



Edge Deployment - Model Compression

Srinivas Rana, PhD

Talk at IIIT-H
2nd April, 2025



WADHWANI AI

Motivation

Model	Parameters (M)	FLOPs (GFLOPs)	FLOPs per Parameter	Model Size (MB)
AlexNet	61M	1.0	16.4	233
VGG-16	138M	15.5	112.3	528
VGG-19	144M	19.6	136.1	549
ResNet-18	11.7M	1.8	153.8	44
ResNet-50	25.6M	4.1	160.2	98
ResNet-101	44.6M	7.6	170.6	167
ResNet-152	60.2M	11.3	187.5	230
InceptionV3	23.8M	5.7	239.3	92
Xception	22.9M	8.4	366.3	88
MobileNetV1	4.2M	0.57	135.7	16
MobileNetV2	3.5M	0.3	85.7	14
EfficientNet-B0	5.3M	0.39	73.6	20
EfficientNet-B7	66M	37	560.6	255

Motivation

- Model complexity
 - Models can also be memory intensive (10s to 100s of million parameters and more) - memory footprint
 - Powerful infrastructure required to perform inference in a reasonably acceptable amount of time - computational footprint

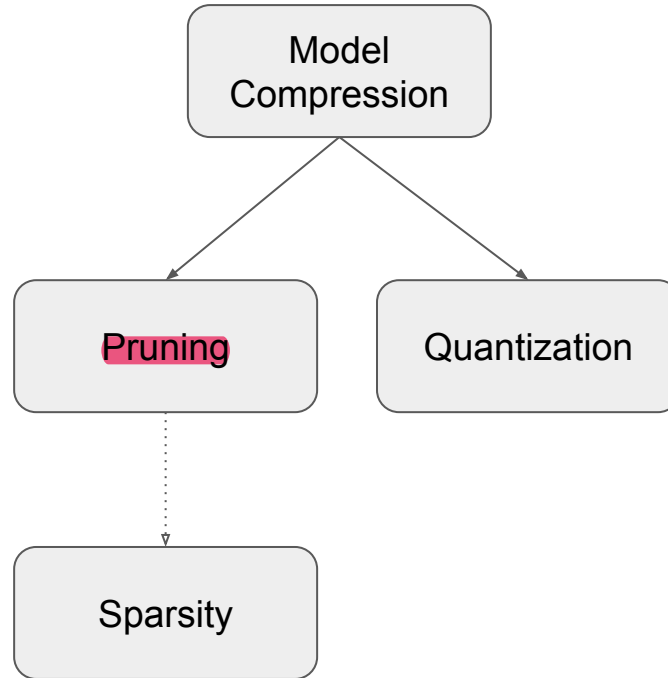
Model	Parameters (M)	FLOPs (GFLOPs)	FLOPs per Parameter	Model Size (MB)
AlexNet	61M	1.0	16.4	233
VGG-16	138M	15.5	112.3	528
VGG-19	144M	19.6	136.1	549
ResNet-18	11.7M	1.8	153.8	44
ResNet-50	25.6M	4.1	160.2	98
ResNet-101	44.6M	7.6	170.6	167
ResNet-152	60.2M	11.3	187.5	230
InceptionV3	23.8M	5.7	239.3	92
Xception	22.9M	8.4	366.3	88
MobileNetV1	4.2M	0.57	135.7	16
MobileNetV2	3.5M	0.3	85.7	14
EfficientNet-B0	5.3M	0.39	73.6	20
EfficientNet-B7	66M	37	560.6	255

Motivation

Can we reduce these two footprints, i.e. reduce the model size and the number of computations, whilst retaining the model accuracy??

This strictly relates to **inference** alone and not training.

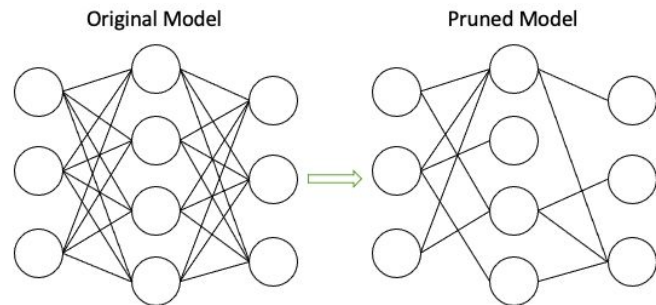
Terminologies



Pruning

Pruning

- DNNs are over-parameterized^[1]
 - Many parameters encode the same / similar information in the network
- Pruning eliminates these additional weights that do not contribute / add additional value to model performance
- Sparse tensors leading to fewer FLOPs → faster inference, smaller model
- Inherently regularised and hence more robust to noise
- Model drops in accuracy but can be recovered by finetuning

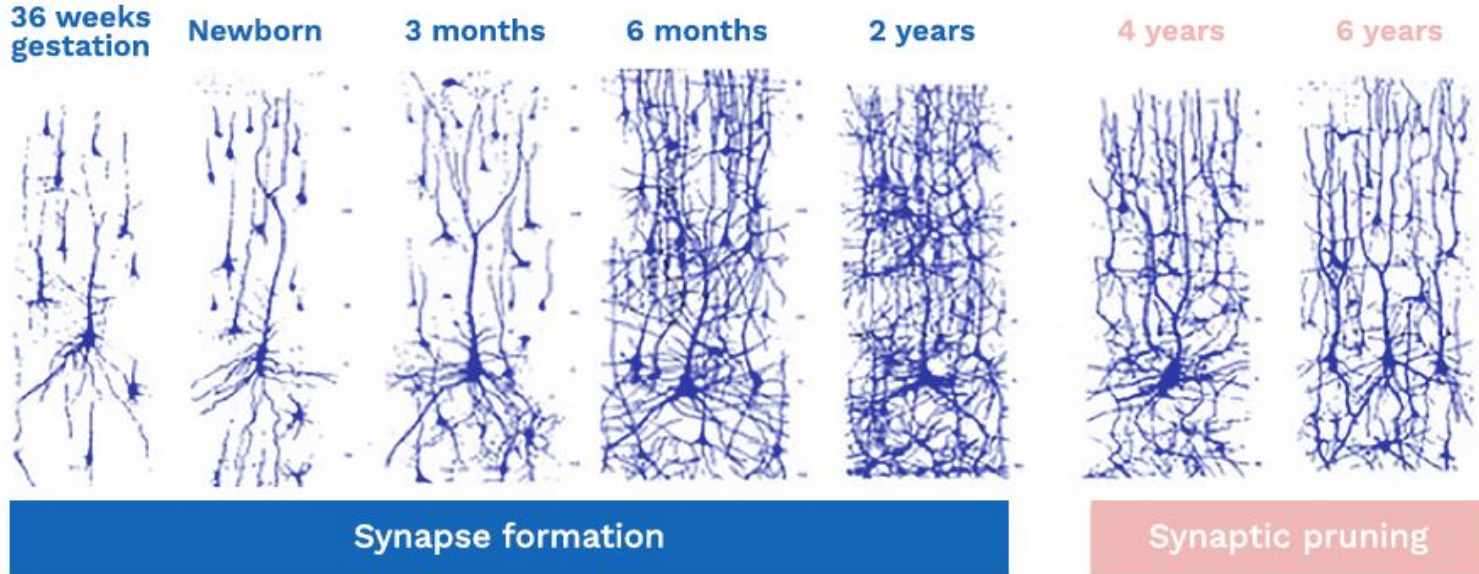


[1] Blier, Léonard & Ollivier, Yann. (2018). Do Deep Learning Models Have Too Many Parameters? An Information Theory Viewpoint.



Pruning

- Pruning is seen in nature too!

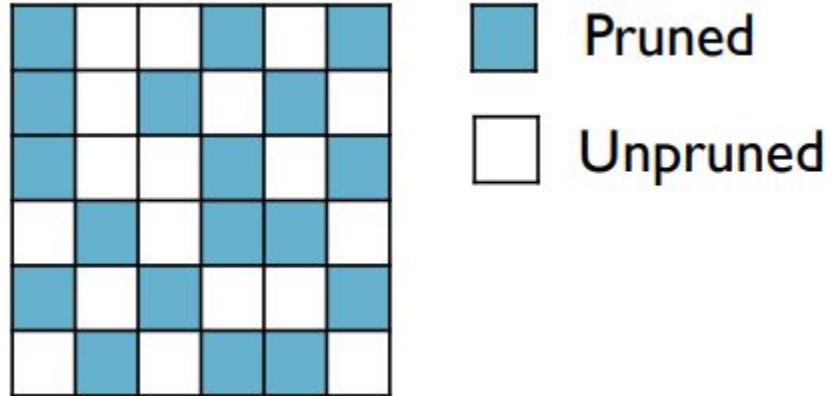


The Lancet Advancing Early Childhood Development: From Science to Scale



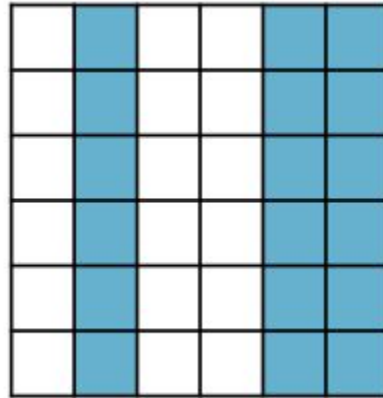
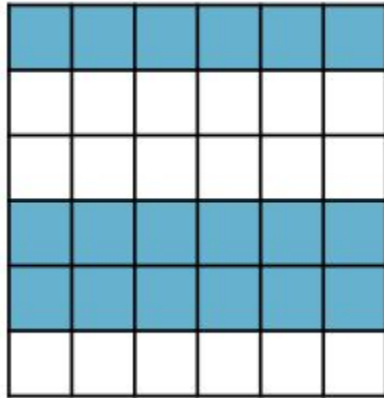
Pruning Strategies - Unstructured



- Pick out $x\%$ of the lowest magnitude weights and remove them
- This is known as unstructured pruning



Pruning Strategies - Structured

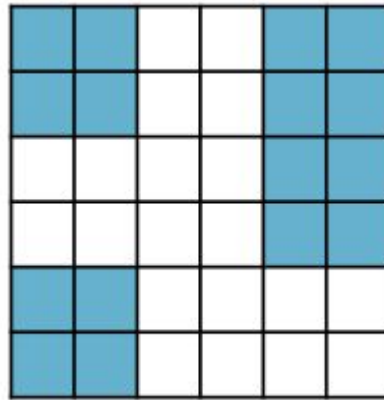
- Pick out the lowest $x\%$ L1-norm for each row / column and remove them
- This is known as structured pruning





 Pruned
 Unpruned

Pruning Strategies - Block

- Pick out x % of the lowest L1-norm in 2D blocks and remove the entire block
- This is known as block pruning



 Pruned

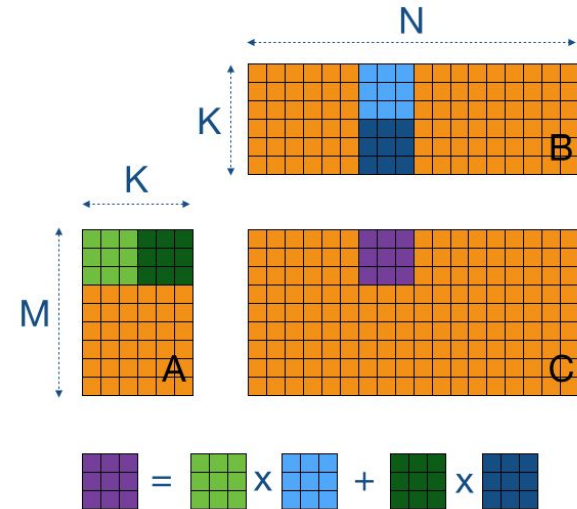
 Unpruned

<https://openai.com/blog/block-sparse-gpu-kernels/>



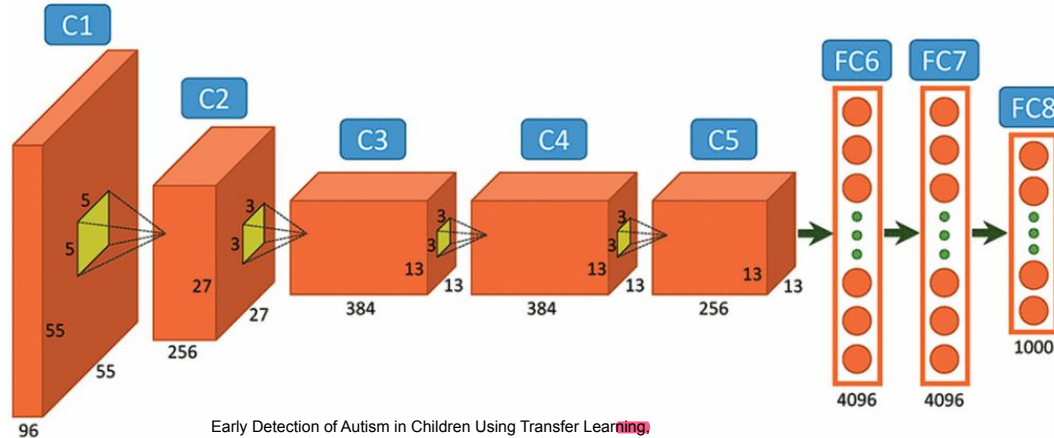
Why structured

- Limited support for sparse matrices in deep learning frameworks which hinder deployment of unstructured sparse matrices
 - May **slow down inference which** can affect real time applications demanding high throughputs
- Tiled matrix multiplications in GPU Tensor Cores
 - **Block based methods are suited to take** advantage of this architecture



Let's Prune!

Layer Type	Output Shape	Kernel/Filters	Parameters
Conv1	55 × 55 × 96	11×11×3, Stride 4	$(11 \times 11 \times 3 \times 96) + 96 = 34,944$
Conv2	27 × 27 × 256	5×5×96, Stride 1	$(5 \times 5 \times 96 \times 256) + 256 = 614,656$
Conv3	13 × 13 × 384	3×3×256, Stride 1	$(3 \times 3 \times 256 \times 384) + 384 = 885,120$
Conv4	13 × 13 × 384	3×3×384, Stride 1	$(3 \times 3 \times 384 \times 384) + 384 = 1,327,488$
Conv5	13 × 13 × 256	3×3×384, Stride 1	$(3 \times 3 \times 384 \times 256) + 256 = 884,992$
FC6	1 × 1 × 4096	6×6×256	$(6 \times 6 \times 256 \times 4096) + 4096 = 37,752,832$
FC7	1 × 1 × 4096	4096	$(4096 \times 4096) + 4096 = 16,781,312$
FC8	1 × 1 × 1000	4096	$(4096 \times 1000) + 1000 = 4,097,000$
Total Parameters	-	-	61,100,840



Pruning Recipe

- Load pretrained model
- Choose which parameters to prune and choice of pruning method
- Prune to required percentage
- Fine tune model to regain any lost accuracy
- Plot a *sparsity vs accuracy* curve to visualise and determine the sweet spot!
- Can an equivalent smaller dense model give similar performance?



Other Pruning Techniques

- Iterative Pruning
 - Prune, train, and repeat for n iterations
 - Usually shown to be beneficial for higher sparsity levels
- Non magnitude techniques
 - Adaptive pruning - Monitor gradients and prune the weights corresponding to the gradients which move away more
 - Low rank matrix factorization
- Pruning from scratch
 - Open research problem
 - No single right idea on what sort of pruning mask to start from
 - Imposing pruning on pretrained models underperforms anyway likely because the model is stuck in a local minimum, and the optimizer settings being used aren't the best to get out of it



Quantization

Quantization

- Perform computations and store tensors using lower precision data types instead of the conventional FP32 precision
 - Typically INT8 but lower widths are within scope too
- FP32 \rightarrow INT8
 - 4x reduction in size

Quantization

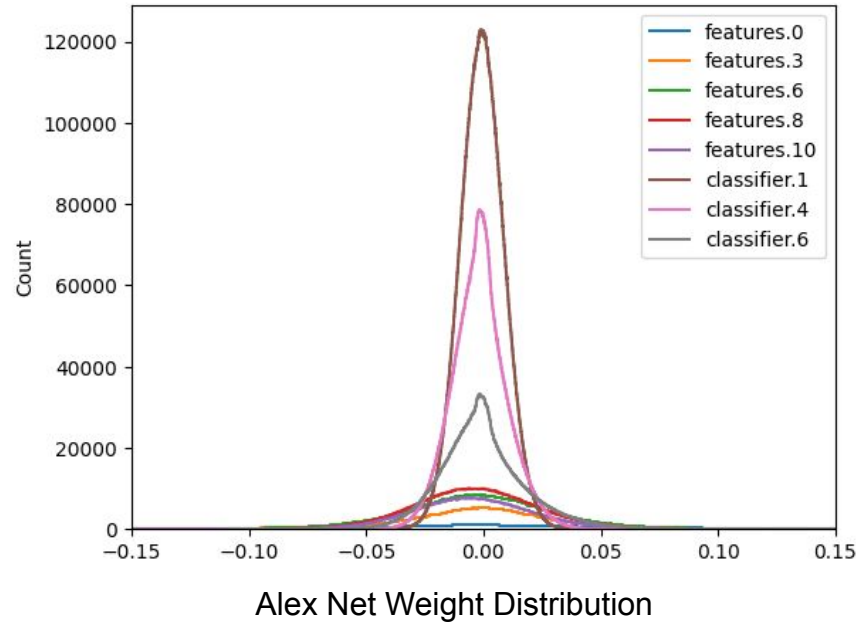
- Casting directly from higher precision to lower precision can lead to very large errors

Data Type	Value	% Deviation	Memory (bits)
FP64	3.141592653589793	-	64
FP32	3.141592653	5.97e-9	32
FP16	3.1415	9.39e-4	16
INT8	3	4.5	8

- 2^n values $\rightarrow n$ bits but in a ML context can lead to a lot of quantization noise
 - FP32 \rightarrow INT8 : 6.8e38 \rightarrow 256

Quantization

- Fortunately neural network model weights usually have a very small range close to 0



Quantization



$$\text{zero point} = \frac{f_{\max} + f_{\min}}{2} \quad \text{scale factor} = \frac{f_{\max} - f_{\min}}{q_{\max} - q_{\min}}$$

$$Q = \frac{F}{\text{scale factor}} + \text{zero point}$$



Quantization Strategies

- Dynamic Quantization
 - Weights quantized, activations in FP and quantized at compute time
- Static Quantization
 - Weights quantized, activations quantized, calibration post training
- Quantization Aware Training (QAT)
 - Weights and activations quantized, quantization numerics modelled while training

Recipe



How to Proceed

- Examine your model architecture
 - Which layers need compression - bar charts are very helpful here!
 - Sanity check: Model file size == state_dict size
 - Use the target size to determine which layers need compression
- Pruning
 - L1, L2, structured, unstructured, ... - sparsity vs accuracy curves to determine your accuracy ceilings
 - Own masks
 - Can smaller dense models with a similar number of parameters give you similar performance?
- Quantization
 - Dynamic, static
 - QAT

Suggested Reading

- [Pruning and Quantization for Deep Neural Network Acceleration: A Survey](#)
- [PyTorch Pruning](#)
- [OpenAI Block Sparsity](#)
- [GPU Architecture](#)
- [Tiled Matrix Multiplication](#)
- [Matrix Multiplication NVIDIA](#)
- [PyTorch Quantization](#)
- [Dynamic Quantization](#)
- [Static Quantization and QAT](#)
- [LLM Pruning](#)
- [MobileLLM](#)
- [The Era of 1-bit LLMs](#)
- [QuEST: Stable Training of LLMs with 1-Bit Weights and Activations](#)



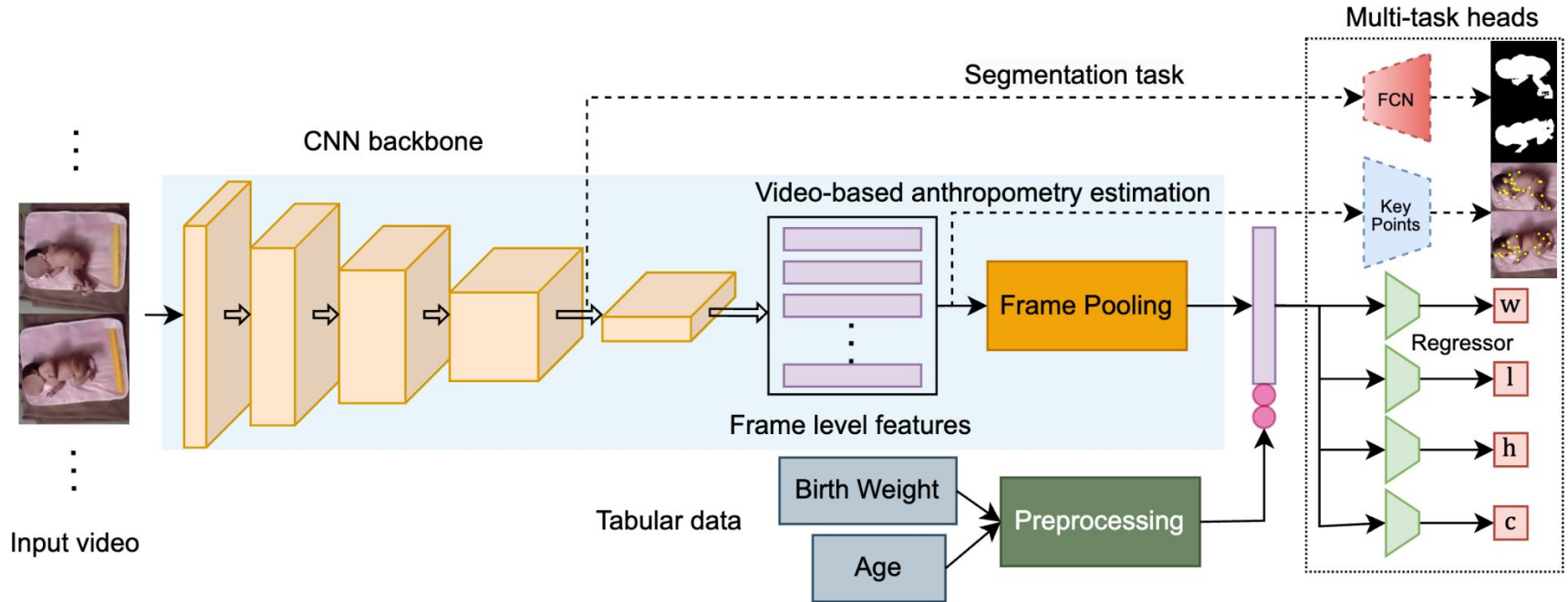


Use Case:

**Newborn
Anthropometry**



Model Architecture



Multi-level Pruning

Pruning (% Filters)	Size (MB)	GFLOPs	MAE (g)
-	105.74	5.37	134.2
25	54.51	3.02	129.4
25 → 33 (50)	26.48	1.34	134.4
50	26.48	1.34	143.9

- Why is the size half when pruning 25% of the filters?
- Why did the MAE improve after pruning?



Thank you

Copyright © Wadhvani AI 2025 All rights reserved

All data and information contained in this document are copyrighted by WIAI and may not be duplicated, copied, modified or otherwise adapted without our written permission. Your use of this document does not grant you any ownership to our content.

Wadhvani AI is a program of the AI Unit of National Entrepreneurship Network (NEN).



WADHWANI AI



www.wadhwaniai.org