Module 2: Implementing Transfer Learning Concepts

Module 2 Course Objectives:

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

1. Demonstrate basic coding proficiencies required to implement feature extraction with a computer vision model.
2. Demonstrate basic coding proficiencies required to implement fine-tuning with a computer vision model.
3. Demonstrate basic coding proficiencies required to implement LoRA with a language model.
4. Compare and contrast the benefits of different transfer learning strategies.

How to Transfer Learning

This module focuses on the practical implementation of transfer learning techniques. Unlike the previous module, which explored the 'why' and ‘what’ behind transfer learning, this module is dedicated to the 'how'—demonstrating key methodologies for achieving learning transfer such as feature extraction, fine-tuning, and Low-Rank Adaptation (LoRA). In the Hands-on portion of this module, we will leverage a model’s pre-trained capabilities to explore different transfer learning strategies, modifying and optimizing its behavior for new tasks. Since this module is heavily focused on implementation, this instructional section will be a bit shorter to make room for the hands-on content!

Extracting Features

As a quick recap from the previous module, **feature extraction** is a fundamental transfer learning technique in which a pre-trained model extracts meaningful representations from input data without any additional model training. Instead of training a model from scratch, we leverage the learned representations from a deep network trained on a large dataset (such as ImageNet) and apply them to a new task. This method is particularly beneficial when working with smaller datasets, as it allows us to use well-established feature hierarchies without requiring extensive computational resources.

Feature extraction is widely used in computer vision because early layers in deep neural networks learn generic image features like edges, textures, and shapes, which remain useful across different tasks. By freezing these layers and only training a new classifier on top, we can efficiently adapt pre-trained models to new domains with minimal computational overhead.

Feature Extraction Workflow

Here is a rundown of the general steps for performing feature extraction.

1. **Select a Pre-Trained Model:** Choose a model trained on a large dataset, such as ResNet, EfficientNet, or BLIP, depending on the target task. The closer the Source model is to your Target task and domain, the less work you'll need to do!
2. **Load the Pre-Trained Model:** Use frameworks like PyTorch or TensorFlow to load the model with pre-trained weights.
3. **Freeze the Early Layers:** Lock ("freeze") the weights of the initial pre-trained model's layers to retain their learned features.
4. **Modify the Output Layer(s):** Replace the top layers of the network to adapt the network for the specific task. For example, if re-using a trained image classification model (such as ResNet) for a new classification task, replace the original fully connected layer with a new one containing the classes you're looking for.
5. **Preprocess the Input Data:** Ensure the dataset being used is suitable for your Source model. This may mean preprocessing data, ensuring it's normalized and formatted to match the input specifications of the model.
6. **Extract Features:** Pass input data through the frozen layers to obtain feature representations.
7. **Train the New Output Layers:** Use the extracted features as input to train the model's new output layers on the target dataset.

8. **Evaluate and Optimize:** Assess model performance on validation data, and fine-tune hyperparameters as necessary.

Time for a tune-up

Fine-tuning is a more flexible transfer learning technique that involves unfreezing some or all of a pre-trained model's layers and retraining them on a new dataset. Unlike feature extraction, which keeps most of the model unchanged, fine-tuning allows the model to adapt more specifically to a new task by updating weight parameters.

Fine-tuning is particularly beneficial when the target dataset differs significantly from the original dataset used for pre-training. By adjusting specific layers, fine-tuning enables the model to learn new feature representations while still leveraging its previously acquired knowledge.

Fine-Tuning Workflow

Here is a rundown of the general steps for performing fine-tuning.

1. **Select a Pre-Trained Model:** Choose a suitable pre-trained model such as ResNet, ViT, or BLIP, depending on the task. As above, choose something close to your target.
2. **Load the Pre-Trained Model:** Use a deep learning framework to load the model with pre-trained weights.
3. **Determine Layers to Unfreeze:** Identify and selectively unfreeze layers that should be retrained based on the similarity between the original and new tasks.
4. **Modify the Output Layers:** Adjust the model's output layers to match the prediction target of the new dataset.
5. **Adjust Learning Rates:** Apply a lower learning rate to pre-trained layers and a higher learning rate to newly initialized layers to prevent overwriting learned representations.
6. **Preprocess the Input Data:** Ensure images are sized, normalized, and formatted to match model requirements.
7. **Train the Model:** Perform training with backpropagation, monitoring validation loss to prevent overfitting.
8. **Evaluate and Optimize:** Use performance metrics such as accuracy and F1 score, and apply hyperparameter tuning if needed.

My friend, LoRA

Low-Rank Adaptation (LoRA) is a parameter-efficient fine-tuning technique that introduces low-rank updates to specific layers of a pre-trained model. Instead of updating all model parameters, LoRA modifies only a subset of them, significantly reducing computational costs while maintaining performance.

What Makes a Model Suitable for LoRA?

LoRA is most effective for models that have many parameters and benefit from attention-based mechanisms, such as:

- **Transformer-based models** (e.g., BERT, GPT, T5) where self-attention layers can be efficiently modified.
- **Vision transformers (ViTs**) process image data through self-attention mechanisms.
- **Multimodal models** like BLIP, which integrate both vision and language features and require efficient adaptation without retraining the entire model.

LoRA Workflow

1. **Select a Pre-Trained Model:** Choose a model that supports LoRA, such as transformers (e.g., BERT, GPT), vision transformers (e.g., ViT), or multimodal models like BLIP.
2. **Identify Target Layers for LoRA:** Determine which layers will receive low-rank updates, typically attention layers in transformers or specialized layers in multimodal networks.
3. **Apply LoRA Adaptation:** Insert LoRA layers that introduce low-rank modifications without altering the entire model architecture.
4. **Preprocess the Input Data:** Ensure compatibility with the pre-trained model by resizing, normalizing, and tokenizing input.
5. **Train the Model with LoRA:** Fine-tune the model using LoRA-adapted layers, optimizing only the additional parameters.
6. **Evaluate and Optimize:** Assess model performance and refine LoRA parameters to achieve optimal results.

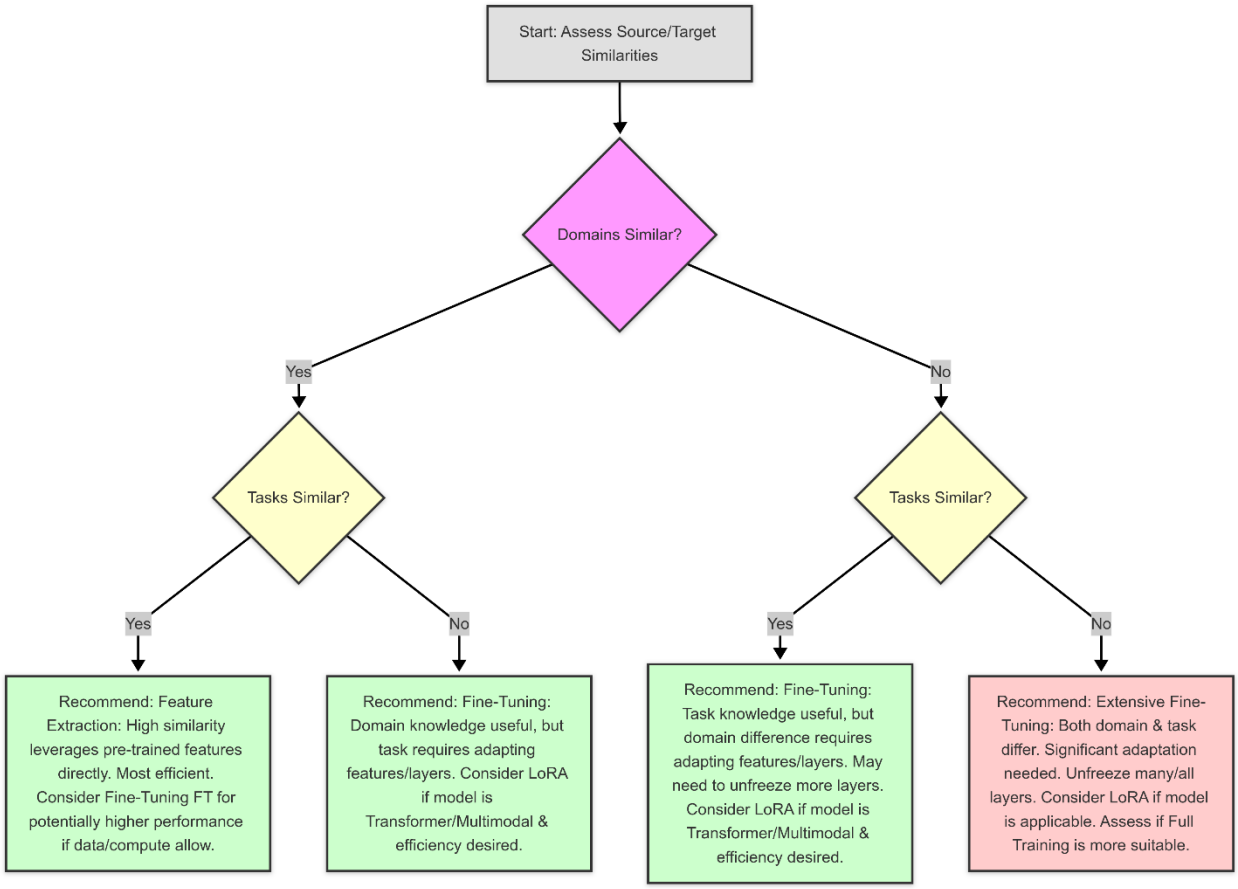
Comparing Transfer Learning Strategies

When selecting a transfer learning approach, consider the following criteria:

- **Computational Efficiency:** Feature extraction is less computationally intensive than fine-tuning, while LoRA is an efficient alternative for adapting large models.
- **Data Availability:** Fine-tuning is most beneficial when a large dataset is available, while feature extraction and LoRA perform well on limited data.
- **Task Complexity:** If the new task is significantly different from the original training data, fine-tuning or LoRA may be necessary to adapt representations effectively.

- **Model Adaptability:** LoRA is best suited for large transformer-based and multimodal models, while feature extraction and fine-tuning work well across a variety of architectures.
- **Training Time:** Feature extraction requires the least training time, while fine-tuning and LoRA require more time but provide better adaptability.

By evaluating these factors, practitioners can make informed decisions on the best approach for their specific application, balancing computational efficiency with model performance. Below is a quick flow chart to assist with your decision making.



Transfer Learning Techniques Exercise

It's time for the gauntlet. We're going to be implementing three different techniques to give you a wide breadth of experience using these tools.

Link here: [02_transfer_learning_LoRA.ipynb]