Transfer Learning

Welcome to Practicum AI's Transfer Learning Course! In these modules, we'll build on our understanding of deep learning to explore how knowledge can be transferred between models, enhancing efficiency and performance across a variety of tasks. Each module in this course will provide hands-on experiences to implement and evaluate transfer learning techniques in real-world scenarios. In Module 1, we will dig into the foundational concepts of transfer learning, explore its benefits, and examine its applications across different domains. By the end, you'll be ready to articulate key principles, differentiate between various strategies, and understand the basics of leveraging pre-trained models effectively.

Module 1: Transfer Learning Concepts

Module 1 Course Objectives:

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

1. Define transfer learning and explain its benefits in deep learning.
2. Identify common deep learning tasks where transfer learning is used.
3. List the steps involved in a typical transfer learning workflow.
4. Differentiate between various types of transfer learning techniques (e.g. feature extraction, fine-tuning, LoRA).
5. Explain how transfer learning leverages knowledge from a pre-trained model.

Deep Learning Refresher

This module builds on the material from our *Practicum AI Beginner Series* and assumes a foundation in Python and deep learning basics. We will briefly revisit foundational deep learning concepts, focusing on how they relate to transfer learning. Additionally, we will expand on certain topics, such as learning rates and layer freezing strategies, and discuss how hyperparameters play a critical role in transfer learning:

Learning Rate: Choosing an appropriate learning rate is crucial for transfer learning. Pre-trained layers typically require lower learning rates (e.g., 1e-5 to 1e-4) to avoid overwriting their learned features. Newly added layers, however, often benefit from a higher learning rate (e.g., 1e-3).

Layer Freezing: During fine-tuning or some other model training technique, specifying which layers to "freeze" depends on domain/task similarity. "Freezing" layers is a coding technique of specifying pre-trained layers of a model to *not* update their parameter weights during training. Freezing the earlier layers and fine-tuning the later layers is a common strategy when the source and target domains are similar. In dissimilar domains, unfreezing more layers allows the model to adapt to new features. After training a model with some frozen layers and some trainable layers, it is common to unfreeze the whole model, lower the learning rate, and finetune the whole model.

Conceptually, it's important to keep in mind that with most network architectures, early layers tend to be more general and the later layers more specific.

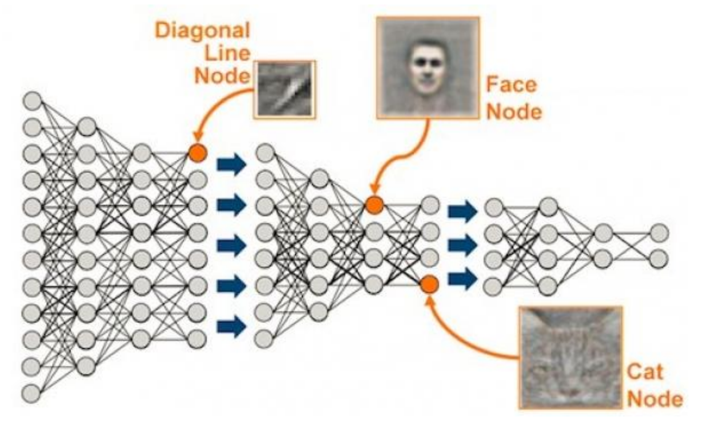

Figure 1. Representation of a deep neural network showing earlier layers capturing abstract features while later layers capture more specific features. Source - https://sdat.ir/en/sdat-blog/item/1-deep-learning, Accessed 29JAN25.

As seen in Figure 1's representation above, computer vision models tend to capture features like horizontal or vertical lines in the earliest layers, while later layers tend to capture more complex combinations of features. The same is true for language models, with earlier layers primarily focusing on encoding low-level features of the input text, like individual words and their basic grammatical structure. They are essentially capturing the "building blocks" of the language, like how early layers in a vision network might detect edges in an image. Later layers in a deep network combine information from the earlier layers to learn more complex semantic relationships, context, and sentiment within a text.

This tendency motivates the idea of freezing some layers and leaving other layers unfrozen. For very similar tasks and/or domains, you may only want to unfreeze one to a few layers of a model before fine-tuning. For more divergent tasks and/or domains, you might want to unfreeze most or all the layers.

Batch Size: Remember that the best batch size is a complex tradeoff among several considerations,

Transfer Learning Terms

Here is a rundown of some common terms and their meanings in the context of this course:

Task

- A specific problem the model is trained to solve, such as image classification, sentiment analysis, or object detection.
- Example: Classifying animals in photos.

Domain

- The type of data the model operates on, including its features and distribution.
- Example: A dataset of animal images or a collection of medical X-rays.

Source

- Refers to the pre-trained model, its task, or its domain from which knowledge is transferred.
- Example: A model trained on ImageNet. The source task (image classification) and domain (cropped images of 1,000 categories of objects).

Target

- Refers to the new model, task, or domain where transfer learning is applied.
- Example: Fine-tuning a model trained on ImageNet to classify dog breeds (same target task, slightly different domain).

including GPU compute and memory capacity, model update frequency, and sample variability. Smaller batch sizes may be required when working with large pre-trained models due to memory constraints. However, the larger variability in small samples can introduce noisier gradient updates, so careful monitoring of validation performance is needed.

Regularization: Techniques like dropout are essential when fine-tuning pre-trained models to prevent overfitting, especially when the target dataset is small.

Optimizer Choice: Optimizers like Adam or SGD with momentum are commonly used. For fine-tuning, employing optimizers that allow for differential learning rates across layers can be advantageous.

Epochs: Transfer learning often requires fewer epochs compared to training from scratch. Early stopping can be used to avoid overfitting while ensuring efficient training.

By understanding and tuning these hyperparameters, you can effectively harness the power of transfer learning for diverse applications.

What is Transfer Learning?

Transfer learning is a technique that leverages knowledge gained from a pre-trained model to solve a new but related problem. Instead of training a model from scratch, transfer learning uses a model already trained on (ideally) a large and similar dataset to transfer its learned features to a different domain and/or task, often requiring significantly less data and computational resources.

For example, a model trained to recognize objects in general images (like dogs, cars, and trees) can be adapted to identify specific types of medical abnormalities in X-rays. This approach reduces the need for large, labeled datasets, which can be expensive and time-consuming to create, and allows researchers and developers to build effective models more efficiently.

Key Techniques in Transfer Learning

- **Fine-Tuning:** Adapting the pre-trained model to the new task by unfreezing some or all its layers and retraining with the new dataset, typically with a small learning rate to prevent large changes in parameter values.
- **Low-Rank Adaptation (LoRA):** A specialized fine-tuning technique that adapts pre-trained models with fewer parameters by introducing low-rank updates, making transfer learning more computationally efficient.
- **Feature Extraction:** In this technique, the pre-trained model's layers are used as a fixed feature extractor. All layers of the base model are frozen, and only task-specific layers are trained on the new dataset. For example, in Python using TensorFlow, you can load a pre-trained model like ResNet and freeze its layers by setting trainable=False. Add a

dense layer at the top with the number of target classes and compile the model to train it. This method is efficient when the target domain closely aligns with the source domain, such as using ImageNet-trained models for animal classification tasks.

Key Uses of Transfer Learning

- **Domain Transfer:** Leveraging knowledge from one *general area* and applying it to a distinctly *different area*, even if the specifics vary quite a bit. The focus for Domain Transfer is on leveraging foundational knowledge across potentially significant domain differences.
    - **Example:** Imagine a model initially trained to classify land types (forest, water, city) using **satellite images** from space. Domain Transfer allows adapting this knowledge to analyze **low-altitude drone footage** for farming. While both involve images of the ground, the *type* of imagery (satellite vs. drone), perspective, and specific goal (general land type vs. detailed crop health) are significantly different. The model transfers its basic image understanding to the new context.
- **Domain Adaptation:** Fine-tuning a model to handle small *differences or conditions within the same general area*. The core task and domain are similar, but the specific data characteristics or environment have shifted, requiring the model to adjust. The focus is often on aligning these different data variations. While Domain Adaptation and Domain Transfer seem similar, they differ in the scale of change required. Domain Adaptation has a smaller gap between the Target and Source domains, and so more efficient techniques targeting just the gap can be used. For Domain Transfer, fine-tuning large portions of the model might be necessary due to the larger gap between the Target and Source.
    - **Example:** Consider a self-driving car model trained extensively on **U.S. roads**. Domain Adaptation is used to make it work effectively on **Indian roads**. The *general domain* (driving on roads) and core task (identifying cars, pedestrians, signs) remain the same. However, the model needs to adapt to *specific variations* like different lighting conditions, road markings, sign designs, and traffic patterns unique to India. It's learning to apply its existing driving knowledge under slightly different conditions.
- **Task Transfer:** Applying a model trained for one type of problem (e.g., image classification) to a different type of problem (e.g., object detection), transferring learned features to a new context.
    - **Example:** A model trained for image classification being adapted for object detection to localize objects in an image. For example, a ResNet model pre-trained on ImageNet can be adapted into a Faster R-CNN architecture by adding bounding box regression layers, allowing it to identify specific objects like animals in wildlife conservation studies.

- **Task Adaptation:** Modifying or fine-tuning the transferred knowledge to suit the specific requirements of the new task, such as adding or replacing layers to accommodate task-specific outputs.
    - **Example:** Adapting a facial recognition model to perform emotion detection by replacing the final classification layers with ones that predict different emotions instead of identities. By modifying the last dense layers of a VGGFace model and training it on datasets like the popular Facial Expression Recognition 2013 dataset (FER-2013), the model can accurately classify emotions such as happiness, sadness, anger, and surprise, making it useful for human-computer interaction applications.

Benefits of Transfer Learning

- **Efficiency**: Reduces the time and computational power needed to train a model.
- **Performance**: Often leads to better results, especially when working with limited data.
- **Adaptability**: Enables the application of cutting-edge models to niche or specialized domains.

Transfer learning has revolutionized fields like computer vision, natural language processing, and more, enabling rapid advancements and reducing barriers for new applications. In this course, we will explore the principles of transfer learning and its practical implementation across a variety of tasks.

Steps in a Typical Transfer Learning Workflow

The workflow for transferring learning is like that of training or pre-training. Here we'll go over the key steps involved in a typical transfer learning workflow:

Step 1: Define the Problem and Gather Data

The first step in transfer learning is to clearly define the problem and identify the target task, ensuring that you have sufficient and suitable data for the application. If the source domain (e.g., agricultural images, sentiment analysis on online product reviews, etc.) is like the target domain (e.g., satellite images, sentiment analysis on social media posts, etc.), the pre-trained model's learned features are more likely to transfer effectively. When domains or tasks differ significantly, such as adapting a general image recognition model to medical X-rays, additional preprocessing or intermediate steps may be needed. During this stage, data is collected, cleaned, and split into training, validation, and test sets while considering the distribution of classes and the volume of data available.

Step 2: Select a Pre-Trained Model

Choosing a pre-trained model involves evaluating its compatibility with the target task and domain. Popular models like ResNet, EfficientNet, or Vision Transformers are often used for vision-related tasks, while BERT and GPT are common for NLP tasks. For example, if you are working on sentiment analysis, a language model such as BERT pre-trained on large text corpora is a good starting point. Similarly, for object detection in traffic surveillance, a YOLO model pre-trained on COCO datasets may be ideal. When tasks and domains align closely, the selected model requires minimal adaptation; however, for tasks like depth estimation or pose detection, more advanced architectures or customizations may be needed.

Transfer Learning Pitfalls

Let's lay out terminology for some of common issues encountered during knowledge transfer. We'll explore these in more depth in the last module of this course:

- **Catastrophic Forgetting**: The model loses previously learned knowledge when fine-tuned on a new task.
- **Negative Transfer**: Knowledge from the source task harms performance on the target task due to domain or task mismatch.
- **Domain Shift**: Differences in data distributions between source and target domains cause poor generalization.

Step 3: Adapt the Model for the Target Task

Adapting a pre-trained model typically involves modifying its architecture to meet the requirements of the target task. For instance, replacing the final classification head of a ResNet or other model with a fully connected layer suited to classify species of plants. This step also considers the domain's data characteristics—for example, grayscale medical images versus RGB images of crops—to ensure compatibility between the model and the input data. Models trained for speech-to-text tasks, such as Whisper, might require fine-tuning for languages or accents not included in the original training data.

Step 4: Freeze and Unfreeze Layers

Deciding which layers to freeze or unfreeze depends on the similarity between the source and target tasks and the size of the available dataset. If the domain and task are closely aligned, such as reusing an ImageNet-trained model for a similar image classification task, freezing early layers and fine-tuning only the later ones is sufficient. For tasks with more significant domain differences, such as adapting a wildlife recognition model to underwater species, more layers should be unfrozen to allow the model to learn task-specific features. For small datasets, freezing most of the model prevents overfitting. For example, in NLP tasks like machine translation, only fine-tuning the attention layers of a pre-trained transformer model can yield excellent results while conserving computational resources.

Step 5: Train the Model

Training involves optimizing the model on the target dataset while leveraging the knowledge encoded in the pre-trained layers. For instance, when training a model to recognize defective parts in a factory setting, augmentation techniques such as rotations or flips can be used to mimic variations in real-world conditions. Smaller learning rates are often employed during fine-tuning to prevent catastrophic forgetting of the pre-trained knowledge.

Step 6: Evaluate the Model

Evaluation helps measure how well the model performs on unseen data and diagnose potential shortcomings. Metrics such as precision, recall, and F1-score are particularly useful for imbalanced datasets, as seen in fraud detection tasks. For vision models, visualizing confusion matrices or overlaying predicted bounding boxes on images can reveal systematic errors. For example, a model trained to detect cracks in pavement might fail on poorly lit images, signaling the need for additional data or preprocessing. Comparing performance against a baseline model trained from scratch can also highlight the benefits of transfer learning.

Step 7: Deploy and Fine-Tune

Once trained and evaluated, the model can be deployed in real-world settings. In production environments, it's essential to monitor performance and retrain periodically if data shifts occur. For instance, a sentiment analysis model deployed in a social media context might require retraining to keep up with evolving language trends or slang. Post-deployment, fine-tuning with new data, such as user feedback or additional labeled examples, ensures that the model remains effective and relevant over time.

Step 8: Iterate and Optimize

The final step is to refine and optimize the model continuously. Experimenting with different pre-trained models, such as comparing ResNet with EfficientNet for image tasks, can yield insights into what works best for the given problem. For example, models trained on aerial imagery may require fine-tuning strategies like domain adaptation to account for seasonal variations in vegetation. Additionally, collecting more data, refining hyperparameters like learning rate schedules, and implementing better regularization techniques such as dropout further improve performance. Continuous evaluation and iteration are crucial to adapting the model to changing real-world conditions.

Fine-Tuning Exercise

Now that we've gone over the basic concepts of transfer learning, let's dip our toes into one of the simpler techniques: Fine-tuning. We'll look at a case study for adapting models to better identify agricultural images and compare that to a model we'll train ourselves on a smaller dataset.

- The GitHub repository for this workshop is at https://github.com/PracticumAI/transfer_learning
- The SCINet Workshop site is at https://scinet.usda.gov/events/2025-04-22-transfer-learning

Follow the directions on the SCINet Workshop site to get things set up.

Work through notebook 01_transfer_learning_fine_tuning.ipynb.