ODTÜ METU

TÜBİTAK

EURO 4SEE

Optimizing Deep Learning Systems for Hardware

Assoc. Prof. Erdem AKAGÜNDÜZ, METU

ncc@ulakbim.gov.tr

Part III : Model-level Optimization

- PIII.a : Model compression
- PIII.b : Efficient architectures

Optimizing a DL model on HW

	Model-Level	System-Level
Scope	Neural network structure & parameters	Hardware, runtime, parallelism strategies
Examples	Pruning, quantization, distillation…	Pipeline parallelism, compiler optimizations
Target	Model size, accuracy-efficiency tradeoff	Latency, throughput, hardware utilization
Who applies it?	ML researchers & algorithm designers	Systems engineers, ML infra-teams

Part III: Model-Level Optimizations

- Definition: Techniques applied directly to the neural network architecture or weights to improve efficiency.
- Focus: Reduce computation, memory, and power usage without significantly impacting accuracy.
- Why it matters?
  - No system-level change
  - Smaller models = faster inference
  - Better fit for resource-constrained devices (edge, mobile)
  - Lower energy and deployment costs

Part III.a: Compression

- a) Model Compression
  - Pruning
  - Quantization
  - Knowledge Distillation
- b) Architecture Optimization
  - Case: Efficient CNNs (e.g., MobileNet)

Model Compression?

- Definition: Techniques that reduce the size and complexity of a deep learning model
- Goal: **Maintain** accuracy while improving:
  - Inference speed
  - Memory footprint
  - Energy efficiency

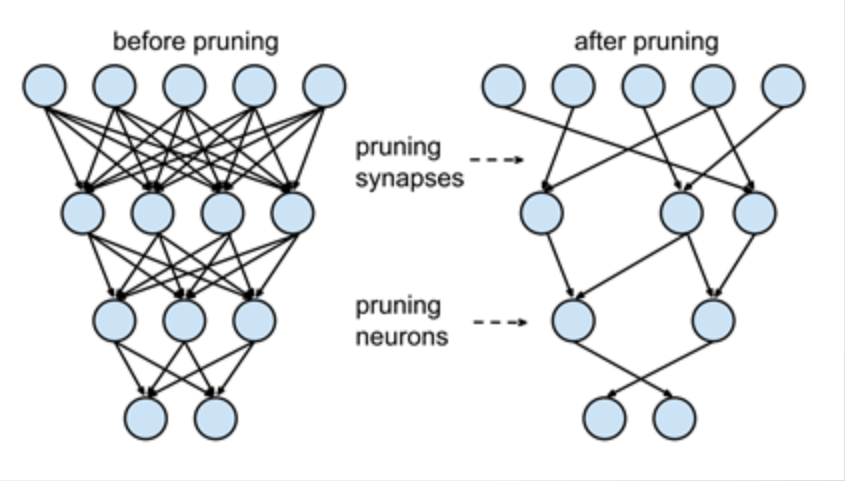
Why Model Compression?

- Real-world drivers:
  - Edge deployment (IoT, smartphones)
  - Faster inference in production systems
  - Lower costs in datacenters (compute + energy)

- Compression ≠ just "shrinking". It's about smart tradeoffs.

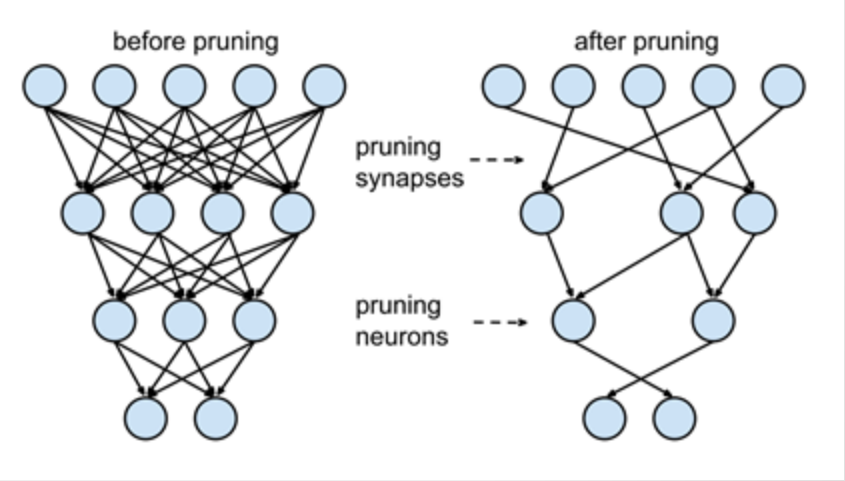
Pruning

- Definition: Eliminate weights or entire neurons that contribute little to output
- Types:
  - Weight pruning (unstructured)
  - Neuron/channel pruning (structured)
  - Visualization:



Pruning

- Pros:
  - Reduces parameter count and computation
  - Works well post-training ("fine-tune" stage)
- Cons:
  - May require retraining/fine-tuning
  - Unstructured pruning may not lead to real speedups without specialized libraries/hardware



Pruning - but how?

- Magnitude-Based Pruning
  - Remove weights with smallest absolute values
  - Simple and effective; often used post-training
- Structured Pruning
  - Remove entire filters, channels, or layers
  - Hardware-friendly → leads to real speedups
- Dynamic / Iterative Pruning
  - Gradually prune during training (e.g., Lottery Ticket Hypothesis)
  - Allows the network to adapt and recover
- Learning-Based Pruning
  - Use optimization or learn mask weights (e.g., L0 regularization)
  - More complex but can yield better performance

Pruning - but how?

Method	Pros	Cons
Magnitude-Based	Simple, fast, widely used	May not accelerate inference without support
Structured Pruning	Leads to real speedups on hardware	Can reduce accuracy more if not carefully tuned
Iterative/Dynamic	Allows model to adapt gradually	Requires longer training/fine-tuning cycles
Learning-Based	Can find better pruning masks	Computationally complex and harder to tune

Quantization

- Definition: Replace high-precision floats with lower-bit representations

- Common types:
  - Post-training quantization (PTQ)
  - Quantization-aware training (QAT)
- Popular formats:
  - int8, float16, bfloat16

Post-Training Quantization

- Definition: Apply quantization to a trained model without retraining
- Workflow:
  - Train model as usual (float32)
  - Convert weights/activations to lower precision (e.g., int8)

✓ Fast and easy to apply
✓ Doesn’t require access to training data (in some versions)
× May suffer accuracy drop, especially for sensitive models (e.g., Transformers)
× Limited control over quantization effects

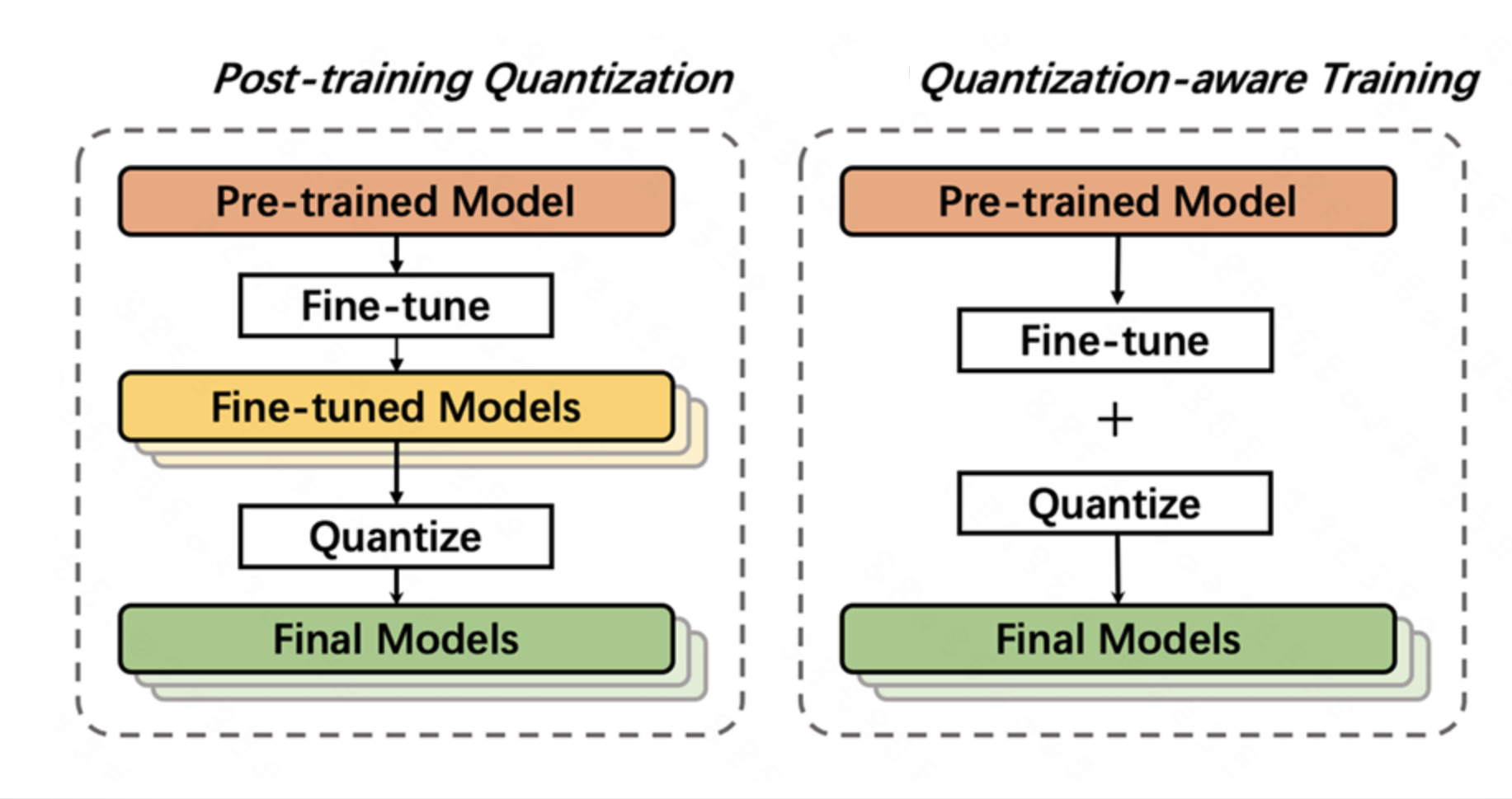
Quantization-aware Training

- Definition: Simulate quantization effects during training
- Workflow:
  - Insert fake quantization ops during training (e.g., simulate int8 rounding)
  - Model learns to be robust to quantization noise

✓ Much smaller accuracy drop vs. PTQ
✓ Works well for complex tasks (e.g., NLP, object detection)

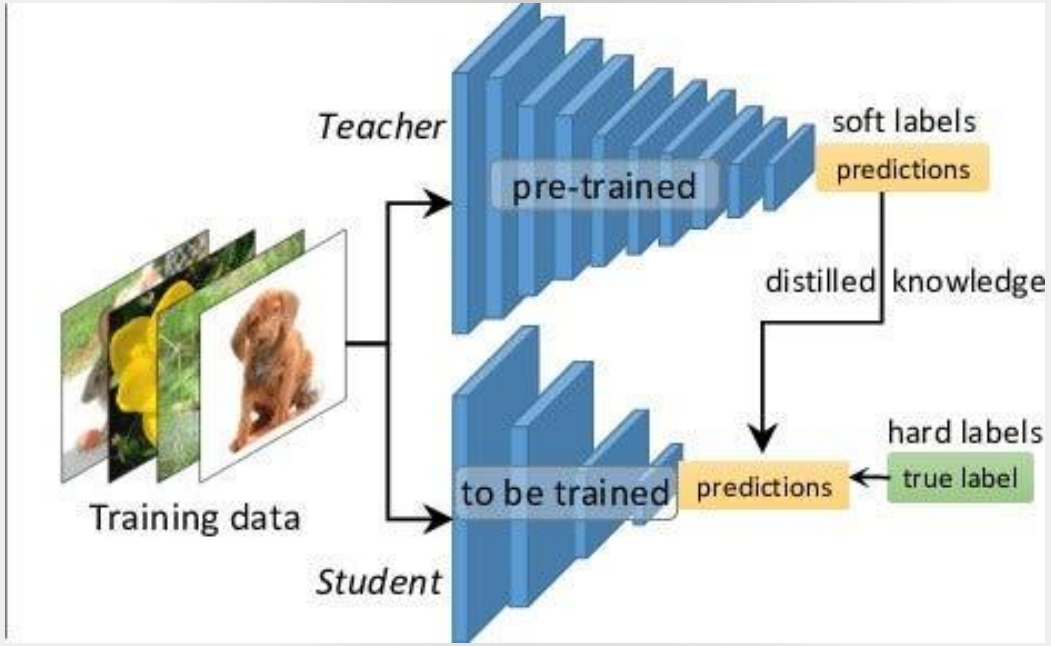
✗ Requires access to training pipeline and data
✗ Slower training due to extra ops
✗ Usually applied to a trained model

PQT vs QAT


Post-training Quantization
Pre-trained Model
Fine-tune
Fine-tuned Models
Quantize
Final Models
Quantization-aware Training
Pre-trained Model
Fine-tune
+
Quantize
Final Models

Knowledge Distilation

- Train a "student" model to mimic a larger, pre-trained "teacher" model



Knowledge Distilation

- Benefits:
  - Student model is smaller and faster
  - Retains much of teacher's performance

- Use Cases:
  - Deploying large LLMs in mobile settings (e.g., TinyBERT from BERT)
  - Ensembles distilled into single model for efficiencyTrain a "student" model to mimic a larger, pre-trained "teacher" model

Model Compression Trade-offs

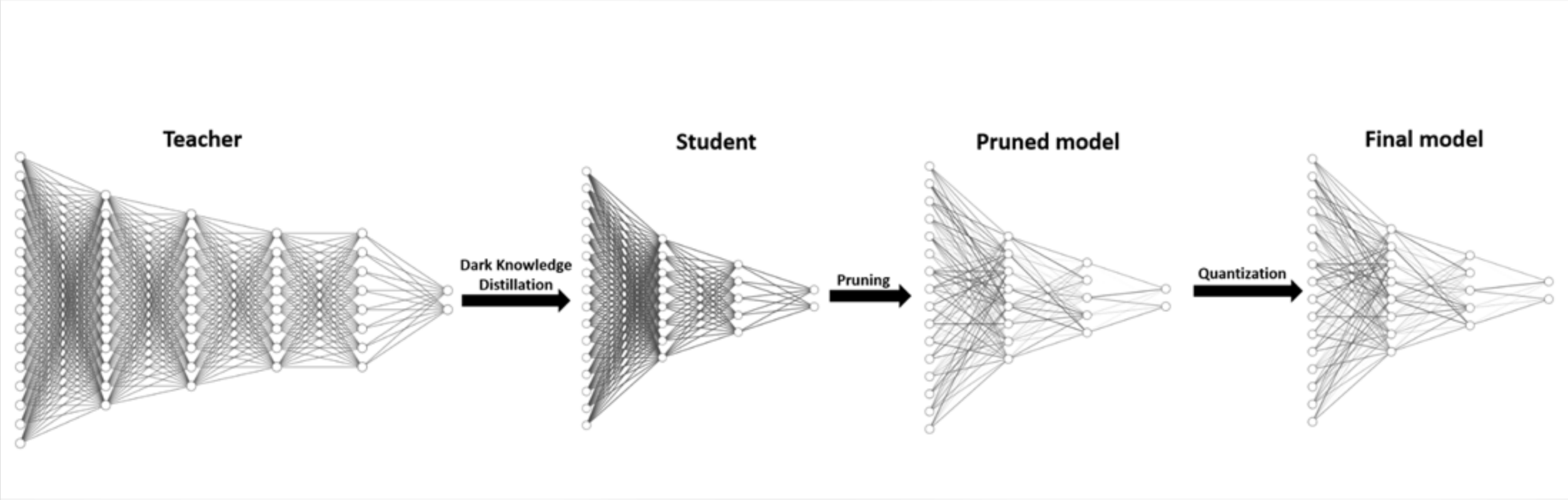
Technique	Description	When to Use
Pruning	Remove unnecessary weights/connections	Overparameterized models
Quantization	Lower precision (e.g., float32 → int8)	Speed + memory critical environments
Knowledge Distillation	Train small model to mimic large one	When deploying compact models

Model Compression Trade-offs

Technique	Size ↓	Speed ↑	Accuracy Impact
Pruning	✅✅	✅	Medium
Quantization	✅✅✅	✅✅	Low–Medium
Knowledge Distill.	✅✅	✅✅	Low

Combined?

- Yes, if you have a pre-trained teacher



Next: Part III.b

- PIII.a : Model compression
- PIII.b : Efficient architectures

EURO 4SEE

Thanks!

Co-funded by
the European Union

EuroHPC
Joint Undertaking

This project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 101191697. The JU receives support from the Digital Europe Programme and Germany, Türkiye, Republic of North Macedonia, Montenegro, Serbia, Bosnia and Herzegovina.