Lecture 12 - Contents

An overview of the main sections in this lecture.

This outline is for guidance. Navigate the slides with the left/right arrow keys.

Lecture 12:

Knowledge Distillation and Model Compression

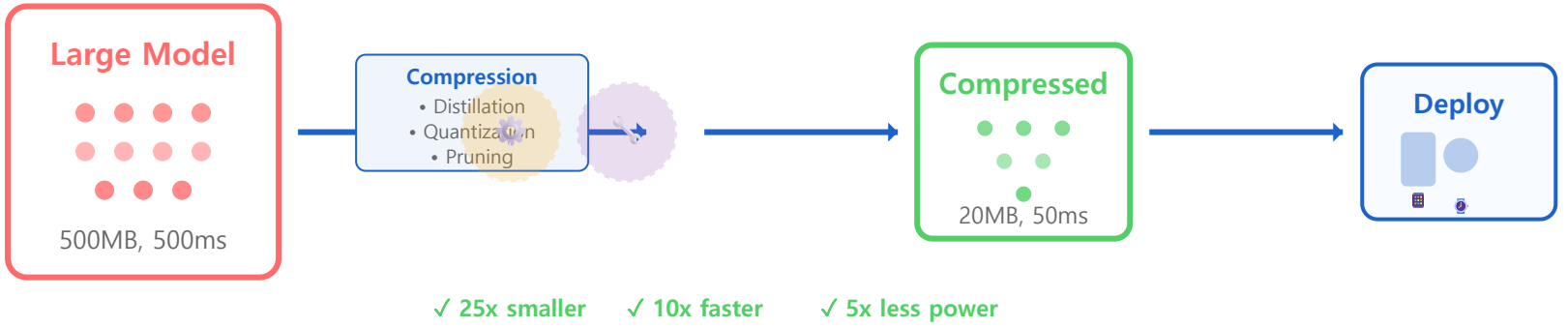
Efficient Medical AI: Compression for Clinical Deployment

Model Compression Overview

왜 압축이 필요한가?

- 의료 현장의 제한된 컴퓨팅 리소스 (모바일, 웨어러블, POC 장비)
- 실시간 진단 요구사항 (저지연 추론)
- 배터리 효율성 및 전력 소비 최소화
- 개인정보 보호를 위한 온-디바이스 처리

Compression Pipeline



주요 압축 방법론

Knowledge Distillation

Teacher 모델의 지식을 작은 Student 모델로 전달

Quantization

가중치 비트 수 감소 (FP32→INT8)

Pruning

불필요한 가중치 및 뉴런 제거

핵심 트레이드오프: 모델 크기/속도 향상 ↔ 정확도 유지

Part 1/3:

Knowledge Distillation for Medical Models

1. Teacher-Student Framework
2. Soft Target Training
3. Temperature Scaling
4. Feature Distillation
5. Attention Transfer
6. Progressive & Multi-Teacher Distillation

Teacher-Student Framework

🎓 Teacher Model

Size: Large (millions to billions of parameters)

Performance: Highest accuracy

Purpose: Knowledge provision

Examples: ResNet-152, BERT-Large

🎯 Student Model

Size: Small (10-100x smaller)

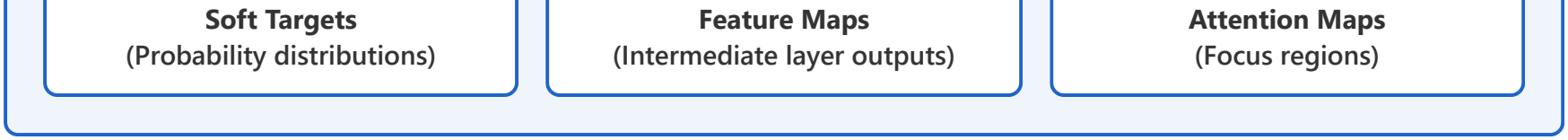
Performance: Approaches teacher

Purpose: Practical deployment

Examples: MobileNet, DistilBERT

Knowledge Transfer

Transferred Knowledge


Soft Targets
(Probability distributions)
Feature Maps
(Intermediate layer outputs)
Attention Maps
(Focus regions)

Soft Target Training

Hard Targets

[0, 0, 1, 0]

One-hot vector
Only correct class is 1

✗ No inter-class relationship information

Soft Targets

[0.05, 0.15, 0.70, 0.10]

Probability distribution
Probabilities for all classes

✓ Learns inter-class similarity

Benefits of Soft Targets

✓ Richer information: Transfers inter-class relationships learned by Teacher

✓ Improved generalization: Prevents overfitting

✓ Medical imaging: Utilizes disease similarity information (e.g., borderline benign/malignant cases)

Temperature Scaling

Softmax with Temperature

$$p_i = \exp(z_i/T) / \Sigma \exp(z_j/T)$$

T = Temperature Parameter

T = 1

Standard Softmax

[0.05, 0.10, 0.80, 0.05]

Sharp Distribution

T = 3~5

Optimal for Distillation

[0.15, 0.20, 0.45, 0.20]

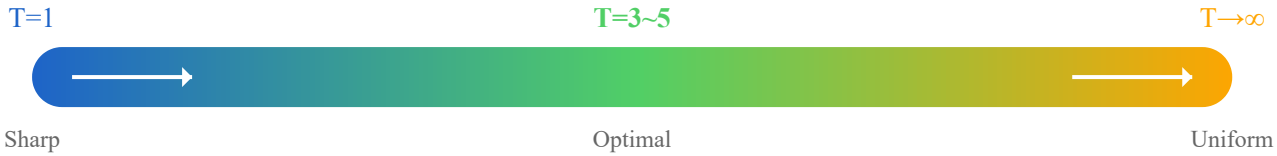
Soft Distribution

T → ∞

Extreme Case

[0.25, 0.25, 0.25, 0.25]

Uniform Distribution



Practical Tip: T=3~4 is effective for medical image classification
Higher T → Richer inter-class relationship information → Improved Student learning

Feature Distillation

Feature-Level Distillation

Matching intermediate layers outputs of Teacher and Student

Layer Matching Strategy

1:1 Matching

Teacher Layer 12 → Student Layer 6

Multiple Matching

Matching multiple layers simultaneously

Adaptive Matching

Using learnable transformations

Feature Distillation Loss

$$L_{feature} = ||f_T(x) - f_S(x)||^2$$

f_T: Teacher features, f_S: Student features

Attention Transfer

Attention Transfer

Transfer teacher model's attention maps to student to learn important regions

Application in Medical Imaging

Lesion Location Learning

Student learns lesion regions that teacher focuses on

X-ray/CT Analysis

Transfer attention to abnormal areas like pneumonia, tumors

Improved Interpretability

Provide clinically understandable diagnostic rationale

Attention Transfer Loss

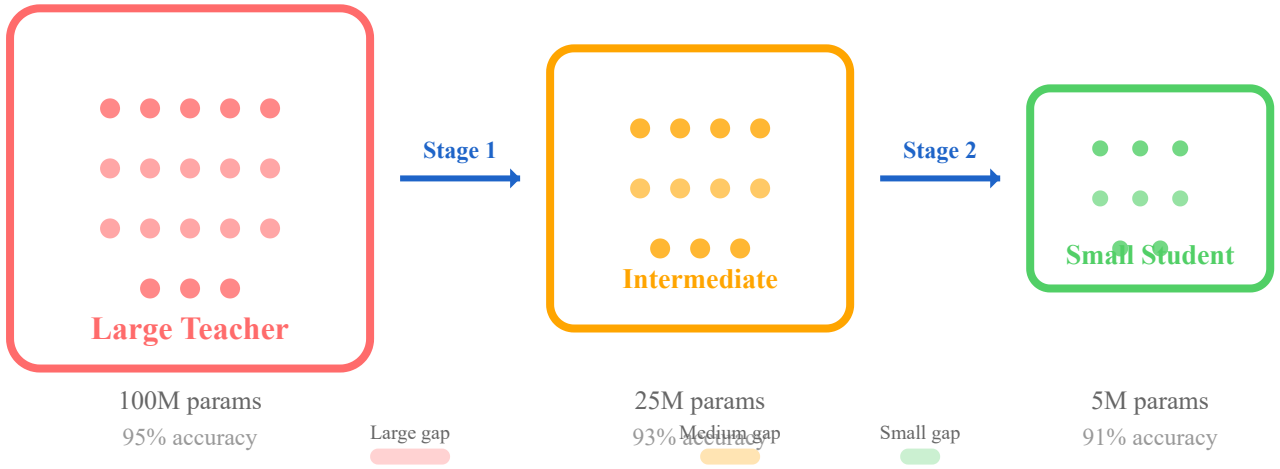
$$L_{AT} = ||A_T - A_S||_2$$

A: Attention maps (spatial activation)

Progressive Distillation

Progressive Distillation

Gradual knowledge transfer from large model to small model through stages



Stage 1

Teacher (Large)

↓

Intermediate Model

(Medium Size)

Stage 2

Intermediate Model

↓

Student (Small)

(Final Deployment)

Benefits

✓ Reduced knowledge gap: Gradually reducing large differences through stages

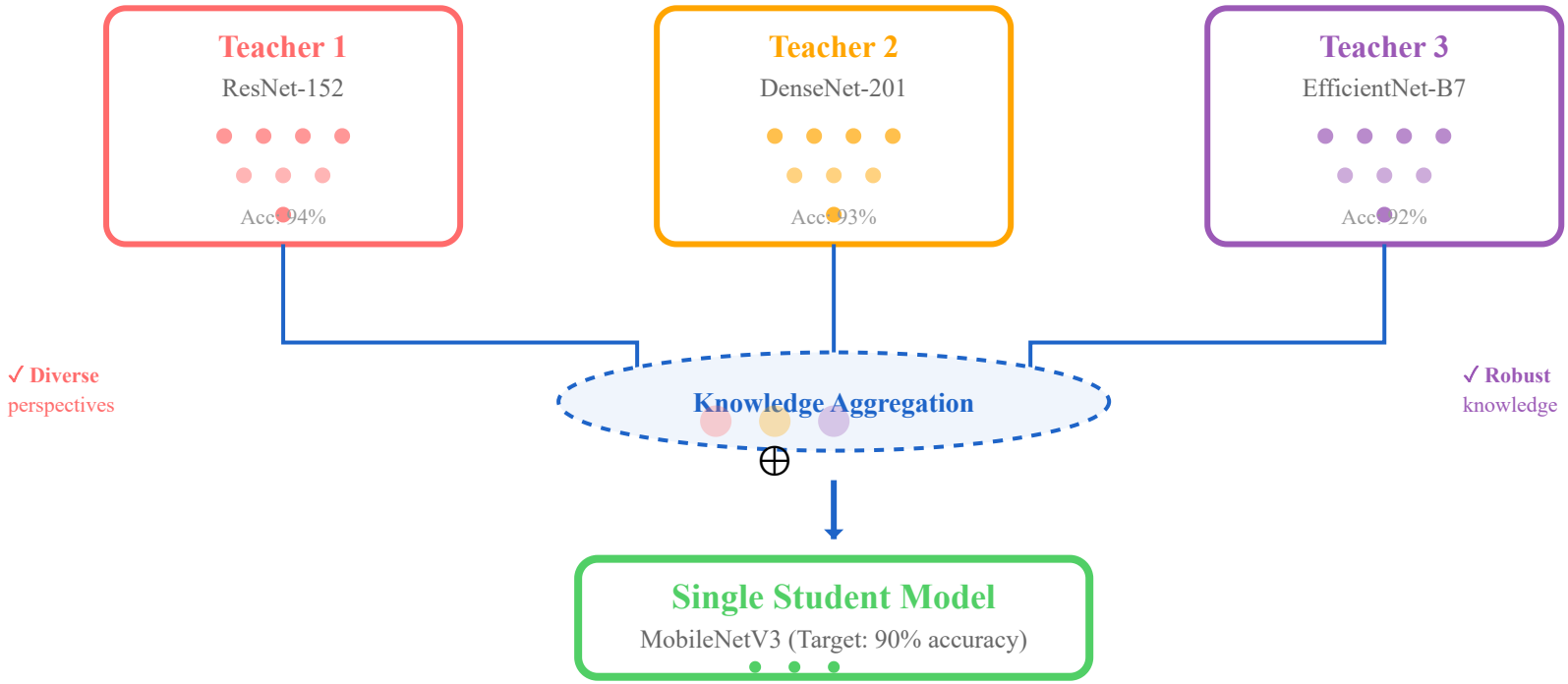
✓ Stable learning: Prevention of sharp performance degradation

✓ Multiple model sizes available: Selection based on deployment environment

Multi-Teacher Distillation

Multi-Teacher Distillation

Transferring ensemble knowledge from multiple Teacher models to Student



Multiple Teachers

Teacher 1

Teacher 2

Teacher 3

ResNet-152

DenseNet-201

EfficientNet-B7

⬇️ Knowledge Aggregation ⬇️

Single Student Model

MobileNetV3

Knowledge Integration Methods

Average

Weighted Average

Attention-based

Part 2/3:

Quantization and Pruning

1. INT8/INT4 Quantization
2. Mixed Precision Strategies
3. Structured Pruning
4. Magnitude Pruning
5. Lottery Ticket Hypothesis
6. Dynamic Sparsity

INT8/INT4 Quantization

Integer Quantization

Converting floating-point (FP32/FP16) to integers (INT8/INT4) to reduce model size

Bit Count Comparison

FP32

32 bits
4 bytes
Default training precision

INT8

8 bits
1 byte
75% memory reduction

INT4

4 bits
0.5 bytes
87.5% memory reduction

Benefits of Quantization

- Reduced memory usage
- Improved inference speed
- Lower power consumption

Accuracy Trade-off: INT8 typically shows less than 1% accuracy degradation
INT4 requires Quantization-Aware Training

Mixed Precision Strategies

Mixed Precision Strategy

Achieving a balance between performance and efficiency by using different precision levels for each layer

Strategy Examples

Input Layer

INT8/FP16
Fast processing enabled

Hidden Layers

INT8
Most computations

Output Layer

FP16/FP32
Accurate probability calculation

Layer Sensitivity Analysis

Sensitive Layers → **Maintain FP16/FP32**

Less Sensitive Layers → **Apply INT8/INT4**

Tools

PyTorch Quantization, TensorRT, ONNX Runtime

Structured Pruning

Structured Pruning

Structural compression by removing entire channels, filters, or layers

Types of Structured Pruning

Channel Pruning

[■][■][□][■]
Channel-level removal

Filter Pruning

Filter 1: ■■■■
Filter 2: XXXX
Filter 3: ■■■■

Layer Pruning

Layer 1 → ■
Layer 2 → X
Layer 3 → ■

Advantages of Structured Pruning

✓ Hardware-friendly: No special hardware required

✓ Real speed improvement: Maintains dense matrix operations

✓ Memory reduction: Actual reduction in parameter count

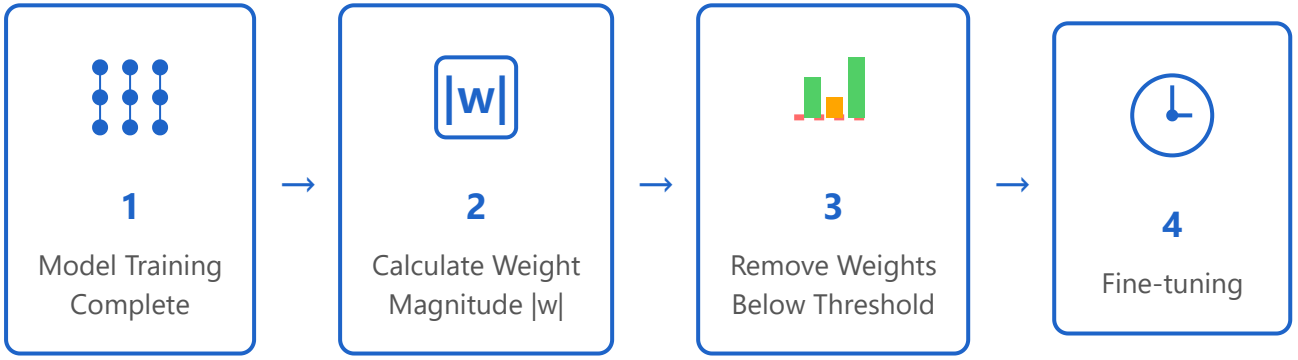
Medical Application: X-ray classification model with 40% channel removal → 2x faster inference, accuracy 98.5% → 97.8% (less than 1% decrease)

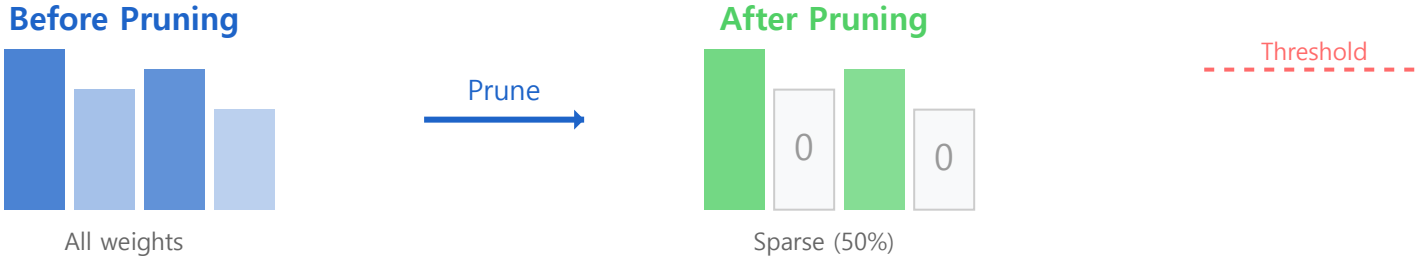
Magnitude Pruning

Magnitude-Based Pruning

Increase sparsity by setting small weight values to 0

Pruning Process





Threshold Setting Strategies

θ

Global Threshold

Single threshold for entire network

θ₁ θ₂ θ₃

Layer-wise Threshold

Different threshold per layer

Top-k

Top-k Pruning

Keep only top k% weights

Sparsity Example

Dense: [0.8, 0.3, -0.5, 0.1, -0.9]

↓ Threshold = 0.4

Sparse: [0.8, 0, -0.5, 0, -0.9]

40% Sparsity Achieved

Lottery Ticket Hypothesis

Lottery Ticket Hypothesis

A large network contains small subnetworks ("winning tickets")
that can achieve good performance when trained from scratch

Finding the Winning Ticket

① **Random Init** Initialize the network randomly

↓

② **Train** Train the entire network

↓

③ **Prune** Remove unimportant weights

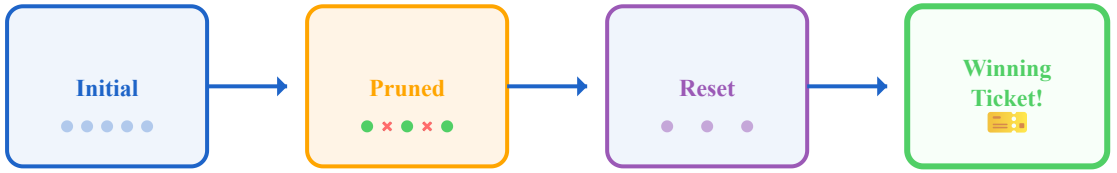
↓

④ **Reset** Rewind to original initialization

↓

⑤ **Retrain** Train only the subnetwork (Winning Ticket!)

Network Evolution



Key Findings

- Can achieve original performance with only 10-20% of weights
- Training speed is also faster
- Initialization method is crucial

Dynamic Sparsity

Dynamic Sparsity

Dynamically adjusting pruning patterns during training to explore optimal sparse structures

Static vs Dynamic Sparsity

Static Sparsity

- Prune once → Fixed
- Simple but suboptimal
- Performance limitations

Dynamic Sparsity

- Continuous reconstruction during training
- Complex but optimized
- Better performance

Dynamic Pruning Methods

RigL

Random Sparse Training
Periodic weight grow/prune

SET

Sparse Evolutionary Training
Evolutionary algorithm based

DST

Dynamic Sparse Training
Gradient-based reconstruction

Advantages: Maintain sparsity from early training → Reduced training cost
Final sparse structure is more efficient

Part 3/3:

Edge Deployment in Healthcare

1. Mobile Health Apps & Wearable Devices
2. Point-of-Care Systems
3. Latency & Battery Optimization
4. Model Serving on Edge
5. Performance-Size Tradeoffs
6. Case Studies & Hands-on

Mobile Health Apps

Mobile Health Applications

Optimization and deployment of medical AI models running on smartphones

Mobile Constraints

Memory

2-8 GB RAM
Model size limitation

Computation

Limited FLOPS
CPU/NPU utilization

Battery

Minimize power consumption
Long-term usage

Optimization Strategies

- ✓ Model size < 50MB (ideally < 20MB)
- ✓ INT8 quantization required
- ✓ Utilize MobileNet, EfficientNet architectures
- ✓ Convert to TensorFlow Lite, Core ML

Medical Use Cases

Skin Lesion Classification

Diabetic Retinopathy Screening

ECG Analysis

Wearable Device Models

Wearable Device AI

Real-time AI processing on ultra-compact devices such as smartwatches and fitness trackers

Wearable Device Specifications

Apple Watch

RAM: 1GB

CPU: Low Power

Battery: 18 hours

Fitbit

RAM: < 512MB

Very Limited

Battery: 5-7 days

Extreme Model Requirements

Model Size:	< 5MB (Ideal: < 1MB)
Inference Time:	< 100ms (Real-time processing)
Precision:	INT8 or INT4
Batch Size:	1 (Single sample)

Wearable Medical Applications

❤️ **Heart Rate Anomaly Detection**

😴 **Sleep Pattern Analysis**

🚶 **Fall Detection**

🫀 **Arrhythmia Monitoring**

Point-of-Care Systems

POC (Point-of-Care) Diagnostic Systems

Portable AI equipment for diagnostics performed immediately at the patient's bedside

POC System Examples

Portable Ultrasound

- Butterfly iQ, Lumify
- Real-time image analysis
- Automatic lesion detection

Blood Analyzer

- Immediate blood testing
- AI-based result interpretation
- Emergency room & outpatient use

Retinal Camera

- Diabetic retinopathy screening
- Offline operation
- Use in developing countries

POC System Requirements

🎯 **Accuracy First:** Medical device level (95%+)

⚡ **Fast Response:** Results within 5-30 seconds

🔌 **Offline Operation:** No network required

🛡️ **Privacy Protection:** No external data transmission

Deployment Considerations: FDA/CE certification required, clinical validation essential, Medical staff training and user interface are critical

Latency Optimization

Latency Optimization

Inference speed improvement techniques for real-time medical applications

Inference Latency Breakdown

Latency Components

Data Preprocessing
Image resizing, normalization
5-20ms

Model Inference
Largest proportion
50-500ms

Postprocessing
Result interpretation, visualization
5-10ms

Optimization Techniques

Model Compression

- Quantization
- Pruning
- Distillation

2-10x speedup

Inference Engine

- TensorRT
- ONNX Runtime
- TFLite

1.5-3x speedup

Hardware Acceleration

- GPU
- NPU/TPU
- FPGA

5-100x speedup

Real-time Processing Standards

Soft Real-time

< 100ms

General diagnostic assistance

Hard Real-time

< 10ms

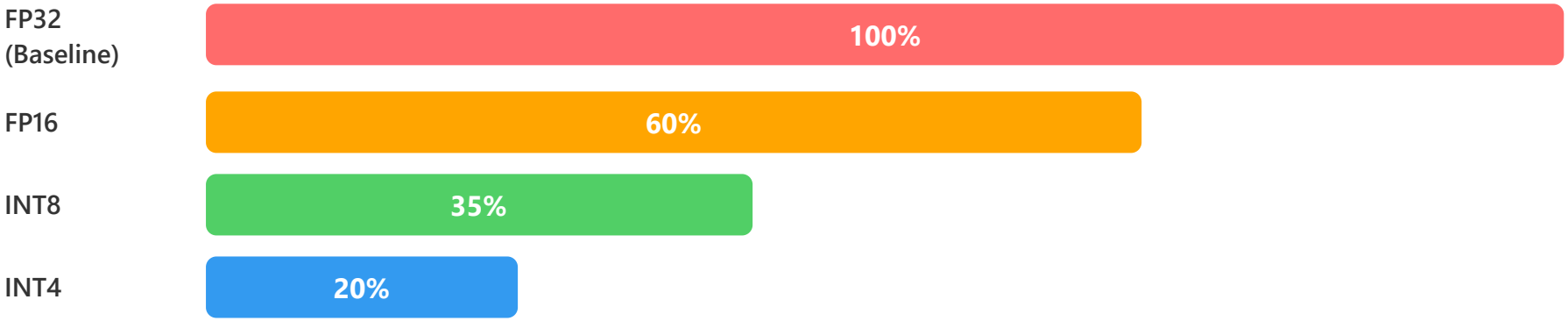
Surgical robots, emergency systems

Battery Efficiency

Battery Efficiency Optimization

Minimizing power consumption for long-term use of mobile/wearable devices

Power Consumption During Model Execution



Power Saving Strategies

1. **Aggressive Quantization**
   Reduce computational load using INT8/INT4

2. **On-Demand Inference**
   Run model only when needed (event-based)

3. **Batch Processing**
   Collect multiple samples and process at once

4

Hardware Accelerator Utilization

Use NPU/DSP (more efficient than CPU)

Trade-off: Battery life vs Model performance/accuracy
In medical applications, optimization must be done without compromising safety

Model Serving on Edge

Model Serving on Edge Devices

Infrastructure for efficiently deploying and running compressed models

Edge Inference Frameworks

TensorFlow Lite

- Provided by Google
- Android/iOS support
- Extensive hardware
- .tflite format

Core ML

- Apple exclusive
- iOS/macOS optimized
- Neural Engine utilization
- .mlmodel format

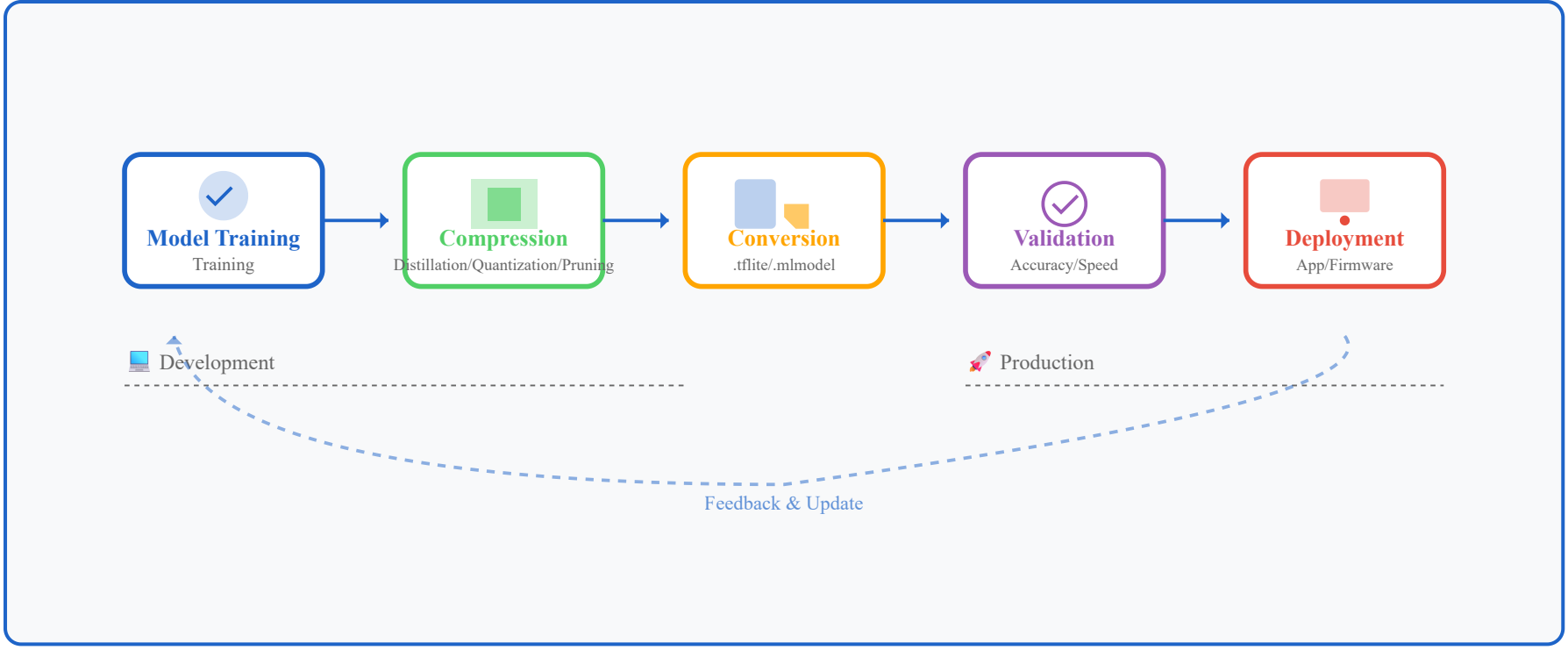
ONNX Runtime

- Cross-platform
- Various backends
- Model optimization tools
- .onnx format

PyTorch Mobile

- PyTorch based
- Android/iOS
- Easy conversion
- .ptl format

Deployment Pipeline



Best Practices

✓ Device-specific benchmarking required (test on actual hardware)

✓ OTA (Over-The-Air) update support

✓ Fallback mechanism (cloud inference)

✓ Monitoring and logging

Performance vs Size Tradeoffs

Performance-Size Tradeoffs

Finding the balance between model size, accuracy, and speed according to compression levels

Pareto Frontier

Large Model

Size: 500MB
Accuracy: 95%
Speed: 500ms

Medium Model

Size: 100MB
Accuracy: 93%
Speed: 100ms

Small Model

Size: 20MB
Accuracy: 90%
Speed: 20ms

Tiny Model

Size: 5MB
Accuracy: 85%
Speed: 5ms

Model Selection Criteria

Cloud Deployment

Large/Medium Model

Accuracy prioritized

Mobile App

Medium/Small Model

Balanced performance

Wearable

Small/Tiny Model

Size minimization

Decision-Making Guide:

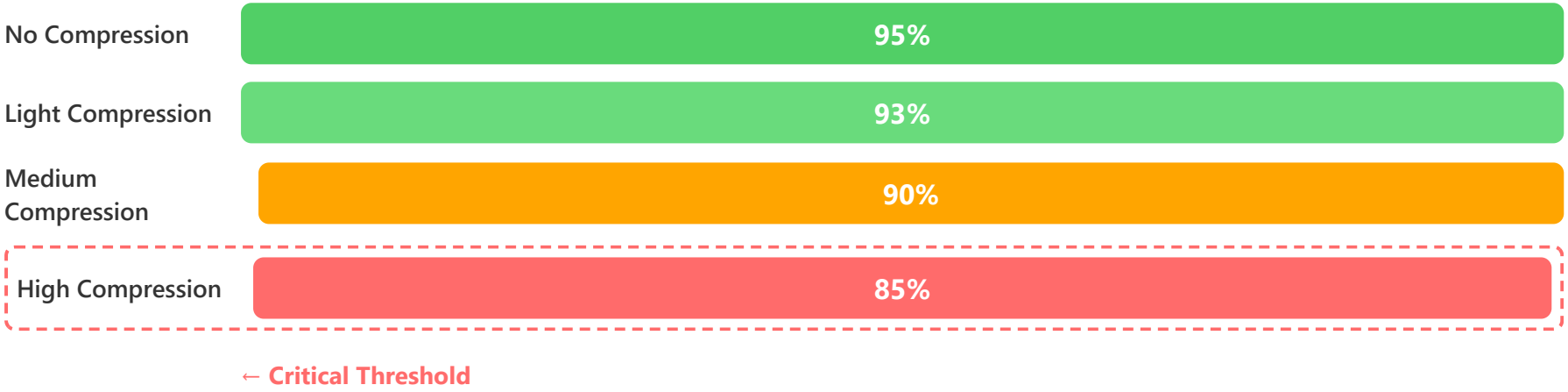
1. Determine minimum acceptable accuracy
2. Identify target device constraints
3. Select the optimal point on the Pareto curve

Accuracy Preservation

Accuracy Preservation

Methods to maximally preserve model performance during compression

Accuracy Changes by Compression Rate



Accuracy Preservation Strategies

1 **Quantization-Aware Training (QAT)**

Simulating quantization during training → Minimizing accuracy loss

2 **Fine-tuning after Compression**

Performance recovery through retraining after compression

3 **Sensitivity Analysis**

Selective compression after layer-wise sensitivity analysis

4 **Knowledge Distillation**

Accuracy preservation by leveraging teacher knowledge

⚠️ **Special Considerations for Medical AI:**

Accuracy degradation directly affects patient safety → Strict threshold setting required

Verify compliance with FDA/CE approval standards

Case Study: Mobile Diagnostics

Case Study: Mobile Skin Cancer Diagnosis App

Implementing a real-time mobile diagnostic system through deep learning model compression

Problem Definition

🎯 **Goal:** Classify benign/malignant from skin lesion images

📱 **Platform:** iOS/Android mobile app

⚡ **Requirements:** Results within 3 seconds, offline operation

Compression Approach

Step 1: Base Model

ResNet-50 (98MB, FP32)

Accuracy: 94.5%

Inference: 2.5s

↓

Step 2: Knowledge Distillation

MobileNetV3 Student

Size: 24MB

Accuracy: 93.2%

↓

Step 3: INT8 Quantization

Size: 6MB (75% reduction)

Accuracy: 92.8%

Inference: 0.8s

↓

Final: TFLite Optimization

Final Size: 5.2MB

Accuracy: 92.5%

Inference: 0.6s ✅

Final Results

94.7%	76%	-2.0%
Size Reduction	Speed Improvement	Accuracy Change

Hands-on: Model Compression

Practice: Model Compression Pipeline

Knowledge Distillation and Quantization Practice using PyTorch

1. Knowledge Distillation Code

```
# Teacher-Student distillation loss
def distillation_loss(student_logits, teacher_logits, labels, T=3):
    # Soft targets
    soft_loss = nn.KLDivLoss()(F.log_softmax(student_logits/T, dim=1),
                               F.softmax(teacher_logits/T, dim=1)) * T*T
    # Hard targets
    hard_loss = nn.CrossEntropyLoss()(student_logits, labels)
    return 0.7 * soft_loss + 0.3 * hard_loss
```

2. INT8 Quantization Code

```
import torch.quantization

# Static quantization
model.qconfig = torch.quantization.get_default_qconfig('qnnpack')
model_prepared = torch.quantization.prepare(model)
# Calibration
model_prepared(calibration_data)
# Apply quantization
model_quantized = torch.quantization.convert(model_prepared)
```

Useful Tools and Libraries

PyTorch

torch.quantization
torch.nn.utils.prune

TensorFlow

TF-MOT (Model Optimization)
Quantization-Aware Training

ONNX

onnxruntime
Cross-framework

Hugging Face

Optimum
Transformer Optimization

Practice Assignment:

Compress ResNet model with CIFAR-10 dataset
(Implement distillation + quantization pipeline)

Deployment Strategies

Deployment Strategies and Checklist

Strategies for successful production deployment of compressed models

Deployment Workflow

Development
- Model compression
- Local testing
- Benchmarking

→

Validation
- A/B testing
- Clinical validation
- Regulatory approval

→

Deployment
- Gradual rollout
- Monitoring
- Feedback collection

→

Maintenance
- Performance tracking
- Model updates
- Version control

Pre-Deployment Checklist ✅

Performance

□ Accuracy target achieved

□ Latency requirements met

□ Within memory constraints

Reliability

□ Edge case testing

□ Error handling

□ Fallback mechanism

Regulatory

□ FDA/CE certification (if required)

□ Privacy protection

□ Explainability

Operations

□ Logging/monitoring

□ OTA updates

□ User training

Recommended Strategies by Platform

Platform	Compression Method	Framework
iOS	Distillation + INT8	Core ML
Android	Distillation + INT8	TF Lite
Web	Lightweight compression	TensorFlow.js
Embedded	INT4 + Pruning	TF Lite Micro

Thank You

강의 요약

Part 1

Knowledge Distillation

- Teacher-Student 프레임워크
- Soft targets & Temperature
- Feature/Attention 전달

Part 2

Quantization & Pruning

- INT8/INT4 양자화
- 구조적/비구조적 가지치기
- 동적 희소성

Part 3

Edge Deployment

- 모바일/웨어러블 최적화
- POC 시스템
- 성능-크기 트레이드오프

핵심 베스트 프랙티스

✓ 압축 전 정확도 목표 및 제약사항 명확히 정의

✓ 여러 압축 기법 조합으로 시너지 효과

✓ 실제 디바이스에서 벤치마크 필수

✓ 의료 AI는 정확도 하락에 특히 민감 - 임상 검증 필수

✓ 배포 후 지속적 모니터링 및 업데이트

추천 도구 모음

압축: PyTorch Quantization, TensorFlow Model Optimization

변환: ONNX, TensorFlow Lite, Core ML Tools

추론: TensorRT, ONNX Runtime, TFLite

벤치마킹: MLPerf Mobile, AI Benchmark

Thank You!

Lecture 12: Knowledge Distillation and Model Compression

Questions? Contact via course forum or office hours