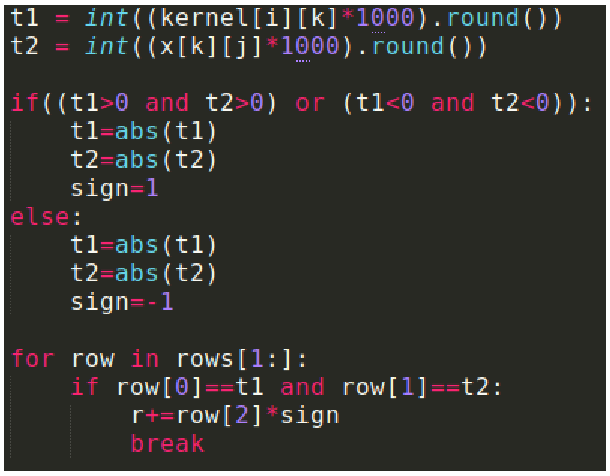
18th March _ Updates

1. Implemented the Code changes in the custom convolution operation inorder to make the lookup more efficient.
   reference_mbm.csv is the file from which I am reading the MBM product values and storing it in a list - **rows**

   Below are the 3 approaches that I tried out.
   a. **The row-wise search approach:**



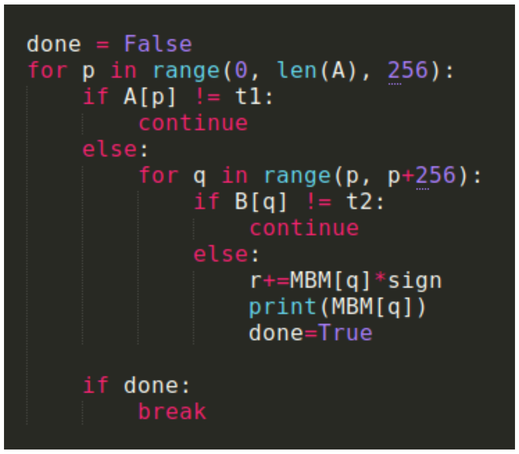
```python
t1 = int((kernel[i][k]*1000).round())
t2 = int((x[k][j]*1000).round())

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

for row in rows[1:]:
    if row[0]==t1 and row[1]==t2:
        r+=row[2]*sign
        break
```

t1 and t2 are the input operands and a search is run through the rows list to find the corresponding product value. This approach in it's worst case had to perform 256x256=65536 search operations before looking up the product value.

   b. **Column-wise search:**

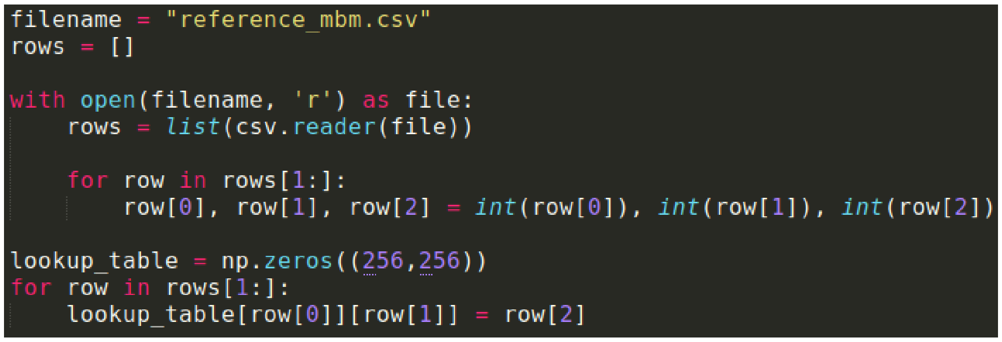


```python
done = False
for p in range(0, len(A), 256):
    if A[p] != t1:
        continue
    else:
        for q in range(p, p+256):
            if B[q] != t2:
                continue
            else:
                r+=MBM[q]*sign
                print(MBM[q])
                done=True

    if done:
        break
```

List(A) -> 1st column, List(B) -> 2nd column, List(MBM) -> 3rd column from the *reference.csv* file. In this approach's worst case scenario, it would only perform 255+256=511 search operations before finding the product value based on the operands t1 and t2.

c. **Look-up table approach:**
Created a look-up table so that the inputs themselves can act as pointers to location of the product value rather than performing search operations.



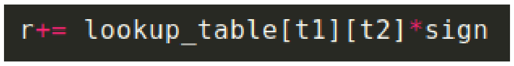
```
filename = "reference_mbm.csv"
rows = []

with open(filename, 'r') as file:
    rows = list(csv.reader(file))

    for row in rows[1:]:
        row[0], row[1], row[2] = int(row[0]), int(row[1]), int(row[2])

lookup_table = np.zeros((256,256))
for row in rows[1:]:
    lookup_table[row[0]][row[1]] = row[2]
```

The lookup_table is filled with product values based on each position's index value.



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

Hence the lookup time is reduced to constant time complexity **O(1)**. This is **currently** used in the implementation of custom conv.

2. Ran 3 inferences using Google Colab: (inference over 1 image)
   a. **Run 1** - Using row-wise approach => ran for **923 / 1104** iterations before disconnecting from the server as the maximum limit for usage of VM in Google Colab is 12 hrs at a time. Hence it disconnected due to time-out.

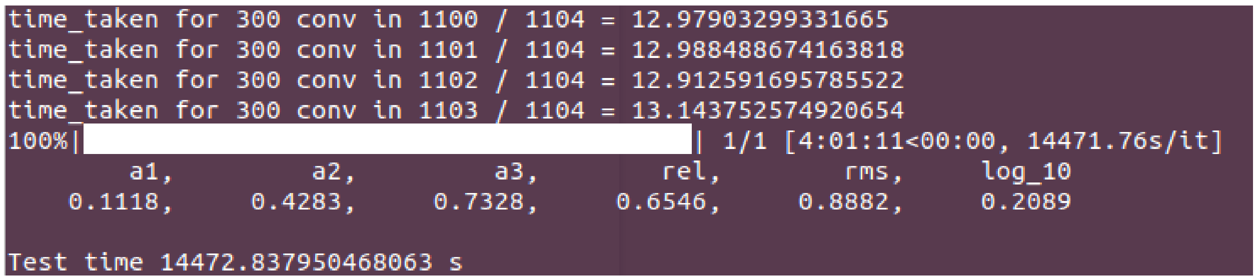
   b. **Run-2** - Using look-up table approach => ran for **672 / 1104** iterations, but the colab sheet disconnected automatically. This is reported to happen frequently in the Colab sheet, as Google tries to allocate VMs based on activity on the sheet rather than background runs. Ran for ~3.5 hours

   c. **Run-3** - Using look-up table approach => ran for entire **1104/1104** iterations for ~7.5 hrs, but encountered with a runtime error.
   RuntimeError: The expanded size of the tensor (20) must match the existing size (1104) at non-singleton dimension 3. Target sizes: [1, 1104, 15, 20]. Tensor sizes: [1, 1104]

Link to colab sheet to check the console after each run:
https://colab.research.google.com/drive/1hR70s85isOyWzyo0-o8970rDvmbcUz6R?usp=sharing

3. Ran inference over 1 image on my local machine. Completed in ~4 hours. Here are the results in terms of error metrics. (Loaded model was trained on 2500 images, 10 epochs, batch_size=4, lr=0.0001)

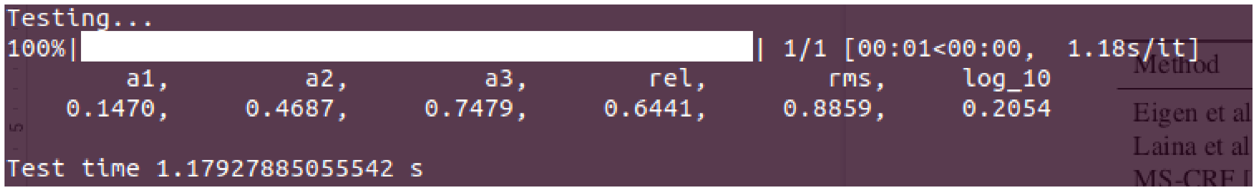
   (a): Inference on 1 image using approx model



```
time_taken for 300 conv in 1100 / 1104 = 12.97903299331665
time_taken for 300 conv in 1101 / 1104 = 12.988488674163818
time_taken for 300 conv in 1102 / 1104 = 12.912591695785522
time_taken for 300 conv in 1103 / 1104 = 13.143752574920654
100%|████████████████████████████████████████| 1/1 [4:01:11<00:00, 14471.76s/it]
        a1,         a2,         a3,        rel,        rms,     log_10
    0.1118,     0.4283,     0.7328,     0.6546,     0.8882,     0.2089

Test time 14472.837950468063 s
```

   (b): Inference on 1 image using accurate model

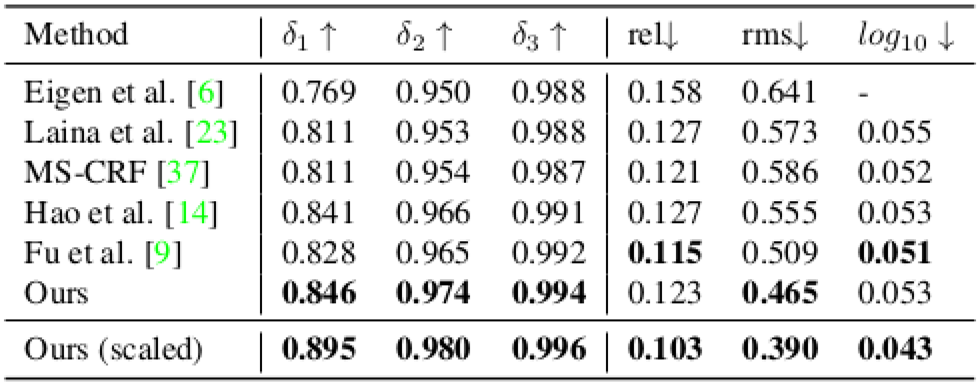


```
Testing...
100%|████████████████████████████████████████████| 1/1 [00:01<00:00,  1.18s/it]
        a1,         a2,         a3,        rel,        rms,     log_10
    0.1470,     0.4687,     0.7479,     0.6441,     0.8859,     0.2054

Test time 1.17927885055542 s
```

   From (a) and (b), it can be observed that the error metrics are comparable between approx and the accurate models.

   The table below depicts the error metrics from the State of the Art (SOTA).



Method	$\delta_1 \uparrow$	$\delta_2 \uparrow$	$\delta_3 \uparrow$	rel$\downarrow$	rms$\downarrow$	$log_{10} \downarrow$
Eigen et al. [6]	0.769	0.950	0.988	0.158	0.641	-
Laina et al. [23]	0.811	0.953	0.988	0.127	0.573	0.055
MS-CRF [37]	0.811	0.954	0.987	0.121	0.586	0.052
Hao et al. [14]	0.841	0.966	0.991	0.127	0.555	0.053
Fu et al. [9]	0.828	0.965	0.992	**0.115**	0.509	**0.051**
Ours	**0.846**	**0.974**	**0.994**	0.123	**0.465**	0.053
Ours (scaled)	**0.895**	**0.980**	**0.996**	**0.103**	**0.390**	**0.043**

4. Running on Colab vs Local machine; Request for access to TU Dresden's Server

   Running inference on colab seems to have a lot of disadvantages due to various policies set up by Google:

   a. Access to VM/ GPU only for 12hrs at a time after which it automatically disconnects.

b. Occasionally, If there is inactivity on the tab on which the google colab sheet is open for more than 30-90 minutes, it automatically disconnects. This prevents inference to be conducted overnight.
c. The resources provided are imbalanced. The time taken for 1 iteration (300 convolutions) keeps fluctuating, for unknown reasons, ranging from anywhere between 17s - 25s. This results in unnecessary increase in time to conduct inference over each image.
d. Overall, the process isn't very reliable as further down the line we need to run inference over 10 images, it may take ~2 days to complete and reliability would be key.

Running inference on my local machine proved to be far more efficient and reliable:

a. The time taken for 1 iteration is (~13s) => the inference time was 2x faster when run on my PC. The inference time for 1 image took 4 hrs.
b. Stable and reliable.
c. My PC may not be able to handle running inference over 10 images for ~40 hours (going by the time taken for 1 img). I have a 2GB GTX 1050 Ti GPU.

I request for access to TU Dresden's server, so that it would be a more reliable and efficient way to run inference over a larger set of test images.

5. I have updated the files and functions with appropriate headers for better understanding of the functionality and changes done by me. I have pushed the codes on Github repo. https://github.com/bALAJi-aDItHYa/MBM_implementation.git