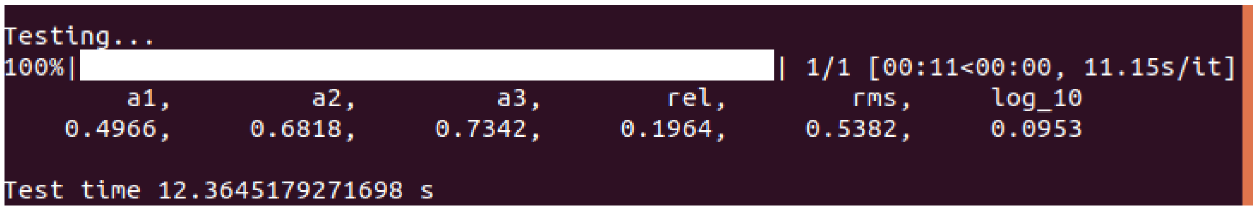
29_30_March_Update

1. The trained model specs are as follows:
   Encoder type: MobileNetv2
   # images (train data): 10K images from NYUv2 depth dataset
   # epochs: 5
   Batch_size: 16
   lr: 0.0001

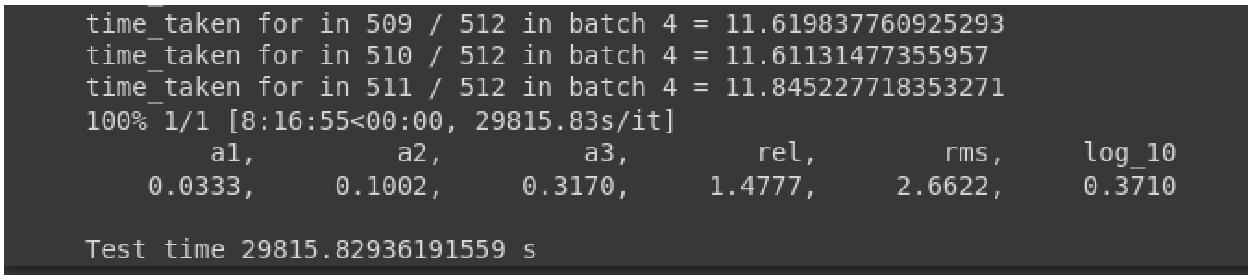
   The expected result for inference over 5 images is given below (with Accurate MobileNetv2 model)



```
Testing...
100%|██████████████████████████████████████████████| 1/1 [00:11<00:00, 11.15s/it]
        a1,         a2,         a3,        rel,        rms,     log_10
    0.4966,     0.6818,     0.7342,     0.1964,     0.5382,     0.0953

Test time 12.3645179271698 s
```

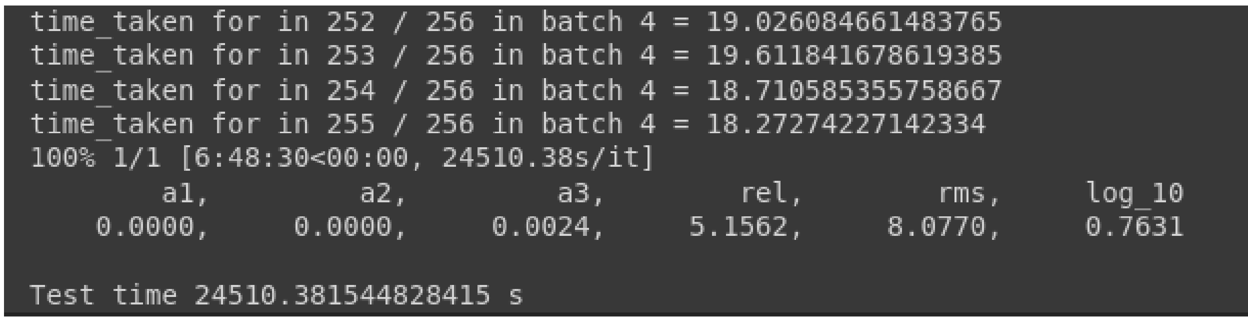
   The following inferences were given over the same 5 images from the NYU test data set. The inference code will be shared in the github repo; all the inferences were run on Google-colab - with the custom convolution part implemented on Python (not C++ extension). The inference was done for following 4 layers:

   a. **Decoder.conv2**



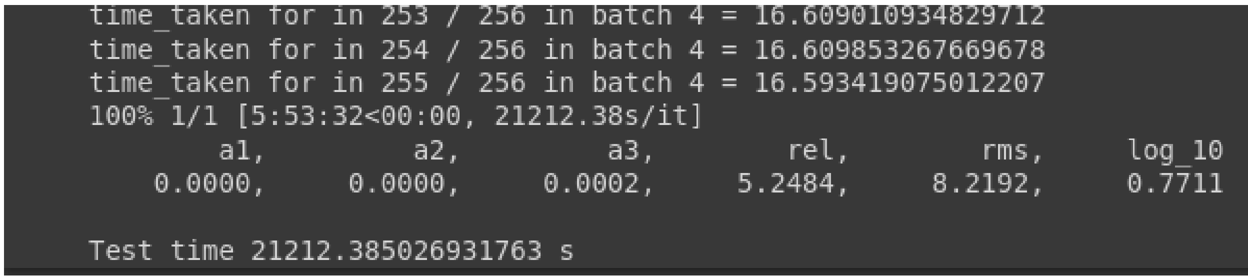
```
time_taken for in 509 / 512 in batch 4 = 11.619837760925293
time_taken for in 510 / 512 in batch 4 = 11.61131477355957
time_taken for in 511 / 512 in batch 4 = 11.845227718353271
100% 1/1 [8:16:55<00:00, 29815.83s/it]
        a1,         a2,         a3,        rel,        rms,     log_10
    0.0333,     0.1002,     0.3170,     1.4777,     2.6622,     0.3710

Test time 29815.82936191559 s
```

   b. **Decoder.up0.convB**



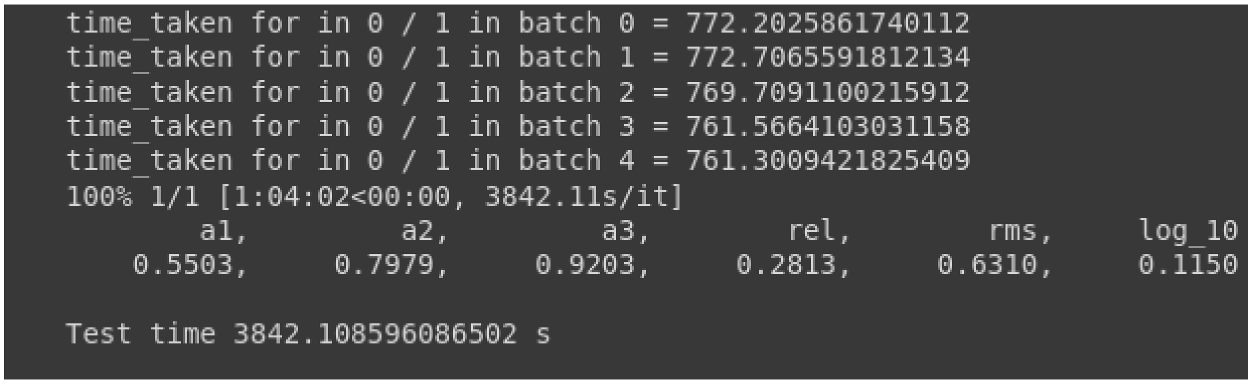
```
time_taken for in 252 / 256 in batch 4 = 19.026084661483765
time_taken for in 253 / 256 in batch 4 = 19.611841678619385
time_taken for in 254 / 256 in batch 4 = 18.710585355758667
time_taken for in 255 / 256 in batch 4 = 18.27274227142334
100% 1/1 [6:48:30<00:00, 24510.38s/it]
        a1,         a2,         a3,        rel,        rms,     log_10
    0.0000,     0.0000,     0.0024,     5.1562,     8.0770,     0.7631

Test time 24510.381544828415 s
```

c. **Decoder.up1.convB**



```
time_taken for in 253 / 256 in batch 4 = 16.609010934829712
time_taken for in 254 / 256 in batch 4 = 16.609853267669678
time_taken for in 255 / 256 in batch 4 = 16.593419075012207
100% 1/1 [5:53:32<00:00, 21212.38s/it]
         a1,          a2,          a3,         rel,         rms,      log_10
     0.0000,      0.0000,      0.0002,      5.2484,      8.2192,      0.7711

Test time 21212.385026931763 s
```

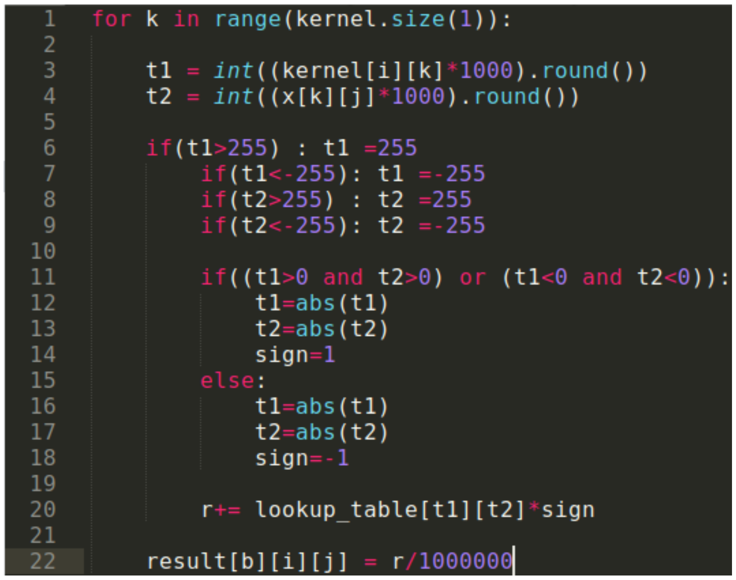
d. **Decoder.conv3**



```
time_taken for in 0 / 1 in batch 0 = 772.2025861740112
time_taken for in 0 / 1 in batch 1 = 772.7065591812134
time_taken for in 0 / 1 in batch 2 = 769.7091100215912
time_taken for in 0 / 1 in batch 3 = 761.5664103031158
time_taken for in 0 / 1 in batch 4 = 761.3009421825409
100% 1/1 [1:04:02<00:00, 3842.11s/it]
         a1,          a2,          a3,         rel,         rms,      log_10
     0.5503,      0.7979,      0.9203,      0.2813,      0.6310,      0.1150

Test time 3842.108596086502 s
```

2. The error-metric values in the case of **conv2**, **up0.convB**, **up1.convB** were observed to be less compared to the expected values. Hence the following changes are made in the code to correct the metrics.

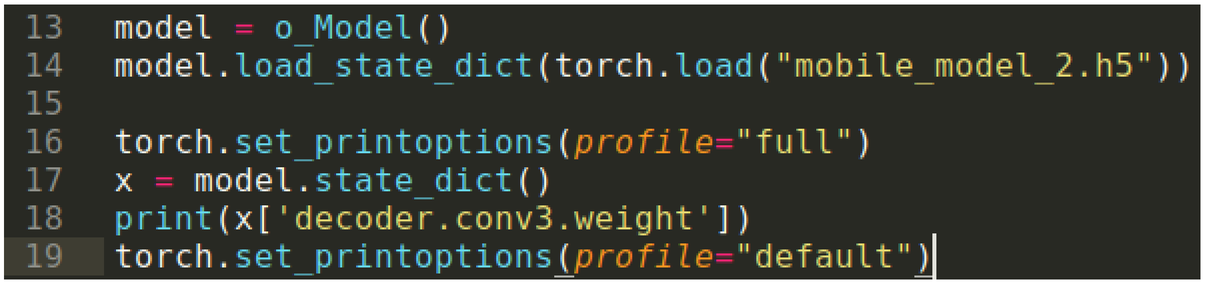
   The reason for the low accuracy is because of the scaling that is done in the



```
for k in range(kernel.size(1)):

    t1 = int((kernel[i][k]*1000).round())
    t2 = int((x[k][j]*1000).round())

    if(t1>255) : t1 =255
        if(t1<-255): t1 =-255
        if(t2>255) : t2 =255
        if(t2<-255): t2 =-255

        if((t1>0 and t2>0) or (t1<0 and t2<0)):
            t1=abs(t1)
            t2=abs(t2)
            sign=1
        else:
            t1=abs(t1)
            t2=abs(t2)
            sign=-1

        r+= lookup_table[t1][t2]*sign

    result[b][i][j] = r/1000000
```

custom convolution operation before lookup. The code snippet is attached above; As it is seen, the scalar scaling up and scaling down operations are done on the incoming multiplicands in **lines 3**, **line 4** and **line 22** respectively. The constant scaling factor of **x1000** was chosen based on values of majority of elements in the image and kernel tensors of the DenseNet-161 architecture.

But this scaling up factor of **x1000** needs some correction for the MobileNetv2 architecture. The script to find the weight values of conv3 layer is given below:



```
model = o_Model()
model.load_state_dict(torch.load("mobile_model_2.h5"))

torch.set_printoptions(profile="full")
x = model.state_dict()
print(x['decoder.conv3.weight'])
torch.set_printoptions(profile="default")
```

The majority of the values in the kernel tensor are in the range of 10^{-2}, hence a scaling factor of **x100** seemed suitable for the kernel. Similarly after monitoring the values of the input image (to the conv3 layer) it was found that the majority of the values were in the range of 10^{0} and 10^{-1}, hence a scaling factor of **x10** seemed suitable for image. After applying these scaling factors to the custom conv operation, the accuracy values of the **decoder.conv3** values improved and are nearer to the expected result.

The tabular form of accurate and approx model error metrics wil be updated here:
https://docs.google.com/spreadsheets/d/1tmXCuR8P1yGrYK8_bC07wBkrz_x6DhPl-a2OtFICZnw/edit?usp=sharing

3. Error metrics definitions:

Rel - average relative error $= 1/n \ * \ \sum |y_p - y'_p| \, / \, y$

RMS - root mean squared error = $\sqrt{(1/n \ * \ \sum (y_p - y'_p)^2)}$

Log10 - average log error = $1/n \ * \ \sum |log_{10}(y_p) - log_{10}(y'_p)|$

a_i - Threshold accuracy = % of y_p s.t.

$max(y_p/y'_p, \, y'_p/y_p) = a_i < thr, \, for\, thr = 1.25, \, 1.25^2, \, 1.25^3$

The definitions are referred from here: https://arxiv.org/abs/1812.11941