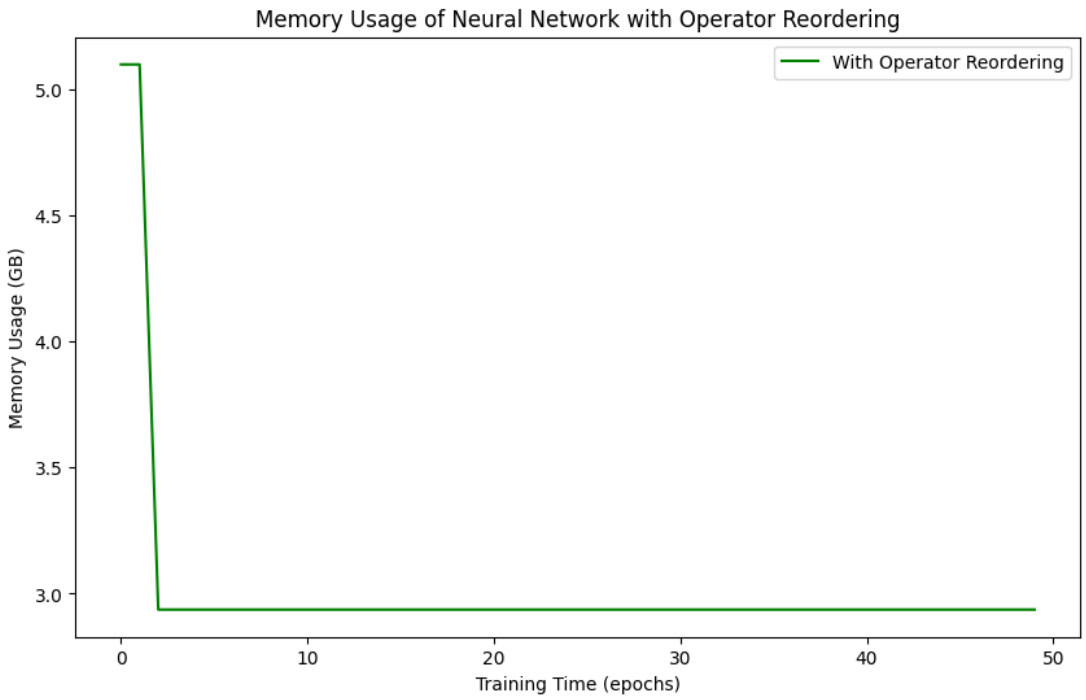
★ On increasing the epochs in the definition of the neural network memory went's on to hike by a huge amount which makes it difficult for any model to run on the microcontroller units .In order to run the neural networks on the MCU's we need to have a model with not having sudden hike in memory usage with operator evaluation serially.

★ In the case of operator reordering MCU's can be deployed to train and run the neural network with the fulfillment of memory requirement without any external loaded hardware. This is concluded by the memory usage graphs below.

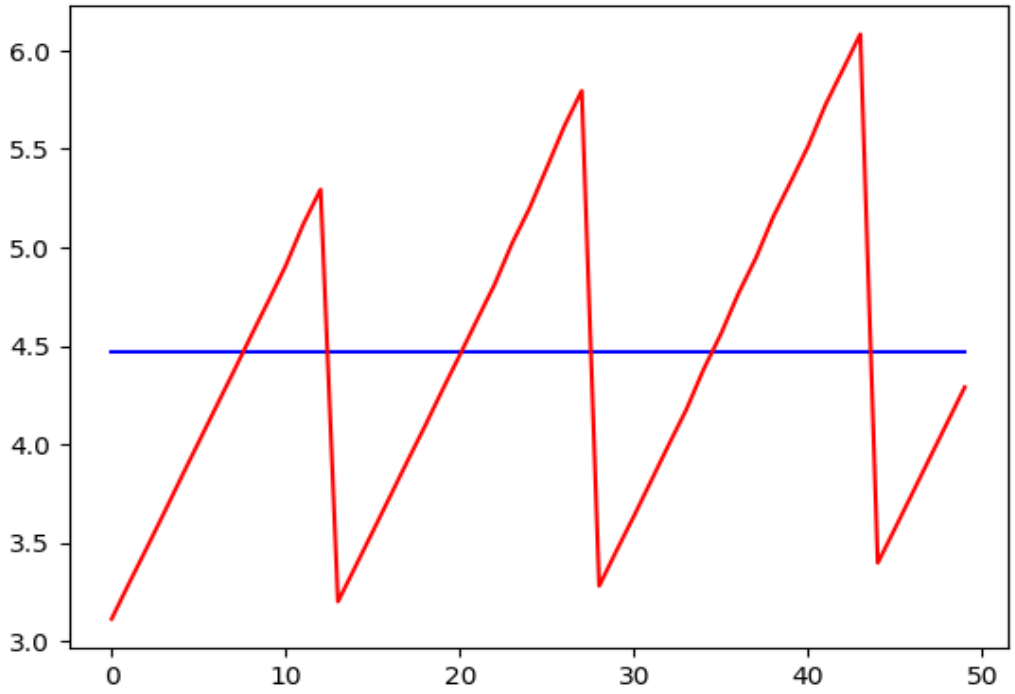


★ It is evident from the memory plot without operator reordering about the memory hike suddenly on operator evaluation of each epoch. Max memory = 6.3 GB.

★ **After applying the neural network memory Reordering:-**
The peak memory usage has dropped down to 5.3GB which is very good based on how important it is for microcontrollers. Reduction in approximately 1GB memory shows how effective is the neural network model.

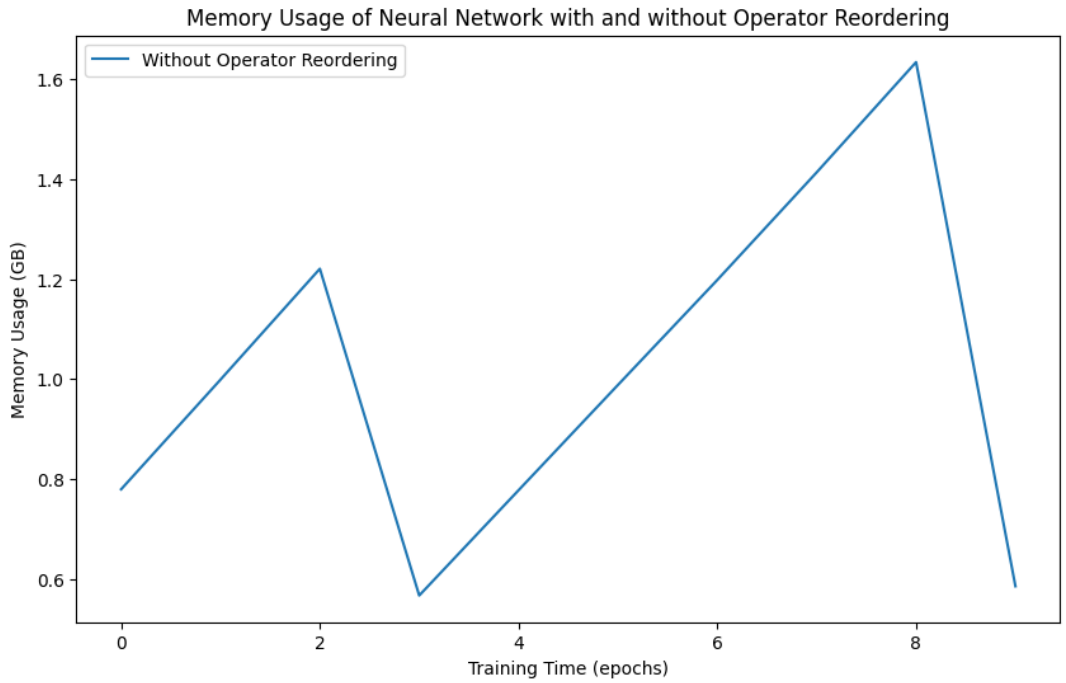


★ This below graph depicts the average memory comparisons between the operator reordering and without the operator reordering so here we can see the stability provided by the operator reordering and peak memory usage is also improved.

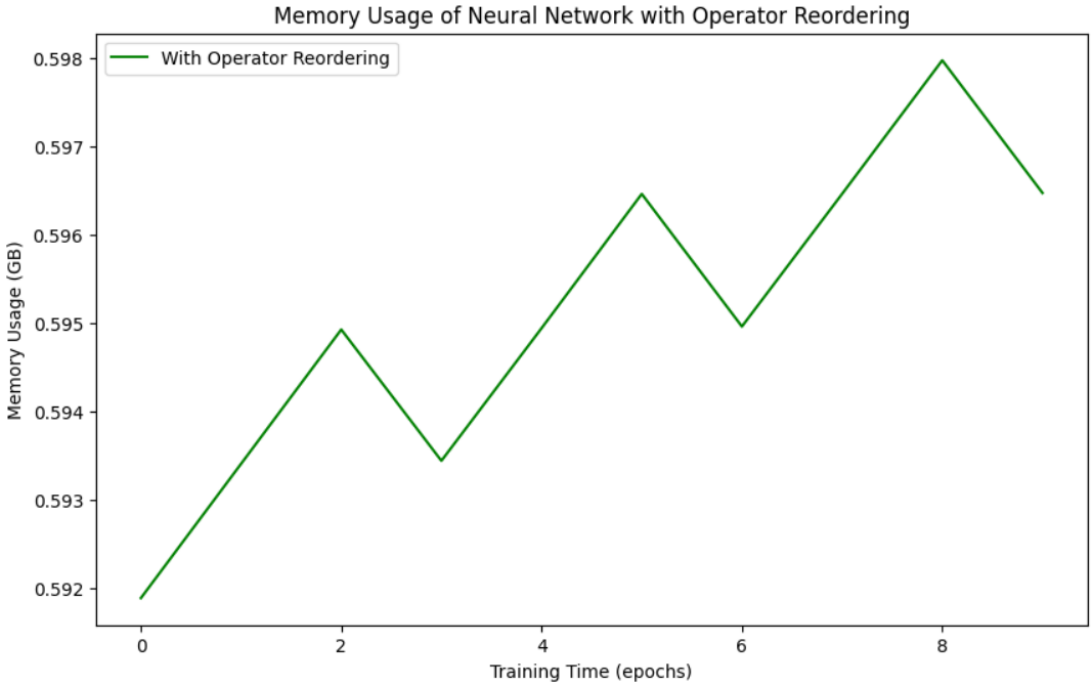


★ From the following graph we can take the observation that the average memory use is approximately 4.5GB.

★ For the case of epochs = 10 the memory usage graphs seems to be :



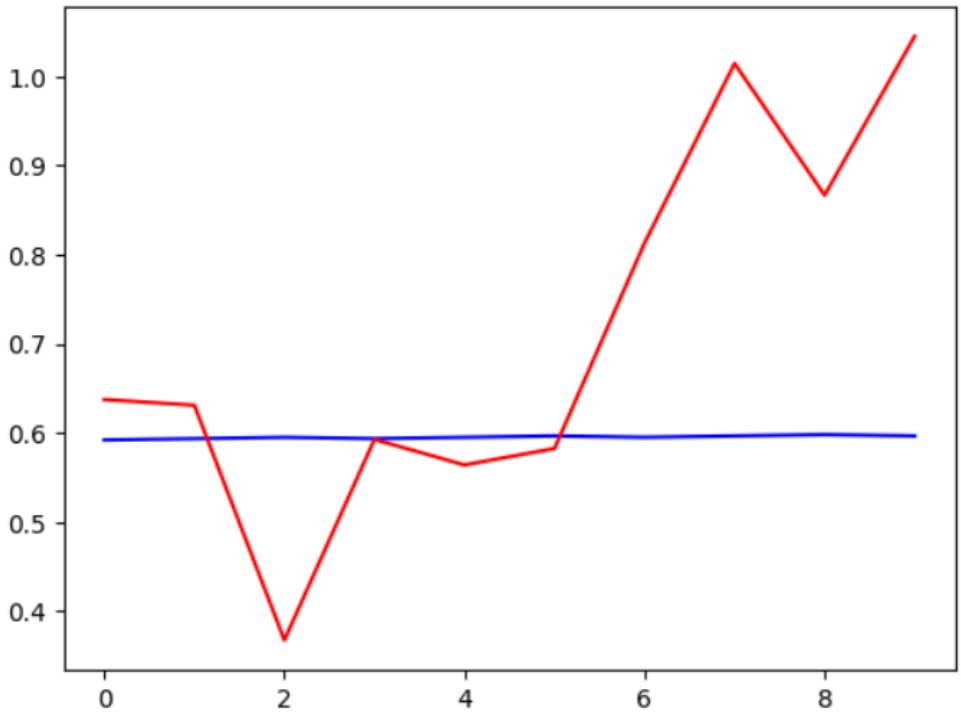
★ With operator reordering graphs :



★ So the memory usage lies between 0.592 to 0.598 GB which is quite a stable range for the 10 epochs whereas for the graph without operator reordering it is

from 0 to 2 GB for just 10 epochs so as the epochs increases it will lead to great memory issues.

★ Combined comparative graph :



★

➔ So , In the conclusion we can see how the operator reordering is helping in reducing the memory consumption at each level of neural network architecture and that is what makes it feasible for running it on Microcontroller units.

****