Network Compression (2)

Aaron Zhao, Imperial College London

Introduction

Neural Network Compression (2)

In this lecture we continue exploring compression techniques:

- **Winograd Transformed Convolution** — replace multiplications with additions
- **Low-rank Approximation** — decompose weight matrices into smaller factors
- **Knowledge Distillation** — train a small student using a large teacher's outputs
- **Chaining Compression Algorithms** — combine orthogonal methods for multiplicative gains

Winograd Transformation

Winograd Transformation: Motivation

Core idea: Replace expensive operations (multiplications) with cheaper ones (additions).

Matrix–vector convolution $F(2,3)$ (2 outputs, filter size 3):

$$F(2,3) = \begin{bmatrix} d_0 & d_1 & d_2 \\ d_3 & d_4 & d_5 \end{bmatrix} \begin{bmatrix} g_0 \\ g_1 \\ g_2 \end{bmatrix}$$

This requires $2 \times 3 = 6$ multiplications.

Can we do it with fewer?

Winograd: Reducing Multiplications

The same result can be computed as:

$$F(2,3) = \begin{bmatrix} m_1 + m_2 + m_3 \\ m_2 - m_3 - m_4 \end{bmatrix}$$

where the four intermediate products are:

$$m_1 = (d_0 - d_2)g_0$$

$$m_2 = (d_1 + d_2)\frac{g_0 + g_1 + g_2}{2}$$

$$m_3 = (d_2 - d_1)\frac{g_0 - g_1 + g_2}{2}$$

$$m_4 = (d_1 - d_3)g_2$$

Now only **4 multiplications** are needed (down from 6), at the cost of a few extra additions.

Winograd: Reducing Multiplications (ii)

Additions are much cheaper than multiplications on hardware.

Winograd: Matrix Form

The Winograd transformation can be written compactly as:

$$Y = A^T[(GgG^T) \cdot (B^T dB)]A$$

where g is the filter and d is the input tile.

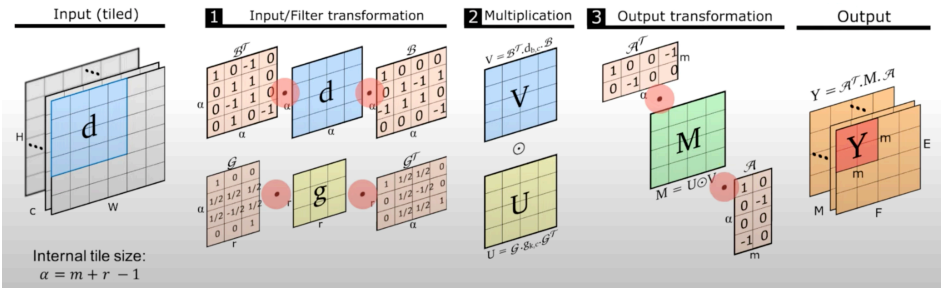

Figure 1: Winograd transformation: filter g and data d are projected into Winograd space for element-wise multiplication

- G, B, A are fixed transform matrices (precomputed, no runtime cost)

Winograd: Matrix Form (ii)

- The point-wise multiplication replaces the full convolution
- Filter transforms GgG^T can be **precomputed offline** since g is fixed at inference time

Low-rank Approximation

Low-rank Approximation: Motivation

Weight matrices in neural networks are often **low-rank** in practice — most of the information is captured by a small number of singular values.

Singular Value Decomposition (SVD): any matrix $W \in \mathcal{R}^{m \times n}$ can be written as:

$$W = U \Sigma V^T$$

where $U \in \mathcal{R}^{m \times m}$, $\Sigma \in \mathcal{R}^{m \times n}$ (diagonal), $V \in \mathcal{R}^{n \times n}$.

Low-rank approximation: keep only the top k singular values:

$$W \approx \widehat{W} = U_k \Sigma_k V_k^T$$

This replaces one $(m \times n)$ matrix with two smaller matrices: $(m \times k)$ and $(k \times n)$, giving compression ratio $\frac{mn}{k(m+n)}$.

Low-rank Approximation: In Practice

For a linear layer $y = Wx$ where $W \in \mathcal{R}^{m \times n}$:

$$y = Wx \approx U_k(\Sigma_k V_k^T x)$$

This is equivalent to **two sequential linear layers** with no bias and a bottleneck of size k — the same structure as an **adapter** or **LoRA** layer.

Key properties:

- The approximation error is bounded by the $(k+1)$-th singular value $\sigma_{\{k+1\}}$
- Compression is effective when the **singular value spectrum decays quickly**
- Can be applied to convolutional layers by reshaping the weight tensor into a 2D matrix
- **No retraining required** for PTQ-style application; fine-tuning recovers accuracy

Low-rank Approximation: For Convolutions

For a convolutional layer with weight $W \in \mathcal{R}^{C_o \times C_i \times K \times K}$:

Approach 1 — reshape and SVD: Reshape W to $(C_o, C_i K^2)$, apply SVD, reshape factors back.

Approach 2 — Tucker decomposition: Decompose W along the channel dimensions only:

$$W \approx G \times_1 U_1 \times_2 U_2$$

where $G \in \mathcal{R}^{r_1 \times r_2 \times K \times K}$ is the core tensor and $U_1 \in \mathcal{R}^{C_o \times r_1}$, $U_2 \in \mathcal{R}^{C_i \times r_2}$ are factor matrices.

The result is equivalent to three sequential convolutions: 1×1 projection, $K \times K$ core convolution, 1×1 projection — similar in spirit to MobileNet's depthwise-separable design.

Knowledge Distillation

The Teacher–Student Paradigm

Knowledge Distillation (KD) transfers the "knowledge" of a large, capable **teacher** model into a smaller **student** model.

The Teacher–Student Paradigm (ii)

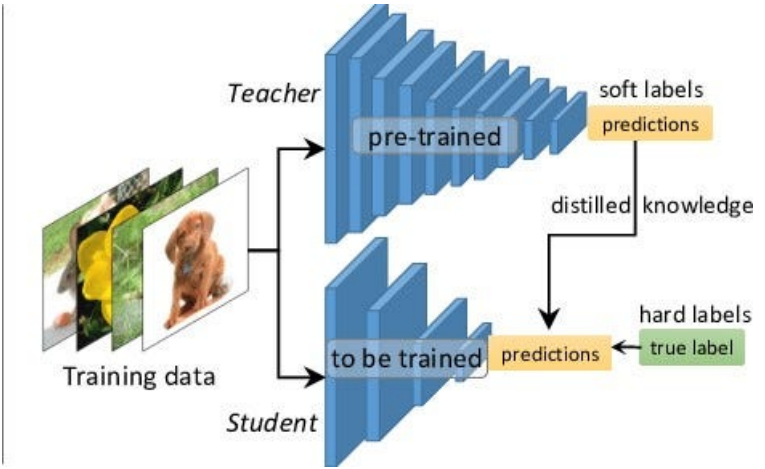

Figure 2: Teacher–student knowledge distillation: the student learns from both ground-truth labels and the teacher's outputs

The Teacher–Student Paradigm (iii)

The student is trained to match not just the hard labels, but the richer information encoded in the teacher's predictions — including confidence scores and inter-class relationships.

Soft Labels

The teacher's output $y_{\text{soft}} = f_{w(X)}$ forms a **soft label** that carries more information than the one-hot ground truth.

The student is trained with a combined loss:

$$\mathcal{L} = \mathcal{L}_{\text{CE}} + \mathcal{L}_{\text{KD}}$$

where the distillation loss is:

$$\mathcal{L}_{\text{KD}} = g(y_{\text{soft}}, f'_{w'}(X))$$

and $f'_{w'}(X)$ is the student's output.

Temperature scaling is often applied to the teacher's logits before softmax to soften the distribution further, making small probabilities more informative.

Logits-based KD

The simplest form of distillation matches the **output logits** of teacher and student:

$$\mathcal{L}_{\text{KD}} = g(y_{\text{soft}}, f'_{w'}(X))$$

Common choices for g:

- l_p **norm** (e.g. MSE between logits)
- **KL divergence** between softmax distributions
- **Cross-entropy** with teacher soft labels

Logits-based KD is easy to implement and works well when teacher and student have similar output spaces.

Activation-based KD

Activation-based KD extends distillation beyond the final layer by aligning intermediate representations:

$$\mathcal{L}_{\text{KD}} = \sum_{i=1}^{L} g\left(y_{\text{act}}^{i}, f_{\text{act}}^{i(X)}\right)$$

where L is the number of layers and y_{act}^{i} is the teacher's activation at layer i.

Activation-based KD (ii)

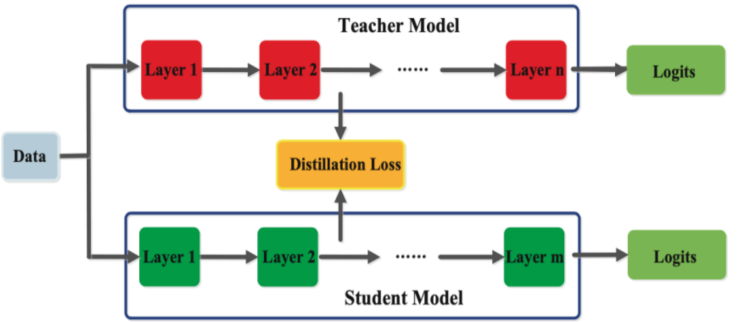

Figure 3: Activation-based KD aligns intermediate feature maps between teacher and student

This encourages the student to develop similar internal representations, not just similar outputs.

Attention-map KD

For activation tensors $x_{\text{act}} \in \mathcal{R}^{C\times H\times W}$, define an **attention map** p : $\mathcal{R}^{C\times H\times W} \to \mathcal{R}^{H\times W}$:

- Sum of absolute values: $p(x) = \sum_{i=1}^{C} |x_i|$
- Power-weighted sum: $p(x) = \sum_{i=1}^{C} |x_i|^m$

Attention-map KD (ii)

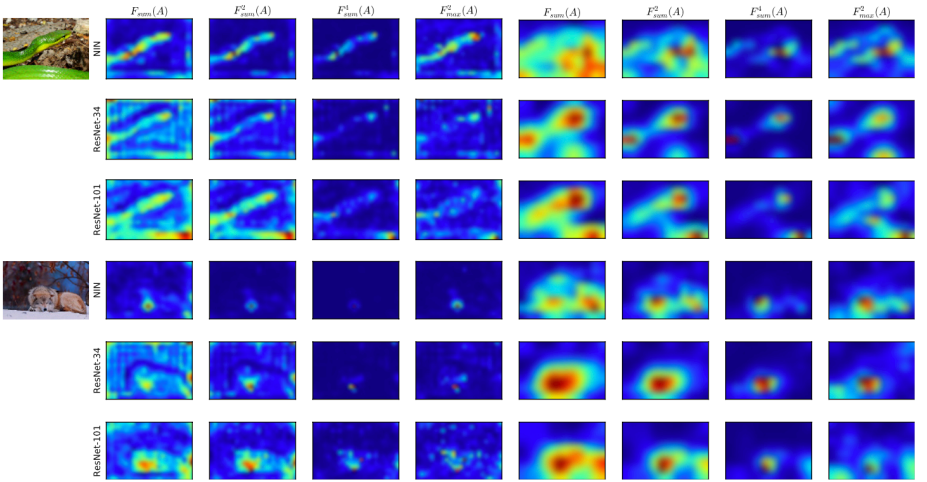

Figure 4: Attention maps of high-performing networks are often similar

Attention-map KD (iii)

High-performing networks tend to produce similar attention maps. The distillation loss encourages the student to match the teacher's normalized attention maps:

$$\mathcal{L} = \mathcal{L}_{\mathrm{CE}} + \frac{\beta}{2} \sum_{i=1}^{L} \left\| \frac{Q_S^i}{\|Q_S^i\|_2} - \frac{Q_T^i}{\|Q_T^i\|_2} \right\|_p$$

where Q_S^i and Q_T^i are vectorized attention maps for student and teacher at layer i.

KD with Quantization: TinyBERT

KD can be combined with quantization: the **teacher** is the full-precision model and the **student** is the quantized model.

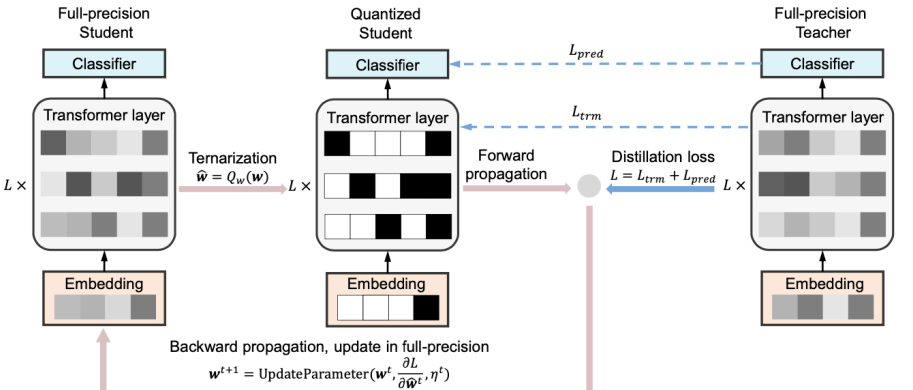

Figure 5: TinyBERT distillation framework: three distillation losses align embedding, attention, and logit layers

KD with Quantization: TinyBERT (ii)

TinyBERT uses three distillation losses:

1. **Embedding + hidden layer loss** — MSE between teacher and student embedding outputs and all Transformer layer outputs
2. **Attention score loss** — MSE between teacher and student attention matrices from all heads and all layers
3. **Logit loss** — soft cross-entropy between teacher and student output logits

TinyBERT Results

		W-E-A (#bits)	Size (MB)	MNLI-m/mm	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE
	BERT	32-32-32	418 (×1)	84.5/84.9	87.5/90.9	92.0	93.1	58.1	89.8/89.4	90.6/86.5	71.1
	TinyBERT	32-32-32	258 (×1.6)	84.5/84.5	88.0/91.1	91.1	93.0	54.1	89.8/89.6	91.0/87.3	71.8
2-bit	Q-BERT	2-8-8	43 (×9.7)	76.6/77.0	-	-	84.6	-	-	-	-
	Q2BERT	2-8-8	43 (×9.7)	47.2/47.3	67.0/75.9	61.3	80.6	0	4.4/4.7	81.2/68.4	52.7
	TernaryBERT$_{TWN}$ (ours)	2-2-8	28 (×14.9)	83.3/83.3	86.7/90.1	91.1	92.8	55.7	87.9/87.7	91.2/87.5	72.9
	TernaryBERT$_{LAT}$ (ours)	2-2-8	28 (×14.9)	83.5/83.4	86.6/90.1	91.5	92.5	54.3	87.9/87.6	91.1/87.0	72.2
	TernaryTinyBERT$_{TWN}$ (ours)	2-2-8	18 (×23.2)	83.4/83.8	87.2/90.5	89.9	93.0	53.0	86.9/86.5	91.5/88.0	71.8
8-bit	Q-BERT	8-8-8	106 (×3.9)	83.9/83.8	-	-	92.9	-	-	-	-
	Q8BERT	8-8-8	106 (×3.9)	-/-	88.0/-	90.6	92.2	58.5	89.0/-	89.6/-	68.8
	8-bit BERT (ours)	8-8-8	106 (×3.9)	84.2/84.7	87.1/90.5	91.8	93.7	60.6	89.7/89.3	90.8/87.3	71.8
	8-bit TinyBERT (ours)	8-8-8	65 (×6.4)	84.4/84.6	87.9/91.0	91.0	93.3	54.7	90.0/89.4	91.2/87.5	72.2

Figure 6: TinyBERT achieves better accuracy than quantization alone under the same bit-width budget

- KD acts as a form of regularization during quantization-aware fine-tuning
- TinyBERT generally achieves **better accuracy** than direct quantization at the same bit-width
- The three-level distillation preserves both structural (attention) and semantic (logit) knowledge

Chaining Compression Algorithms

Chaining Compression Techniques

Many compression techniques operate on **orthogonal** aspects of a model:

- **Pruning** reduces the number of non-zero parameters
- **Quantization** reduces the bit-width of each parameter
- **Lossless coding** (e.g. Huffman) exploits statistical redundancy in the compressed representation

Because they target different sources of redundancy, their gains **multiply** when chained.

Chaining Compression Techniques (ii)

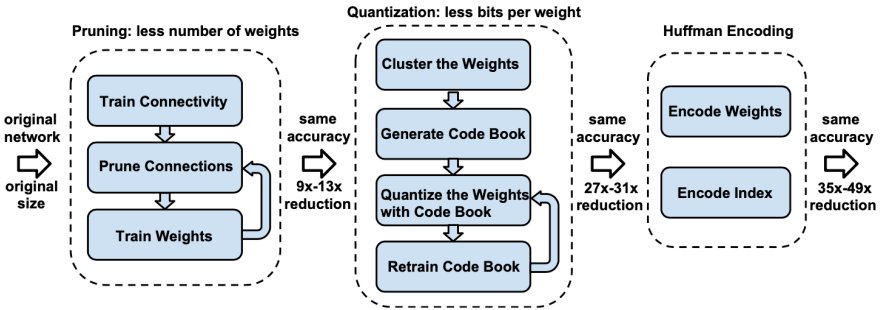

Figure 7: Deep Compression pipeline: prune → re-train → quantize → re-train → Huffman encode

Chaining: Experimental Results

Example: fine-grained pruning + fixed-point quantization on a VGG variant trained on CIFAR-10.

Table 1: Chaining pruning and quantization on CIFAR-10 (VGG variant)

Method	Bitwidth	Density	Compression	Top-1 Acc.
Baseline	32	100%	–	91.37%
Fixed-point	4	100%	8×	89.64%
Dynamic fixed-point (DFP)	4	100%	8×	90.63%
Fine-grained pruning	32	15.65%	6.39×	91.12%
Pruned + fixed	6	15.65%	33.92×	90.59%
Pruned + DFP	6	15.65%	33.92×	91.04%

- Quantization alone: up to 8 × compression, $< 1\%$ accuracy drop
- Pruning + quantization: **33.92 × compression** with comparable accuracy

Chaining: Experimental Results (ii)

- Pruning can act as regularization, sometimes **improving** accuracy

Lossy and Lossless Compression

Lossy compression modifies model parameters — accuracy may degrade, re-training helps recover it:

- Pruning
- Quantization

Lossless compression exploits statistical redundancy without changing values — no accuracy impact, no re-training needed:

- Huffman coding (encodes frequent weight values with shorter codes)
- Run-length encoding (encodes runs of zeros after pruning)

Chaining lossy then lossless stages is the standard pipeline in Deep Compression.

Special Hardware Support

Achieving the theoretical speedup from compression requires **hardware support**:

- **Sparse matrix multiplication** — skip computation for zero weights (pruning)
- **Huffman decoder/encoder** — compress/decompress weight storage (Huffman coding)
- **Low-precision arithmetic units** — INT4/INT8 multiply-accumulate (quantization)

Special Hardware Support (ii)

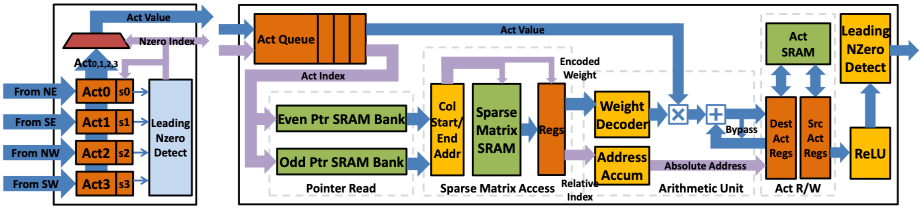

Figure 8: Hardware accelerator components supporting compressed neural network inference

Without matching hardware, compressed models may not achieve wall-clock speedup even if FLOPs are reduced.

Summary

Winograd Transformation:

- Replace convolution multiplications with additions via fixed transform matrices
- Filter transforms are precomputed offline; runtime cost is element-wise multiplication

Low-rank Approximation:

- SVD / Tucker decomposition reduces weight matrices to low-rank factors
- Equivalent to inserting a bottleneck layer; no retraining required for PTQ-style use

Knowledge Distillation:

- **Logits-based** — match teacher output distributions
- **Activation-based** — match intermediate feature maps
- **Attention-map** — match spatial attention patterns
- Combines naturally with quantization (TinyBERT)

Summary (ii)

Chaining:

- Pruning + quantization + Huffman coding gives **multiplicative** compression gains
- Requires hardware support to realize wall-clock speedups