CSC 4315: INTRODUCTION TO ARTIFICIAL INTELLIGENCE

Week 11, Lecture 22

Transfer Learning and Fine-tuning Pre-trained Models

Lecture Overview

This lecture covers transfer learning techniques for leveraging pre-trained models on new tasks with limited data.

Topics Covered:

- Transfer learning fundamentals
- Feature extraction approach
- Fine-tuning strategies
- Data preprocessing requirements
- Complete transfer learning pipeline

1. Transfer Learning Fundamentals

1.1 What is Transfer Learning?

```
import torch
import torch.nn as nn

print("Transfer Learning")
print("="*70)

print("Use knowledge from one task for another")

print("\nWhy it works:")
print("  - Early layers learn generic features (edges, textures)")
print("  - These transfer across different tasks")
print("  - Only adapt final layers for new task")

print("\nBenefits:")
print("  1. Works with small datasets (100s-1000s images)")
print("  2. Faster training")
print("  3. Better performance than training from scratch")

print("\nTwo approaches:")
print("  1. Feature Extraction: Freeze all except classifier")
print("  2. Fine-tuning: Unfreeze some/all, train with low LR")

print("\nWhen to use which:")
print("  Similar domain + Small data -&gt; Feature extraction")
print("  Similar domain + Large data -&gt; Fine-tune all")
print("  Different domain + Large data -&gt; Fine-tune carefully")
print("  Different domain + Small data -&gt; Feature extraction")
```

```
Transfer Learning
======================================================================
Use knowledge from one task for another

Why it works:
  - Early layers learn generic features (edges, textures)
  - These transfer across different tasks
  - Only adapt final layers for new task

Benefits:
  1. Works with small datasets (100s-1000s images)
  2. Faster training
  3. Better performance than training from scratch

Two approaches:
  1. Feature Extraction: Freeze all except classifier
  2. Fine-tuning: Unfreeze some/all, train with low LR

When to use which:
  Similar domain + Small data -&gt; Feature extraction
  Similar domain + Large data -&gt; Fine-tune all
  Different domain + Large data -&gt; Fine-tune carefully
  Different domain + Small data -&gt; Feature extraction
```

2. Feature Extraction

2.1 Freezing Pre-trained Layers

```
print("Feature Extraction: Freeze Pre-trained Layers")
print("="*70)

print("Loading and modifying pre-trained model:")
print("""
from torchvision import models

# Load pre-trained ResNet
model = models.resnet50(pretrained=True)

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

# Replace classifier
num_classes = 10
model.fc = nn.Linear(model.fc.in_features, num_classes)

# Only classifier trains
trainable = sum(p.numel() for p in model.parameters() if p.requires_grad)
total = sum(p.numel() for p in model.parameters())
print(f"Trainable: {trainable:,} / {total:,}")
""")

print("\nAdvantages:")
print("  - Very fast training")
print("  - Works with small datasets")
print("  - Less risk of overfitting")

print("\nDisadvantages:")
print("  - Limited adaptation to new domain")
print("  - May not achieve best performance")
```

```
Feature Extraction: Freeze Pre-trained Layers
======================================================================
Loading and modifying pre-trained model:

from torchvision import models

# Load pre-trained ResNet
model = models.resnet50(pretrained=True)

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

# Replace classifier
num_classes = 10
model.fc = nn.Linear(model.fc.in_features, num_classes)

# Only classifier trains
trainable = sum(p.numel() for p in model.parameters() if p.requires_grad)
total = sum(p.numel() for p in model.parameters())
print(f"Trainable: {trainable:,} / {total:,}")

Advantages:
  - Very fast training
  - Works with small datasets
  - Less risk of overfitting

Disadvantages:
  - Limited adaptation to new domain
  - May not achieve best performance
```

3. Fine-tuning

3.1 Gradual Unfreezing

```
print("Fine-tuning Strategies")
print("="*70)

print("Strategy 1: Train classifier, then unfreeze all")
print("""
# Phase 1: Train classifier only
for param in model.parameters():
    param.requires_grad = False
model.fc.requires_grad_(True)
train(epochs=5, lr=0.01)

# Phase 2: Unfreeze all, low LR
for param in model.parameters():
    param.requires_grad = True
train(epochs=10, lr=0.0001)  # Much lower LR!
""")

print("\nStrategy 2: Gradual unfreezing (discriminative LR)")
print("""
# Different LR for different layers
optimizer = optim.Adam([
    {'params': model.layer4.parameters(), 'lr': 1e-4},
    {'params': model.layer3.parameters(), 'lr': 1e-5},
    {'params': model.layer2.parameters(), 'lr': 1e-6},
    {'params': model.fc.parameters(), 'lr': 1e-3},
])
""")

print("\nKey Tips:")
print("  - Use lower LR than training from scratch")
print("  - Typically 10-100x lower")
print("  - Train classifier first, then unfreeze")
print("  - Monitor validation loss carefully")
```

```
Fine-tuning Strategies
======================================================================
Strategy 1: Train classifier, then unfreeze all

# Phase 1: Train classifier only
for param in model.parameters():
    param.requires_grad = False
model.fc.requires_grad_(True)
train(epochs=5, lr=0.01)

# Phase 2: Unfreeze all, low LR
for param in model.parameters():
    param.requires_grad = True
train(epochs=10, lr=0.0001)  # Much lower LR!

Strategy 2: Gradual unfreezing (discriminative LR)

# Different LR for different layers
optimizer = optim.Adam([
    {'params': model.layer4.parameters(), 'lr': 1e-4},
    {'params': model.layer3.parameters(), 'lr': 1e-5},
    {'params': model.layer2.parameters(), 'lr': 1e-6},
    {'params': model.fc.parameters(), 'lr': 1e-3},
])

Key Tips:
  - Use lower LR than training from scratch
  - Typically 10-100x lower
  - Train classifier first, then unfreeze
  - Monitor validation loss carefully
```

3.2 Data Preprocessing

```
print("Data Preprocessing for Transfer Learning")
print("="*70)
```

```
print("IMPORTANT: Use same preprocessing as pre-training!")
print("""
# ImageNet normalization (most pre-trained models)
normalize = transforms.Normalize(
    mean=[0.485, 0.456, 0.406],
    std=[0.229, 0.224, 0.225]
)

train_transform = transforms.Compose([
    transforms.RandomResizedCrop(224),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    normalize,
])

test_transform = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    normalize,
])
""")

print("\nInput size requirements:")
print("  ResNet: 224x224 (flexible with GAP)")
print("  EfficientNet-B0: 224x224")
print("  EfficientNet-B7: 600x600")
print("  ViT-Base: 224x224 or 384x384")
```

```
Data Preprocessing for Transfer Learning
======================================================================
IMPORTANT: Use same preprocessing as pre-training!

# ImageNet normalization (most pre-trained models)
normalize = transforms.Normalize(
    mean=[0.485, 0.456, 0.406],
    std=[0.229, 0.224, 0.225]
)

train_transform = transforms.Compose([
    transforms.RandomResizedCrop(224),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    normalize,
])

test_transform = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    normalize,
])

Input size requirements:
  ResNet: 224x224 (flexible with GAP)
  EfficientNet-B0: 224x224
  EfficientNet-B7: 600x600
  ViT-Base: 224x224 or 384x384
```

4. Complete Example

4.1 Full Transfer Learning Pipeline

```
print("Complete Transfer Learning Pipeline")
print("="*70)

print("""
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import models, transforms
from torch.utils.data import DataLoader

# 1. Load pre-trained model
model = models.resnet50(pretrained=True)

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

# 3. Replace classifier
model.fc = nn.Sequential(
    nn.Linear(2048, 512),
    nn.ReLU(),
    nn.Dropout(0.3),
    nn.Linear(512, num_classes)
)

# 4. Setup training
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.fc.parameters(), lr=0.001)

# 5. Train classifier (Phase 1)
for epoch in range(5):
    model.train()
    for images, labels in train_loader:
        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

# 6. Fine-tune (Phase 2)
for param in model.parameters():
    param.requires_grad = True

optimizer = optim.Adam(model.parameters(), lr=0.0001)
for epoch in range(10):
    # Training loop...
    pass
""")
```

```
Complete Transfer Learning Pipeline
======================================================================

import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import models, transforms
from torch.utils.data import DataLoader

# 1. Load pre-trained model
model = models.resnet50(pretrained=True)

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

# 3. Replace classifier
model.fc = nn.Sequential(
    nn.Linear(2048, 512),
    nn.ReLU(),
    nn.Dropout(0.3),
```

```
    nn.Linear(512, num_classes)
)

# 4. Setup training
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.fc.parameters(), lr=0.001)

# 5. Train classifier (Phase 1)
for epoch in range(5):
    model.train()
    for images, labels in train_loader:
        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

# 6. Fine-tune (Phase 2)
for param in model.parameters():
    param.requires_grad = True

optimizer = optim.Adam(model.parameters(), lr=0.0001)
for epoch in range(10):
    # Training loop...
    pass
```

Summary

Key Takeaways:

- Transfer learning leverages pre-trained features
- Feature extraction: freeze all, train classifier only
- Fine-tuning: unfreeze and train with low LR
- Use same preprocessing as pre-training
- Train classifier first, then fine-tune
- Use lower LR for fine-tuning (10-100x lower)

Practice Exercises:

1. Load pre-trained ResNet and modify for 5 classes
2. Compare feature extraction vs fine-tuning
3. Implement discriminative learning rates
4. Transfer learning on custom dataset
5. Visualize which layers change during fine-tuning