Generating Images with Variational Autoencoders

The applications we've described thus far are all supervised learning problems. There are well-defined inputs and outputs, and the task remains (using a convolutional network) to learn a sophisticated function mapping input to output. Are there unsupervised learning problems that can be solved with convolutional networks? Recall that unsupervised learning requires "understanding" the structure of input datapoints. For image modeling, a good measure of understanding the structure of input images is being able to "sample" new images that come from the input distribution.

What does "sampling" an image mean? To explain, let's suppose we have a dataset of dog images. Sampling a new dog image requires the generation of a new image of a dog that *is not in the training data*! The idea is that we would like a picture of a dog that could have reasonably been included with the training data, but was not. How could we solve this task with convolutional networks?

Perhaps we could train a model to take in word labels like "dog" and predict dog images. We might possibly be able to train a supervised model to solve this prediction problem, but the issue remains that our model could generate only one dog picture given the input label "dog." Suppose now that we could attach a random tag to each dog—say "dog3422" or "dog9879." Then all we'd need to do to get a new dog image would be to attach a new random tag, say "dog2221," to get out a new picture of a dog.

Variational autoencoders formalize these intuitions. Variational autoencoders consist of two convolutional networks: the encoder and decoder network. The encoder network is used to transform an image into a flat "embedded" vector. The decoder network is responsible for transforming the embedded vector into images. Noise is added to ensure that different images can be sampled by the decoder. Figure 6-13 illustrates a variational autoencoder.

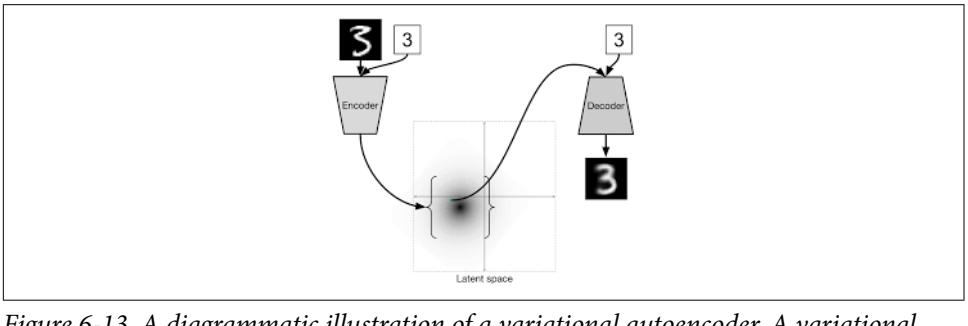

Figure 6-13. A diagrammatic illustration of a variational autoencoder. A variational autoencoder consists of two convolutional networks, the encoder and decoder.

There are more details involved in an actual implementation, but variational autoencoders are capable of sampling images. However, naive variational encoders seem to generate blurry image samples, as Figure 6-14 demonstrates. This blurriness may be

because the L^2 loss doesn't penalize image blurriness sharply (recall our discussion about L^2 not penalizing small deviations). To generate crisp image samples, we will need other architectures.

Figure 6-14. Images sampled from a variational autoencoder trained on a dataset of faces. Note that sampled images are quite blurry.

Adversarial models

The L2 loss sharply penalizes large local deviations, but doesn't severely penalize many small local deviations, causing blurriness. How could we design an alternate loss function that penalizes blurriness in images more sharply? It turns out that it's quite challenging to write down a loss function that does the trick. While our eyes can quickly spot blurriness, our analytical tools aren't quite so fast to capture the problem.

What if we could somehow "learn" a loss function? This idea sounds a little nonsensical at first; where would we get training data? But it turns out that there's a clever idea that makes it feasible.

Suppose we could train a separate network that learns the loss. Let's call this network the discriminator. Let's call the network that makes the images the generator. The generator can be set to duel against the discriminator until the generator is capable of producing images that are photorealistic. This form of architecture is commonly called a generative adversarial network, or GAN.

Faces generated by a GAN (Figure 6-15) are considerably crisper than those generated by the naive variational autoencoder (Figure 6-14)! There are a number of other promising results that have been achieved by GANs. The CycleGAN, for example, appears capable of learning complex image transformations such as transmuting horses into zebras and vice versa. Figure 6-16 shows some CycleGAN image transformations.

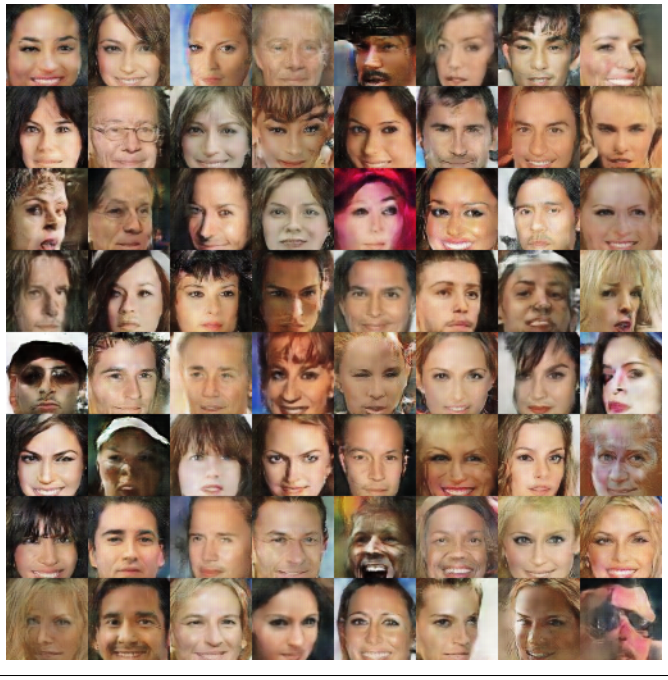

Figure 6-15. Images sampled from a generative adversarial network (GAN) trained on a dataset of faces. Note that sampled images are less blurry than those from the variational autoencoder.

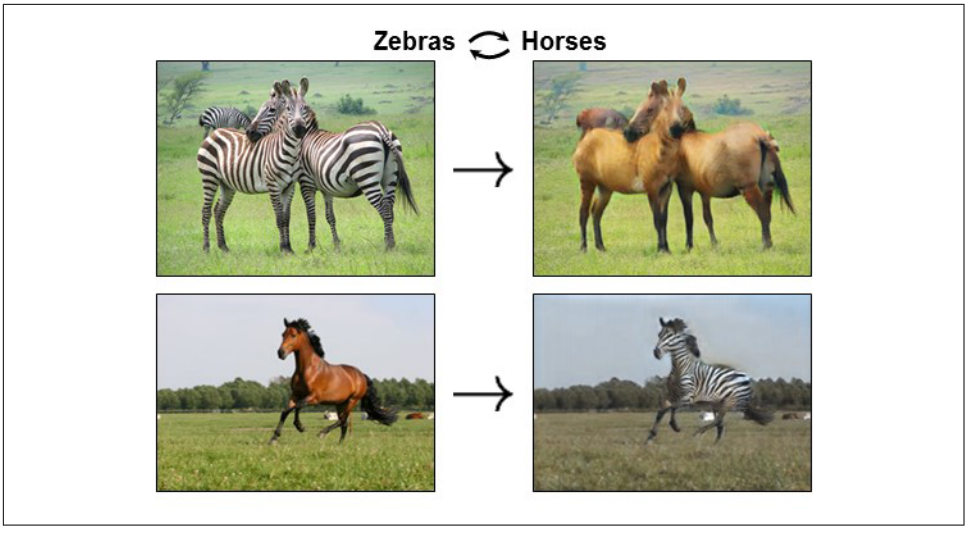

Figure 6-16. The CycleGAN is capable of performing complex image transformations, such as transforming images of horses into those of zebras (and vice versa).

Unfortunately, generative adversarial networks are still challenging to train in practice. Making generators and discriminators learn reasonable functions requires a deep bag of tricks. As a result, while there have been many exciting GAN demonstrations, GANs have not yet matured into a state where they can be widely deployed in industrial applications.

Training a Convolutional Network in TensorFlow

In this section we consider a code sample for training a simple convolutional neural network. In particular, our code sample will demonstrate how to train a LeNet-5 convolutional architecture on the MNIST dataset using TensorFlow. As always, we recommend that you follow along by running the full code sample from the GitHub repo associated with the book.

The MNIST Dataset

The MNIST dataset consists of images of handwritten digits. The machine learning challenge associated with MNIST consists of creating a model trained on the training set of digits that generalizes to the validation set. Figure 6-17 shows some images drawn from the MNIST dataset.