- Then comes the tall stack of nine inception modules, interleaved with a couple max pooling layers to reduce dimensionality and speed up the net.
- Next, the average pooling layer uses a kernel the size of the feature maps with VALID padding, outputting 1×1 feature maps: this surprising strategy is called *global average pooling*. It effectively forces the previous layers to produce feature maps that are actually confidence maps for each target class (since other kinds of features would be destroyed by the averaging step). This makes it unnecessary to have several fully connected layers at the top of the CNN (like in AlexNet), considerably reducing the number of parameters in the network and limiting the risk of overfitting.
- The last layers are self-explanatory: dropout for regularization, then a fully connected layer with a softmax activation function to output estimated class probabilities.

This diagram is slightly simplified: the original GoogLeNet architecture also included two auxiliary classifiers plugged on top of the third and sixth inception modules. They were both composed of one average pooling layer, one convolutional layer, two fully connected layers, and a softmax activation layer. During training, their loss (scaled down by 70%) was added to the overall loss. The goal was to fight the vanishing gradients problem and regularize the network. However, it was shown that their effect was relatively minor.

ResNet

Last but not least, the winner of the ILSVRC 2015 challenge was the *Residual Network* (or *ResNet*), developed by Kaiming He et al.,[12] which delivered an astounding top-5 error rate under 3.6%, using an extremely deep CNN composed of 152 layers. The key to being able to train such a deep network is to use *skip connections* (also called *shortcut connections*): the signal feeding into a layer is also added to the output of a layer located a bit higher up the stack. Let's see why this is useful.

When training a neural network, the goal is to make it model a target function $h(\mathbf{x})$. If you add the input $\mathbf{x}$ to the output of the network (i.e., you add a skip connection), then the network will be forced to model $f(\mathbf{x}) = h(\mathbf{x}) - \mathbf{x}$ rather than $h(\mathbf{x})$. This is called *residual learning* (see Figure 13-12).

12 "Deep Residual Learning for Image Recognition," K. He (2015).

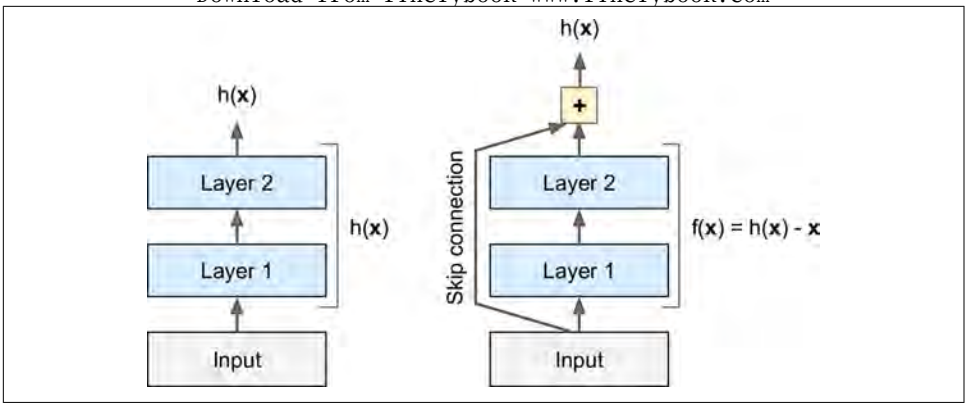

Figure 13-12. Residual learning

When you initialize a regular neural network, its weights are close to zero, so the network just outputs values close to zero. If you add a skip connection, the resulting network just outputs a copy of its inputs; in other words, it initially models the identity function. If the target function is fairly close to the identity function (which is often the case), this will speed up training considerably.

Moreover, if you add many skip connections, the network can start making progress even if several layers have not started learning yet (see Figure 13-13). Thanks to skip connections, the signal can easily make its way across the whole network. The deep residual network can be seen as a stack of *residual units*, where each residual unit is a small neural network with a skip connection.

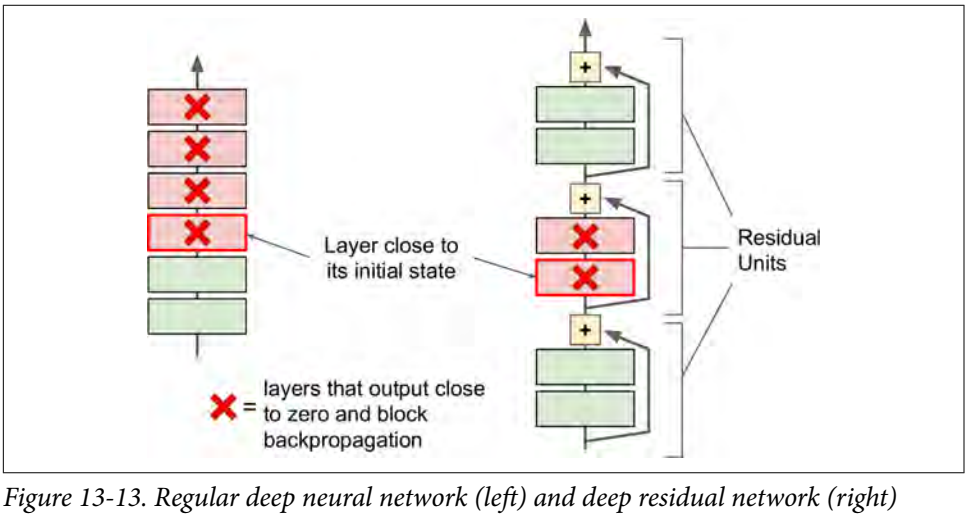

Figure 13-13. Regular deep neural network (left) and deep residual network (right)

Now let's look at ResNet's architecture (see Figure 13-14). It is actually surprisingly simple. It starts and ends exactly like GoogLeNet (except without a dropout layer), and in between is just a very deep stack of simple residual units. Each residual unit is composed of two convolutional layers, with Batch Normalization (BN) and ReLU activation, using 3 × 3 kernels and preserving spatial dimensions (stride 1, SAME padding).

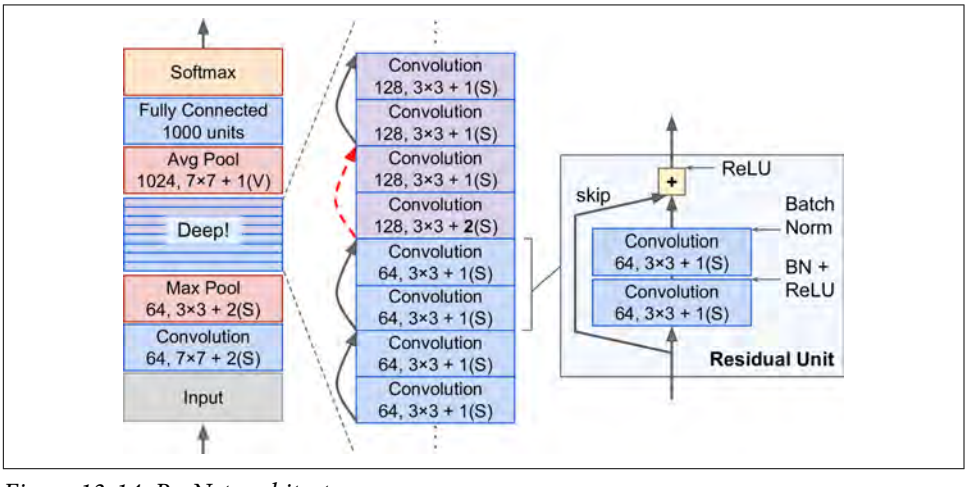

Figure 13-14. ResNet architecture

Note that the number of feature maps is doubled every few residual units, at the same time as their height and width are halved (using a convolutional layer with stride 2). When this happens the inputs cannot be added directly to the outputs of the residual unit since they don't have the same shape (for example, this problem affects the skip connection represented by the dashed arrow in Figure 13-14). To solve this problem, the inputs are passed through a 1 × 1 convolutional layer with stride 2 and the right number of output feature maps (see Figure 13-15).

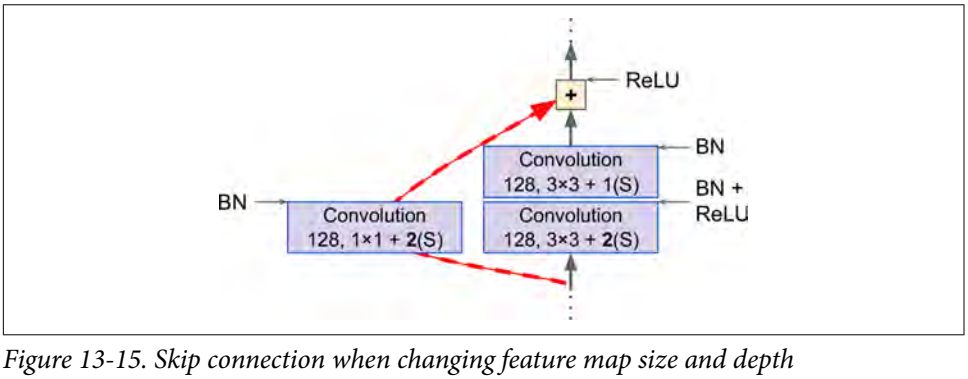

Figure 13-15. Skip connection when changing feature map size and depth

ResNet-34 is the ResNet with 34 layers (only counting the convolutional layers and the fully connected layer) containing three residual units that output 64 feature maps, 4 RUs with 128 maps, 6 RUs with 256 maps, and 3 RUs with 512 maps.

ResNets deeper than that, such as ResNet-152, use slightly different residual units. Instead of two 3 × 3 convolutional layers with (say) 256 feature maps, they use three convolutional layers: first a 1 × 1 convolutional layer with just 64 feature maps (4 times less), which acts a a bottleneck layer (as discussed already), then a 3 × 3 layer with 64 feature maps, and finally another 1 × 1 convolutional layer with 256 feature maps (4 times 64) that restores the original depth. ResNet-152 contains three such RUs that output 256 maps, then 8 RUs with 512 maps, a whopping 36 RUs with 1,024 maps, and finally 3 RUs with 2,048 maps.

As you can see, the field is moving rapidly, with all sorts of architectures popping out every year. One clear trend is that CNNs keep getting deeper and deeper. They are also getting lighter, requiring fewer and fewer parameters. At present, the ResNet architecture is both the most powerful and arguably the simplest, so it is really the one you should probably use for now, but keep looking at the ILSVRC challenge every year. The 2016 winners were the Trimps-Soushen team from China with an astounding 2.99% error rate. To achieve this they trained combinations of the previous models and joined them into an ensemble. Depending on the task, the reduced error rate may or may not be worth the extra complexity.

There are a few other architectures that you may want to look at, in particular VGGNet[13] (runner-up of the ILSVRC 2014 challenge) and Inception-v4[14] (which merges the ideas of GoogLeNet and ResNet and achieves close to 3% top-5 error rate on ImageNet classification).

There is really nothing special about implementing the various CNN architectures we just discussed. We saw earlier how to build all the individual building blocks, so now all you need is to assemble them to create the desired architecture. We will build ResNet-34 in the upcoming exercises and you will find full working code in the Jupyter notebooks.

13 "Very Deep Convolutional Networks for Large-Scale Image Recognition," K. Simonyan and A. Zisserman (2015).

14 "Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning," C. Szegedy et al. (2016).

TensorFlow Convolution Operations

TensorFlow also offers a few other kinds of convolutional layers:

- `conv1d()` creates a convolutional layer for 1D inputs. This is useful, for example, in natural language processing, where a sentence may be represented as a 1D array of words, and the receptive field covers a few neighboring words.
- `conv3d()` creates a convolutional layer for 3D inputs, such as 3D PET scan.
- `atrous_conv2d()` creates an *atrous convolutional layer* ("à trous" is French for "with holes"). This is equivalent to using a regular convolutional layer with a filter dilated by inserting rows and columns of zeros (i.e., holes). For example, a 1×3 filter equal to `[[1,2,3]]` may be dilated with a *dilation rate* of 4, resulting in a *dilated filter* `[[1, 0, 0, 0, 2, 0, 0, 0, 3]]`. This allows the convolutional layer to have a larger receptive field at no computational price and using no extra parameters.
- `conv2d_transpose()` creates a *transpose convolutional layer*, sometimes called a *deconvolutional layer*,[15] which *upsamples* an image. It does so by inserting zeros between the inputs, so you can think of this as a regular convolutional layer using a fractional stride. Upsampling is useful, for example, in image segmentation: in a typical CNN, feature maps get smaller and smaller as you progress through the network, so if you want to output an image of the same size as the input, you need an upsampling layer.
- `depthwise_conv2d()` creates a *depthwise convolutional layer* that applies every filter to every individual input channel independently. Thus, if there are f_n filters and $f_{n'}$ input channels, then this will output $f_n \times f_{n'}$ feature maps.
- `separable_conv2d()` creates a *separable convolutional layer* that first acts like a depthwise convolutional layer, then applies a 1×1 convolutional layer to the resulting feature maps. This makes it possible to apply filters to arbitrary sets of inputs channels.

Exercises

1. What are the advantages of a CNN over a fully connected DNN for image classification?
2. Consider a CNN composed of three convolutional layers, each with 3×3 kernels, a stride of 2, and SAME padding. The lowest layer outputs 100 feature maps, the

15 This name is quite misleading since this layer does *not* perform a deconvolution, which is a well-defined mathematical operation (the inverse of a convolution).