Compressão de Redes Neurais

Vítor Yeso Fidelis Freitas

Motivação

- Aplicações de modelos de Deep Learning em ambientes com recursos computacionais limitados (Embarcados, celulares, FPGA, computadores pessoais)
- Eficiência Energética (Grandes modelos consomem muita energia)

Redes Neurais geralmente são treinadas utilizando muitos recursos.

- Pesos em ponto flutuante 32 ou 64 bits

- Muitos valores de pesos por rede (por exemplo: GPT-3 tem 175 bilhões de parâmetros)

Pruning e Quantização de Redes Neurais

Pruning: Processo que tenta aumentar a quantidade de zeros de uma Rede Neural

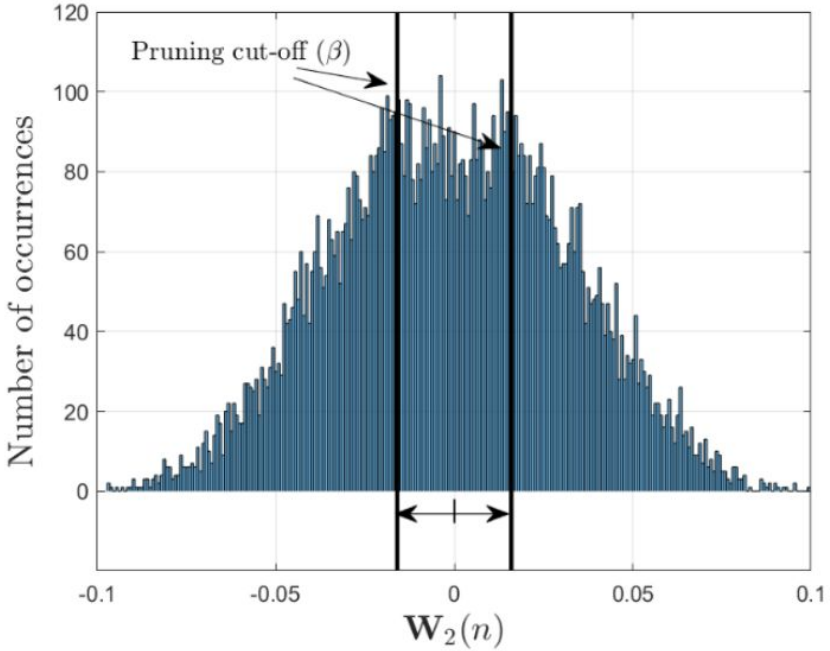
Quantização: Processo que tenta diminuir a resolução de bits dos parâmetros de uma Rede Neural

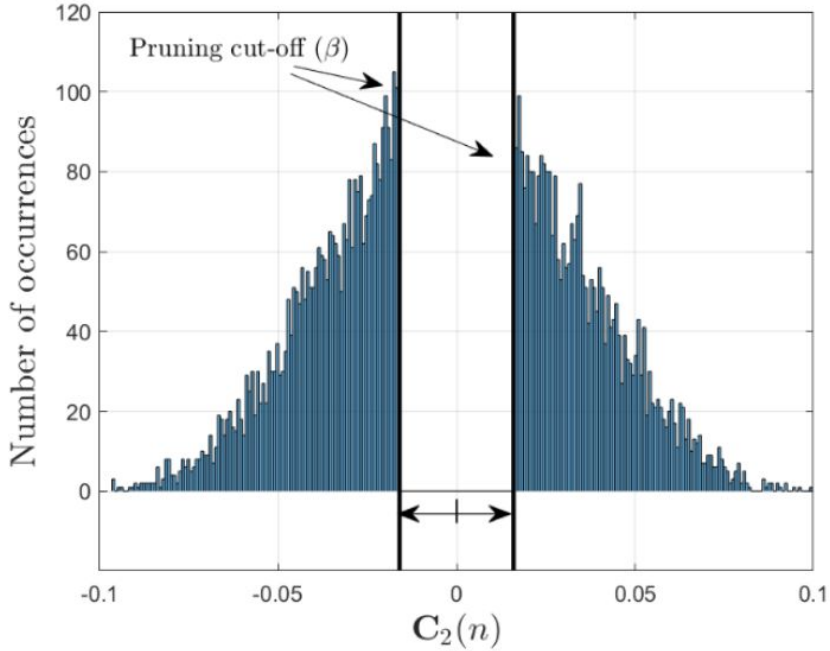
Pruning

$$P\left(\mathbf{W}_k(n), \beta_k\right) = \begin{cases} \mathbf{W}_k(n) & \text{if } |\mathbf{W}_k(n)| \geq \beta_k \\ 0 & \text{if } |\mathbf{W}_k(n)| < \beta_k \end{cases} \tag{1}$$

$$\beta_k = \gamma \times \sigma_k$$

Pruning


Pruning cut-off (β)
Number of occurrences
120
100
80
60
40
20
0
-0.1
-0.05
0
0.05
0.1
$\mathbf{W}_2(n)$

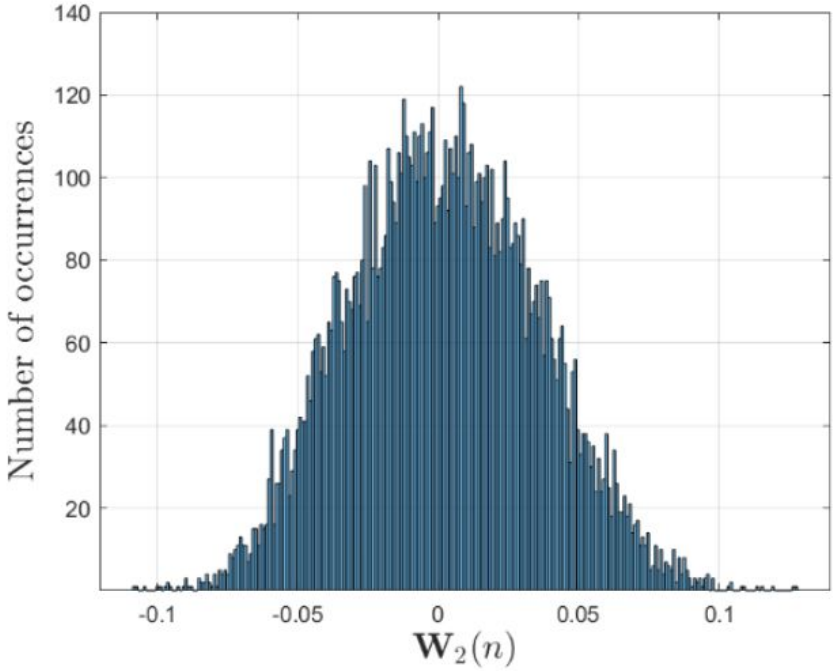

Pruning cut-off (β)
Number of occurrences
120
100
80
60
40
20
0
-0.1
-0.05
0
0.05
0.1
$\mathbf{C}_2(n)$

Quantização

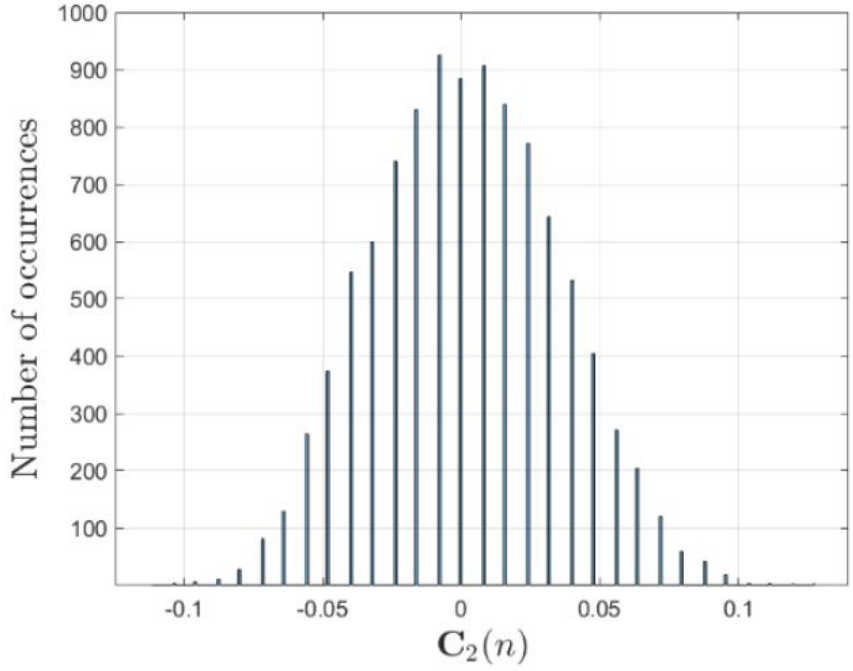
$$\mathbf{C}_k(n) = Q\left(\mathbf{W}_k(n), q_k\right) = \left\lceil \frac{\mathbf{W}_k(n)}{q_k} \right\rfloor \times q_k$$

$$q_k = \frac{\max\left\{\left|\mathbf{W}_k(n)\right|\right\}}{2^{b-1} - 1}.$$

Quantização



(a) Before quantization.



(b) After quantization.

Formas de aplicar Pruning e Quantização

- No final do treinamento
- No final de cada época de treinamento
- No final de cada iteração de treinamento

Implementação

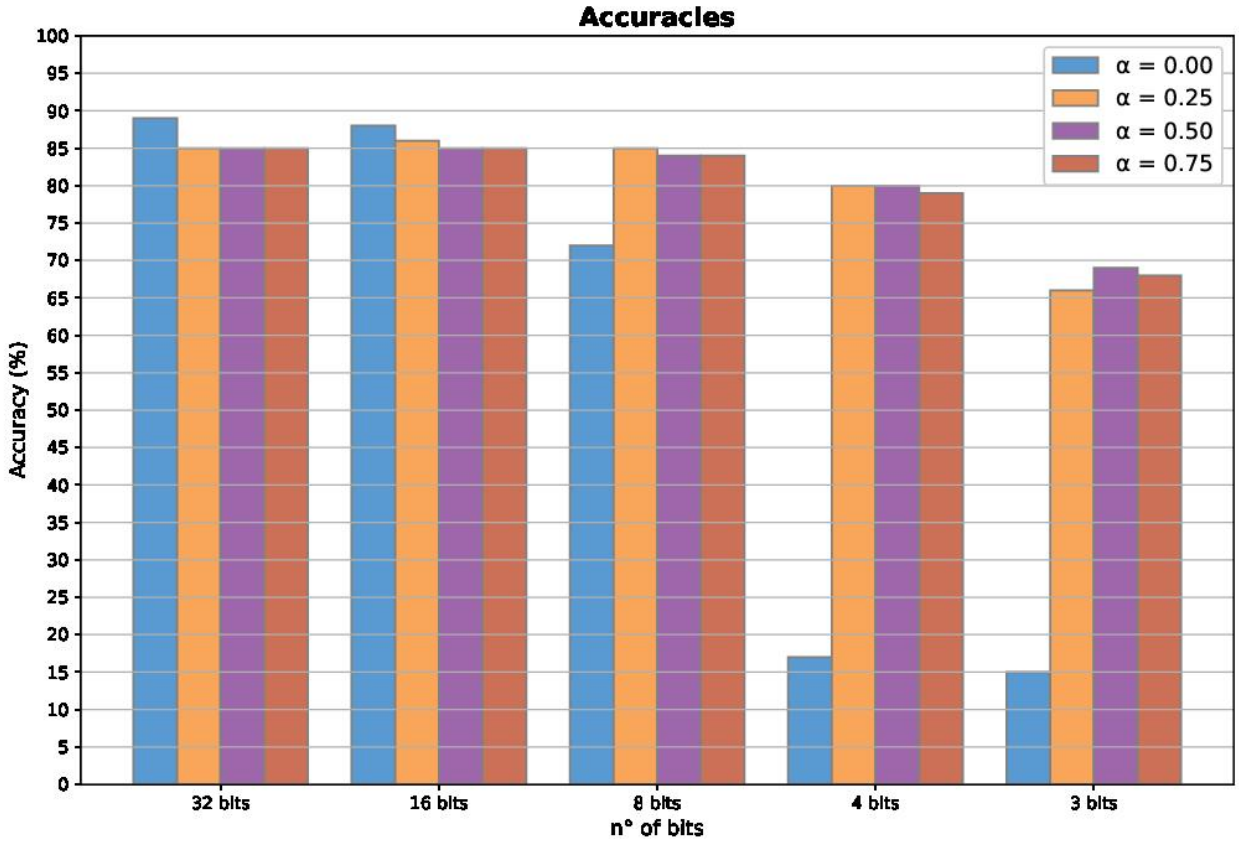
```python
def pruning(model, gamma):
    if gamma > 0.0:
        with torch.no_grad():
            betas = []
            for p in model.parameters():
                # transformando o layer em um vetor unidimensional
                flatted_weights = torch.flatten(p)

                # calculando o desvio padrao
                std = torch.std(flatted_weights, -1)

                # calculando o beta para a camada
                beta = gamma * std
                betas.append(beta)

                # mascara booleana, que indica qual peso devera ser mantido
                mask = torch.gt(p.abs(), torch.ones_like(p) * beta)

                # multiplicando os pesos da camada pelo tensor booleano
                p.multiply_(mask)
```

Implementação

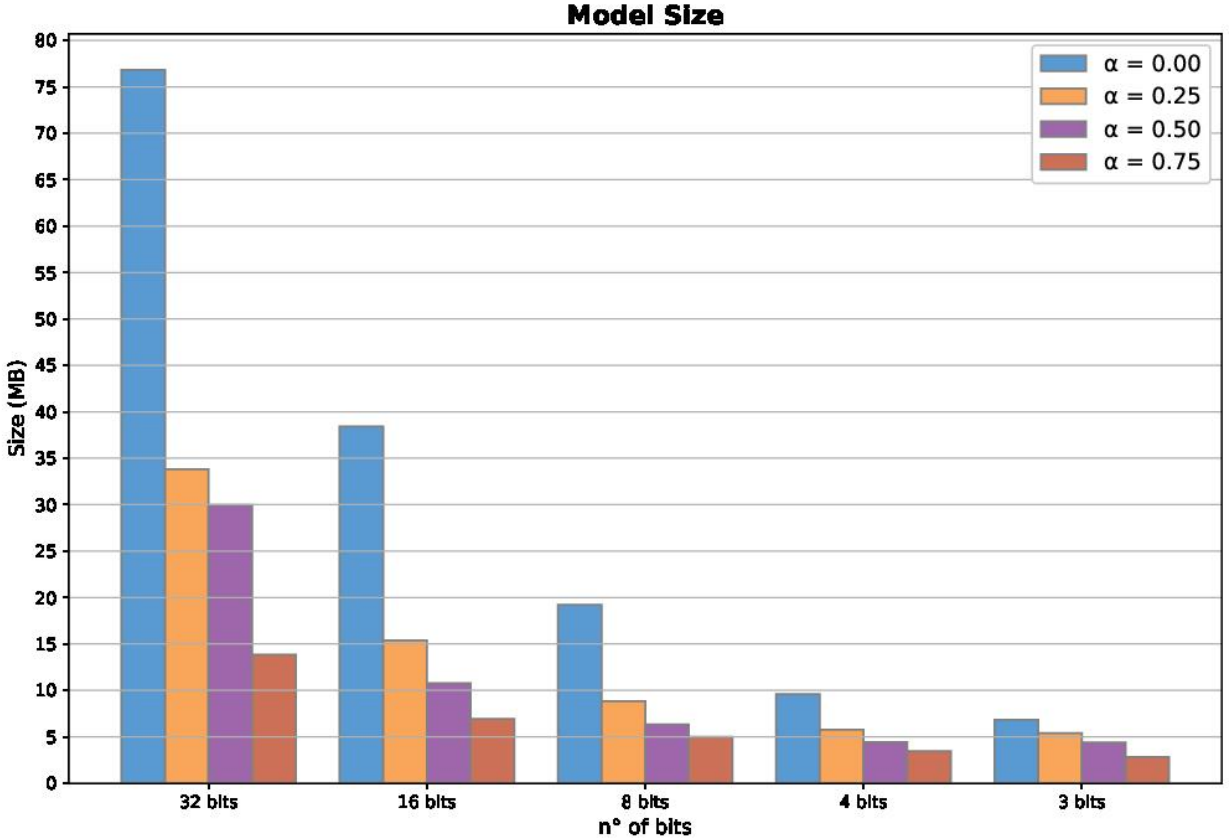
```
def quantization(model, b, betas=[]):
    if b >= 1:
        if len(betas) > 0:
            with torch.no_grad():
                for p, beta in zip(model.parameters(), betas):
                    flatted_weights = torch.flatten(p)

                    qk_prime = (torch.max(torch.abs(flatted_weights)) - beta) / ( (2**(b - 1)) - 1 )
                    torch.round(p/qk_prime, out=p)

                    p.multiply_(qk_prime)
        else:
            with torch.no_grad():
                for p in model.parameters():
                    flatted_weights = torch.flatten(p)
                    beta = 0.0;
                    qk_prime = (torch.max(torch.abs(flatted_weights)) - beta) / ( (2**(b - 1)) - 1 )
                    torch.round(p/qk_prime, out=p)
                    p.multiply_(qk_prime)
```

Dataset CIFAR10

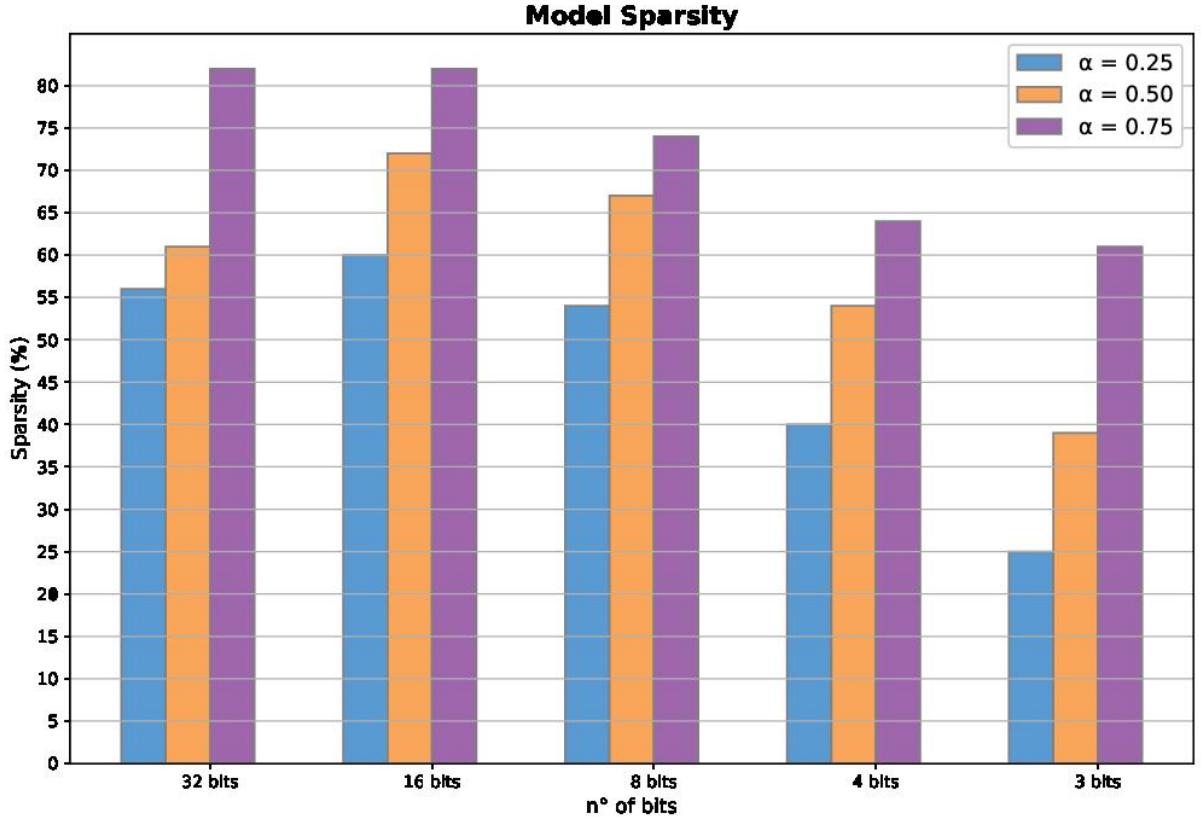
Resultados


Accuracies
α = 0.00
α = 0.25
α = 0.50
α = 0.75
Accuracy (%)
100
95
90
85
80
75
70
65
60
55
50
45
40
35
30
25
20
15
10
5
0
32 bits
16 bits
8 bits
4 bits
3 bits
n° of bits

Resultados


Model Size
α = 0.00
α = 0.25
α = 0.50
α = 0.75
Size (MB)
0
5
10
15
20
25
30
35
40
45
50
55
60
65
70
75
80
32 bits
16 bits
8 bits
4 bits
3 bits
n° of bits

Resultados


Model Sparsity
α = 0.25
α = 0.50
α = 0.75
Sparsity (%)
0
5
10
15
20
25
30
35
40
45
50
55
60
65
70
75
80
32 bits
16 bits
8 bits
4 bits
3 bits
n° of bits

Obrigado! 😀

Códigos e resultados disponíveis em:

https://github.com/vitoryeso/dl_model_compression