**Introduction**

Transfer learning involves the approach in which knowledge learned in one or more source tasks is transferred and used to improve the learning of a related target task.

**Why transfer learning ?**

Many deep neural networks trained on natural images exhibit a curious phenomenon in common: on the first layer they learn features similar to Gabor filters and color blobs. Such first-layer features appear not to specific to a particular dataset or task but are general in that they are applicable to many datasets and tasks. As finding these standard features on the first layer seems to occur regardless of the exact cost function and natural image dataset, we call these first-layer features general. For example, in a network with an N-dimensional softmax output layer that has been successfully trained towards a supervised classification objective, each output unit will be specific to a particular class. We thus call the last-layer features specific.

In transfer learning we first train a base network on a base dataset and task, and then we repurpose the learned features, or transfer them, to a second target network to be trained on a target dataset and task. This process will tend to work if the features are general, that is, suitable to both base and target tasks, instead of being specific to the base task.

In practice, very few people train an entire Convolutional Network from scratch because it is relatively rare to have a dataset of sufficient size. Instead, it is common to pre-train a ConvNet on a very large dataset (e.g. ImageNet, which contains 1.2 million images with 1000 categories), and then use the ConvNet either as an initialization or a fixed feature extractor for the task of interest.

**Transfer learning scenarios**

Depending on both the size of the new dataset and the similarity of the new dataset to the original dataset, the approach for using transfer learning will be different. Keeping in mind that ConvNet features are more generic in the early layers and more original-dataset specific in the later layers, here are some common rules of thumb for navigating the four major scenarios:

1. The *target* dataset is **small** and **similar** to the *base* training dataset.  
   Since the target dataset is small, it is not a good idea to fine-tune the ConvNet due to the risk of overfitting. Since the *target* data is similar to the *base* data, we expect higher-level features in the ConvNet to be relevant to this dataset as well. Hence, we:
   * Remove the fully connected layers near the end of the pretrained *base* ConvNet
   * Add a new fully connected layer that matches the number of classes in the *target* dataset
   * Randomize the weights of the new fully connected layer and freeze all the weights from the pre-trained network
   * Train the network to update the weights of the new fully connected layers
2. The *target* dataset is **large** and **similar** to the *base* training dataset.   
   Since the *target* dataset is large, we have more confidence that we won’t overfit if we try to fine-tune through the full network. Therefore, we:
   * Remove the last fully connected layer and replace with the layer matching the number of classes in the *target* dataset
   * Randomly initialize the weights in the new fully connected layer
   * Initialize the rest of the weights using the pre-trained weights, i.e., unfreeze the layers of the pre-trained network
   * Retrain the entire neural network
3. The *target* dataset is **small** and **different** from the *base* training dataset.   
   Since the data is small, overfitting is a concern. Hence, we train only the linear layers. But as the *target* dataset is very different from the *base* dataset, the higher level features in the ConvNet would not be of any relevance to the *target* dataset. So, the new network will only use the lower level features of the *base* ConvNet. To implement this scheme, we:
   * Remove most of the pre-trained layers near the beginning of the ConvNet
   * Add to the remaining pre-trained layers new fully connected layers that match the number of classes in the new dataset
   * Randomize the weights of the new fully connected layers and freeze all the weights from the pre-trained network
   * Train the network to update the weights of the new fully connected layers
4. The *target* dataset is large and different from the *base* training dataset.   
   As the *target* dataset is large and different from the *base* dataset, we can train the ConvNet from scratch. However, in practice, it is beneficial to initialize the weights from the pre-trained network and fine-tune them as it might make the training faster. In this condition, the implementation is the same as in case 3.

