Transfer Learning

Humans have an inherent ability to transfer knowledge across tasks. What we acquire as knowledge while learning about one task, we utilize in the same way to solve related tasks. The more related the tasks, the easier it is for us to transfer, or cross-utilize our knowledge. Some simple examples would be,

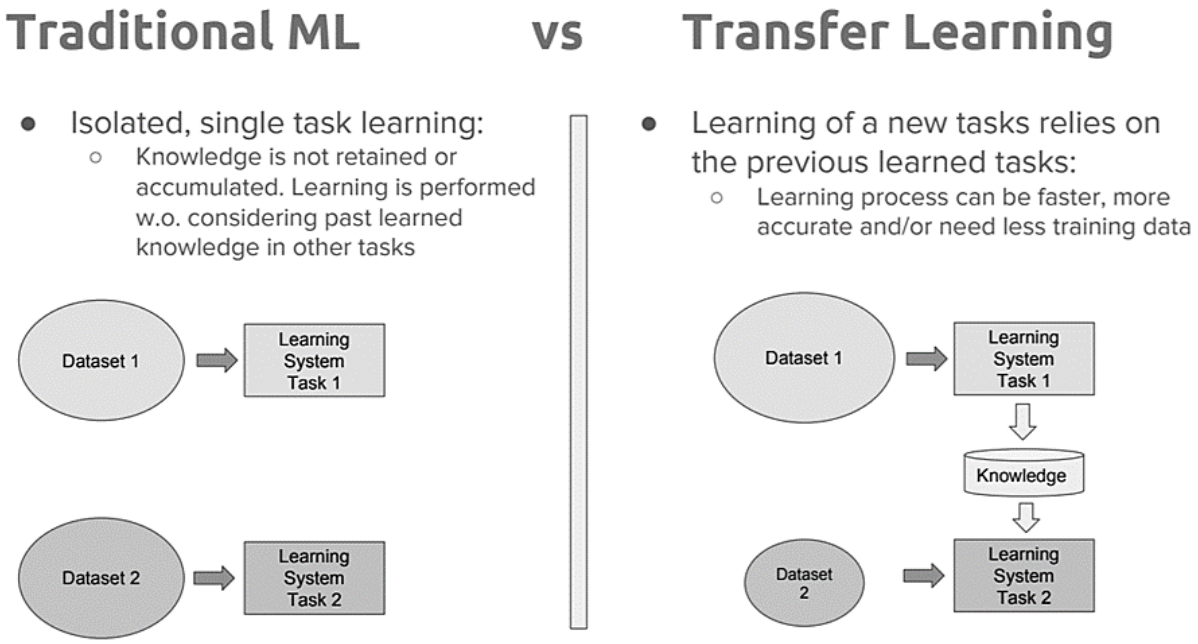
Know how to ride a motorbike ➦ Learn how to ride a car

Know how to play classic piano ➦ Learn how to play jazz piano

Know math and statistics ➦ Learn machine learning

In each of the above scenarios, we don't learn everything from scratch when we attempt to learn new aspects or topics. We transfer and leverage our knowledge from what we have learnt in the past!

Conventional machine learning and deep learning algorithms, so far, have been traditionally designed to work in isolation. These algorithms are trained to solve specific tasks. The models have to be rebuilt from scratch once the feature-space distribution changes. Transfer learning is the idea of overcoming the isolated learning paradigm and utilizing knowledge acquired for one task to solve related ones.



The NIPS 1995 workshop Learning to Learn: Knowledge Consolidation and Transfer in Inductive Systems is believed to have provided the initial motivation for research in this field.
Situation where what has been learned in one setting is exploited to improve generalization in another setting.

How does Transfer Learning Contributing to the Progression of Neural Information Processing ?

Now, a common misconception in the DL community is that without a Google-esque amount of data, you can't possibly hope to create effective deep learning models. While data is a critical part of creating the network, the idea of transfer learning has helped to lessen the data demands. Transfer learning is the process of taking a pre-trained model (the weights and parameters of a network that has been trained on a large dataset by somebody else) and "fine-tuning" the model with your own dataset. The idea is that this pre-trained model will act as a feature extractor. You will remove the last layer of the network and replace it with your own classifier (depending on what your problem space is). You then freeze the weights of all the other layers and train the network normally (Freezing the layers means not changing the weights during gradient descent/optimization).

Let's investigate why this works. Let's say the pre-trained model that we're talking about was trained on ImageNet (a dataset that contains 14 million images with over 1,000 classes). When we think about the lower layers of the network, we know that they will detect features like edges and curves. Now, unless you have a very unique problem space and dataset, your network is going to need to detect curves and edges as well. Rather than training the whole network through a random initialization of weights, we can use the weights of the pre-trained model (and freeze them) and focus on the more important layers (ones that are higher up) for training. If your dataset is quite different than something like ImageNet, then you'd want to train more of your layers and freeze only a couple of the low layers.

We can summarize some of these usage patterns of transfer learning as follows:

- **Classifier:** The pre-trained model is used directly to classify new images.
- **Standalone Feature Extractor:** The pre-trained model, or some portion of the model, is used to pre-process images and extract relevant features.
- **Integrated Feature Extractor:** The pre-trained model, or some portion of the model, is integrated into a new model, but layers of the pre-trained model are frozen during training.
- **Weight Initialization:** The pre-trained model, or some portion of the model, is integrated into a new model, and the layers of the pre-trained model are trained in concert with the new model.

Each approach can be effective and save significant time in developing and training a deep convolutional neural network model.

Understanding Transfer Learning:

The first thing to remember here is that, transfer learning, is not a new concept which is very specific to deep learning. There is a stark difference between the traditional approach of building and training machine learning models, and using a methodology following transfer learning principles.

Traditional learning is isolated and occurs purely based on specific tasks, datasets and training separate isolated models on them. No knowledge is retained which can be transferred from one model to another. In transfer learning, you can leverage knowledge (features, weights etc) from previously trained models for training newer models and even tackle problems like having less data for the newer task!

Let's understand the preceding explanation with the help of an example. Let's assume our task is to identify objects in images within a restricted domain of a restaurant. Let's mark this task in its defined scope as T1. Given the dataset for this task, we train a model and tune it to perform well (generalize) on unseen data points from the same domain (restaurant). Traditional supervised ML algorithms break down when we do not have sufficient training examples for the required tasks in given domains. Suppose, we now must detect objects from images in a park or a café (say, task T2). Ideally, we should be able to apply the model trained for T1, but in reality, we face performance degradation and models that do not generalize well. This happens for a variety of reasons, which we can liberally and collectively term as the model's bias towards training data and domain.

Transfer learning should enable us to utilize knowledge from previously learned tasks and apply them to newer, related ones. If we have significantly more data for task T1, we may utilize its learning, and generalize this knowledge (features, weights) for task T2 (which has significantly less data). In the case of problems in the computer vision domain, certain low-level features, such as edges, shapes, corners and intensity, can be shared across tasks, and thus enable knowledge transfer among tasks! Also, as we have depicted in the earlier figure, knowledge from an existing task acts as an additional input when learning a new target task.

Practical Advice:

There are a few additional things to keep in mind when performing Transfer Learning:

If you wish to use a pretrained network, you may be slightly constrained in terms of the architecture you can use for your new dataset. For example, you can't arbitrarily take out Conv layers from the pretrained network. However, some changes are straight-forward: Due to parameter sharing, you can easily run a pretrained network on images of different spatial size. This is clearly evident in the case of Conv/Pool layers because their forward function is

independent of the input volume spatial size (as long as the strides "fit"). In case of FC layers, this still holds true because FC layers can be converted to a Convolutional Layer: For example, in an AlexNet, the final pooling volume before the first FC layer is of size [6x6x512]. Therefore, the FC layer looking at this volume is equivalent to having a Convolutional Layer that has receptive field size 6x6, and is applied with padding of 0.

Learning rates: It's common to use a smaller learning rate for ConvNet weights that are being fine-tuned, in comparison to the (randomly-initialized) weights for the new linear classifier that computes the class scores of your new dataset. This is because we expect that the ConvNet weights are relatively good, so we don't wish to distort them too quickly and too much (especially while the new Linear Classifier above them is being trained from random initialization).

How do you decide what type of transfer learning you should perform on a new dataset?

This is a function of several factors, but the two most important ones are the size of the new dataset (small or big), and its similarity to the original dataset (e.g. ImageNet-like in terms of the content of images and the classes, or very different, such as microscope images). Keeping in mind that ConvNet features are more generic in early layers and more original-dataset-specific in later layers, here are some common rules of thumb for navigating the 4 major scenarios:

- New dataset is small and similar to original dataset. Since the data is small, it is not a good idea to fine-tune the ConvNet due to overfitting concerns. Since the data is similar to the original data, we expect higher-level features in the ConvNet to be relevant to this dataset as well. Hence, the best idea might be to train a linear classifier on the CNN codes.
- New dataset is large and similar to the original dataset. Since we have more data, we can have more confidence that we won't overfit if we were to try to fine-tune through the full network.
- New dataset is small but very different from the original dataset. Since the data is small, it is likely best to only train a linear classifier. Since the dataset is very different, it might not be best to train the classifier form the top of the network, which contains more dataset-specific features. Instead, it might work better to train the SVM classifier from activations somewhere earlier in the network.
- New dataset is large and very different from the original dataset. Since the dataset is very large, we may expect that we can afford to train a ConvNet from scratch. However, in practice it is very often still beneficial to initialize with weights from a pretrained model. In this case, we would have enough data and confidence to fine-tune through the entire network.

Different Types of Transfer Learning:

- **Inductive Transfer learning:** In this scenario, the source and target domains are the same, yet the source and target tasks are different from each other. The algorithms try to utilize the inductive biases of the source domain to help improve the target task. Depending upon whether the source domain contains labeled data or not, this can be further divided into two subcategories, similar to multitask learning and self-taught learning, respectively.
- **Unsupervised Transfer Learning:** This setting is similar to inductive transfer itself, with a focus on unsupervised tasks in the target domain. The source and target domains are similar, but the tasks are different. In this scenario, labeled data is unavailable in either of the domains.
- **Transductive Transfer Learning:** In this scenario, there are similarities between the source and target tasks, but the corresponding domains are different. In this setting, the source domain has a lot of labeled data, while the target domain has none. This can be further classified into subcategories, referring to settings where either the feature spaces are different or the marginal probabilities.

How to fine tune if the new dataset is very different from the orginal dataset ?

The earlier features of a ConvNet contain more generic features (e.g. edge detectors or color blob detectors), but later layers of the ConvNet becomes progressively more specific to the details of the classes contained in the original dataset.

The earlier layers can help to extract the features of the new data. So it will be good if you fix the earlier layers and retrain the rest of the layers, if you got only small amount of data. If you have large amount of data, you can retrain the whole network with weights initialized from the pre-trained network.

Fine Tuning:

We can give the new dataset to fine tune the pre-trained CNN. Consider that the new dataset is almost similar to the orginal dataset used for pre-training. Since the new dataset is similar, the same weights can be used for extracting the features from the new dataset.

If the new dataset is very small, it's better to train only the final layers of the network to avoid overfitting, keeping all other layers fixed. So remove the final layers of the pre-trained network. Add new layers . Retrain only the new layers. If the new dataset is very much large, retrain the whole network with initial weights from the pretrained model.

Freeze or fine-tune?

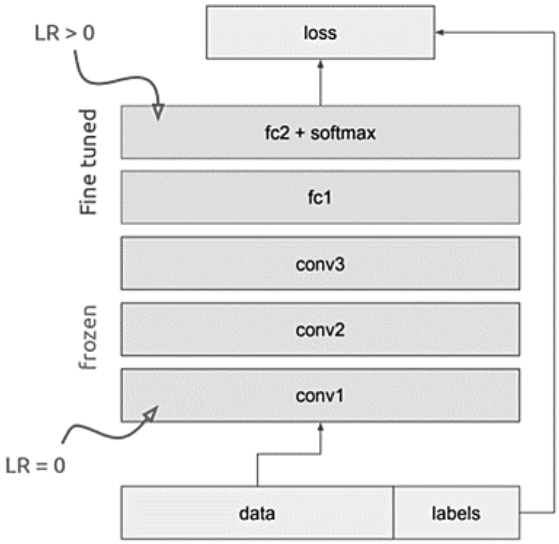
Bottom *n* layers can be frozen or fine tuned.

- **Frozen**: not updated during backprop
- **Fine-tuned**: updated during backprop

Which to do depends on target task:

- **Freeze**: target task labels are scarce, and we want to avoid overfitting
- **Fine-tune**: target task labels are more plentiful

In general, we can set learning rates to be different for each layer to find a tradeoff between freezing and fine tuning



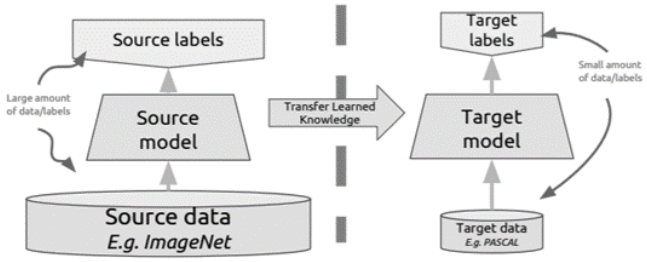
Transfer learning: idea

Instead of training a deep network from scratch for your task:

- Take a network trained on a different domain for a different **source task**
- Adapt it for your domain and your **target task**

Variations:

- Same domain, different task
- Different domain, same task



Transfer learning in DL

Myth: you can't do deep learning unless you have a million labeled examples for your problem.

Reality

- You can learn useful representations from **unlabeled data**
- You can train on a nearby **surrogate objective** for which it is easy to generate labels
- You can **transfer** learned representations from a related task