Rakotoarivony & Sicard

Challenge

This document is intended for the oral restitution carried out in groups on 24 march 2021.

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

Introduction

Learning Hyperparameters

Model A & B

Optimizer : SGD

Momentum = 0.9

Weight decay = 10^{-4}

Lr = 0.1 [1,89]

Lr = 0.01 [90,109]

Lr = 0.001 [110,120]

Introduction

Learning Hyperparameters

Model A & B

Optimizer : SGD

Momentum = 0.9

Weight decay = 10^{-4}

Lr = 0.1 [1,89]

Lr = 0.01 [90,109]

Lr = 0.001 [110,120]

Architecture Hyperparameters

Model A

Grow rate = 12

Number of denseblock = 4

Number of bottleneck = [6,12,24,16]

Model	Accuracy	Score Flops	Score Parameters
densenet_A	93.70 %	0.231	0.088

Introduction

Learning Hyperparameters

Model A & B

Optimizer : SGD

Momentum = 0.9

Weight decay = 10^{-4}

Lr = 0.1 [1,89]

Lr = 0.01 [90,109]

Lr = 0.001 [110,120]

Architecture Hyperparameters

Model A

Grow rate = 12

Number of denseblock = 4

Number of bottleneck = [6,12,24,16]

Model B

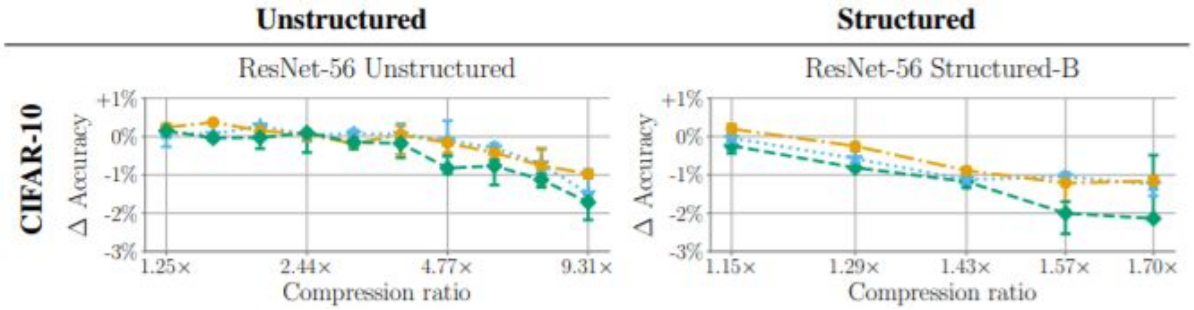
Grow rate = 8

Number of denseblock = 4

Number of bottleneck = [6,10,20,12]

Model	Accuracy	Score Flops	Score Parameters
densenet_A	93.70 %	0.231	0.088
densenet_B	**92.05 %**	**0.089**	**0.029**

Learning rate rewinding



Comparing Rewinding and Fine-tuning in Neural Network Pruning:

https://arxiv.org/abs/2003.02389

Learning rate rewinding

Algorithm: Our pruning algorithm

1. Train to completion
2. Prune the 20% lowest-magnitude weights globally
3. Retrain using learning rate rewinding for the original training time
4. Repeats steps 2 and 3 iteratively until the desired compression ratio is reached

How do we retrain ?

Learning rate rewinding for **t** epochs runs Train (**W_T** , **m** , **T - t**)

With:
W_T , the weights actually of the pruned model
m, the pruned model
T - t, the learning rate schedule from the last t epochs of training

Retrain for 60 epochs:
Lr = 0.1 [1,29]

Lr = 0.01 [30,49]

Lr = 0.001 [50,60]

Learning rate rewinding

Pruning Parameters

Pruned A

Compression ratio: 5 (80 % pruned)

Steps of pruning: 7

Model	Accuracy	Score Flops	Score Parameters
densenet_B	92.05 %	0.089	0.029
densenet_B_pruned_A	92.65 %	0.089	0.007

Learning rate rewinding

Pruning Parameters

Pruned A

Compression ratio: 5 (80 % pruned)

Steps of pruning: 7

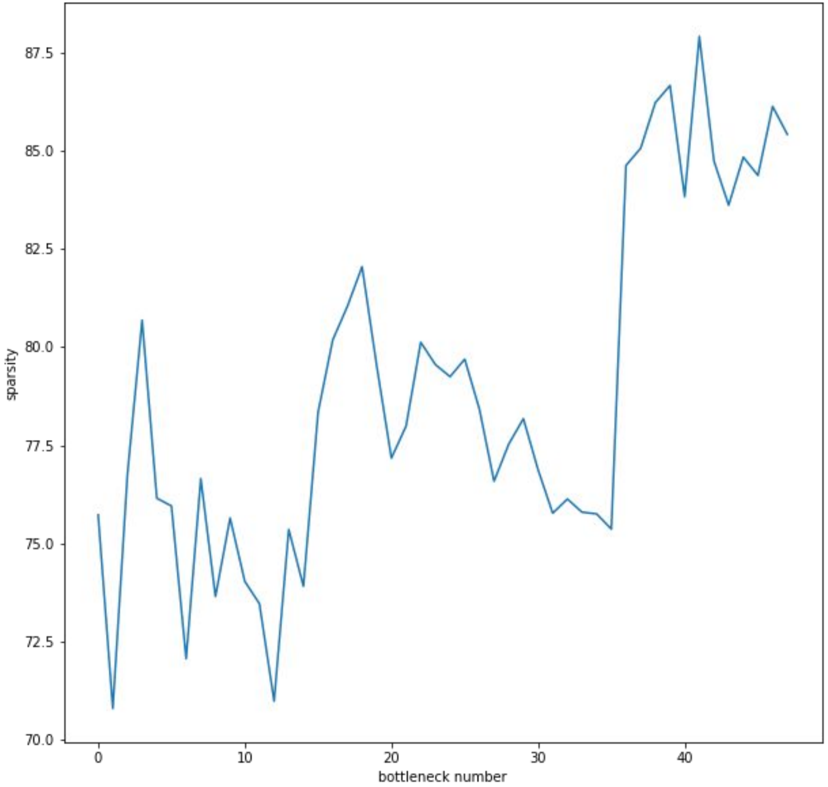
Pruned B

Compression ratio: 10 (89 % pruned)

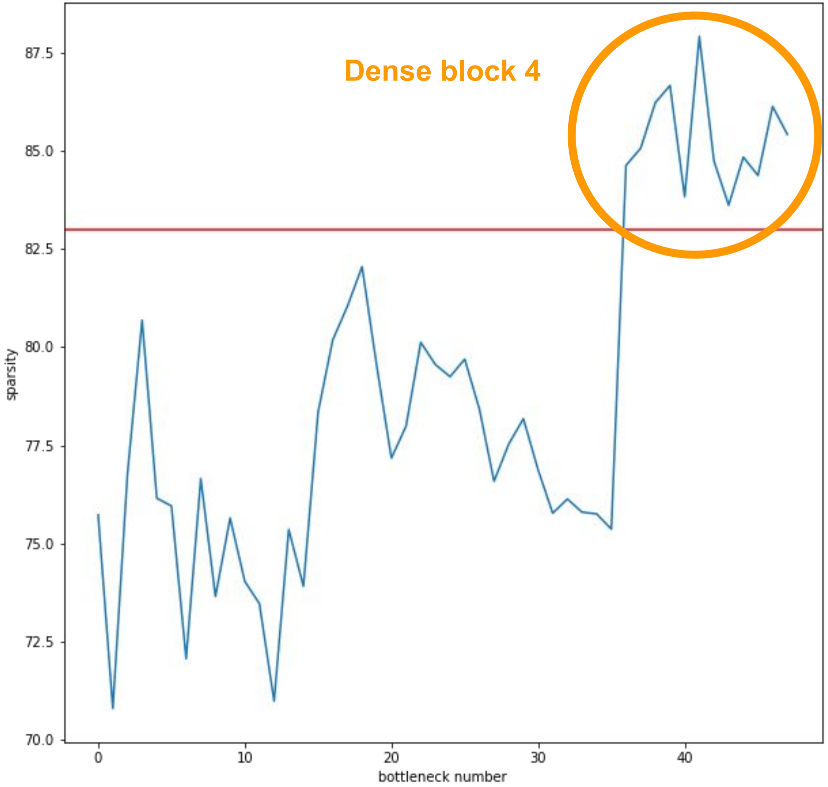
Steps of pruning: 10

Model	Accuracy	Score Flops	Score Parameters
densenet_B	92.05 %	0.089	0.029
densenet_B_pruned_A	**92.65 %**	**0.089**	**0.007**
densenet_B_pruned_B	92.07 %	0.089	0.004

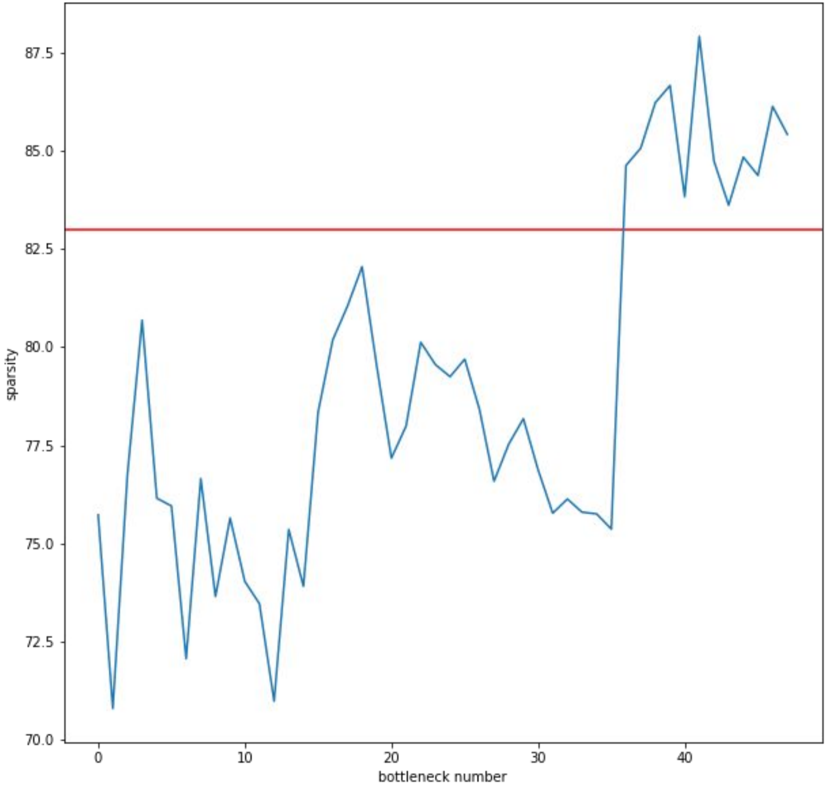
Sparsity of layers



Sparsity of layers



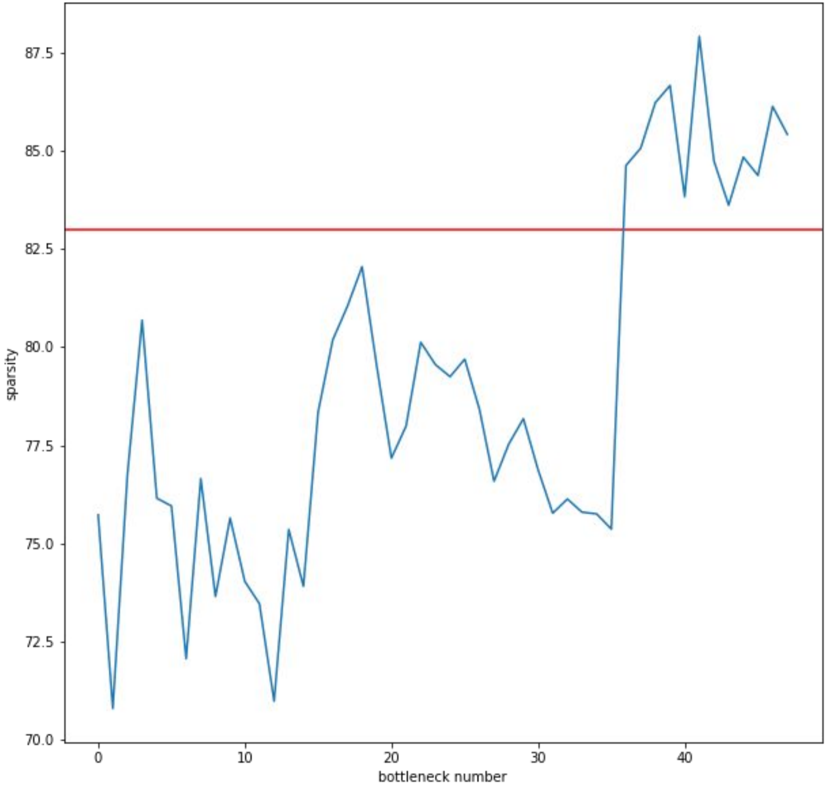
Sparsity of layers



Removal of dense block 4
(all of his bottlenecks have more than **83%** of sparsity)

Need to **modify** (increase) the number of features maps in the transition block 3

Sparsity of layers



Removal of dense block 4
(all of his bottlenecks have more than **83%** of sparsity)

Need to **modify** (increase) the number of features maps in the transition block 3

Model	Accuracy	Score Flops	Score Parameters
densenet_B	92.05 %	0.089	0.029
densenet_B_pruned_A	**92.65 %**	**0.0895**	**0.007**
densenet_B_pruned_A remove_denseblock4	92.12 %	0.0893	0.009

APoT Quantization

Quantization on **N** bits of the parameters of the **Convolutional Layers**
Use of the module QuantConv2D on **4 bits**

```
m = model_quant.conv1
print(m.weight[0].data)
print(m.weight_quant(m.weight, m.weight_mask)[0].data)
```

```
tensor([[[ 0.0000, -0.1658,  0.1490],
         [-0.0000,  0.0000, -0.1612],
         [-0.0536, -0.0825,  0.1160]],

        [[ 0.1253,  0.0759,  0.0000],
         [ 0.0000, -0.0000, -0.0468],
         [-0.0396, -0.1333,  0.1714]],

        [[-0.0000, -0.0782,  0.1518],
         [ 0.1493,  0.0732, -0.0744],
         [ 0.1655, -0.0000,  0.0536]]], device='cuda:0')
tensor([[[ 0.0000, -1.2000,  1.2000],
         [ 0.0000,  0.0000, -1.2000],
         [-0.3000, -0.6000,  0.9000]],

        [[ 1.2000,  0.6000,  0.0000],
         [ 0.0000,  0.0000, -0.3000],
         [-0.3000, -0.9000,  1.2000]],

        [[ 0.0000, -0.6000,  1.2000],
         [ 1.2000,  0.6000, -0.6000],
         [ 1.2000,  0.0000,  0.6000]]], device='cuda:0')
```

- We quantize **post training** because training with quantization is really slow

- Quantization on 4 bits is the best **trade-off accuracy and number of bits** because we lose only approximately 2% of accuracy

- We also try to implement other types of quantization such as BWN, Binary Connect or XNOR but this method is more efficient

Combination of Quantization and Pruning

The main challenge of our project was to **combine quantization and pruning**

- We need to **modify the code of QuantConv2D** in order to implement the pruning (use of mask and only focus on parameters different to 0 to calculate the mean)
- We have to **modify the state dict of the prune model** to implement the quantization (add new parameters)
- We need to retrain the quantize model approximately 30 epochs in order to obtain a good accuracy

Model	Accuracy	Score Flops	Score Parameters
densenet_B	92.05 %	0.089	0.029
densenet_B_pruned_A	92.65 %	0.089	0.007
densenet_B_pruned_A quantized	**90.07 %**	0.067	0.002

Areas for improvement

01

Use of **distillation** and **factorization**

02

Improve our **flops score**

=> Modify the architecture or use of structured pruning

03

Realize tests on **Cifar100**

Find a **better compromise** between the score flops and the score params

Conclusion

Thanks for your attention

Do you have any questions ?

Appendices and notes

Bottleneck : Sequence of
BatchNorm2d - Conv2d - BatchNorm2d - Conv2d

(Trans3)(conv): Conv2d(216, 108, kernel_size=(1, 1), stride=(1, 1), bias=False)
=> output: 204

4217088 ops

we remove 55k or 100k max …

Change size linear 0.086

References

Additive Powers-of-Two Quantization: An Efficient Non-uniform Discretization for Neural Networks:

https://arxiv.org/abs/1909.13144

Comparing Rewinding and Fine-tuning in Neural Network Pruning:

https://arxiv.org/abs/2003.02389