Module 3: Evaluating and Optimizing Transfer Learning

Module 3 Course Objectives:

By the end of this module, you will be able to:

1. Critique a pre-trained model's suitability for use in transfer learning to a new task and select effective candidate models.
2. Assess the trade-offs of transfer learning versus full training and make justifiable choices for their approach.
3. Develop intuition on which, if any, transfer learning techniques to use in model development.
4. Evaluate the outcome of transfer learning techniques (feature extraction, fine-tuning and LoRA) on model performance.
5. Evaluate the outcome of transfer learning techniques compared to a similar model trained without transfer.
6. Troubleshoot and remediate common transfer learning model issues like catastrophic forgetting and negative transfer.

Choosing the Right Pre-Trained Model

Choosing the right pre-trained model is one of the most critical steps in transfer learning. The effectiveness of transfer learning depends on the compatibility between the pre-trained model (source model) and the new task (target task). It's important to note here that we mean compatibility both in the task the architecture was meant to support and in the data the source model was trained on versus the target data. A well-chosen source model can significantly reduce training time, lower computational costs, and improve model accuracy—but a poorly chosen one can lead to negative transfer, where the transferred knowledge actually hurts performance.

Why Pre-Trained Model Selection Matters

Deep learning models learn hierarchical feature representations:

- Early layers detect low-level features (e.g., edges, textures, basic shapes)
- Middle layers capture more abstract patterns (e.g., object parts, textures, language embeddings)
- Later layers learn high-level, task-specific representations (e.g., entire objects, contextual meaning in text)

Note that these are not explicitly programmed in and are, to some degree, generalizations. But when people explore the feature representations in trained models, these patterns are

surprisingly consistent and surprisingly similar to how our own brains process data. This structure is what makes feature extraction and fine-tuning work:

- Feature extraction relies on keeping the early layers frozen and repurposing them for a new task.
- Fine-tuning selectively retrains deeper layers to adapt to the target domain.

The closer the source task (original training domain) is to the target task, the more transferable these features will be.

Key Factors in Choosing a Pre-Trained Model

Selecting a pre-trained model requires analyzing *multiple aspects*, including architecture, dataset similarity, and computational constraints.

Source Model	Source Dataset	Good Transfer Learning Applications	Poor Transfer Learning Applications
ResNet-50 (general image classification)	ImageNet (general image dataset)	Medical image classification, wildlife identification	Satellite imagery classification, X-ray analysis
Faster R-CNN (general object detection)	COCO (common objects in context)	Security surveillance, pedestrian detection, object detection in retail settings	Industrial defect detection (often requires specialized features), image segmentation without object bounding boxes
BERT (natural language processing)	Wikipedia + BookCorpus (textual data)	Sentiment analysis, document classification, question answering	Generative NLP tasks requiring significant creative output, tasks requiring extensive domain-specific knowledge outside of general text
GPT-4 (language generation)	Diverse Internet Text (web pages, books, etc.)	Chatbots, summarization, creative writing, text generation	Scientific article classification (requires deep domain understanding), tasks requiring strict factual accuracy in specialized domains

Transfer Learning

Different model architectures are optimized for different tasks. Choosing the right architecture ensures that the transferred knowledge is effective.

Architecture Type	Best Suited For	Common Examples
CNNs (Convolutional Neural Networks)	Image classification, object detection	ResNet, VGG, EfficientNet
Vision Transformers (ViTs)	High-resolution images, complex visual relationships	ViT, Swin Transformer
Recurrent Neural Networks (RNNs)	Time series, sequential data	LSTMs, GRUs
Transformers (NLP)	Text classification, language generation, question answering	BERT, GPT, T5
Multimodal Models	Image + text understanding, captioning, vision-language models	CLIP, BLIP

CNN-based models (e.g., ResNet, EfficientNet) are well-suited for traditional image tasks. Transformers (e.g., ViTs, BERT) excel at understanding context in images or text. Multimodal models (e.g., CLIP, BLIP) are ideal for vision-language applications.

Larger models are often more accurate but require significant computational resources. Selecting a model that fits within memory and processing constraints is crucial.

Example Model	Relative Size	Number of Parameters	Relative Speed	Best Use Cases
ResNet-18	Small	~11 million	Fast	Embedded devices, real-time applications
ViT-Large	Medium	~307 million	Medium	High-resolution image tasks
Mistral-Large-2	Large	~123 billion	Slow	Small scale language processing, focused on math and coding.
GPT-4	Very Large	~1.8 trillion	Very Slow	Large-scale general language processing

For edge and mobile devices (e.g., phones, low-powered computers, like sensors in factories), small models (e.g., MobileNet, DistilBERT) are usually best. For cloud/server deployments, larger models (e.g., ViT, GPT-4) can be used when computational power is available. For real-time processing, models with lower latency (e.g., EfficientNet, ResNet-50) should be prioritized.

Evaluating Model Suitability for a Target Task

Now that we understand the key factors, let's outline a structured process for evaluating model suitability. Before selecting a pre-trained model, try and answer the following:

- What type of problem are you solving? (Classification, object detection, segmentation, NLP, etc.)
- What is the closest available pre-trained model? (Consider task *and* dataset similarity)
- How large is your dataset? (If small, feature extraction might be preferable)
- What are your computational constraints? (Do you have access to GPUs/TPUs?)
- Do you need real-time inference? (If yes, may be best avoid large transformer-based models)

Case Study: Choosing a Model for Wildlife Image Classification

Imagine you are developing a wildlife recognition system to identify endangered species from camera trap images. For computing resources, you have access to a high-performance computing cluster. Since there might be more than one animal in the images, you need a model that can account for that.

Option 1: ResNet-50 (Pre-Trained on ImageNet)

✅ Well-suited for general image classification
✅ Lightweight and efficient
❌ Would need a highly specialized dataset to classify multiple animals per image
❌ Might not perform well for animals in natural settings

Option 2: Swin Transformer (Pre-Trained on COCO)

✅ More robust to complex backgrounds
✅ Well-suited for multiple objects in images
❌ Computationally expensive

Option 3: Custom Model Trained from Scratch

✅ Fully optimized for the task
❌ Requires a massive, labeled dataset
❌ Computationally expensive

Given the above criteria and circumstances, the best choice is probably Option 2, the Swin Transformer pre-trained on the COCO dataset. If you could tolerate low performance and did not have access to a lot of compute resources, which option might be better?

Transfer Learning vs. Full Training

One of the key decisions in deep learning model development is whether to use **transfer learning** or **train a model from scratch**. While training from scratch offers full control over model architecture and feature learning, transfer learning allows developers to **leverage pre-trained models** for better efficiency, particularly when working with limited data or computational resources.

Comparing Transfer Learning and Full Training

Deep learning models require large datasets and significant computational power. The main question is: **should we reuse an existing model or train one from scratch?**

Key Differences

Approach	Transfer Learning	Full Training
Starting Point	Uses a pre-trained, with weights learned from training on a large dataset	Starts with randomly initialized weights
Data Requirement	Requires a smaller labeled dataset	Needs a large, labeled dataset
Training Time	Generally faster training	Potentially longer, though this is highly architecture dependent.
Computational Cost	Less expensive, can run on consumer GPUs	Requires significant GPU resources
Performance	Often achieves high accuracy with limited data	Can outperform transfer learning if sufficient data is available
Flexibility	Limited by pre-trained model's learned features	Fully customizable for the specific task

Transfer learning works **best** when the **source** and **target tasks** share similarities. Full training is preferable when the dataset is large enough and the problem requires **completely novel feature learning**.

When to Use Transfer Learning

Transfer learning is advantageous when:

- **Data is scarce** – If there are not enough labeled examples to train a deep model from scratch, a pre-trained model can extract useful features.
- **Computation is limited** – Training from scratch is expensive. Transfer learning enables high-performance models on modest hardware.
- **The problem is similar to a well-researched domain** – If a suitable pre-trained model exists, fine-tuning it can yield strong results with minimal effort.
- **Faster deployment is needed** – Transfer learning significantly reduces time-to-production.

Example: Medical Image Classification

A hospital wants to classify chest X-ray images for the presence of pneumonia. Training a deep learning model from scratch would require a large, labeled dataset, which is often difficult and expensive to obtain in the medical domain. Instead, using a pre-trained **DenseNet model, specifically one pre-trained on chest X-ray datasets or a model pre-trained on a large and diverse medical image dataset**, and fine-tuning it on the hospital's specific X-ray images, allows the model to leverage relevant feature extraction while adapting to the specific pneumonia classification task. DenseNet architectures have shown good performance on medical images due to their dense connectivity, which helps in capturing fine-grained details. Models pre-trained on relevant medical datasets are even better, as they have already learned low and mid level features that are relevant to the medical domain.

When to Use Full Training

While transfer learning is often preferable, some scenarios require full training:

A completely novel problem – If no similar pre-trained model exists, transfer learning won't help much.

Large-scale datasets are available – With millions of labeled examples, training from scratch can lead to better performance.

Customization is required – If an existing architecture doesn't suit the problem well, designing a new one from scratch allows greater flexibility.

The pre-trained model's features do not transfer well – If using transfer learning leads to **negative transfer**, full training may be a better option.

Example: Self-Driving Cars

A company developing **autonomous driving systems** may need to train an object detection model **from scratch**. While pre-trained models like YOLO or Faster R-CNN can detect general objects, self-driving cars require **specialized training** on road conditions, pedestrians, and vehicle behaviors.

Trade-Offs Between Transfer Learning and Full Training

Making the right choice involves balancing **data availability, computational resources, and task complexity**.

- **Data Requirements -** The single biggest determinant of whether to use transfer learning or full training is the availability of **high-quality labeled data**. If data is **scarce or expensive to collect**, transfer learning is almost always the better option.
    - **Transfer learning** is **highly effective for small datasets** (e.g., <100,000 images for vision tasks).
    - **Full training** requires a **massive dataset** (e.g., ImageNet has **14 million labeled images**).
- **Computational Constraints -** Deep learning models require **significant compute resources**. If access to large-scale compute power is **limited**, transfer learning is a more practical choice.
    - **Transfer learning** enables training models on **consumer-grade GPUs** (e.g., NVIDIA RTX 3090).
    - **Full training** often requires **dedicated cloud clusters or TPUs** (e.g., Google's TPUs or NVIDIA A100).
- **Training Time and Cost -** Training from scratch can be extremely **time-intensive and expensive**. For organizations on **tight budgets or timelines**, transfer learning provides a **cost-effective alternative**.
    - A **BERT language model** trained from scratch can cost **$50,000+ in compute resources**.
    - Fine-tuning **BERT for a specific NLP task** costs a fraction of that.

Case Study: Image Classification for Agriculture

Imagine a small research lab developing a deep learning model to **classify crop diseases** using images taken by farmers. They must decide whether to **train a model from scratch** or **use transfer learning** with a pre-trained model.

Option 1: Train from Scratch

✅ Fully customizable for agricultural disease detection
✅ No reliance on external pre-trained models
❌ Requires **hundreds of thousands of labeled crop images**
❌ Takes **weeks to train on high-end GPUs**
❌ Very expensive in terms of computational resources

Option 2: Transfer Learning with ResNet

- ✅ Uses a **pre-trained ResNet model** trained on ImageNet
- ✅ Fine-tuned on a **smaller dataset of crop disease images**
- ✅ Trains in **a few hours** instead of weeks
- ✅ Requires significantly **less computational power**
- ❌ May not capture unique agricultural disease patterns as well as a fully customized model

Given the above criteria, the best choice is probably Option 2!

Evaluating Transfer Learning Techniques

Once a pre-trained model has been selected and the decision to use transfer learning has been made, the next step is to **determine the best transfer learning approach**. Not all transfer learning techniques are equally effective for every task. Choosing the right method—**feature extraction, fine-tuning, or Low-Rank Adaptation (LoRA)**—depends on factors such as dataset size, task complexity, and computational constraints.

There are **several other transfer learning approaches** (such as those mentioned in the previous modules) that may be valuable depending on the use case. However, due to time constraints, we will **focus on three primary approaches in our hands-on work** while briefly discussing other techniques used in the field.

Overview of Transfer Learning Techniques

There are several approaches to transfer learning, each balancing **performance, efficiency, and adaptability**.

Technique	Description	Best Used When…
Feature Extraction	Uses a pre-trained model's frozen layers as a feature extractor; only the final classifier is trained.	Data is limited, and the target task is similar to the source task.
Fine-Tuning	Some layers of the pre-trained model are unfrozen and retrained on the new dataset.	The target task is moderately different from the source task, and sufficient data is available.
LoRA (Low-Rank Adaptation)	Injects trainable low-rank matrices into selected model layers, reducing memory requirements while adapting the model.	The target task requires adaptation but computational resources are limited.

Technique	Description	Best Used When…
Domain Adaptation	Adapts a model to a new data distribution without modifying the task itself.	The target dataset differs in style or environment from the pre-trained dataset.
Self-Supervised Pretraining	Uses unlabeled data to train feature representations before fine-tuning on a downstream task.	Labeled data is scarce but a large amount of unlabeled data is available.
Knowledge Distillation	A smaller "student" model learns from a larger "teacher" model to retain knowledge efficiently.	Model deployment requires high efficiency without losing accuracy.
Progressive Neural Networks	Adds new modules to a pre-trained model instead of modifying existing weights, preserving past knowledge.	The model needs to continually learn new tasks without catastrophic forgetting.

Each technique is suited for different scenarios, but for this course, we will **focus on feature extraction, fine-tuning, and LoRA** due to their practicality and broad applicability.

Feature Extraction: When and How to Use It

Feature extraction is one of the simplest transfer learning techniques. The pre-trained model's convolutional or transformer layers remain frozen, while a new classifier is trained on top.

When to Use Feature Extraction

- ✅ The dataset is small and lacks diversity.
- ✅ The new task is like the source task.
- ✅ The goal is to deploy a lightweight model with minimal compute requirements.

Example: Classifying Cat and Dog Images

A company wants to build a **cat vs. dog classifier** using deep learning. Instead of training from scratch, they:

1. Use **ResNet-50 trained on ImageNet** as a frozen feature extractor.
2. Remove the final classification layer.
3. Add a **new classifier** to distinguish between cats and dogs.
4. Train only the classifier while keeping all convolutional layers **frozen**.

This method **minimizes training time and data requirements**, making it ideal for small datasets.

Fine-Tuning: When and How to Use It

Fine-tuning **adjusts the weights of selected layers in a pre-trained model**, allowing it to specialize in the target task while retaining useful knowledge from the source task.

When to Use Fine-Tuning

- ✅ The target dataset is **larger and slightly different** from the source dataset.
- ✅ More control over **task-specific feature learning** is required.
- ✅ Compute resources allow for some layers to be retrained.

Example: Detecting Defects in Manufactured Products

A manufacturing company wants to detect **defective parts** in images. Instead of training a model from scratch, they:

1. Use a **pre-trained EfficientNet** model.
2. Freeze **early convolutional layers** but unfreeze **deeper layers** to learn new product-specific features.
3. Train the model on their **custom defect detection dataset**.

Fine-tuning allows the model to **adapt its feature representation**, improving accuracy for the new task.

LoRA (Low-Rank Adaptation): When and How to Use It

LoRA is a parameter-efficient adaptation method that **adds trainable low-rank matrices to frozen layers** instead of fully fine-tuning them. This significantly reduces computational cost while allowing for effective adaptation.

When to Use LoRA

- ✅ The dataset is **moderately sized**, but compute resources are limited.
- ✅ The model is a **large transformer (e.g., GPT, BERT, ViT)** that is expensive to fine-tune fully.
- ✅ The goal is to **preserve the original model's capabilities** while specializing it for a new task.

Example: Adapting a Large Language Model for Finance

A financial services company wants to adapt **GPT-4** for legal document summarization. Instead of fully fine-tuning the model, they:

1. Apply **LoRA to key attention layers** in the transformer model.
2. Train only the LoRA layers on **finance-related text data**.

3. Keep most of the model **frozen**, reducing training time and cost.

LoRA allows **efficient adaptation** while maintaining the benefits of the large pre-trained model.

Performance Evaluation: Measuring the Effectiveness of Transfer Learning

Selecting a technique is just the beginning—evaluating its effectiveness is crucial to ensure the best results.

Key Evaluation Metrics

Metric	Description	Best Used For
Accuracy	Measures the percentage of correctly classified samples.	Classification tasks
Precision & Recall	Precision measures how many predicted positives are correct; recall measures how many actual positives are identified.	Tasks with class imbalance (e.g., fraud detection, medical diagnosis)
F1-Score	Harmonic mean of precision and recall.	General evaluation of classification performance
Mean Average Precision (mAP)	Measures precision across different IoU thresholds.	Object detection
IoU (Intersection over Union)	Measures the overlap between predicted and ground-truth object bounding boxes.	Object detection, segmentation

Fine-tuning and LoRA usually lead to **better performance than feature extraction**, but they also require **more data and computation**. Evaluating models based on metrics ensures that the best trade-off is made.

Common Transfer Learning Issues & Troubleshooting Strategies

While transfer learning provides significant benefits—reducing training time, improving performance with limited data, and lowering computational costs—it is **not without challenges**. Applying pre-trained models to new tasks can lead to **catastrophic forgetting, negative transfer, and domain shift**, among other problems. Understanding these challenges and how to troubleshoot them is critical to ensuring **successful model adaptation**.

Catastrophic Forgetting

One of the most common issues in transfer learning is **catastrophic forgetting**—where a model **forgets previously learned knowledge** when fine-tuned on a new task. This is especially

problematic when fine-tuning **all layers of a pre-trained model** on a small dataset, as the model may **overfit** on the new task while losing generalization.

Why It Happens

- The new dataset is **too small**, and fine-tuning **too many layers** causes the model to forget useful features.
- The **learning rate is too high**, causing rapid weight updates that overwrite important knowledge.
- The source and target domains are **similar enough to benefit from pre-trained knowledge, but not identical**, requiring careful fine-tuning.

Troubleshooting Strategies

✅ **Freeze early layers** and only fine-tune the last few layers to preserve important low-level features.
✅ **Use a lower learning rate** (e.g., 1e-5 to 1e-4) when fine-tuning to avoid drastic weight changes.
✅ **Gradually unfreeze layers**—start with a frozen model, then incrementally unfreeze layers as needed.
✅ **Regularize the model** using dropout or weight decay to prevent overfitting to the new dataset.

Example: Catastrophic Forgetting in Language Models

Imagine fine-tuning **a BERT model pre-trained on Wikipedia** for **medical text classification** using clinical reports. Initially, the model understands **general language structure**, but after fine-tuning on the specialized dataset, it **forgets how to handle everyday sentence structures** and struggles with **non-medical text comprehension**.

Negative Transfer

Negative transfer occurs when **using a pre-trained model worsens performance** on the new task. Instead of helping, the transferred knowledge **interferes** with learning, leading to poor results.

Why It Happens

- The **source and target domains** are too **different**, and the model's learned features **don't apply well**.
- The **wrong layers are fine-tuned**, introducing harmful biases from the source model.
- The pre-trained model has **domain-specific biases** that negatively impact the target task.

Troubleshooting Strategies

✅ **Select a better source model**—choose a pre-trained model that is closer to the target task.
✅ **Use feature extraction instead of fine-tuning** if the dataset is too different.
✅ **Apply domain adaptation techniques**, such as adversarial training, to align feature distributions.
✅ **Experiment with different layers**—sometimes freezing more layers or retraining only the classifier improves results.

Example: Negative Transfer in Medical Imaging

Imagine using **a ResNet model trained on ImageNet** to classify **MRI scans**. Since **ImageNet contains natural images**, the features (e.g., edges, colors, textures) may **not transfer well** to MRI scans, which rely on **grayscale pixel intensities**. Instead, a model trained on **medical X-ray datasets** would likely perform better.

Domain Shift

Domain shift occurs when the **distribution of the new dataset is significantly different** from the data the pre-trained model was originally trained on. This can cause the model to **perform poorly** because it has never seen data like the target dataset before.

Why It Happens

- The new dataset has **different lighting, background, or resolution** than the source dataset.
- The target domain includes **rare objects or classes** that were underrepresented in the source dataset.
- The input modality changes (e.g., **satellite images vs. drone images**, **scientific text vs. casual speech**).

Troubleshooting Strategies

✅ **Use data augmentation** to simulate variations in the dataset and improve generalization.
✅ **Fine-tune the model** on a subset of the target domain before full adaptation.
✅ **Use domain adaptation techniques** such as adversarial learning or contrastive learning to align distributions.
✅ **Normalize or preprocess images** to match the conditions of the original dataset.

Example: Domain Shift in Object Detection

A company uses **a pre-trained YOLO model** to detect cars in a sunny urban setting. However, when deployed in **rainy, nighttime, or rural environments**, performance **drops drastically**. This is because the original model was trained on **well-lit city streets**. Applying **domain adaptation** (e.g., using GANs to simulate night/rain conditions) can help the model **generalize better**.

Hyperparameter Tuning for Transfer Learning

Even when avoiding major transfer learning pitfalls, **hyperparameter selection plays a crucial role** in model performance. Below are key considerations when fine-tuning transfer learning models.

Key Hyperparameters to Tune

Hyperparameter	Effect on Transfer Learning	Best Practices
Learning Rate	Controls how much weights are updated during training.	Use a **lower learning rate** (1e-5 to 1e-4) when fine-tuning.
Batch Size	Affects model stability and training speed.	Use **smaller batches** for large models (e.g., 16–32 for ViTs).
Number of Frozen Layers	Determines how much of the pre-trained knowledge is retained.	**Freeze early layers** for general tasks, unfreeze more for domain-specific tasks.
Regularization	Helps prevent overfitting to the new dataset.	Apply **dropout, L2 weight decay, or data augmentation**.

Fine-tuning hyperparameters can **significantly improve performance**, and it is recommended to use **tools like Optuna or Ray Tune** for automated optimization.

Case Study: Adapting a Sentiment Analysis Model

A company wants to fine-tune **BERT** to classify **customer reviews** as positive, neutral, or negative. However, they encounter **common transfer learning issues** along the way.

Issue 1: Catastrophic Forgetting

- The model loses its ability to **understand general sentence structure** when fine-tuning on the customer review dataset.
- **Solution:** Lower the **learning rate** and freeze **early transformer layers**.

Issue 2: Negative Transfer

- The pre-trained model was originally trained on **Wikipedia and news articles**, but the customer reviews contain **slang and informal language**.
- **Solution:** Use **domain adaptation** by first fine-tuning on a **larger dataset of informal text** before specializing on customer reviews.

Issue 3: Domain Shift

- Customer reviews often contain **emoji-based sentiment** that the model struggles with.
- **Solution:** Add **emoji tokenization** and fine-tune the model **on augmented data** that includes emojis.

Pre-Trained Model Evaluation Exercise

In this last exercise, we'll compare different pre-trained models for their suitability as the Source model for feature extraction. Then we'll practice using evaluation metrics to guide us in improving the new model. Happy coding!

The exercise for this module, "Module 3 – Hands On Exercise", can be found in the "handouts" folder of this repo.