

Download from finelybook www.finelybook.com
 randomly shifting the training images by various offsets, flipping them horizontally, and changing the lighting conditions.

AlexNet also uses a competitive normalization step immediately after the ReLU step of layers C1 and C3, called *local response normalization*. This form of normalization makes the neurons that most strongly activate inhibit neurons at the same location but in neighboring feature maps (such competitive activation has been observed in biological neurons). This encourages different feature maps to specialize, pushing them apart and forcing them to explore a wider range of features, ultimately improving generalization. Equation 13-2 shows how to apply LRN.

Equation 13-2. Local response normalization

$$b_i = a_i \left(k + \alpha \sum_{j=j_{\text{low}}}^{j_{\text{high}}} a_j^2 \right)^{-\beta} \quad \text{with} \quad \begin{cases} j_{\text{high}} = \min \left(i + \frac{r}{2}, f_n - 1 \right) \\ j_{\text{low}} = \max \left(0, i - \frac{r}{2} \right) \end{cases}$$

- b_i is the normalized output of the neuron located in feature map i , at some row u and column v (note that in this equation we consider only neurons located at this row and column, so u and v are not shown).
- a_i is the activation of that neuron after the ReLU step, but before normalization.
- k , α , β , and r are hyperparameters. k is called the *bias*, and r is called the *depth radius*.
- f_n is the number of feature maps.

For example, if $r = 2$ and a neuron has a strong activation, it will inhibit the activation of the neurons located in the feature maps immediately above and below its own.

In AlexNet, the hyperparameters are set as follows: $r = 2$, $\alpha = 0.00002$, $\beta = 0.75$, and $k = 1$. This step can be implemented using TensorFlow's `local_response_normalization()` operation.

A variant of AlexNet called *ZF Net* was developed by Matthew Zeiler and Rob Fergus and won the 2013 ILSVRC challenge. It is essentially AlexNet with a few tweaked hyperparameters (number of feature maps, kernel size, stride, etc.).

GoogLeNet

The **GoogLeNet architecture** was developed by Christian Szegedy et al. from Google Research,¹⁰ and it won the ILSVRC 2014 challenge by pushing the top-5 error rate

¹⁰ "Going Deeper with Convolutions," C. Szegedy et al. (2015).

below 7%. This great performance came in large part from the fact that the network was much deeper than previous CNNs (see Figure 13-11). This was made possible by sub-networks called *inception modules*,¹¹ which allow GoogLeNet to use parameters much more efficiently than previous architectures: GoogLeNet actually has 10 times fewer parameters than AlexNet (roughly 6 million instead of 60 million).

Figure 13-10 shows the architecture of an inception module. The notation “ $3 \times 3 + 2(S)$ ” means that the layer uses a 3×3 kernel, stride 2, and SAME padding. The input signal is first copied and fed to four different layers. All convolutional layers use the ReLU activation function. Note that the second set of convolutional layers uses different kernel sizes (1×1 , 3×3 , and 5×5), allowing them to capture patterns at different scales. Also note that every single layer uses a stride of 1 and SAME padding (even the max pooling layer), so their outputs all have the same height and width as their inputs. This makes it possible to concatenate all the outputs along the depth dimension in the final *depth concat layer* (i.e., stack the feature maps from all four top convolutional layers). This concatenation layer can be implemented in TensorFlow using the `concat()` operation, with `axis=3` (axis 3 is the depth).

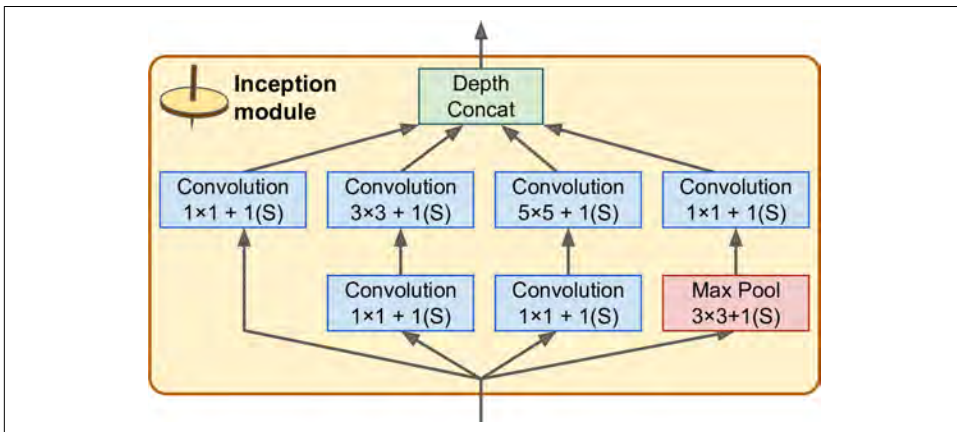


Figure 13-10. Inception module

You may wonder why inception modules have convolutional layers with 1×1 kernels. Surely these layers cannot capture any features since they look at only one pixel at a time? In fact, these layers serve two purposes:

- First, they are configured to output many fewer feature maps than their inputs, so they serve as *bottleneck layers*, meaning they reduce dimensionality. This is par-

¹¹ In the 2010 movie *Inception*, the characters keep going deeper and deeper into multiple layers of dreams, hence the name of these modules.

ticularly useful before the 3×3 and 5×5 convolutions, since these are very computationally expensive layers.

- Second, each pair of convolutional layers ($[1 \times 1, 3 \times 3]$ and $[1 \times 1, 5 \times 5]$) acts like a single, powerful convolutional layer, capable of capturing more complex patterns. Indeed, instead of sweeping a simple linear classifier across the image (as a single convolutional layer does), this pair of convolutional layers sweeps a two-layer neural network across the image.

In short, you can think of the whole inception module as a convolutional layer on steroids, able to output feature maps that capture complex patterns at various scales.



The number of convolutional kernels for each convolutional layer is a hyperparameter. Unfortunately, this means that you have six more hyperparameters to tweak for every inception layer you add.

Now let's look at the architecture of the GoogLeNet CNN (see [Figure 13-11](#)). It is so deep that we had to represent it in three columns, but GoogLeNet is actually one tall stack, including nine inception modules (the boxes with the spinning tops) that actually contain three layers each. The number of feature maps output by each convolutional layer and each pooling layer is shown before the kernel size. The six numbers in the inception modules represent the number of feature maps output by each convolutional layer in the module (in the same order as in [Figure 13-10](#)). Note that all the convolutional layers use the ReLU activation function.

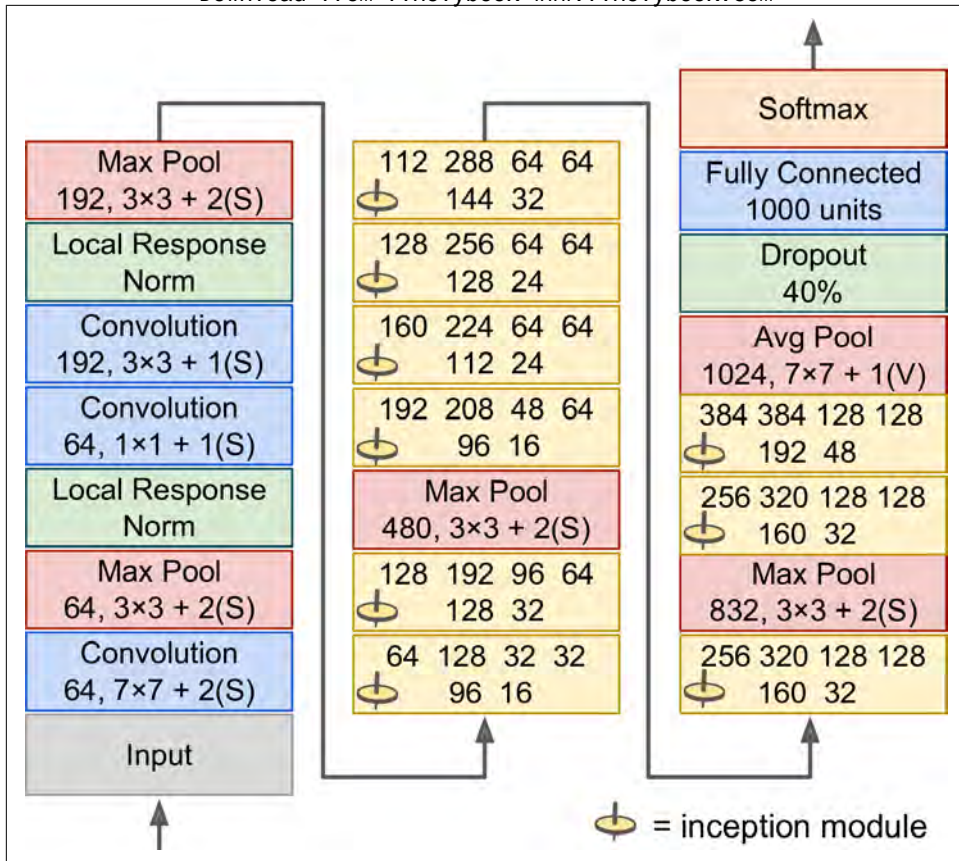


Figure 13-11. GoogLeNet architecture

Let's go through this network:

- The first two layers divide the image's height and width by 4 (so its area is divided by 16), to reduce the computational load.
- Then the local response normalization layer ensures that the previous layers learn a wide variety of features (as discussed earlier).
- Two convolutional layers follow, where the first acts like a *bottleneck layer*. As explained earlier, you can think of this pair as a single smarter convolutional layer.
- Again, a local response normalization layer ensures that the previous layers capture a wide variety of patterns.
- Next a max pooling layer reduces the image height and width by 2, again to speed up computations.

- 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).