11_to_14th April Update

1. Mobilenet_v2 is used as the encoder in the encoder-decoder model that we are using for inference for depth estimation. Hence here is a general overview of the Mobilenet_v2 architecture (as described in [1]):

Operator	t	c	n	s
conv2d	-	32	1	2
bottleneck	1	16	1	1
bottleneck	6	24	2	2
bottleneck	6	32	3	2
bottleneck	6	64	4	2
bottleneck	6	96	3	1
bottleneck	6	160	3	2
bottleneck	6	320	1	1
conv2d 1x1	-	1280	1	1
avgpool 7x7	-	-	1	-
conv2d 1x1	-	k	-	

Here each bottleneck block consists of the following convolution operations:

a. **1x1** point wise convolution - to increase the number of channels
b. **3x3** depthwise convolution (where groups = in_channels)
c. **1x1** pointwise convolution - (layer b + layer c = depthwise seperable conv)

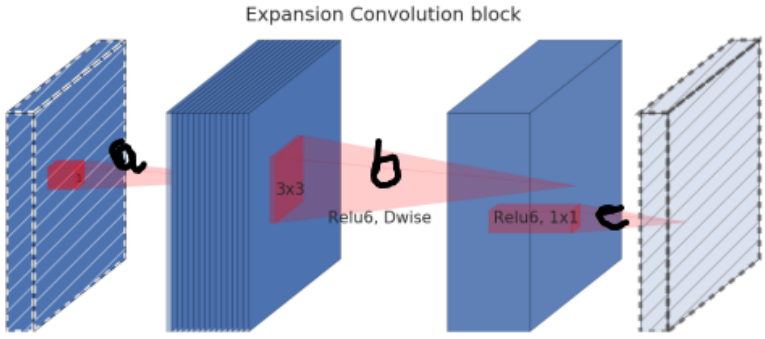


Fig: The figure represents the 3 different conv operations occurring in the bottleneck layer or the inverted Residual block

2. The parameters used to define these blocks are t,c,n and s:
    a. **t**: expansion coefficient - multiple by which the number of channels increases after 1x1 pointwise convolution in the bottleneck block
    b. **c:** out channels from each block.
    c. **n:** Number of inverted residual units
    d. **s:** Stride of current block - this info is fed into the necessary conv layer

3. Pointwise conv vs Depth-Wise separable convolution

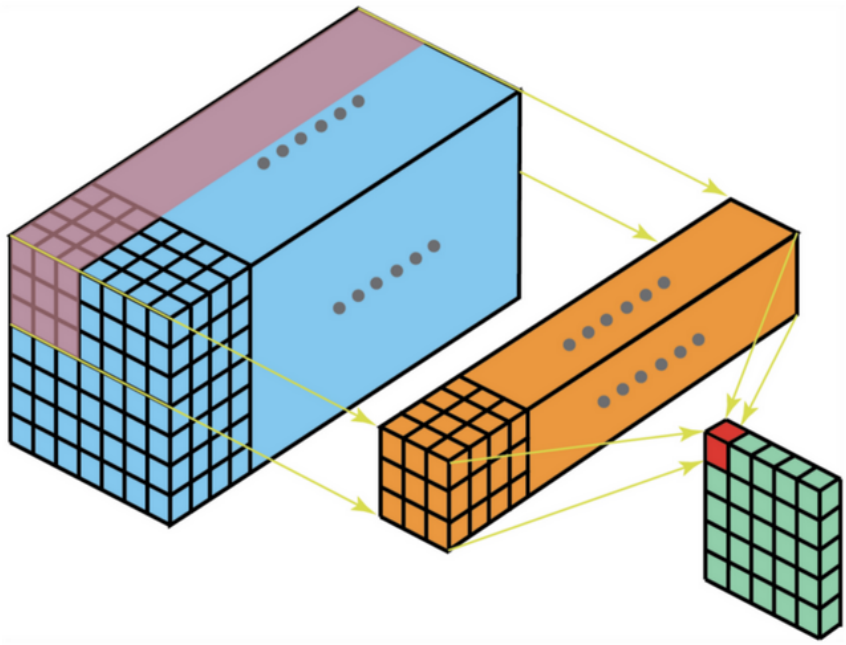


Fig: Pointwise convolution

Let's assume the following:

Input_shape = $h_i * w_i * d_i$

Kernel = $k * k * d_i * d_j$

Output = $h_i * w_i * d_j$

For performing pointwise convolution operation:
The computation cost = $h_i * w_i * d_i * d_j * k$*k

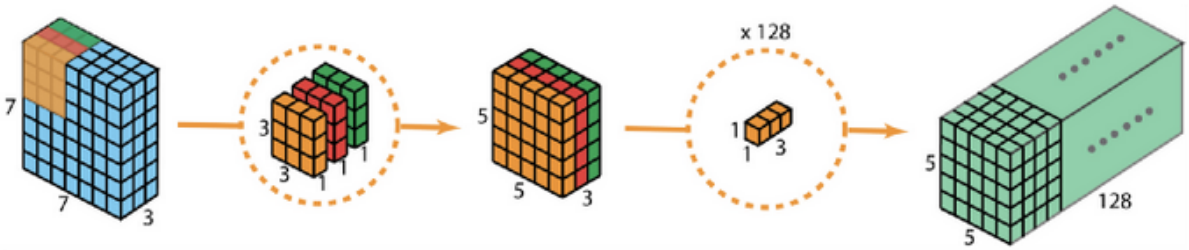


Fig: Depth-wise separable conv with example values

For performing depth-wise separable convolution:
Computation cost = $h_i * w_i * d_i * k * k + h_i * w_i * d_i * d_j * 1 * 1$

$$= h_i * w_i * d_i * k^2 * (1 + d_j/k^2)$$

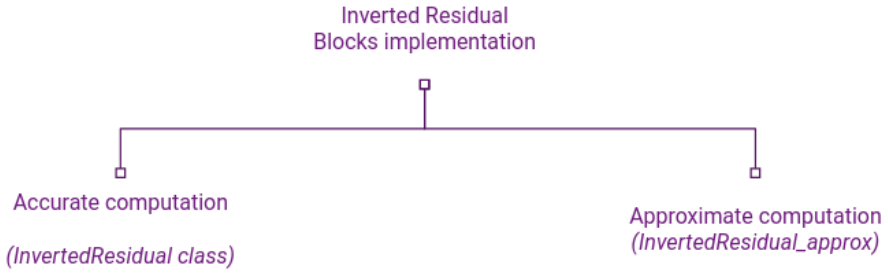
Where,
h_i, w_i = spatial dimensions of input image
d_i = input channels
d_j = output channels
k = kernel dimension

As it can be seen, the speedup by using depthwise separable convolution is about k^2 times. Hence it's used in most network architectures aimed at mobile devices such as MobileNet_v2.

4. The changes in encoder implementation has been provided below: (base model as described in official PyTorch implementation of mobilenetv2 - https://github.com/pytorch/vision/blob/master/torchvision/models/mobilenetv2.py)

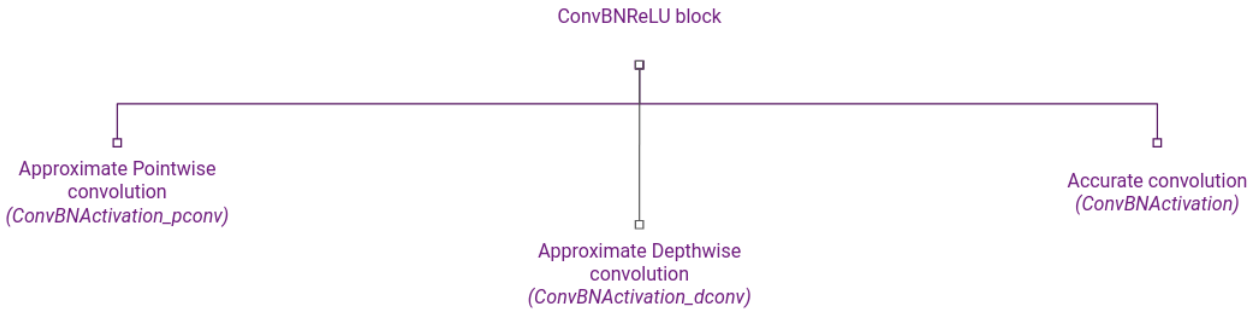
   There is the option to choose between the required specifications based on where the approximation has to be applied.

   A. For inverted Residual Block
   The unique block in which we need to perform approximation can be selected using the **n, c** values from the parameters list.



B. For ConvBNReLU layer

This block performs convolution, ReLU6 activation and batchnormalization operation on the input feature maps. The appropriate class can be selected depending on the number of groups (depthwise/ pointwise conv)



5. Approximate pointwise convolution - implementation details

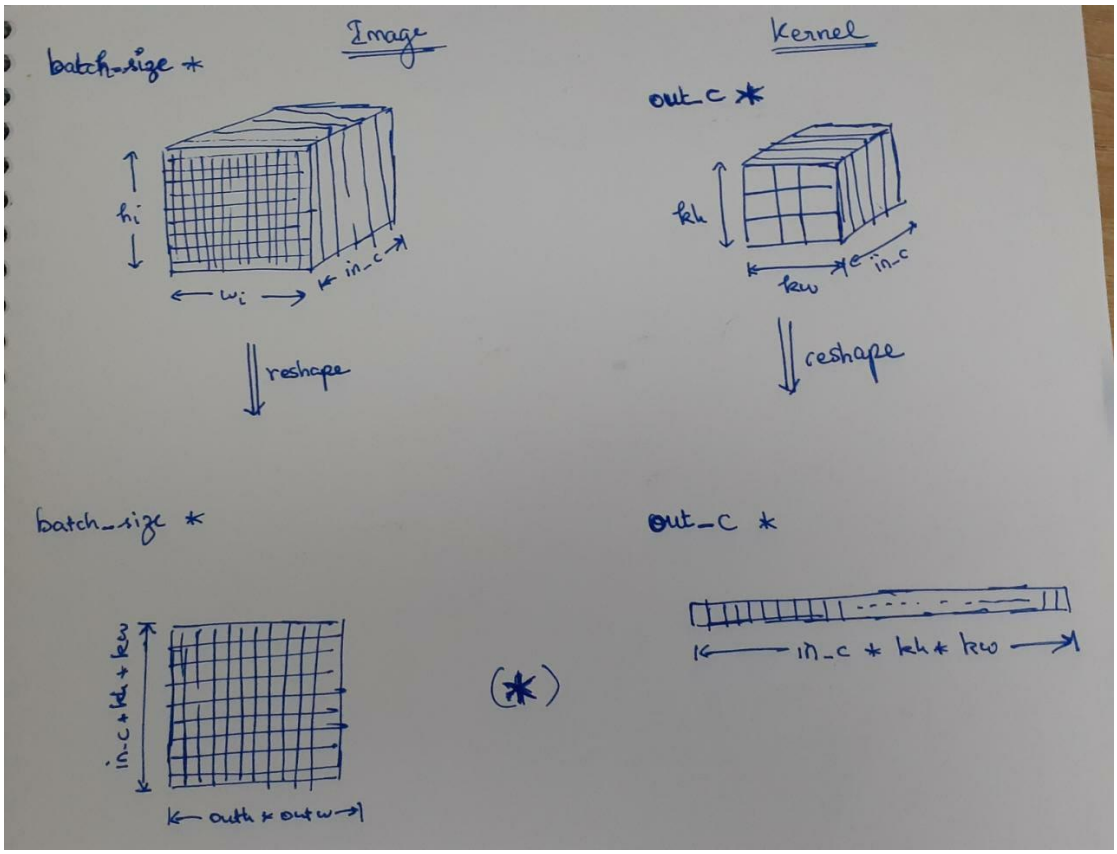


Fig: Describes the way in which the input feature map and kernel are reshaped to perform pointwise convolution. Elements from the Kernel vector are multiplied with each element of the resized_img column wise “*out_c”* number of times.

Hence it can be seen for 1 input --> the number of computations are
$(in_{channel} * k * k) * (out_h * out_w) * out_{channel}$

6. Approximate Depthwise convolution - implementation details

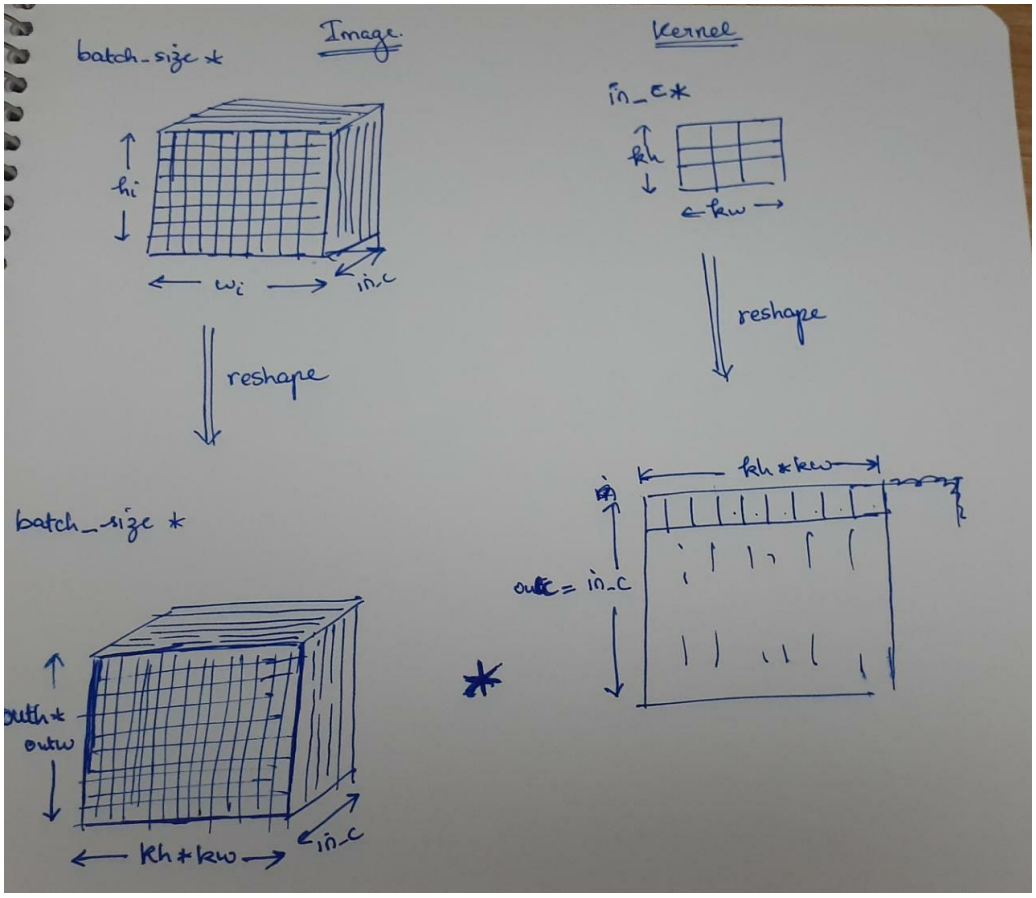


Fig: Input and kernel are reshaped in this manner to perform dwise conv. Here $out_{channel} = in_{channel} = groups$. Dot product is performed between I^{th} row of reshaped kernel to I^{th}channel of reshaped image.

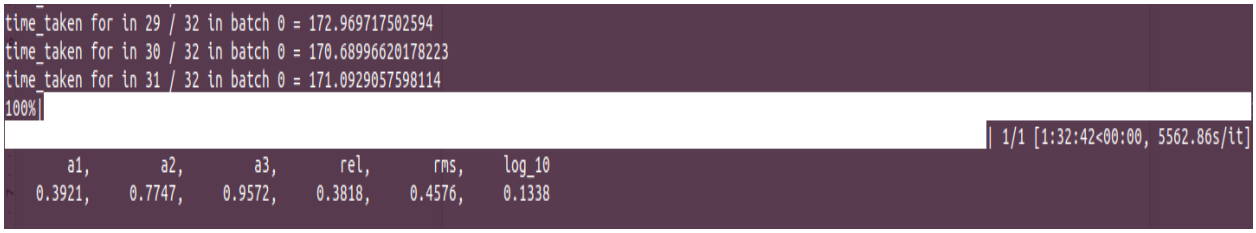
Hence for 1 input --> number of computations will be
$out_h * out_w * k * k * in_{channels}$ (where $in_{channels} = out_{channels}$)

7. Total number of layers over which inference has to be done:
   There are in total -
   1) Initial layer = 1 point-wise conv
   2) 1st bottleneck = 1 depth-wise + 1 point-wise conv
   3) 6 bottleneck layers => (each) = (1 depth-wise + 2 point-wise conv) * n
   4) Final layer = 1 point-wise conv

Hence in total = (16*2 + 1 +1 +1) pw and (16*1 + 1) dw conv
= 35 pw conv and 17 dw conv
= 52 conv layers in total

8. An inference run over 1 image was given for the Initial layer (pw conv) for 1 image and the results are shown below:

Encoder.0.0 - (runtime = 1hr 32min) - local machine



The naming scheme for each layer is followed according to the key values in the saved ***model.state_dict()***. Further updates for inference run on server as well as other layers will be provided in the subsequent updates.

9. References:
[1] https://arxiv.org/abs/1801.04381 - Mobilenet_v2 model description
[2] Images source - https://medium.com/@luis_gonzales/a-look-at-mobilenetv2-inverted-residuals-and-linear-bottlenecks-d49f85c12423
[3] https://github.com/pytorch/vision/blob/master/torchvision/models/mobilenetv2.py - PyTorch implementation of Mobilenet_v2

The updated codes for Encoder inference has been added in the github repo for your reference.