

```
def plot_image(image, shape=[28, 28]):
    plt.imshow(image.reshape(shape), cmap="Greys", interpolation="nearest")
    plt.axis("off")

for digit_index in range(n_test_digits):
    plt.subplot(n_test_digits, 2, digit_index * 2 + 1)
    plot_image(X_test[digit_index])
    plt.subplot(n_test_digits, 2, digit_index * 2 + 2)
    plot_image(outputs_val[digit_index])
```

Figure 15-6 shows the resulting images.

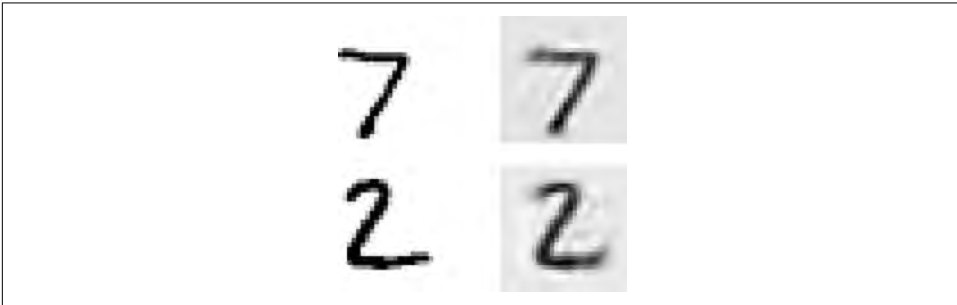


Figure 15-6. Original digits (left) and their reconstructions (right)

Looks close enough. So the autoencoder has properly learned to reproduce its inputs, but has it learned useful features? Let's take a look.

## Visualizing Features

Once your autoencoder has learned some features, you may want to take a look at them. There are various techniques for this. Arguably the simplest technique is to consider each neuron in every hidden layer, and find the training instances that activate it the most. This is especially useful for the top hidden layers since they often capture relatively large features that you can easily spot in a group of training instances that contain them. For example, if a neuron strongly activates when it sees a cat in a picture, it will be pretty obvious that the pictures that activate it the most all contain cats. However, for lower layers, this technique does not work so well, as the features are smaller and more abstract, so it's often hard to understand exactly what the neuron is getting all excited about.

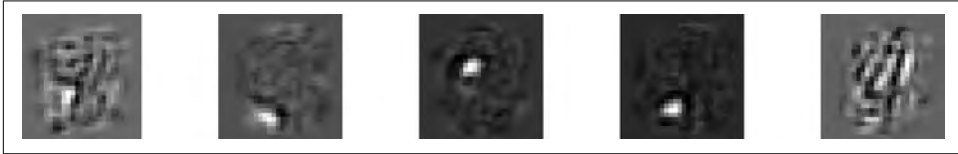
Let's look at another technique. For each neuron in the first hidden layer, you can create an image where a pixel's intensity corresponds to the weight of the connection to the given neuron. For example, the following code plots the features learned by five neurons in the first hidden layer:

```
with tf.Session() as sess:
    [...] # train autoencoder
```

Download from finelybook [www.finelybook.com](http://www.finelybook.com)  
`weights1_val = weights1.eval()`

```
for i in range(5):  
    plt.subplot(1, 5, i + 1)  
    plot_image(weights1_val.T[i])
```

You may get low-level features such as the ones shown in [Figure 15-7](#).



*Figure 15-7. Features learned by five neurons from the first hidden layer*

The first four features seem to correspond to small patches, while the fifth feature seems to look for vertical strokes (note that these features come from the stacked denoising autoencoder that we will discuss later).

Another technique is to feed the autoencoder a random input image, measure the activation of the neuron you are interested in, and then perform backpropagation to tweak the image in such a way that the neuron will activate even more. If you iterate several times (performing gradient ascent), the image will gradually turn into the most exciting image (for the neuron). This is a useful technique to visualize the kinds of inputs that a neuron is looking for.

Finally, if you are using an autoencoder to perform unsupervised pretraining—for example, for a classification task—a simple way to verify that the features learned by the autoencoder are useful is to measure the performance of the classifier.

## Unsupervised Pretraining Using Stacked Autoencoders

As we discussed in [Chapter 11](#), if you are tackling a complex supervised task but you do not have a lot of labeled training data, one solution is to find a neural network that performs a similar task, and then reuse its lower layers. This makes it possible to train a high-performance model using only little training data because your neural network won't have to learn all the low-level features; it will just reuse the feature detectors learned by the existing net.

Similarly, if you have a large dataset but most of it is unlabeled, you can first train a stacked autoencoder using all the data, then reuse the lower layers to create a neural network for your actual task, and train it using the labeled data. For example, [Figure 15-8](#) shows how to use a stacked autoencoder to perform unsupervised pretraining for a classification neural network. The stacked autoencoder itself is typically trained one autoencoder at a time, as discussed earlier. When training the classifier, if