SCALER School of Technology × Priyansh Saxena

Transfer Learning and CNN Applications

Feature Extraction • Fine-Tuning • Detection • Segmentation

Priyansh Saxena

Scaler School of Technology | Deep Learning I: Neural Networks | Week 8 — Session 1/2

May 2026

The Problem

Training from scratch:

- ResNet-152 on ImageNet
- 2 weeks of training
- 8 GPUs
- 1.2 million labeled images

Your reality:

- 500 labeled images
- 1 laptop (maybe a Colab GPU)
- Due Friday
- No budget for labeling

How do we use deep learning with limited data and compute?

The Solution: Transfer Learning

Key Idea

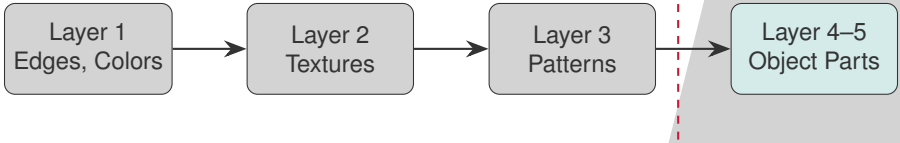
Use knowledge learned on a **large dataset** (ImageNet) to solve a **different task** with limited data.

Two approaches:

1. **Feature Extraction** — freeze the pre-trained network, train a new head
2. **Fine-Tuning** — unfreeze some layers, adapt with a small learning rate

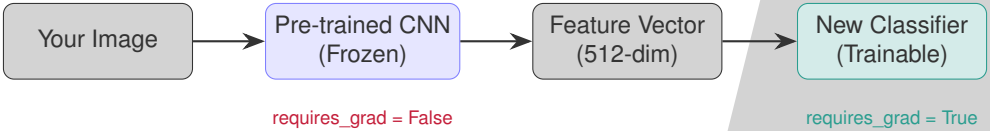
Like a chef trained in French cuisine adapting to Italian — knife skills transfer, only the flavors change.

What CNN Layers Learn



Early layers detect universal features.
Later layers specialize for the training task.

Feature Extraction Strategy



- Load pre-trained model (e.g., ResNet-50)
- Freeze all layers — no gradient updates
- Replace final FC layer with your classifier
- Train only the new head (fast, needs little data)

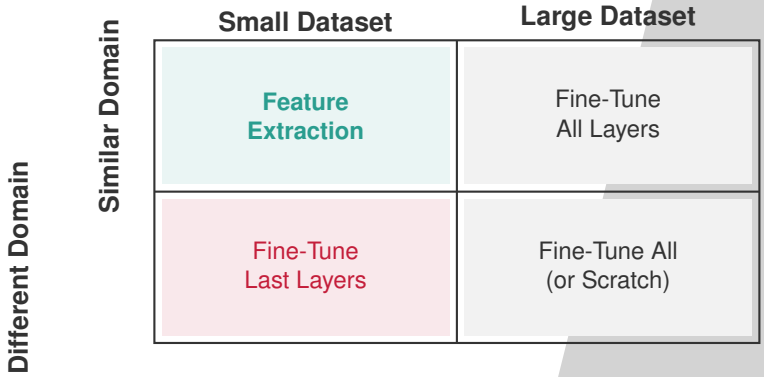
Feature Extraction in PyTorch

Code Pattern

```
model = resnet50(weights='IMAGENET1K_V1')

# Freeze all parameters
for param in model.parameters():
   param.requires_grad = False

# Replace classifier head
model.fc = nn.Linear(2048, num_classes)

# Only optimize the new head
optimizer = Adam(model.fc.parameters(), lr=0.001)
```

Result: Training only 0.1% of parameters, yet high accuracy.

When Does Feature Extraction Work?



Rule of thumb: Start with feature extraction. Unfreeze more if needed.

Quick Check

Question

Q1: You have 10,000 X-ray images and want to detect pneumonia.
Starting with an ImageNet pre-trained model — feature extraction or fine-tuning?

Q2: Which layers of a pre-trained CNN are *most likely* to transfer across different image domains?

Quick Check

Question

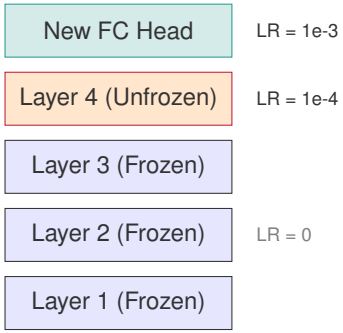
Q1: You have 10,000 X-ray images and want to detect pneumonia.
Starting with an ImageNet pre-trained model — feature extraction or fine-tuning?

Q2: Which layers of a pre-trained CNN are *most likely* to transfer across different image domains?

A1: Fine-tune — different domain (medical), decent data size.
A2: Early layers (edges, textures) — they're universal.

Fine-Tuning: The Idea

Unfreeze some pre-trained layers and train end-to-end with a **small learning rate**.



Critical: Use 10–100× smaller LR for pre-trained layers.
Otherwise you destroy useful features in the first epoch.

Fine-Tuning Strategies

1. **Conservative:** Unfreeze only the last 1–2 blocks
   - Safe for small datasets
   - Minimal risk of catastrophic forgetting
2. **Aggressive:** Unfreeze all layers with very small LR
   - Best for large datasets or very different domains
3. **Gradual Unfreezing:** Start head-only, unfreeze more each epoch
   - Safest for beginners
   - Less chance of catastrophic forgetting

Always: Train the new head first for 1–2 epochs before unfreezing.

Practical Fine-Tuning Tips

Key Practices

- **Differential learning rates:** pre-trained layers get 10–100$\times$ smaller LR
- **Freeze batch norm:** keep BN in eval mode even while training
- **Train head first:** 1–2 epochs with backbone frozen, then unfreeze
- **Monitor val loss:** overfitting happens fast with small datasets
- **Use early stopping:** stop when val loss plateaus or increases

“When in doubt, start frozen. Unfreeze only if accuracy plateaus.”

Coming Up Next

Part 2: Domain Adaptation, Object Detection, and Image Segmentation

The Domain Shift Problem

The Problem

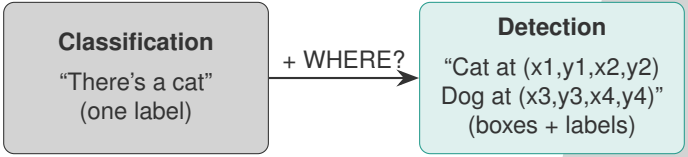
Model trained on domain A, deployed on domain B $\rightarrow$ **performance drops.**

Examples:

- Studio photos $\rightarrow$ phone camera photos
- US hospital X-rays $\rightarrow$ rural clinic X-rays
- Synthetic training data $\rightarrow$ real-world deployment

Simplest fix: Fine-tune on a small labeled set from the target domain.
Even 100–500 target samples can significantly close the gap.

From Classification to Detection



Detection answers two questions: WHAT is in the image and WHERE is it?

Use cases: self-driving cars, security cameras, medical imaging, retail analytics

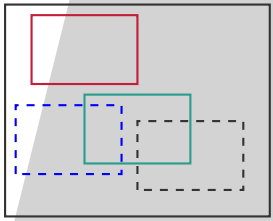
The Naive Approach: Sliding Window

Idea: Slide a window across the image at every position and scale. Classify each crop.

Problem:

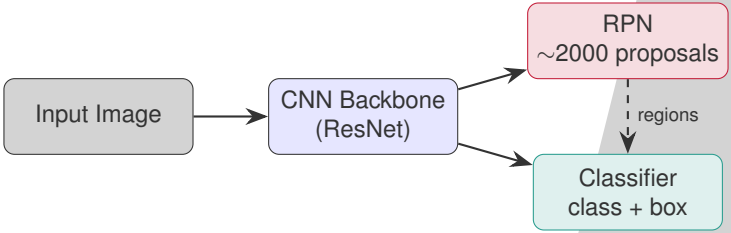
- 1000×1000 image
- 100 different scales
- Stride of 10 pixels
- $=$ Millions of forward passes

Way too slow for real-world use.



Many overlapping windows

Faster R-CNN: Two-Stage Detection



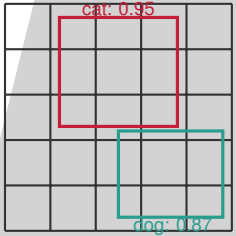
- **Stage 1:** Region Proposal Network (RPN) proposes “interesting” regions
- **Stage 2:** Classify each region + refine the bounding box
- Backbone features are **shared** — compute once, reuse many times

YOLO: You Only Look Once

Single-Shot Detection

- Divide image into $S \times S$ grid
- Each cell predicts:
  - B bounding boxes + confidence
  - C class probabilities
- **One forward pass** for everything
- Real-time capable (30+ FPS)

Trade-off: Faster but slightly less accurate on small objects.



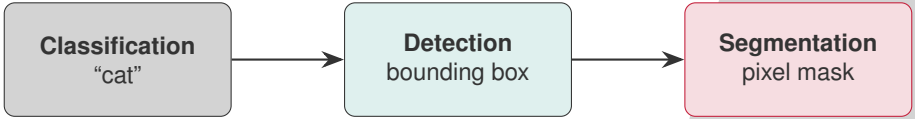
Detection: Two Families

	Two-Stage (R-CNN)	Single-Shot (YOLO)
Speed	Slower	**Real-time**
Accuracy	**Higher**	Slightly lower
Small objects	**Better**	Weaker
Use case	Medical, high-stakes	Video, robotics

Both use **pre-trained CNN backbones** — transfer learning is the foundation!

In practice, YOLO dominates industry because speed usually matters more than the last 1% accuracy.

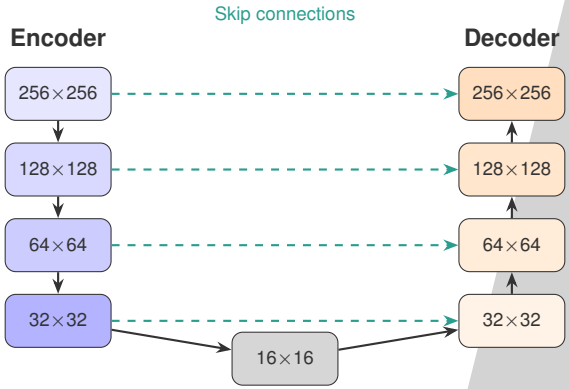
Beyond Boxes: Pixel-Level Predictions



- **Semantic segmentation:** classify every pixel (all cats = one color)
- **Instance segmentation:** distinguish individuals (cat 1 vs. cat 2)

Why? Medical imaging needs **exact boundaries**, not rectangles.

U-Net: Encoder-Decoder with Skip Connections



- **Encoder:** downsamples, builds semantic features
- **Decoder:** upsamples back to full resolution
- **Skip connections:** preserve fine spatial details lost during downsampling

Segmentation Applications

Medical	Autonomous Driving	Satellite
Tumor boundaries	Road vs. sidewalk	Land use mapping
Cell segmentation	Cars vs. pedestrians	Crop classification
Organ delineation	Lane detection	Deforestation tracking

All use the same principle: **encoder-decoder + skip connections + pre-trained backbone.**

Quick Check

Question

Q1: YOLO vs. Faster R-CNN — which would you pick for processing 30fps dashcam video in a self-driving car?

Q2: What do skip connections in U-Net help preserve that downsampling destroys?

Quick Check

Question

Q1: YOLO vs. Faster R-CNN — which would you pick for processing 30fps dashcam video in a self-driving car?

Q2: What do skip connections in U-Net help preserve that downsampling destroys?

A1: YOLO — real-time speed is critical at 30fps.
A2: Fine spatial details — edges, boundaries, exact object outlines.

Practical Tips for Training CNNs

Rules of Thumb

1. **Always start pre-trained** — training from scratch is a last resort
2. **Data augmentation** — flips, crops, color jitter, rotation (free extra data)
3. **LR schedule** — warmup + cosine decay or step decay
4. **Progressive resizing** — train 128×128 first, then 224×224
5. **Monitor both losses** — if train $\downarrow$ but val $\uparrow$, stop immediately

Architecture Cheat Sheet

Task	Default Choice	Alternative
Classification	ResNet-50	EfficientNet
Detection (speed)	YOLOv8	SSD
Detection (accuracy)	Faster R-CNN	DETR
Segmentation (medical)	U-Net	—
Segmentation (general)	DeepLabV3	Segment Anything

When in doubt: ResNet-50 backbone + transfer learning.
It's "boring" but it works.

Recap: Three Big Ideas

1. **Transfer Learning** — don't train from scratch
   - Feature extraction (freeze) vs. fine-tuning (unfreeze)
   - Decision depends on data size $\times$ domain similarity
2. **Object Detection** — WHAT + WHERE
   - Faster R-CNN (accurate, two-stage) vs. YOLO (fast, single-shot)
3. **Image Segmentation** — pixel-level understanding
   - Encoder-decoder + skip connections (U-Net)

Next session: What happens when your data has a *time* dimension? $\rightarrow$ **RNNs**

Thank You!

Questions?

Priyansh Saxena

All lecture materials available at:

ww.priyanshsaxena.com/sst-deep-learning-nn