The art of speed (Part 1)

Beginner's Companion to Optimize Deep Learning Model Inference

CodeSeoul MLA
May 20, 2023

Outline

1. (Part 1) Motivation
   - The Yin and Yang of Deep Learning: Training vs Inference
2. Background
   - Understanding inference performance
     - Model complexity
     - FLOPs
     - Speed
   - Optimization techniques
     - Weight quantization
     - Model pruning
3. (Part 2) Hands-on tutorial
   - Prerequisites (To be announced ~05/31 via Discord)
   - Quantization
   - Model pruning
   - Benchmarking
   - (Optional) Deployment with OpenVINO
4. References

Overview

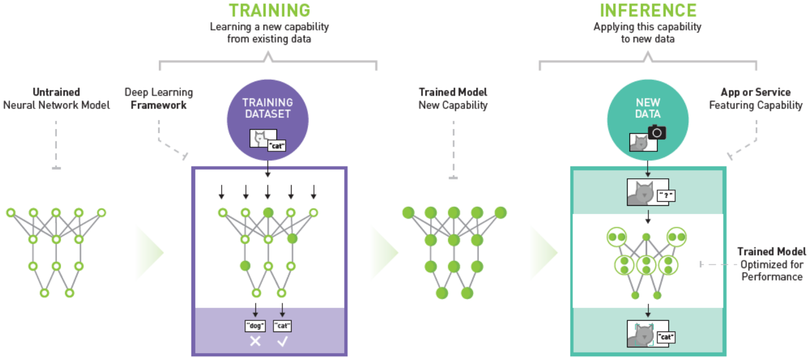

Figure 1: High-level deep learning workflow showing training, then followed by inference. [8]

Training vs Inference

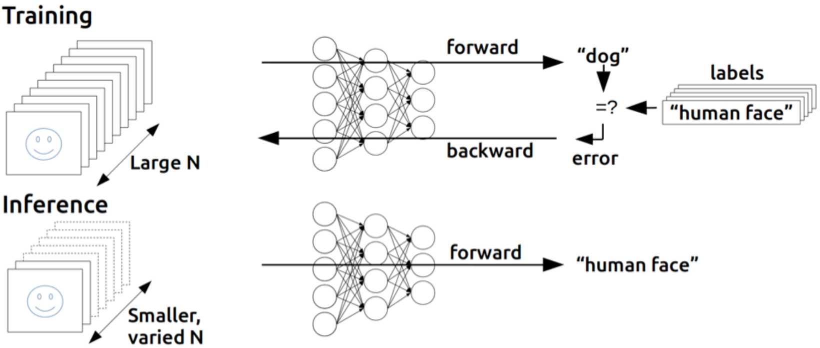

Figure 2: Training uses multiple inputs in large batches to train a deep neural network, while inference extracts information from new inputs in smaller batches using the trained network. [2]

Why do we need inference optimization?

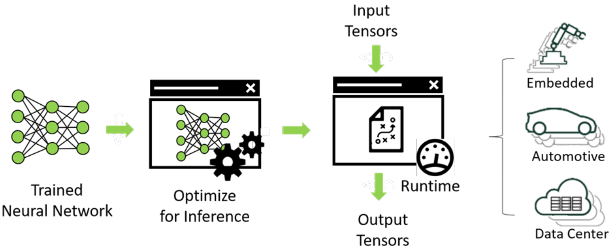

Figure 3: Inference optimization for deployment. [1]

- **Improved performance**: optimized models can deliver faster and more responsive predictions
- **Efficient resource utilization**: leads to cost savings, as less powerful hardware, or multiple models simultaneously
- **Deployment flexibility**: expands reach and applicability (mobile, edge, embedded systems, etc.)

Existing challenges...

- Accuracy-efficiency trade-off
- Model complexity and size
- Hardware and platform variability
- Compatibility with frameworks and libraries

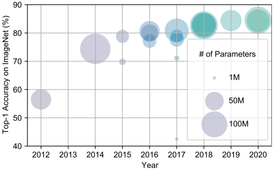

Figure 4: Timeline of top NN models with the number of parameters (from ImageNet). [3]

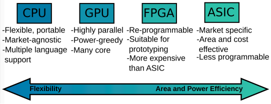

Figure 5: Hardware platforms comparison [3]

Figure 6: Deep learning frameworks landscape

To sum up

- To optimize the model's **performance**, **accuracy**, and **generalization** on the training data.
- Focuses on **latency**, **memory usage**, **power consumption** as it directly impacts the real-time performance.

Number of parameters

A sample linear model in Tensorflow:

```python
model = keras.Sequential([
    keras.layers.Dense(5, activation='relu', input_shape=(3,)),
    keras.layers.Dense(8, activation='relu', trainable=False),
    keras.layers.Dense(10, activation='relu', trainable=False),
    keras.layers.Dense(15, activation='relu'),
    keras.layers.Dense(4, activation='softmax'),
])
```

Checking model summary in Tensorflow:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 5)                 20
_________________________________________________________________
dense_1 (Dense)              (None, 8)                 48
_________________________________________________________________
dense_2 (Dense)              (None, 10)                90
_________________________________________________________________
dense_3 (Dense)              (None, 15)                165
_________________________________________________________________
dense_4 (Dense)              (None, 4)                 64
=================================================================
Total params: 387
Trainable params: 249
Non-trainable params: 138
```

FLOPs

A computational cost of a model:

- A widely used metric as the proxy, **FLOPs**, measures *the number of floating-point arithmetic operations* performed in a deep learning model. It includes operations such as additions, subtractions, multiplications, and divisions involving floating-point numbers.
- Alternative to FLOPs, **MACs**, short for *the number of multiply-accumulate operations* [6], count both multiplication and addition as a single unit of operation.

You can check these repos to count FLOPs of your (tf/pt) model:
`https://github.com/Mr-TalhaIlyas/Tensorflow-Keras-Model-Profiler`
`https://github.com/sovrasov/flops-counter.pytorch`

In convolutional neural networks (CNNs), which heavily rely on convolution operations, MACs are often used as a metric to estimate the computational workload.

FLOPs

A sample pretrained CNN model in Tensorflow:

```
form tensorflow.keras.applications import VGG16

model = VGG16(include_top=True, weights="imagenet", input_tensor=None,
              input_shape=None, pooling=None, classes=1000,
              classifier_activation="softmax")
```

Using library to print number of FLOPs:

```
from model_profiler import model_profiler

Batch_size = 128
profile = model_profiler(model, Batch_size)

print(profile)
```

Output printed in terminal:

```
| Model Profile                      | Value              | Unit     |
|------------------------------------|--------------------|----------|
| Selected GPUs                      | ['0', '1']         | GPU IDs  |
| No. of FLOPs                       | 0.30932349055999997 | BFLOPs  |
| GPU Memory Requirement             | 7.4066760912537575 | GB       |
| Model Parameters                   | 138.357544         | Million  |
| Memory Required by Model Weights   | 527.7921447753906  | MB       |
```

FLOPs

An example from FLOPS evaluations in papers:

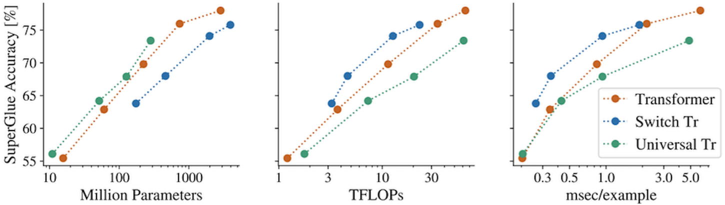

Figure 7: Comparison of standard Transformers, Universal Transformers and Switch Transformers in terms of the number of parameters, FLOPs, and throughput[4]

The relationship between **the number of parameters** and **FLOPs** in deep learning models is **not** necessarily *linear*. The relationship can vary depending on several factors, including the model architecture, layer types, and specific operations used. For example, convolutional layers tend to involve more FLOPs due to the convolution operations, while fully connected layers may have a higher number of parameters but lower FLOPs.

Speed

Performance means **how fast** the model processes the live data. Its two key metrics, *latency* and *throughput* are fundamentally interconnected:

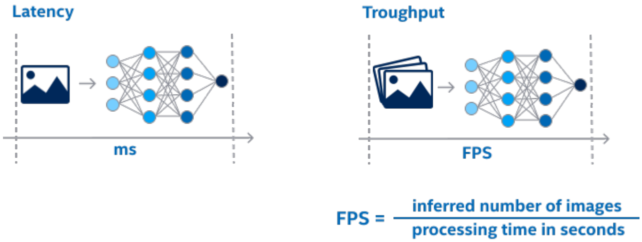

Figure 8: Understanding latency and throughput. [5]

- **Latency** measures the required time to process a single input (ms)
- **Throughput** measures overall number of inferences per second (or frames per second, FPS for visual processing)

Speed

If we were to be given two models, performing equally well on a given task, which one should we choose?

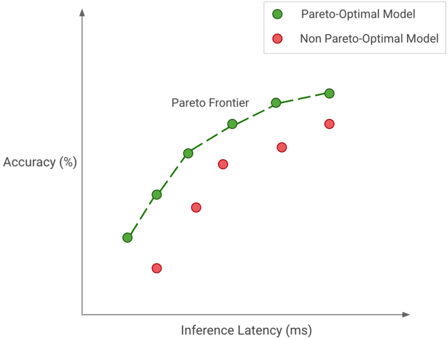

Figure 9: Pareto Optimality: Green dots represent pareto-optimal models (together forming the pareto-frontier), where none of the other models (red dots) get better accuracy with the same inference latency, or the other way around. [7]

Weight quantization

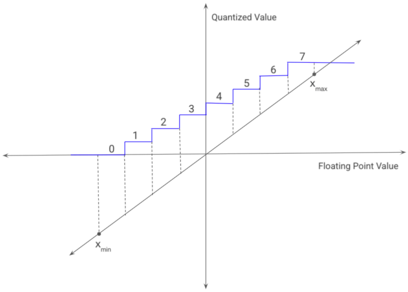

Figure 10: Quantizing floating-point continuous values to discrete values. [7]

Quantization and dequantization steps

$$\text{quantize}(x) = x_q = \text{round}\left(\frac{x}{s}\right) + z \tag{1}$$

$$\text{dequantize}(x) = \hat{x} = s(x_q - z) \tag{2}$$

where s scale value, z zero-point value; more details at [7]

Model pruning

Model pruning **reduces the size** of a neural network by *removing unnecessary connections or weights*, improving computational efficiency without compromising performance.

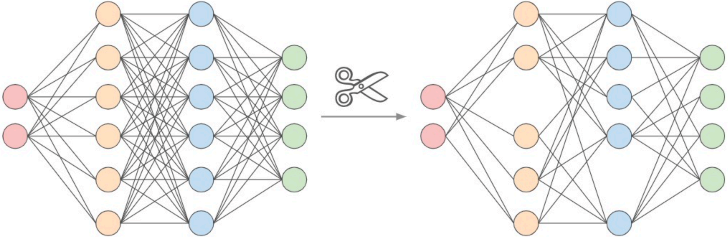

Figure 11: A simplified illustration of pruning weights (connections) and neurons (nodes) in a neural network comprising of fully connected layers [7]

Model pruning

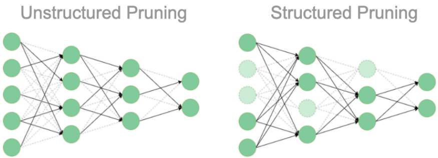

Figure 12: Understanding unstructured and structured pruning [9]

Model Architecture	Sparsity Type	Sparsity %	FLOPs	Top-1 Accuracy %	Source
MobileNet v2 - 1.0	Dense (Baseline)	0%	1x	72.0%	Sandler et al. [133]
	Unstructured	75%	0.27x	67.7%	Zhu et al. [167]
	Unstructured	75%	0.52x	71.9%	Evci et al. [54]
	Structured (block-wise)	85%	0.11x	69.7%	Elsen et al.
	Unstructured	90%	0.12x	61.8%	Zhu et al. [167]
	Unstructured	90%	0.12x	69.7%	Evci et al. [54]

Figure 13: A sample of various sparsity results on the MobileNet v2 architecture with depth multiplier = 1.0. [7]

References I

[1] Michael Andersch. *Dev Stack: How to Optimize Your Model for Inference?* URL: https://www.devstack.co.kr/inference-optimization-using-tensorrt/.

[2] Michael Andersch. *Inference: The Next Step in GPU-Accelerated Deep Learning*. URL: https://developer.nvidia.com/blog/inference-next-step-gpu-accelerated-deep-learning/.

[3] Maurizio Capra et al. "Hardware and software optimizations for accelerating deep neural networks: Survey of current trends, challenges, and the road ahead". In: *IEEE Access* 8 (2020), pp. 225134–225180.

[4] William Fedus, Barret Zoph, and Noam Shazeer. "Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity". In: *The Journal of Machine Learning Research* 23.1 (2022), pp. 5232–5270.

[5] *Introduction to Performance Optimization - OpenVINO Documentation*. URL: https://docs.openvino.ai/2022.1/openvino_docs_optimization_guide_dldt_optimization_guide.html.

References II

[6] Jeff Johnson. "Rethinking floating point for deep learning". In: *arXiv preprint arXiv:1811.01721* (2018).

[7] Gaurav Menghani. "Efficient deep learning: A survey on making deep learning models smaller, faster, and better". In: *ACM Computing Surveys* 55.12 (2023), pp. 1–37.

[8] Brad Nemire. *NVIDIA Deep Learning Inference Platform Performance Study*. URL: https://developer.nvidia.com/blog/nvidia-deep-learning-inference-platform-performance-study/.

[9] neuralmagic. *Part 1: What is Pruning in Machine Learning?* URL: https://neuralmagic.com/blog/pruning-overview/.

Thank you for your attention!

- Lecture contents:
  - Slides: https://github.com/CodeSeoul/machine-learning/230520-inference-p1/lecture.pdf
- Connect with us:
  - Discord: https://discord.gg/HFknCs8
  - GitHub: https://github.com/CodeSeoul/machine-learning
  - Donations: We are a registered non-profit and run on donations from people like you! NongHyeop/농협은행 | 301 0275 2831 81 (코드서울)