to the one you are trying to tackle, then just reuse the lower layers of this network: this is called *transfer learning*. It will not only speed up training considerably, but will also require much less training data.

For example, suppose that you have access to a DNN that was trained to classify pictures into 100 different categories, including animals, plants, vehicles, and everyday objects. You now want to train a DNN to classify specific types of vehicles. These tasks are very similar, so you should try to reuse parts of the first network (see Figure 11-4).

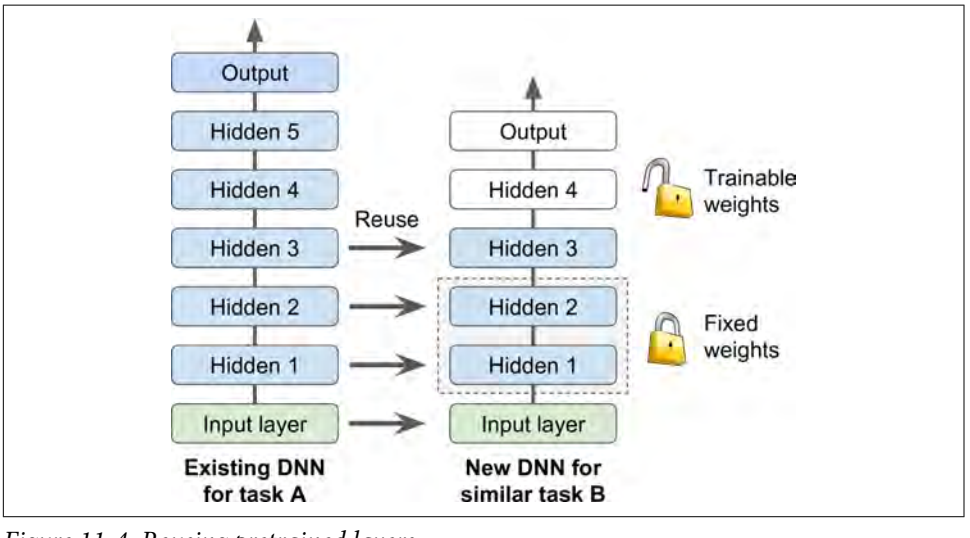

Figure 11-4. Reusing pretrained layers

If the input pictures of your new task don't have the same size as the ones used in the original task, you will have to add a preprocessing step to resize them to the size expected by the original model. More generally, transfer learning will work only well if the inputs have similar low-level features.

Reusing a TensorFlow Model

If the original model was trained using TensorFlow, you can simply restore it and train it on the new task:

```
[...] # construct the original model

with tf.Session() as sess:
    saver.restore(sess, "./my_original_model.ckpt")
    [...] # Train it on your new task
```

However, in general you will want to reuse only part of the original model (as we will discuss in a moment). A simple solution is to configure the `Saver` to restore only a subset of the variables from the original model. For example, the following code restores only hidden layers 1, 2, and 3:

```
[...] # build new model with the same definition as before for hidden layers 1-3

init = tf.global_variables_initializer()

reuse_vars = tf.get_collection(tf.GraphKeys.TRAINABLE_VARIABLES,
                               scope="hidden[123]")
reuse_vars_dict = dict([(var.name, var.name) for var in reuse_vars])
original_saver = tf.Saver(reuse_vars_dict) # saver to restore the original model

new_saver = tf.Saver() # saver to save the new model

with tf.Session() as sess:
    sess.run(init)
    original_saver.restore("./my_original_model.ckpt") # restore layers 1 to 3
    [...] # train the new model
    new_saver.save("./my_new_model.ckpt") # save the whole model
```

First we build the new model, making sure to copy the original model's hidden layers 1 to 3. We also create a node to initialize all variables. Then we get the list of all variables that were just created with `"trainable=True"` (which is the default), and we keep only the ones whose scope matches the regular expression `"hidden[123]"` (i.e., we get all trainable variables in hidden layers 1 to 3). Next we create a dictionary mapping the name of each variable in the original model to its name in the new model (generally you want to keep the exact same names). Then we create a `Saver` that will restore only these variables, and we create another `Saver` to save the entire new model, not just layers 1 to 3. We then start a session and initialize all variables in the model, then restore the variable values from the original model's layers 1 to 3. Finally, we train the model on the new task and save it.

The more similar the tasks are, the more layers you want to reuse (starting with the lower layers). For very similar tasks, you can try keeping all the hidden layers and just replace the output layer.

Reusing Models from Other Frameworks

If the model was trained using another framework, you will need to load the weights manually (e.g., using Theano code if it was trained with Theano), then assign them to the appropriate variables. This can be quite tedious. For example, the following code shows how you would copy the weight and biases from the first hidden layer of a model trained using another framework: