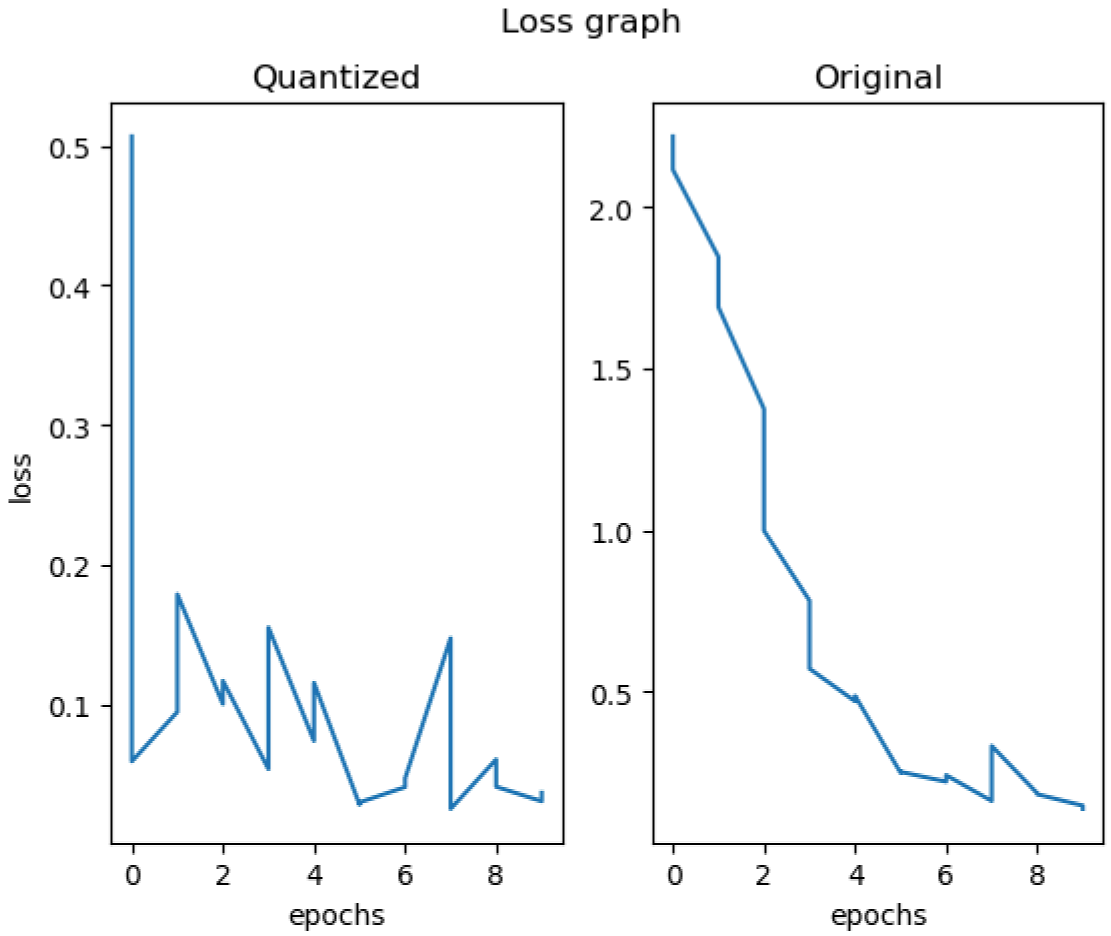
Name: Abel Haris Harsono
UID: 303583434

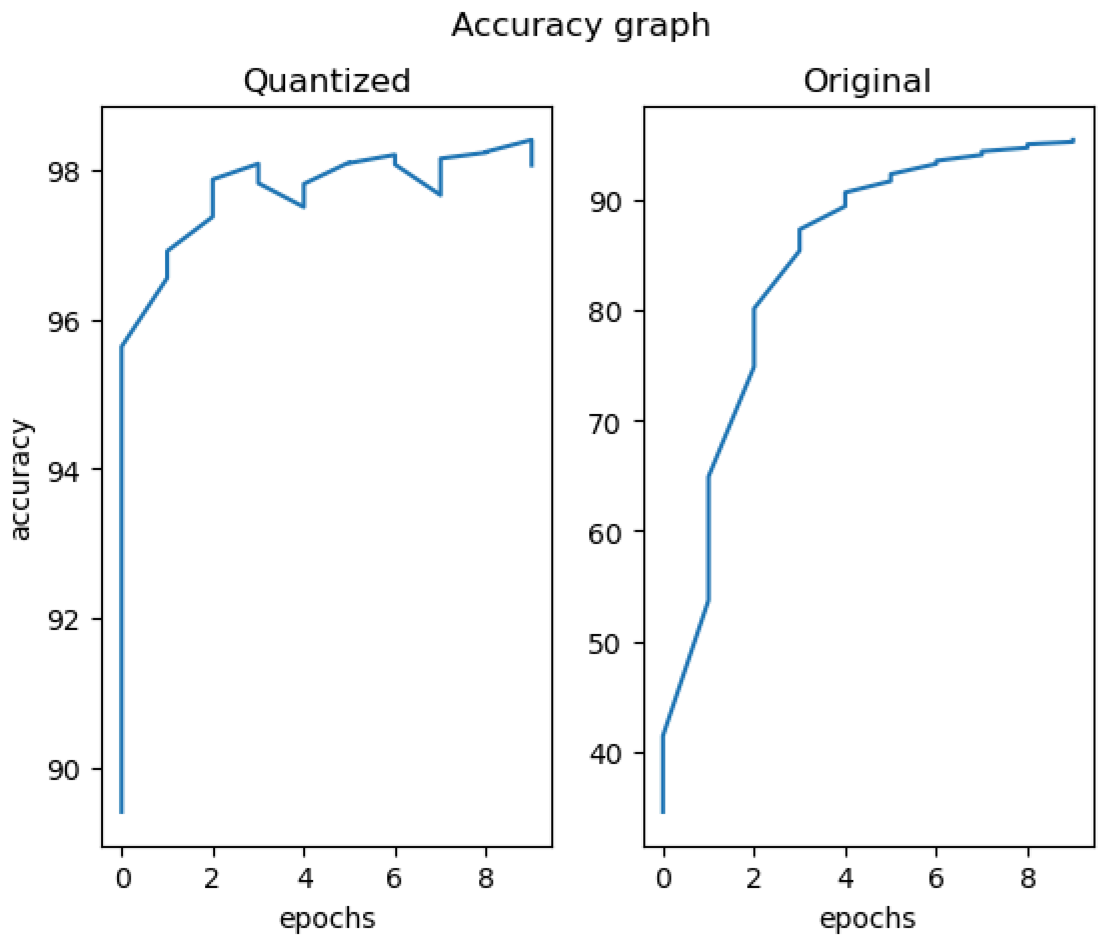
II.1. Implementation of LSQ on LeNet-5
As per finding, a model size won't necessarily determine the accuracy of the model itself since in this case, as the model size decreases, the accuracy gets better. Though that may be because of the fact that the quantization training starts with the weights of the originally trained model.

Finding Summaries
- compression rate = ~28 % (original model size = 253.586 KB)
- final accuracy of quantized model = ~ 95 %
- final accuracy of original model = ~ 98 %

Based on the graph the loss function seems to exhibit similar trend:



Other than the sudden drop in loss, both graphs seem to have a general downward trend. That initial huge drop is because the parameters of the quantized model are initialized from the trained model.

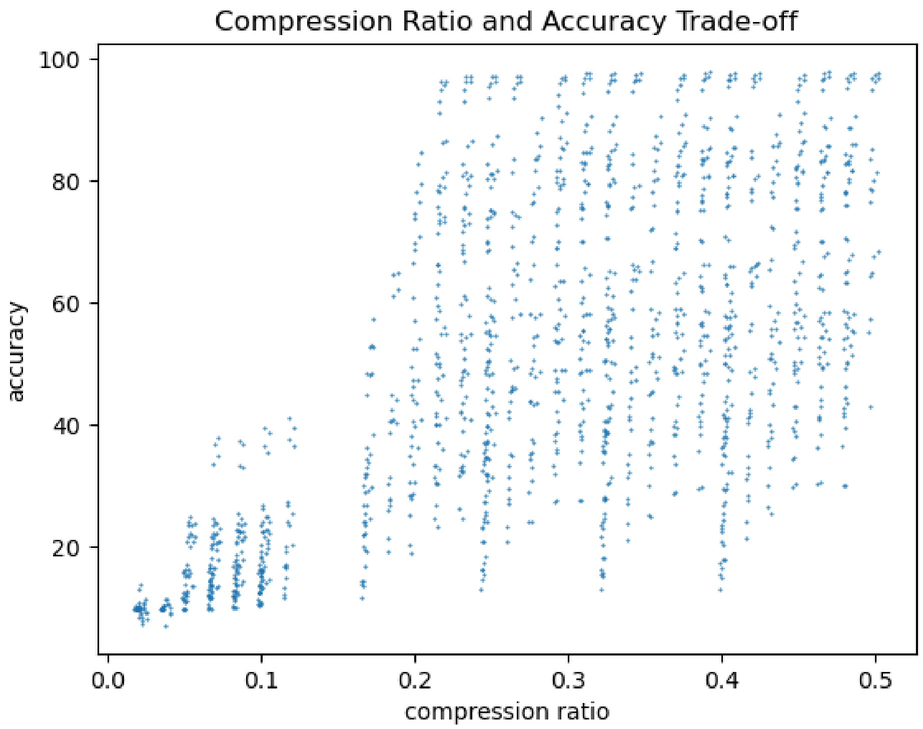


The accuracy here also shows a similar trend as the loss with the sudden spikes every now and then. The higher accuracy is also because of the weights being initialized from the original trained model.

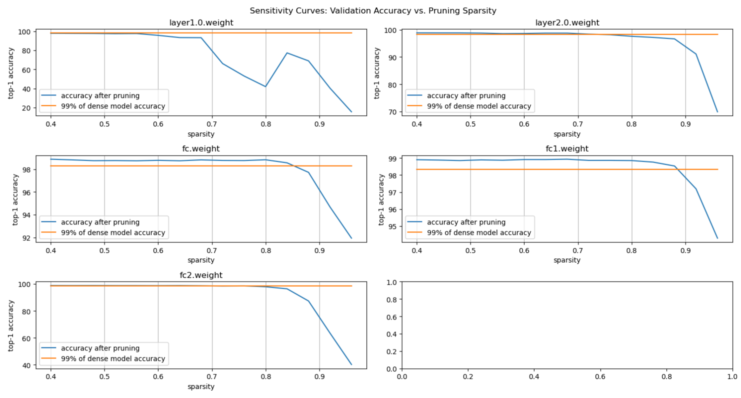
Note: For calculating size, I directly saved the relevant parameters of each model

II.2. Magnitude Pruning (fine grained)

Magnitude pruning algorithm: Fine grained pruning where the importance of each element is indicated by their square



It's pretty hard to say how the compression ratio will directly affect the accuracy as shown by the graph. This is mainly due to fact that each layer of the LeNet5 having a different sensitivity to pruning:



So naturally, having the same compression ratio won't necessarily produce the same level of accuracy.

In the general case however, the graph seems to suggest that bigger models would be more likely to acquire greater accuracy.

II.3. Structured Pruning

Pruning Pattern

- Structure: Channel Pruning
- Sparsity Ratio: 0.3 for all layers

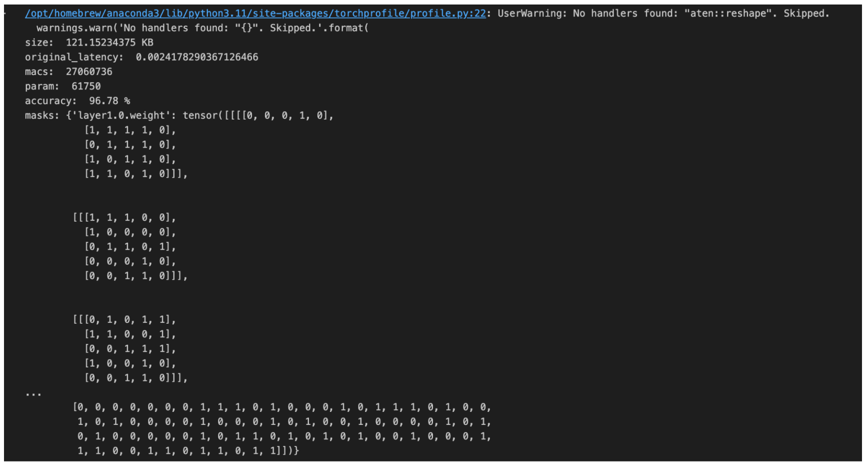
Channel pruned:

```
[tensor([[0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.]]),
 tensor([[0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.]]),
tensor([[0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.]]),
tensor([[1., 1., 1., 1., 1.],
        [1., 1., 1., 1., 1.],
        [1., 1., 1., 1., 1.],
        [1., 1., 1., 1., 1.],
        [1., 1., 1., 1., 1.]]),
tensor([[0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.]]),
tensor([[1., 1., 1., 1., 1.],
        [1., 1., 1., 1., 1.],
        [1., 1., 1., 1., 1.],
        [1., 1., 1., 1., 1.],
        [1., 1., 1., 1., 1.]])]
```

Note: There is only one pattern as there are only 2 convolutional layers and the fcs are not channel pruned but are fine grained pruned as per **II.2**. This pattern applies to both output channels of the convolutional layers. In essence, it will only keep 2 out of the 6 5x5 matrix for each output channel of the conv layers.

- Structure: Fine grained pruned
- Sparsity: 50% of each layer

Fine grain prune: Too long to put here. Please refer to magnitude.ipynb at the very bottom



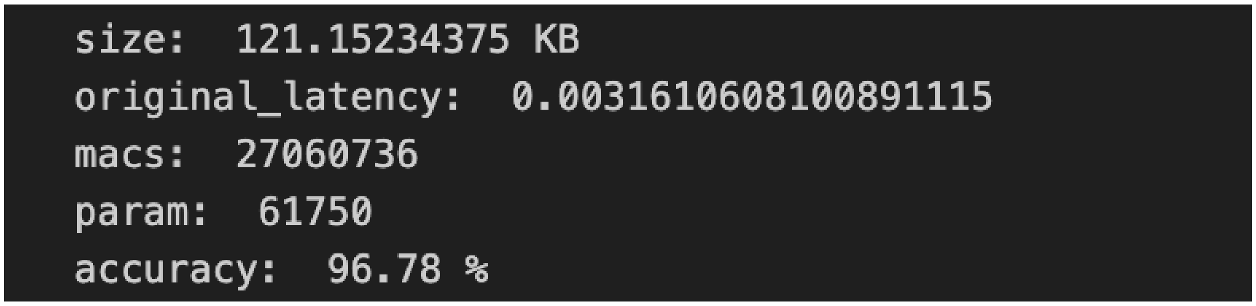
```
/opt/homebrew/anaconda3/lib/python3.11/site-packages/torchprofile/profile.py:22: UserWarning: No handlers found: "aten::reshape". Skipped.
  warnings.warn('No handlers found: "{}". Skipped.'.format(
size:  121.15234375 KB
original_latency:  0.0024178290367126466
macs:  27060736
param:  61750
accuracy:  96.78 %
masks: {'layer1.0.weight': tensor([[[[0, 0, 0, 1, 0],
          [1, 1, 1, 1, 0],
          [0, 1, 1, 1, 0],
          [1, 0, 1, 1, 0],
          [1, 1, 0, 1, 0]]],


        [[[1, 1, 1, 0, 0],
          [1, 0, 0, 0, 0],
          [0, 1, 1, 0, 1],
          [0, 0, 0, 1, 0],
          [0, 0, 1, 1, 0]]],


        [[[0, 1, 0, 1, 1],
          [1, 1, 0, 0, 1],
          [0, 0, 1, 1, 1],
          [1, 0, 0, 1, 0],
          [0, 0, 1, 1, 0]]],
...
        [0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0,
         1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1,
         0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1,
         1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1]])}
```

Speed
Channel Pruning: 0.0027301716804504395 s
Fine grained: 0.0031610608100891115 s

Although it is quite clear here that channel pruning wins over fine grained, the latency for fine grained is not truly shown. This is due to the fact that pytorch obviously won't ignore any parameter by simply changing it to 0. Instead, they keep on computing with that parameter. We can see this as we check the macs:



```
size:  121.15234375 KB
original_latency:  0.0031610608100891115
macs:  27060736
param:  61750
accuracy:  96.78 %
```

The macs and param are the same as per the original model.

Though, regardless, channel pruning should still win most of the time given a particular ratio in terms of latency since it gets rid of entire channels (can get rid of more weights for a given sparsity ratio compared to fine grain pruning). However, for speed and size, channel pruning does sacrifice a significant amount of accuracy.

Accuracy and Compression Rate
As mentioned, channel pruning provides considerably worse accuracy than its fine grained counterpart.

Here are some range of values:

Ratio	Accuracy	Size
0.1	85.06 %	120.21484375 KB
0.3	40.4 %	119.3046875 KB
0.5	27.25 %	118.54296875 KB
0.7	25.12 %	117.71484375 KB

As the table shows, with more percentage of the layer being pruned, the accuracy drops way more than fine grained. Though, let me just reiterate that the sudden big drop could be attributed to the fact that the fc layer of this channel pruned LeNet5 was also fine grained pruned from **II.2**.

These levels of accuracy however can be greatly improved by fine-tuning the model again.