Implementing ML algorithms using C-model

Priyankar Sarkar

October 2021

1 Introduction

This document can be used a reference for implementing any ML algorithm using the C-model of the hardware accelerator. It is a walk-through of the whole process, starting from creating a tensor, loading them into memory-pool, performing various operations on them using the processing elements and finally convert the output tensor into a suitable image using available scripts.

2 Steps involved

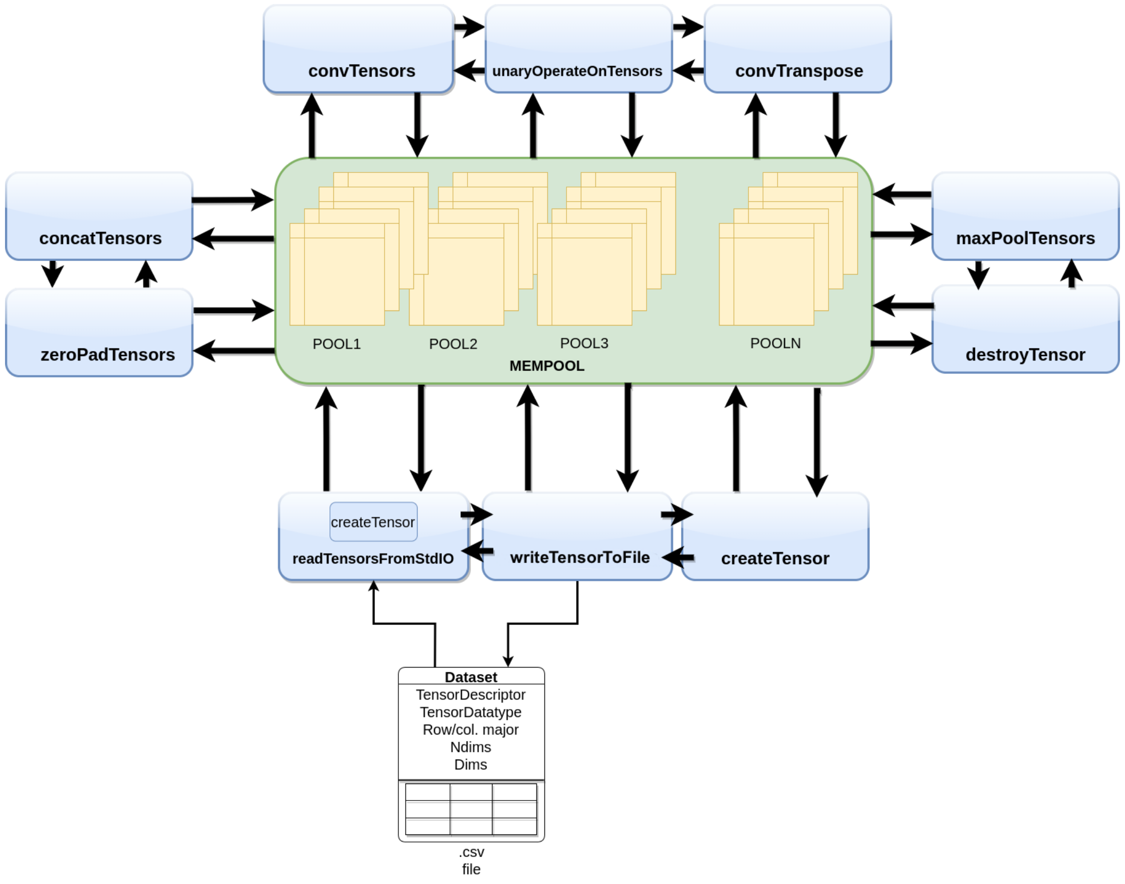

Figure 1: Brief Overview of C Model

1. **Read** a **tensor** from file.

   The tensors are stored in **.csv** file. One can use the **readTensorFromFile()** method to read the file and create a Tensor into the memory-pool.The format of the csv file is as follows:

  - Line 1: Datatype of the tensor.
  - Line 2: Storage type of the tensor('1' for row-major, '0' for column-major)
  - Line 3: Number of dimensions of the tensor.
  - Line 4: Dimension array representing the shape of the tensor.
  - Line 5 onwards: Data.

2. **Create** additional **tensors**.

   Use the **createTensor()** function. One need not worry about which pool to allocate to store the tensor. This function automatically allocates the tensor in a best-fit manner.

3. Use the various **processing elements** available.

   Their implementations are described in a separate directory.One can test them individually using the scripts available. The list of available functions are:

   - convTensors()
   - maxPoolonTensors()
   - dilateTensors()
   - depadTensors()
   - unaryOperateOnTensors()
   - batchNormalization()
   - zeroPadTensors()

4. **Destroy** unnecessary tensors.

   It is important to clean up the memory-pool as its size is fixed and can be used by a limited number of tensors at a certain point of time.Use the **destroyTensor()** method to destroy them

5. **Write** the tensor to **File**.

   This is the final step where a tensor is stored in a **.csv** file using **writeTensorToFile()**. The arrangement of data in the output file is same as that of the read file.

3 Example

3.1 UNET Architecture

The architecture contains two paths. First path is the contraction path (also called as the encoder) which is used to capture the context in the image. The encoder is just a traditional stack of convolutional and max pooling layers. The second path is the symmetric expanding path (also called as the decoder) which is used to enable precise localization using transposed convolutions. Thus it is an end-to-end fully convolutional network(FCN), i.e. it only contains convolutional layers and does not contain any Dense layer because of which it can accept image of any size.

3.1.1 Flowchart

Below contains the flowchart of the implementation of this architecture. Note that the previously mentioned steps of creating necessary tensors and destroying them at certain stages has been taken care of.

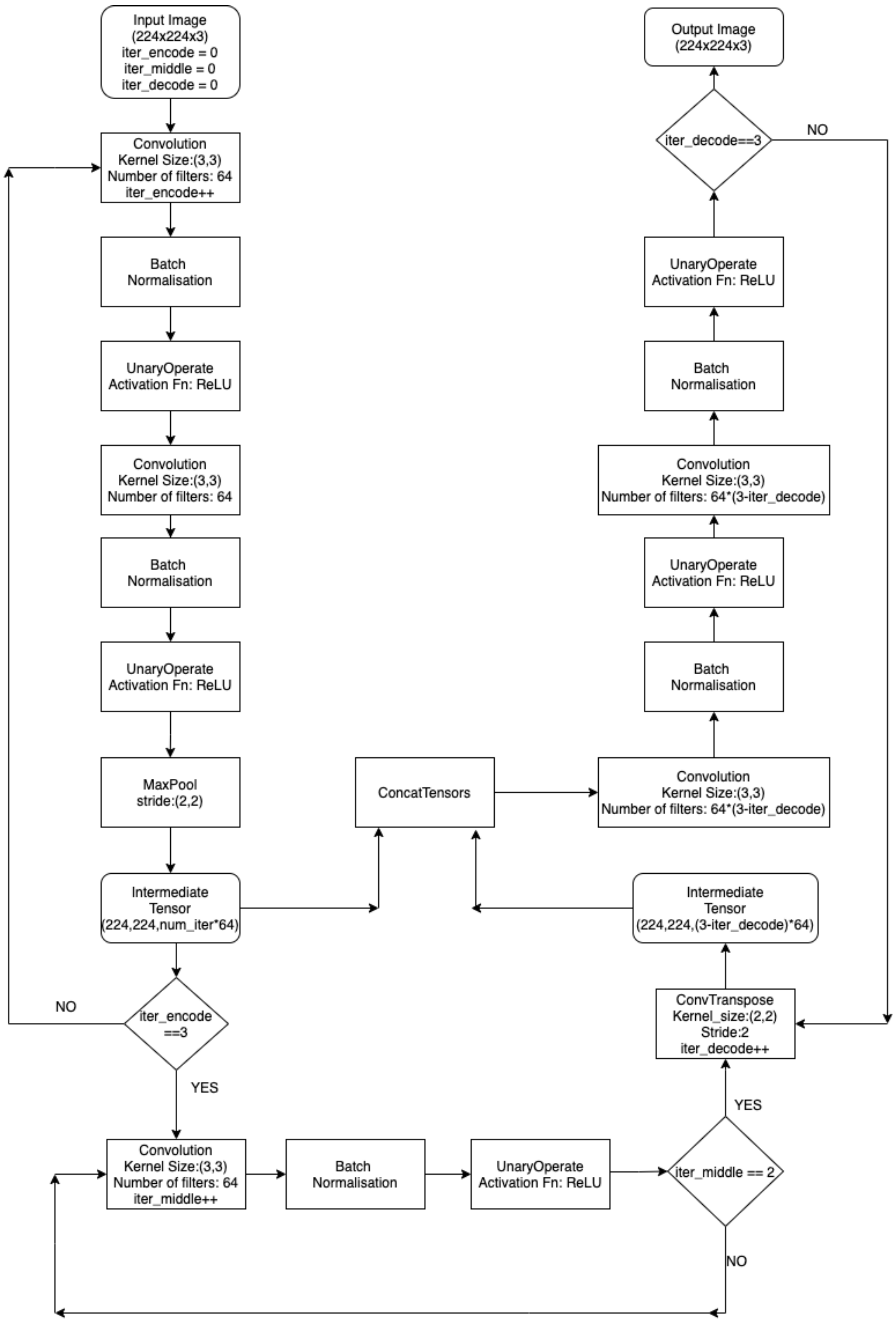

Figure 2: UNET Architecture Flowchart