implemented using a convolution in some cases, but in the general case requires a third operation to be implemented. Care must be taken to coordinate this transpose operation with the forward propagation. The size of the output that the transpose operation should return depends on the zero padding policy and stride of the forward propagation operation, as well as the size of the forward propagation's output map. In some cases, multiple sizes of input to forward propagation can result in the same size of output map, so the transpose operation must be explicitly told what the size of the original input was.

These three operations—convolution, backprop from output to weights, and backprop from output to inputs—are sufficient to compute all of the gradients needed to train any depth of feedforward convolutional network, as well as to train convolutional networks with reconstruction functions based on the transpose of convolution. See Goodfellow (2010) for a full derivation of the equations in the fully general multi-dimensional, multi-example case. To give a sense of how these equations work, we present the two dimensional, single example version here.

Suppose we want to train a convolutional network that incorporates strided convolution of kernel stack $\mathbf{K}$ applied to multi-channel image $\mathbf{V}$ with stride s as defined by $c(\mathbf{K}, \mathbf{V}, s)$ as in equation 9.8. Suppose we want to minimize some loss function $J(\mathbf{V}, \mathbf{K})$. During forward propagation, we will need to use c itself to output $\mathbf{Z}$, which is then propagated through the rest of the network and used to compute the cost function J. During back-propagation, we will receive a tensor $\mathbf{G}$ such that $G_{i,j,k} = \frac{\partial}{\partial Z_{i,j,k}} J(\mathbf{V}, \mathbf{K})$.

To train the network, we need to compute the derivatives with respect to the weights in the kernel. To do so, we can use a function

$$g(\mathbf{G}, \mathbf{V}, s)_{i,j,k,l} = \frac{\partial}{\partial K_{i,j,k,l}} J(\mathbf{V}, \mathbf{K}) = \sum_{m,n} G_{i,m,n} V_{j,(m-1)\times s+k,(n-1)\times s+l}. \tag{9.11}$$

If this layer is not the bottom layer of the network, we will need to compute the gradient with respect to $\mathbf{V}$ in order to back-propagate the error farther down. To do so, we can use a function

$$h(\mathbf{K}, \mathbf{G}, s)_{i,j,k} = \frac{\partial}{\partial V_{i,j,k}} J(\mathbf{V}, \mathbf{K}) \tag{9.12}$$

$$= \sum_{\substack{l,m \\ \text{s.t.} \\ (l-1)\times s+m=j}} \sum_{\substack{n,p \\ \text{s.t.} \\ (n-1)\times s+p=k}} \sum_{q} K_{q,i,m,p} G_{q,l,n}. \tag{9.13}$$

Autoencoder networks, described in chapter 14, are feedforward networks trained to copy their input to their output. A simple example is the PCA algorithm,