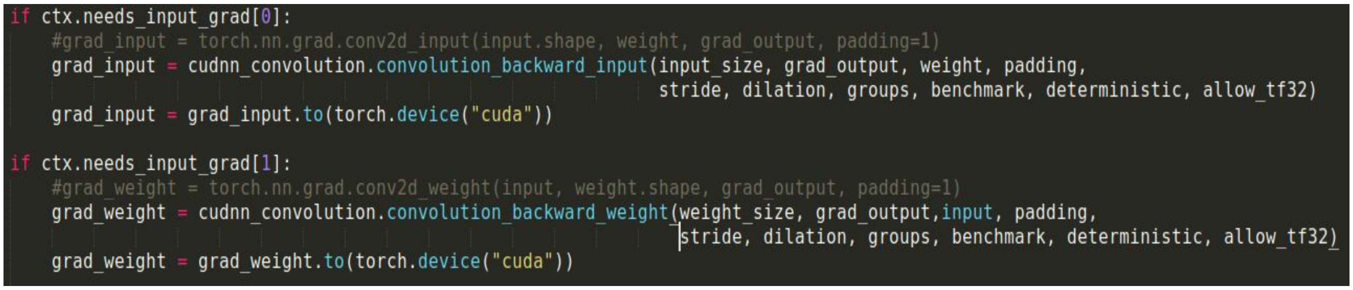
UPDATE - 4: CUSTOM CONV LAYER - Training & Inference

1. **Custom conv layer:**
    - The idea of training the custom layer from scratch seemed too slow. Even after calling of cudnn C++ APIs for convolution operation in backward(), it was too slow to train the model as is.

      Code snippet of calling cudnn C++ APIs for convolution operation:



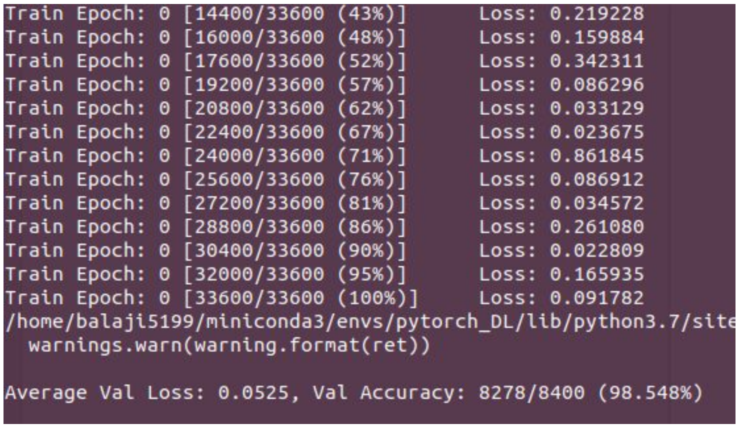
```
if ctx.needs_input_grad[0]:
    #grad_input = torch.nn.grad.conv2d_input(input.shape, weight, grad_output, padding=1)
    grad_input = cudnn_convolution.convolution_backward_input(input_size, grad_output, weight, padding,
                                                              stride, dilation, groups, benchmark, deterministic, allow_tf32)
    grad_input = grad_input.to(torch.device("cuda"))

if ctx.needs_input_grad[1]:
    #grad_weight = torch.nn.grad.conv2d_weight(input, weight.shape, grad_output, padding=1)
    grad_weight = cudnn_convolution.convolution_backward_weight(weight_size, grad_output,input, padding,
                                                                stride, dilation, groups, benchmark, deterministic, allow_tf32)
    grad_weight = grad_weight.to(torch.device("cuda"))
```

- The next step taken was to make two definitions of Network class, one having the pre-existing convolution layers - *Normie_Net* and the other containing the custom convolution layers *Net*. (as defined under *Net_py.py* file)

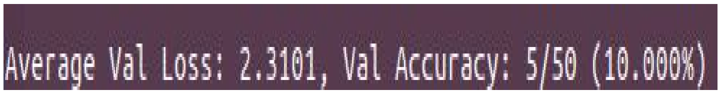
  The former one was used to train over the training dataset and the learned parameters are loaded into the network having a custom convolution layer defined under it.

- The implementation is done as follows - training and validation done on MNIST dataset of 10,000 28x28 images with 80-20 split. The testing was done on 50 images. The result is displayed below: a) validation accuracy; b) test accuracy



```
Train Epoch: 0 [14400/33600 (43%)]      Loss: 0.219228
Train Epoch: 0 [16000/33600 (48%)]      Loss: 0.159884
Train Epoch: 0 [17600/33600 (52%)]      Loss: 0.342311
Train Epoch: 0 [19200/33600 (57%)]      Loss: 0.086296
Train Epoch: 0 [20800/33600 (62%)]      Loss: 0.033129
Train Epoch: 0 [22400/33600 (67%)]      Loss: 0.023675
Train Epoch: 0 [24000/33600 (71%)]      Loss: 0.861845
Train Epoch: 0 [25600/33600 (76%)]      Loss: 0.086912
Train Epoch: 0 [27200/33600 (81%)]      Loss: 0.034572
Train Epoch: 0 [28800/33600 (86%)]      Loss: 0.261080
Train Epoch: 0 [30400/33600 (90%)]      Loss: 0.022809
Train Epoch: 0 [32000/33600 (95%)]      Loss: 0.165935
Train Epoch: 0 [33600/33600 (100%)]     Loss: 0.091782
/home/balaji5199/miniconda3/envs/pytorch_DL/lib/python3.7/site
  warnings.warn(warning.format(ret))

Average Val Loss: 0.0525, Val Accuracy: 8278/8400 (98.548%)
```



```
Average Val Loss: 2.3101, Val Accuracy: 5/50 (10.000%)
```

- Problems:
    1. The test accuracy is very low when done with the custom layer loaded with trained weights. (~10%) and the average test loss also seems high.
    2. The time taken to even run on 50 test images takes up way more than expected time.
    3. When custom convolution layer is replaced in place of a pre-existing conv layer with high number of trainable parameters, the code simply doesn't execute because its too slow.

- Possible reasons are:
    1. There must be a mismatch in the convolution implementation in the pre-existing conv layers and the custom conv layers.
    2. Need for a better and improved implementation of the convolution operation.

*The **custom_conv2d.py** code has not yet been updated into github repo. Some changes are still being made.*
*The updated **Net_py.py** has been pushed into the github repo.*

Some useful links that I have used have been given below.[1, 2]

2. Brief Working Detail:

- **Software/ packages details :-**
  python 3.7.9; pytorch 1.7.0; torchvision 0.8.1; torchsummary 1.5.1; cudatoolkit 10.2.89; And other common packages in their latest versions

- **Github Repo:-** Constant update to the codes done here.
  https://github.com/bALAJi-aDItHYa/MBM_implementation.git

3. Further Work:

- Look into the C++ backend implementation of Conv2d and try to replicate it. - look into C++ - pytorch backend

4. References:

[1] Pytorch C++ backend

1) https://pytorch.org/cppdocs/api/file_torch_csrc_api_include_torch_nn_functional_conv.h.html#file-torch-csrc-api-include-torch-nn-functional-conv-h - conv.h header file
2) https://pytorch.org/docs/stable/_modules/torch/nn/modules/conv.html
3) https://pytorch.org/tutorials/advanced/cpp_frontend.html - possible lead? (c++ front-end for pytorch)

[2] Some doubts raised in PyTorch forum

1) https://discuss.pytorch.org/t/runtimeerror-tensor-for-argument-2-weight-is-on-cpu-but-expected-it-to-be-on-gpu-while-checking-arguments-for-cudnn-batch-norm-doubt/111385
2) https://discuss.pytorch.org/t/improve-the-efficiency-of-custom-conv2d-layer/111847