2022年11月12日 星期六

Final project T2G3

OUT LINE:

1.Training with data augmentation

2.Pruning original model

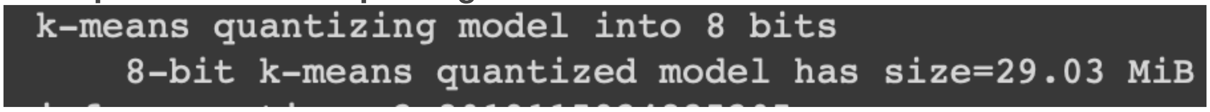
3. Quantization pruned model

You can run our colab demo here:

https://colab.research.google.com/drive/1Az42YmAv6R964N5ppswpigiX7L0-pgSU?usp=sharing#scrollTo=alyOL6SR9HiH

Final result:

8-bit quantization after pruning the our model:

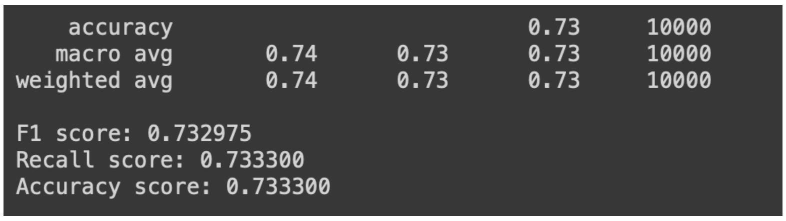


```
 k-means quantizing model into 8 bits
     8-bit k-means quantized model has size=29.03 MiB
```

Accuracy=73.33% after quantization-aware training

8-bit model has flops=532846480.00

inference time:2.6539 s



```
    accuracy                           0.73     10000
   macro avg       0.74      0.73      0.73     10000
weighted avg       0.74      0.73      0.73     10000

F1 score: 0.732975
Recall score: 0.733300
Accuracy score: 0.733300
```

Data Augmentation：

Our model is the original resnet-9 with batch normalization

(The network structure is so long we will show it in appendix)

our training process is similar to the original training process, but we add grid mask data augmentation on the raw data.

GridMask Data Augmentation: https://arxiv.org/abs/2001.04086

Our group’s presentation is also about this paper and the ppt has been submitted.

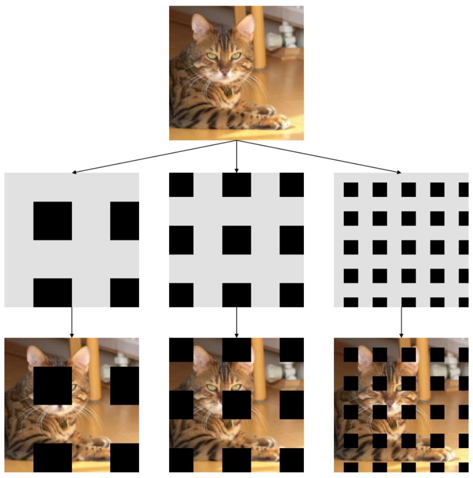


Figure 3. This image shows examples of GridMask. First, we produce a mask according to the given parameters (r, d, δ_x, δ_y). Then we multiply it with the input image. The result is shown in the last row. In the mask, gray value is 1, representing the reserved regions; black value is 0, for regions to be deleted.

Parameters selection： According to the experiment results given by the paper on cifar-10,we choose our parameters on grid mask as:

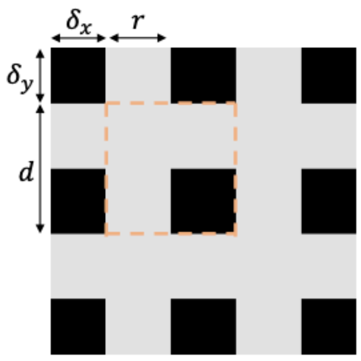
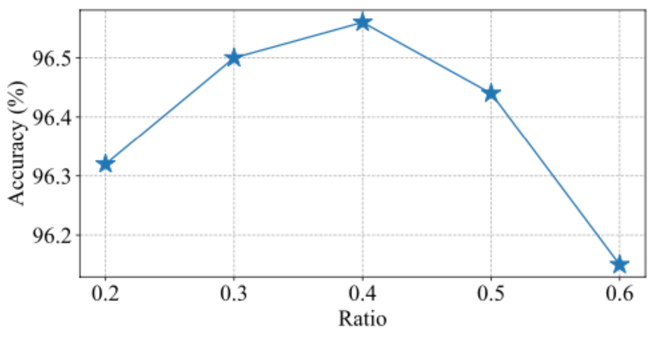


Figure 4. The dotted square shows one unit of the mask.

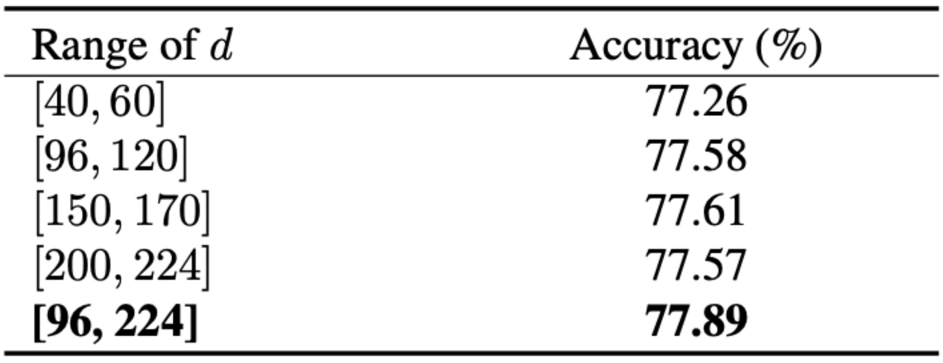
keep ratio : 0.4



(b) CIFAR10

range of d:(results of image net)

For



Range of d	Accuracy (%)
$[40, 60]$	77.26
$[96, 120]$	77.58
$[150, 170]$	77.61
$[200, 224]$	77.57
$[96, 224]$	**77.89**

Table 4. Results on different ranges of d.

the

resolution of cifar10， we choose the range of d as:[12,18]

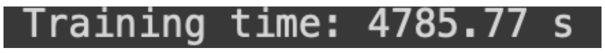
rotate： random rotate mask from 0 to 2π

Result of training and inference:

Metric of params, flops and inference time will be shown in the second(pruning) and third(Quantization) part.

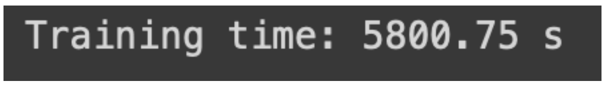
Training:

Original method：

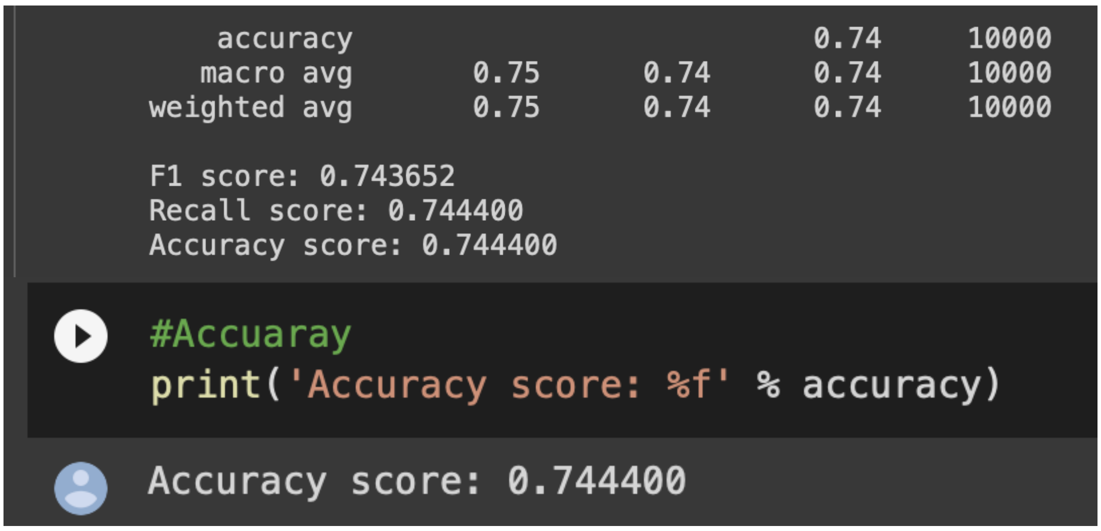


```
Training time: 4785.77 s
```

augmentation：



```
Training time: 5800.75 s
```



```
    accuracy                           0.74     10000
   macro avg       0.75      0.74      0.74     10000
weighted avg       0.75      0.74      0.74     10000

F1 score: 0.743652
Recall score: 0.744400
Accuracy score: 0.744400
```

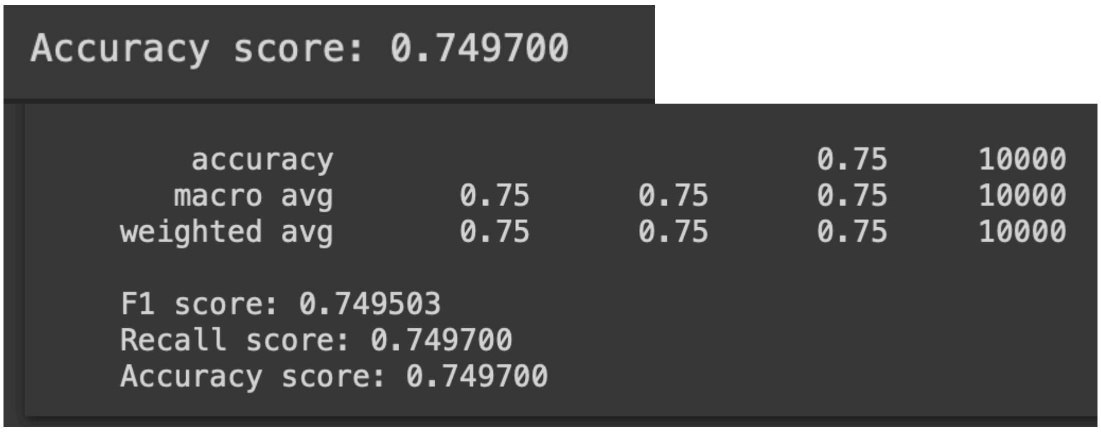
```
#Accuaray
print('Accuracy score: %f' % accuracy)
```

```
Accuracy score: 0.744400
```

The training time is close to the original training time:(our data augmentation method does take much time to run)

Inference：

Original method：



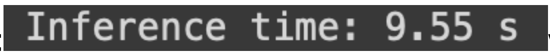
```
Accuracy score: 0.749700
```

```
    accuracy                           0.75     10000
   macro avg       0.75      0.75      0.75     10000
weighted avg       0.75      0.75      0.75     10000

F1 score: 0.749503
Recall score: 0.749700
Accuracy score: 0.749700
```

Data augmentation：

The result is improved, which means our method works.

Inference time: 

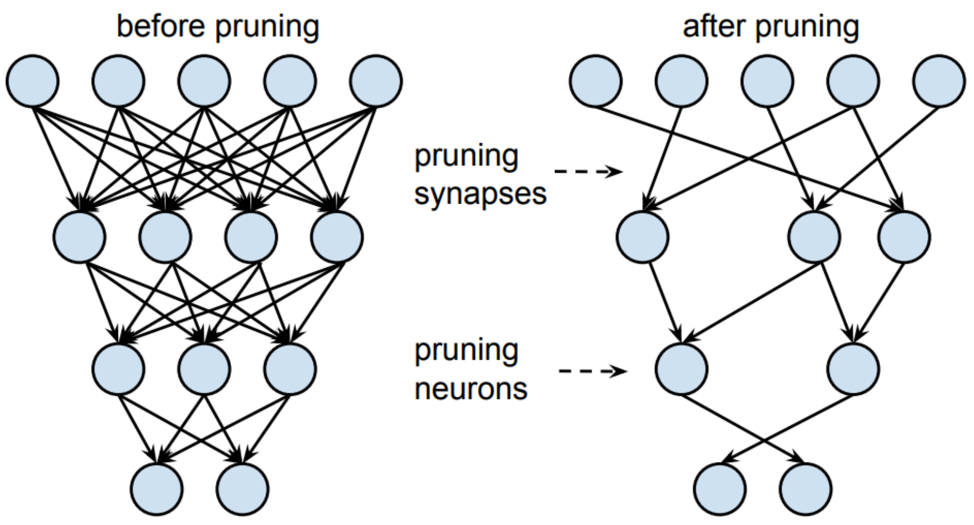
```
Inference time: 9.55 s
```

we will use this result later.

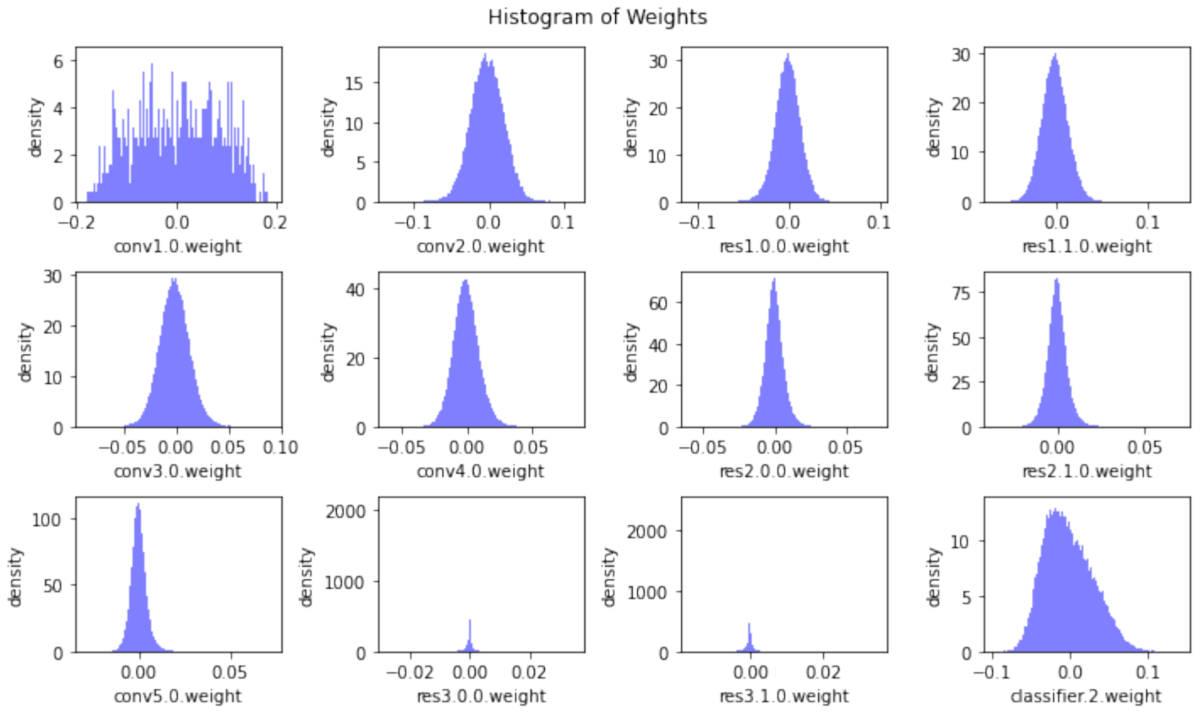
Pruning

We calculate the absolute value of weight as its importance. The closer the parameter value is to 0, the less important it is. Then, we calculate the pruning threshold so that all synapses with importance smaller than threshold will be removed. The threshold is calculated by choosing the proportion of parameters should be set to 0.

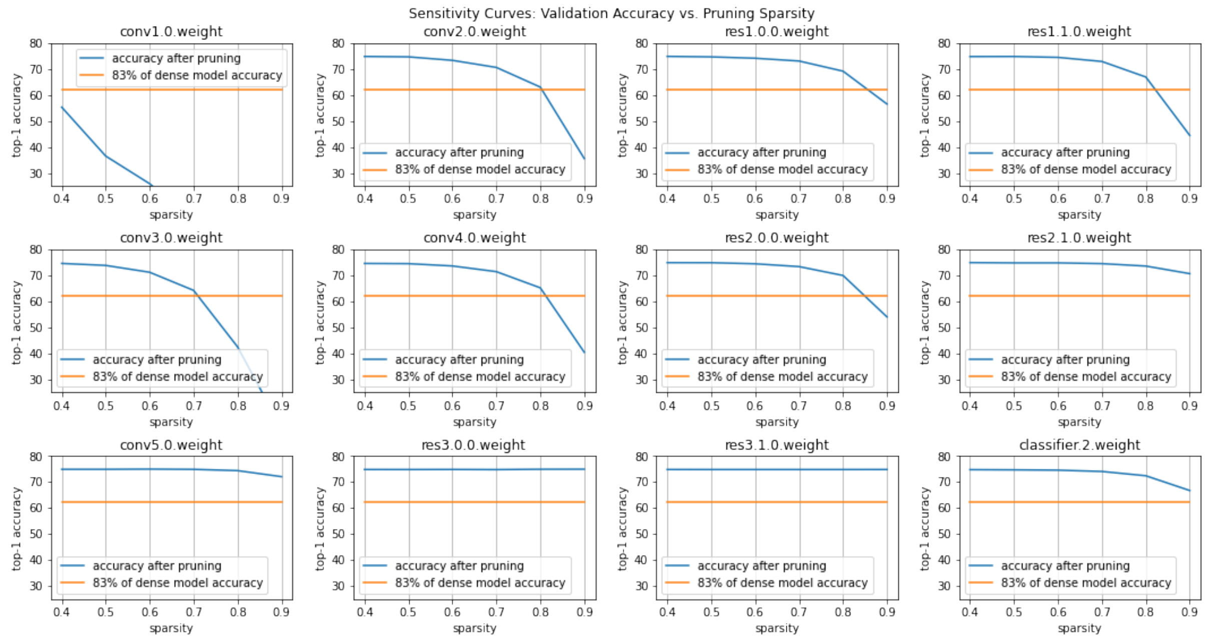
We will explain the implementation process while explaining the code below.



First ,calculate the dense model weight distribution in each layer:

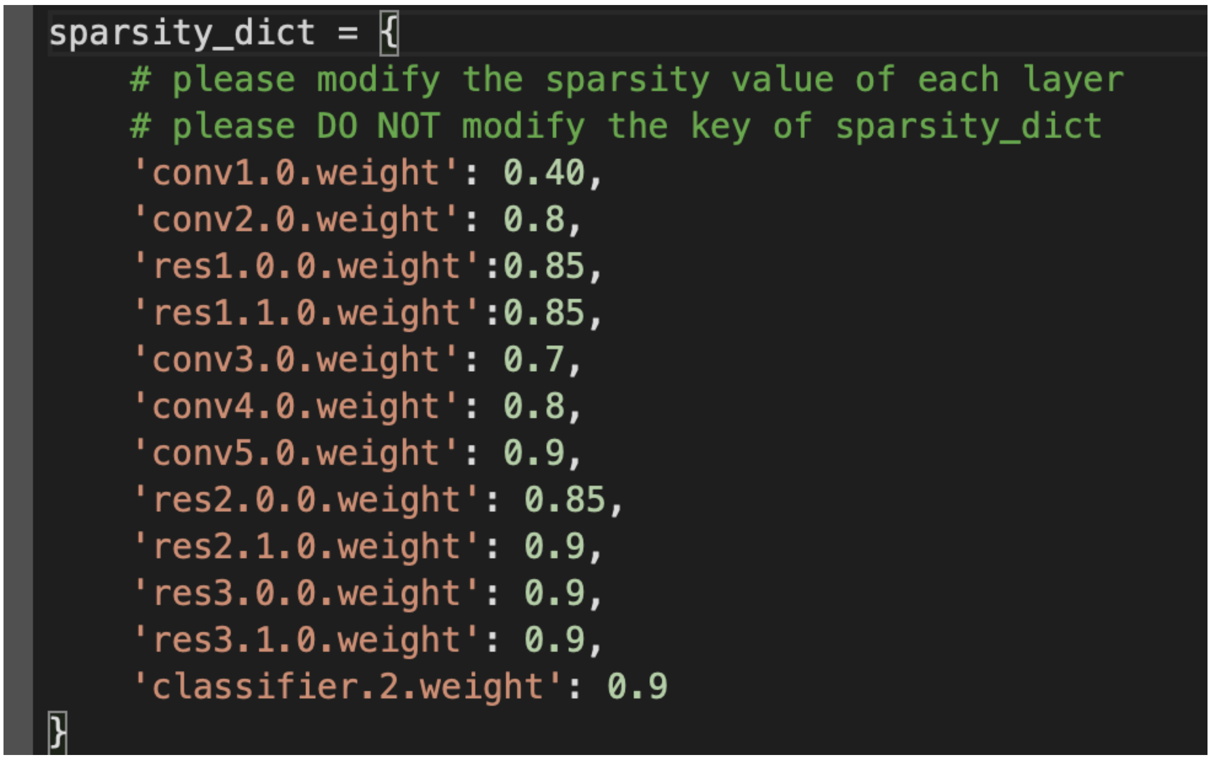


Then we will do sensitive scan on the parameters



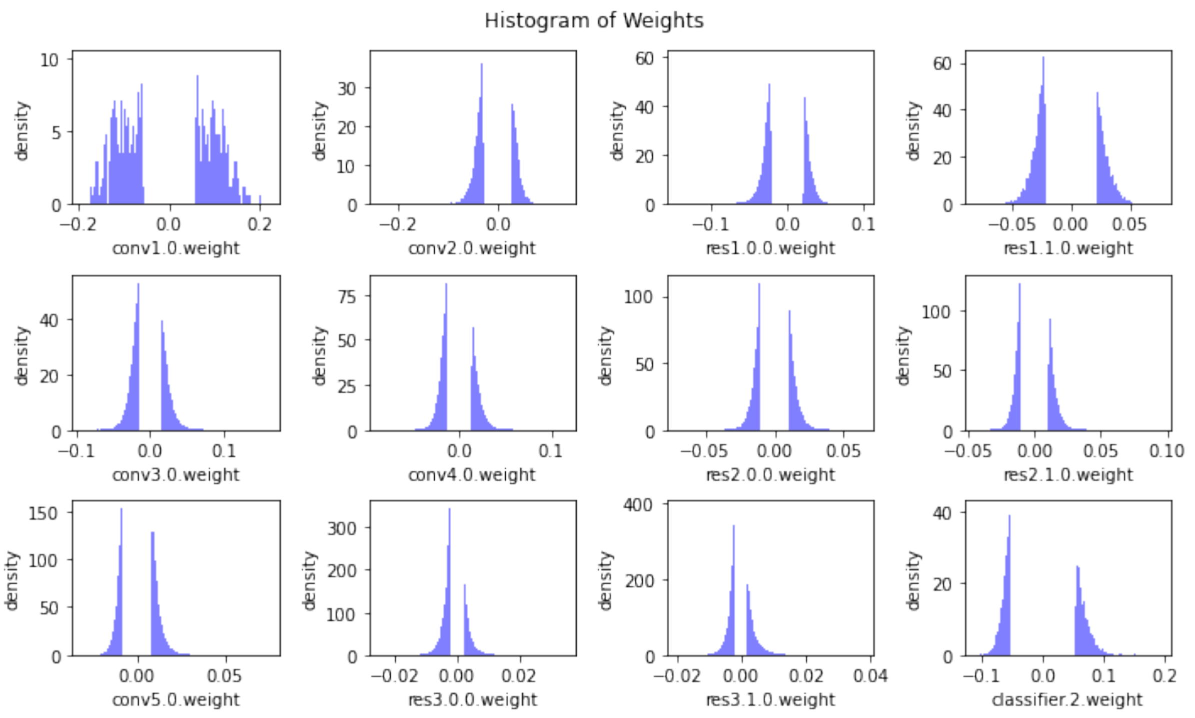
We need to scan the sparsity(which determines how much of the parameter will be set to 0)

We will set the threshold(sparsity) based on the form above, that is:



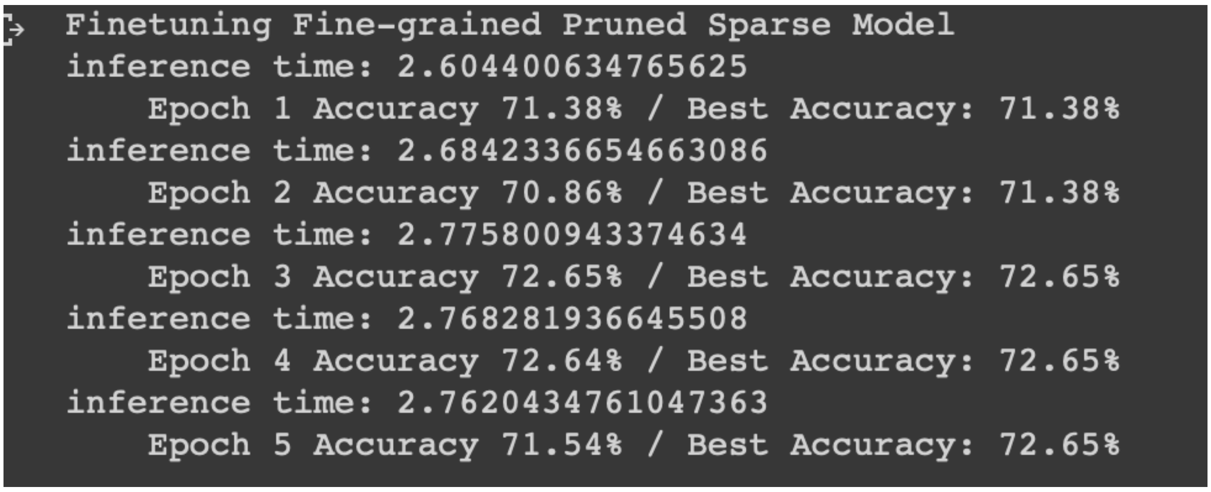
```
sparsity_dict = {
    # please modify the sparsity value of each layer
    # please DO NOT modify the key of sparsity_dict
    'conv1.0.weight': 0.40,
    'conv2.0.weight': 0.8,
    'res1.0.0.weight':0.85,
    'res1.1.0.weight':0.85,
    'conv3.0.weight': 0.7,
    'conv4.0.weight': 0.8,
    'conv5.0.weight': 0.9,
    'res2.0.0.weight': 0.85,
    'res2.1.0.weight': 0.9,
    'res3.0.0.weight': 0.9,
    'res3.1.0.weight': 0.9,
    'classifier.2.weight': 0.9
}
```

The parameter distribution after pruning:



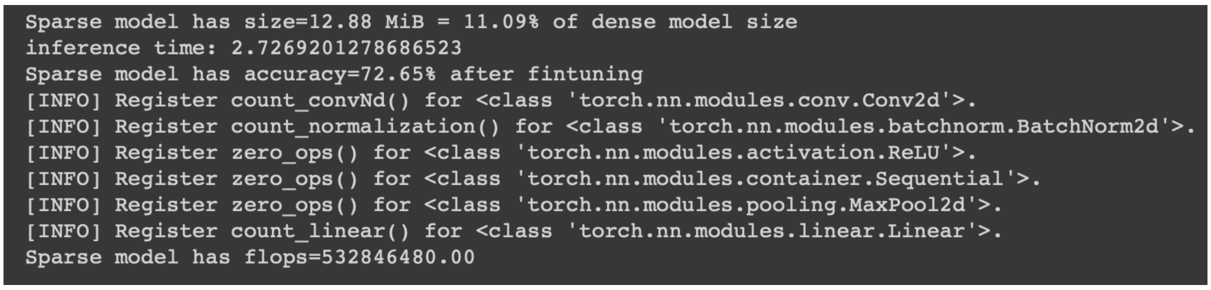
6

Then fine-tuning the model to improve the performance(train another 5 epochs)



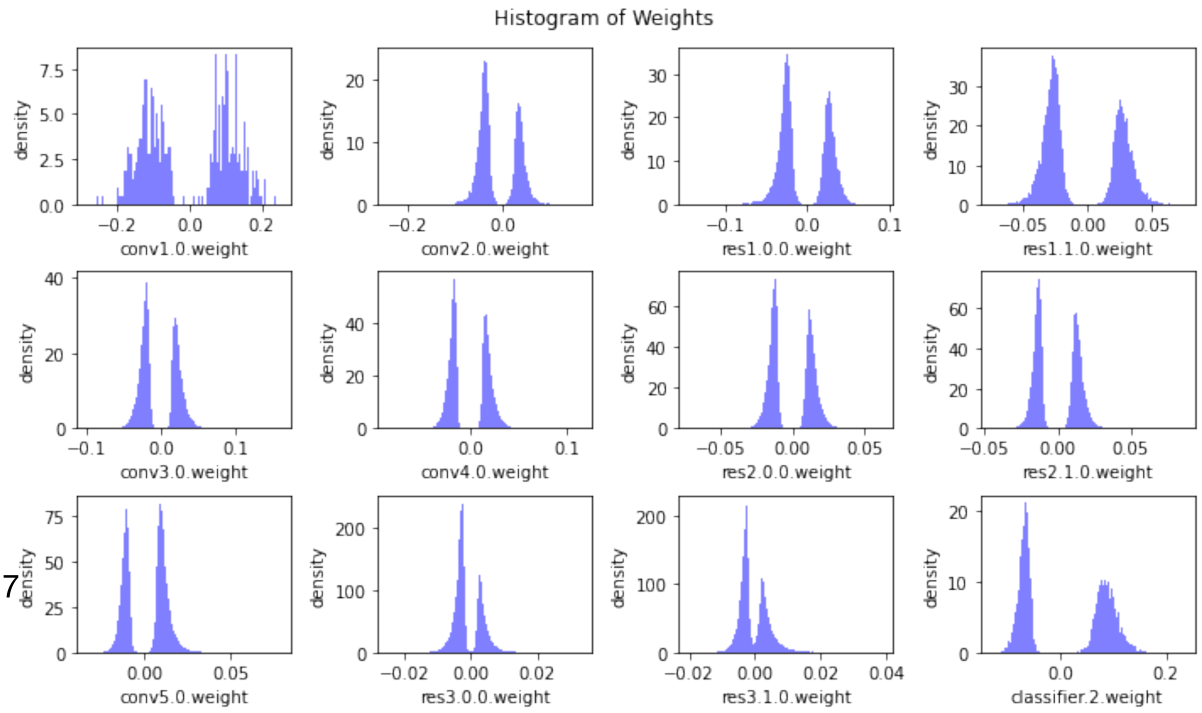
```
Finetuning Fine-grained Pruned Sparse Model
inference time: 2.604400634765625
    Epoch 1 Accuracy 71.38% / Best Accuracy: 71.38%
inference time: 2.6842336654663086
    Epoch 2 Accuracy 70.86% / Best Accuracy: 71.38%
inference time: 2.775800943374634
    Epoch 3 Accuracy 72.65% / Best Accuracy: 72.65%
inference time: 2.768281936645508
    Epoch 4 Accuracy 72.64% / Best Accuracy: 72.65%
inference time: 2.7620434761047363
    Epoch 5 Accuracy 71.54% / Best Accuracy: 72.65%
```

Result after fine-tuning:



```
Sparse model has size=12.88 MiB = 11.09% of dense model size
inference time: 2.7269201278686523
Sparse model has accuracy=72.65% after fintuning
[INFO] Register count_convNd() for <class 'torch.nn.modules.conv.Conv2d'>.
[INFO] Register count_normalization() for <class 'torch.nn.modules.batchnorm.BatchNorm2d'>.
[INFO] Register zero_ops() for <class 'torch.nn.modules.activation.ReLU'>.
[INFO] Register zero_ops() for <class 'torch.nn.modules.container.Sequential'>.
[INFO] Register zero_ops() for <class 'torch.nn.modules.pooling.MaxPool2d'>.
[INFO] Register count_linear() for <class 'torch.nn.modules.linear.Linear'>.
Sparse model has flops=532846480.00
```

Parameter distribution after fine-tuning:

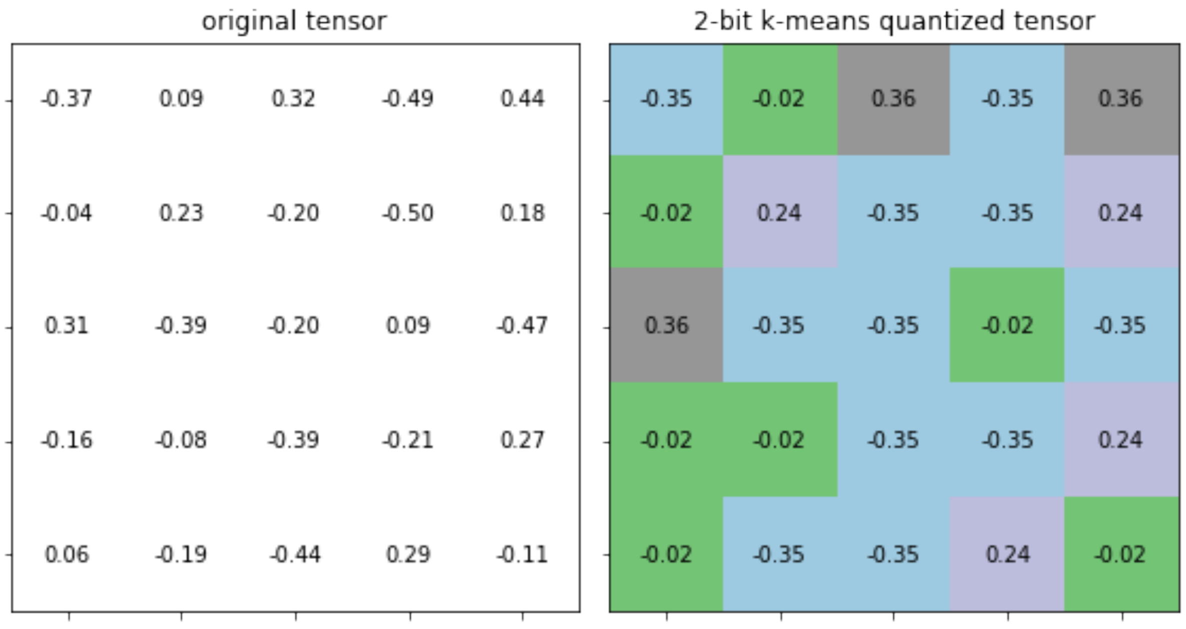


Quantization pruned model:

K-means quantization: https://arxiv.org/pdf/1510.00149.pdf

How to apply?

To put it simply, a n-bit k-means quantization will divide synapses into 2^n clusters, and synapses in the same cluster will share the same weight value.



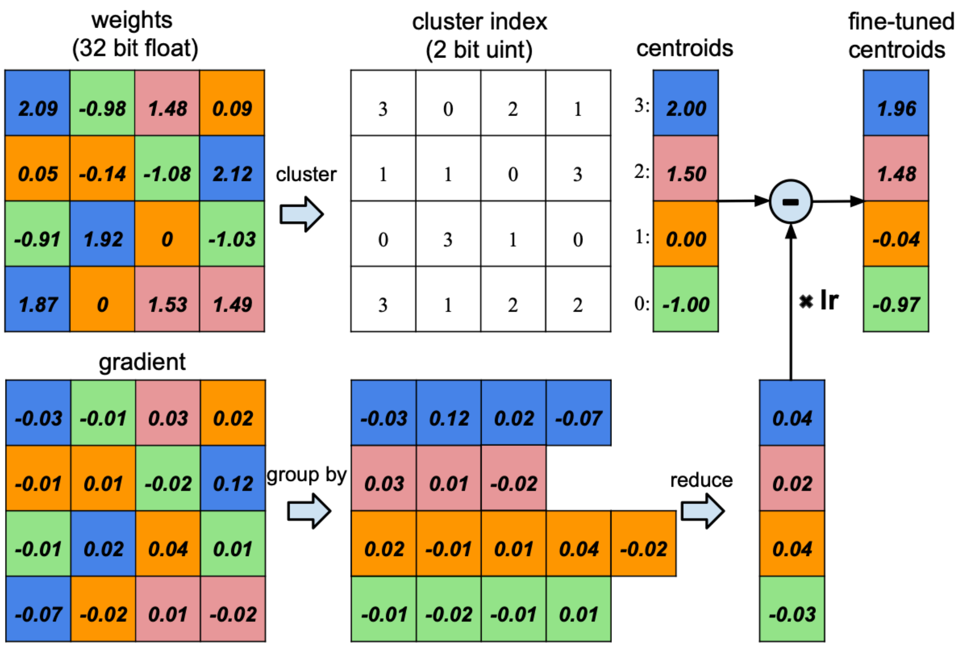
original tensor

-0.37	0.09	0.32	-0.49	0.44
-0.04	0.23	-0.20	-0.50	0.18
0.31	-0.39	-0.20	0.09	-0.47
-0.16	-0.08	-0.39	-0.21	0.27
0.06	-0.19	-0.44	0.29	-0.11

2-bit k-means quantized tensor

-0.35	-0.02	0.36	-0.35	0.36
-0.02	0.24	-0.35	-0.35	0.24
0.36	-0.35	-0.35	-0.02	-0.35
-0.02	-0.02	-0.35	-0.35	0.24
-0.02	-0.35	-0.35	0.24	-0.02

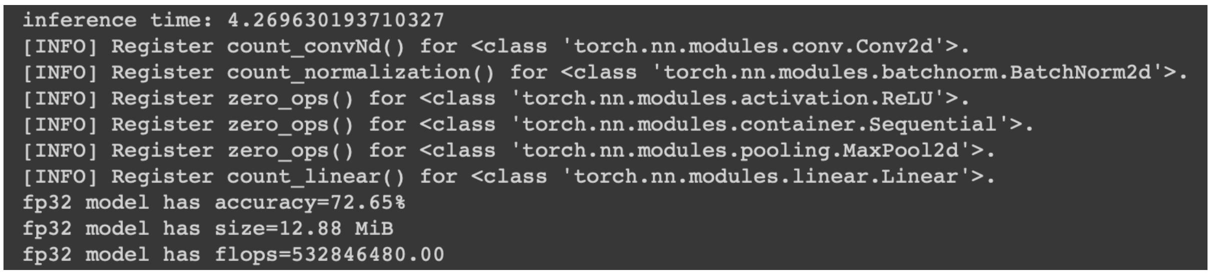
But after quantization, the networks' performance will decrease.
Therefore, we need to retrain the model.But the parameters of each



category will have different gradients.So when updating, the parameter is updated according to the average value of parameter gradient in each cluster.

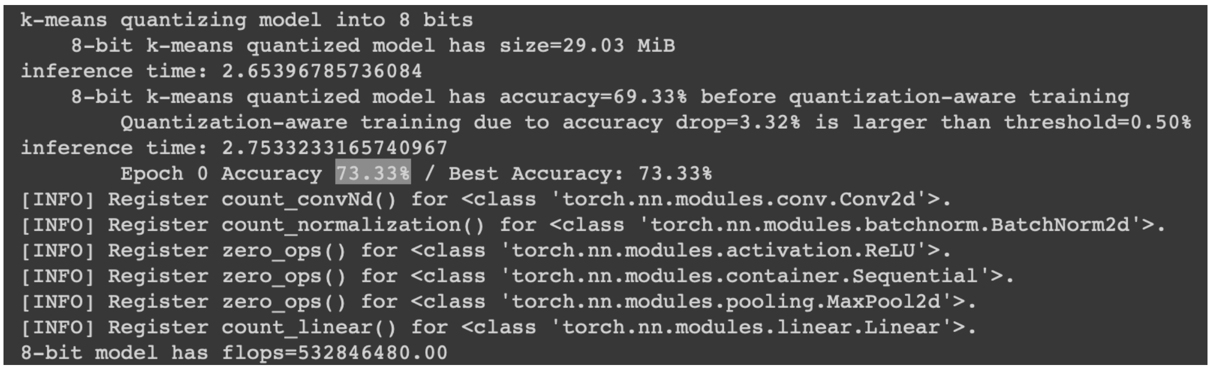
Results after k-means quantization:

Original Pruned model:

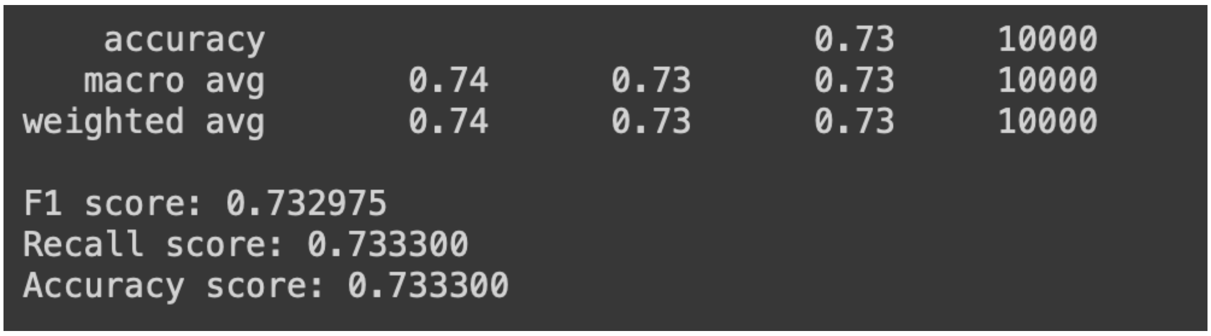


```
inference time: 4.269630193710327
[INFO] Register count_convNd() for <class 'torch.nn.modules.conv.Conv2d'>.
[INFO] Register count_normalization() for <class 'torch.nn.modules.batchnorm.BatchNorm2d'>.
[INFO] Register zero_ops() for <class 'torch.nn.modules.activation.ReLU'>.
[INFO] Register zero_ops() for <class 'torch.nn.modules.container.Sequential'>.
[INFO] Register zero_ops() for <class 'torch.nn.modules.pooling.MaxPool2d'>.
[INFO] Register count_linear() for <class 'torch.nn.modules.linear.Linear'>.
fp32 model has accuracy=72.65%
fp32 model has size=12.88 MiB
fp32 model has flops=532846480.00
```

Final result:
8-bit k-means after fin-tuning:(we also tried 4-bit and 2-bit but the performance dropped a lot)



```
k-means quantizing model into 8 bits
    8-bit k-means quantized model has size=29.03 MiB
inference time: 2.65396785736084
    8-bit k-means quantized model has accuracy=69.33% before quantization-aware training
        Quantization-aware training due to accuracy drop=3.32% is larger than threshold=0.50%
inference time: 2.7533233165740967
        Epoch 0 Accuracy 73.33% / Best Accuracy: 73.33%
[INFO] Register count_convNd() for <class 'torch.nn.modules.conv.Conv2d'>.
[INFO] Register count_normalization() for <class 'torch.nn.modules.batchnorm.BatchNorm2d'>.
[INFO] Register zero_ops() for <class 'torch.nn.modules.activation.ReLU'>.
[INFO] Register zero_ops() for <class 'torch.nn.modules.container.Sequential'>.
[INFO] Register zero_ops() for <class 'torch.nn.modules.pooling.MaxPool2d'>.
[INFO] Register count_linear() for <class 'torch.nn.modules.linear.Linear'>.
8-bit model has flops=532846480.00
```



```
    accuracy                           0.73     10000
   macro avg       0.74      0.73      0.73     10000
weighted avg       0.74      0.73      0.73     10000

F1 score: 0.732975
Recall score: 0.733300
Accuracy score: 0.733300
```