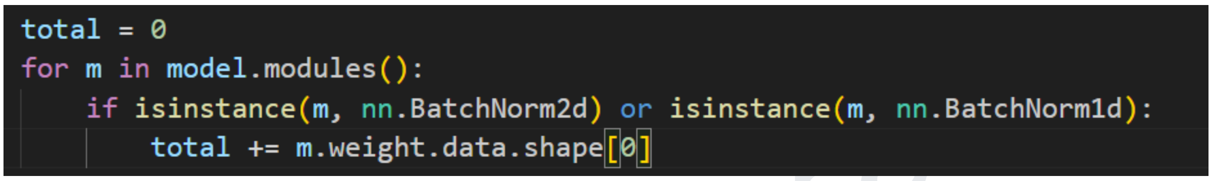
Network Sliming Documentation

It slims the model on the basis of the scaling factor of BatchNorm layers taking in input as model and percentage of pruning.

Scaling Factor: It is a learnable parameter that controls the scale of the normalised outputs.It is used in batchnorm layers and if its value is near zero then it don't contribute to accuracy while adding to computation cost.

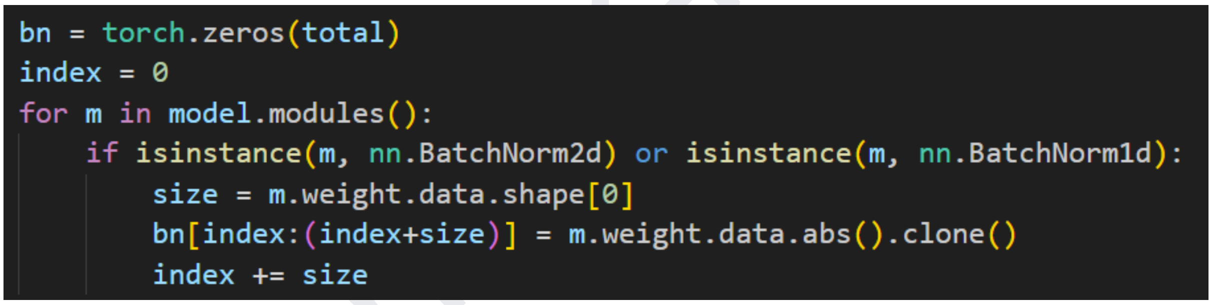
#WORKING

1. It first calculates the total number of BatchNorm layer



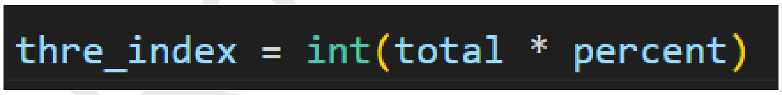
```
total = 0
for m in model.modules():
    if isinstance(m, nn.BatchNorm2d) or isinstance(m, nn.BatchNorm1d):
        total += m.weight.data.shape[0]
```

2. Then it makes a tensor named "bn" which stores the scaling factor of each batch norm layers and also an index is maintained.



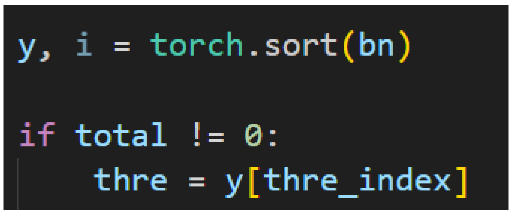
```
bn = torch.zeros(total)
index = 0
for m in model.modules():
    if isinstance(m, nn.BatchNorm2d) or isinstance(m, nn.BatchNorm1d):
        size = m.weight.data.shape[0]
        bn[index:(index+size)] = m.weight.data.abs().clone()
        index += size
```

3. Then it calculates the threshold_index based on the percentage of pruning you want to apply.



```
thre_index = int(total * percent)
```

4. Then the tensor containing the scaling factor of each batchnorm layer is sorted and using the threshold_index we calculate the threshold .

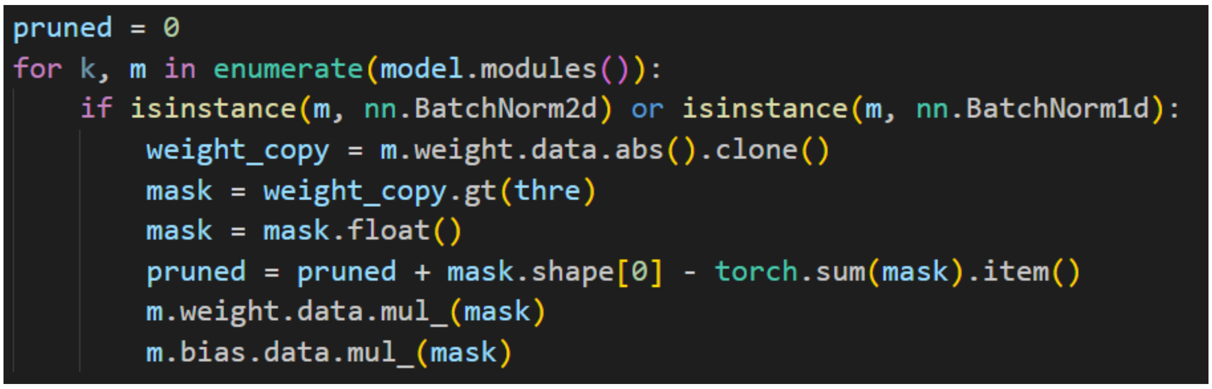


```
y, i = torch.sort(bn)

if total != 0:
    thre = y[thre_index]
```

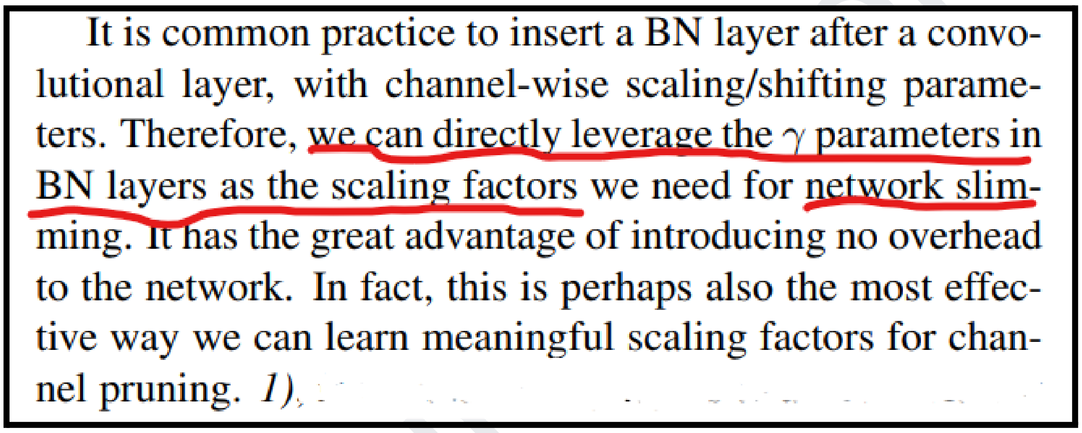
5. Then it creates a mask by checking that which weights are above threshold and which are not

6. Then the mask is applied on actual weights of the model leading some weights of the model to run zero which makes the model smaller and lighter.

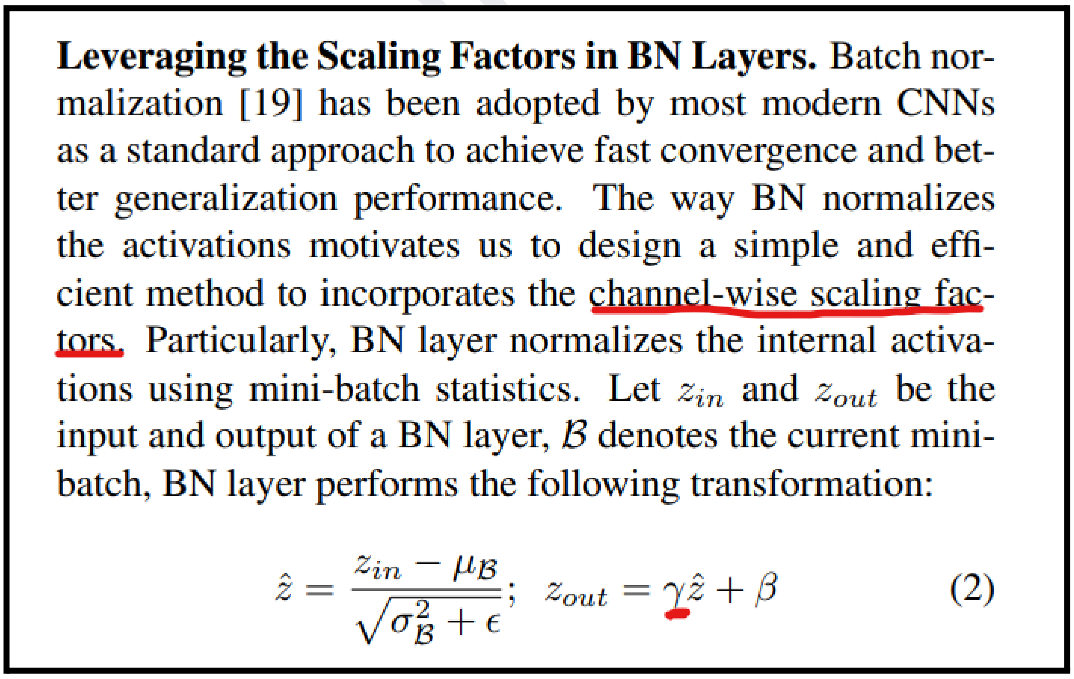


```python
pruned = 0
for k, m in enumerate(model.modules()):
    if isinstance(m, nn.BatchNorm2d) or isinstance(m, nn.BatchNorm1d):
        weight_copy = m.weight.data.abs().clone()
        mask = weight_copy.gt(thre)
        mask = mask.float()
        pruned = pruned + mask.shape[0] - torch.sum(mask).item()
        m.weight.data.mul_(mask)
        m.bias.data.mul_(mask)
```

Supporting Paper: Learning Efficient Convolutional Networks through Network Slimming (https://arxiv.org/abs/1708.06519)

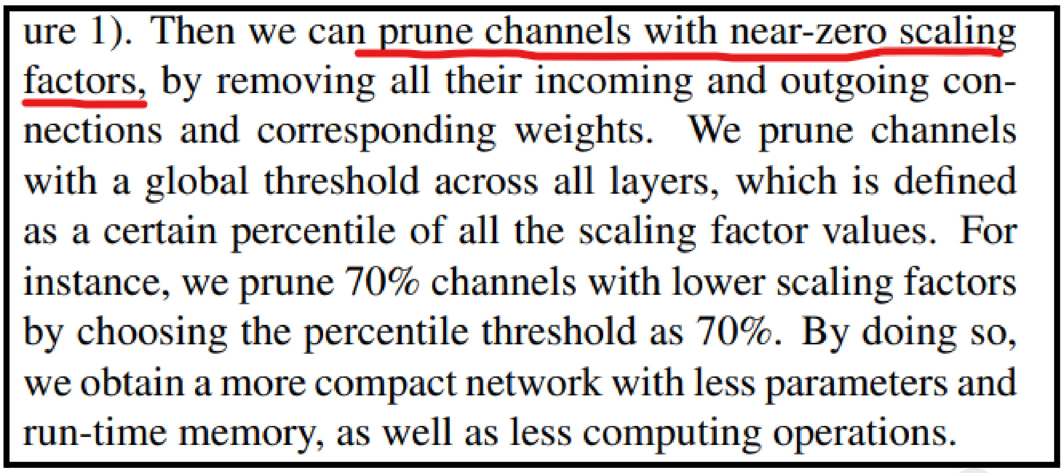


It is common practice to insert a BN layer after a convolutional layer, with channel-wise scaling/shifting parameters. Therefore, we can directly leverage the γ parameters in BN layers as the scaling factors we need for network slimming. It has the great advantage of introducing no overhead to the network. In fact, this is perhaps also the most effective way we can learn meaningful scaling factors for channel pruning. *1)*,



Leveraging the Scaling Factors in BN Layers. Batch normalization [19] has been adopted by most modern CNNs as a standard approach to achieve fast convergence and better generalization performance. The way BN normalizes the activations motivates us to design a simple and efficient method to incorporates the channel-wise scaling factors. Particularly, BN layer normalizes the internal activations using mini-batch statistics. Let z_{in} and z_{out} be the input and output of a BN layer, $\mathcal{B}$ denotes the current mini-batch, BN layer performs the following transformation:

$$\hat{z} = \frac{z_{in} - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}; \quad z_{out} = \gamma \hat{z} + \beta \tag{2}$$



ure 1). Then we can prune channels with near-zero scaling factors, by removing all their incoming and outgoing connections and corresponding weights. We prune channels with a global threshold across all layers, which is defined as a certain percentile of all the scaling factor values. For instance, we prune 70% channels with lower scaling factors by choosing the percentile threshold as 70%. By doing so, we obtain a more compact network with less parameters and run-time memory, as well as less computing operations.