

Data Augmentation

One last regularization technique, data augmentation, consists of generating new training instances from existing ones, artificially boosting the size of the training set. This will reduce overfitting, making this a regularization technique. The trick is to generate realistic training instances; ideally, a human should not be able to tell which instances were generated and which ones were not. Moreover, simply adding white noise will not help; the modifications you apply should be learnable (white noise is not).

For example, if your model is meant to classify pictures of mushrooms, you can slightly shift, rotate, and resize every picture in the training set by various amounts and add the resulting pictures to the training set (see [Figure 11-10](#)). This forces the model to be more tolerant to the position, orientation, and size of the mushrooms in the picture. If you want the model to be more tolerant to lighting conditions, you can similarly generate many images with various contrasts. Assuming the mushrooms are symmetrical, you can also flip the pictures horizontally. By combining these transformations you can greatly increase the size of your training set.

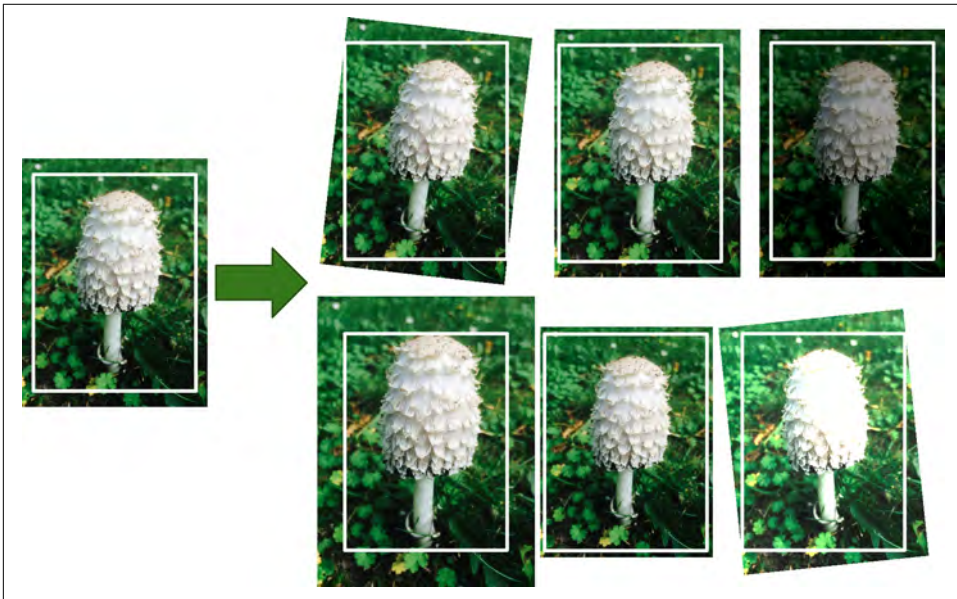


Figure 11-10. Generating new training instances from existing ones

It is often preferable to generate training instances on the fly during training rather than wasting storage space and network bandwidth. TensorFlow offers several image manipulation operations such as transposing (shifting), rotating, resizing, flipping, and cropping, as well as adjusting the brightness, contrast, saturation, and hue (see

Download from [finelybook www.finelybook.com](http://finelybook.com)
the API documentation for more details). This makes it easy to implement data augmentation for image datasets.



Another powerful technique to train very deep neural networks is to add *skip connections* (a skip connection is when you add the input of a layer to the output of a higher layer). We will explore this idea in [Chapter 13](#) when we talk about deep residual networks.

Practical Guidelines

In this chapter, we have covered a wide range of techniques and you may be wondering which ones you should use. The configuration in [Table 11-2](#) will work fine in most cases.

Table 11-2. Default DNN configuration

Initialization	He initialization
Activation function	ELU
Normalization	Batch Normalization
Regularization	Dropout
Optimizer	Adam
Learning rate schedule	None

Of course, you should try to reuse parts of a pretrained neural network if you can find one that solves a similar problem.

This default configuration may need to be tweaked:

- If you can't find a good learning rate (convergence was too slow, so you increased the training rate, and now convergence is fast but the network's accuracy is sub-optimal), then you can try adding a learning schedule such as exponential decay.
- If your training set is a bit too small, you can implement data augmentation.
- If you need a sparse model, you can add some ℓ_1 regularization to the mix (and optionally zero out the tiny weights after training). If you need an even sparser model, you can try using FTRL instead of Adam optimization, along with ℓ_1 regularization.
- If you need a lightning-fast model at runtime, you may want to drop Batch Normalization, and possibly replace the ELU activation function with the leaky ReLU. Having a sparse model will also help.

With these guidelines, you are now ready to train very deep nets—well, if you are very patient, that is! If you use a single machine, you may have to wait for days or