Understanding Deep Learning

Chapter 9: Convolutional Networks

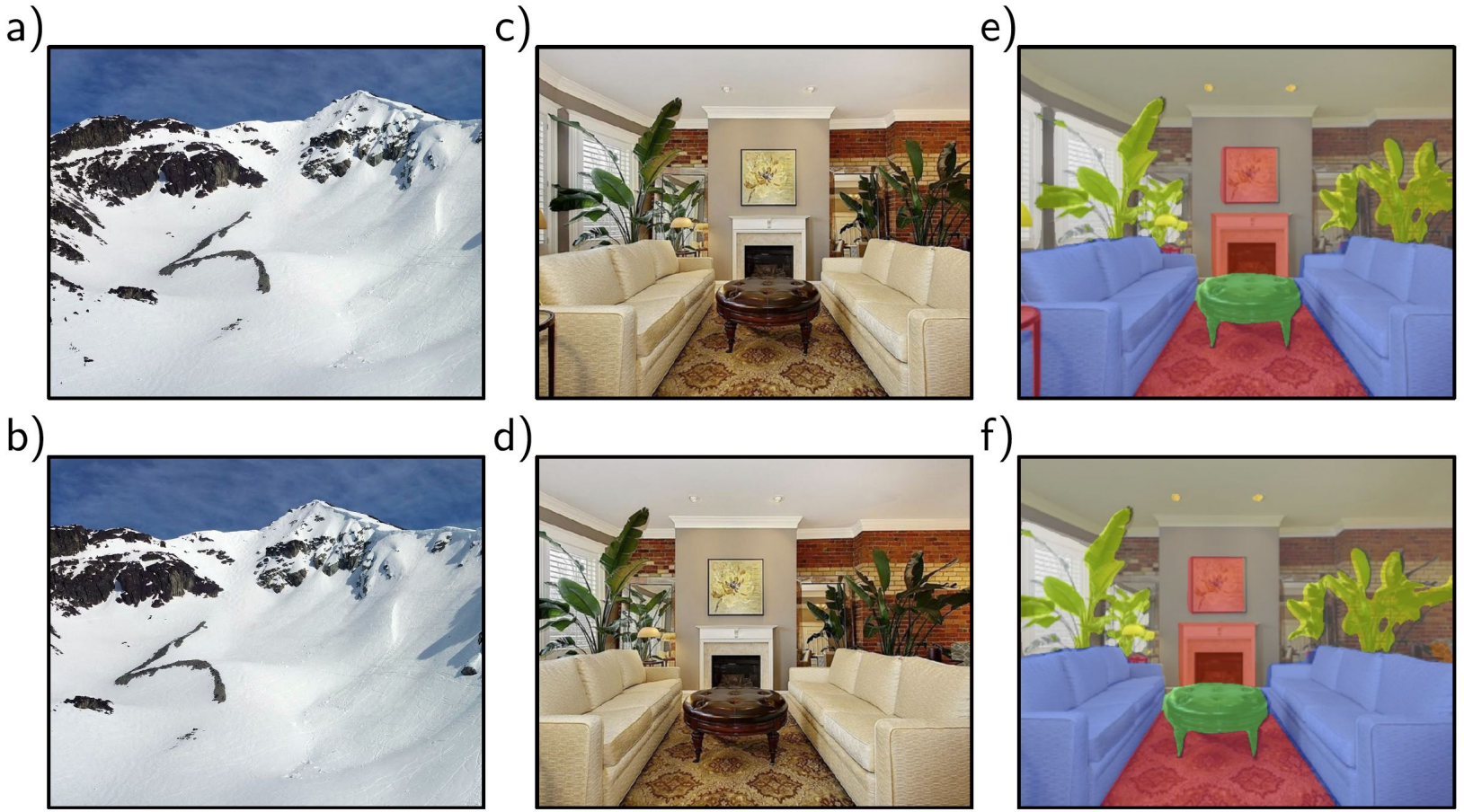


Figure 10.1 Invariance and equivariance for translation. a–b) In image classification, the goal is to categorize both images as "mountain" regardless of the horizontal shift that has occurred. In other words, we require the network prediction to be invariant to translation. c,e) The goal of semantic segmentation is to associate a label with each pixel. d,f) When the input image is translated, we want the output (colored overlay) to translate in the same way. In other words, we require the output to be equivariant with respect to translation. Panels c–f) adapted from Bousselham et al. (2021).

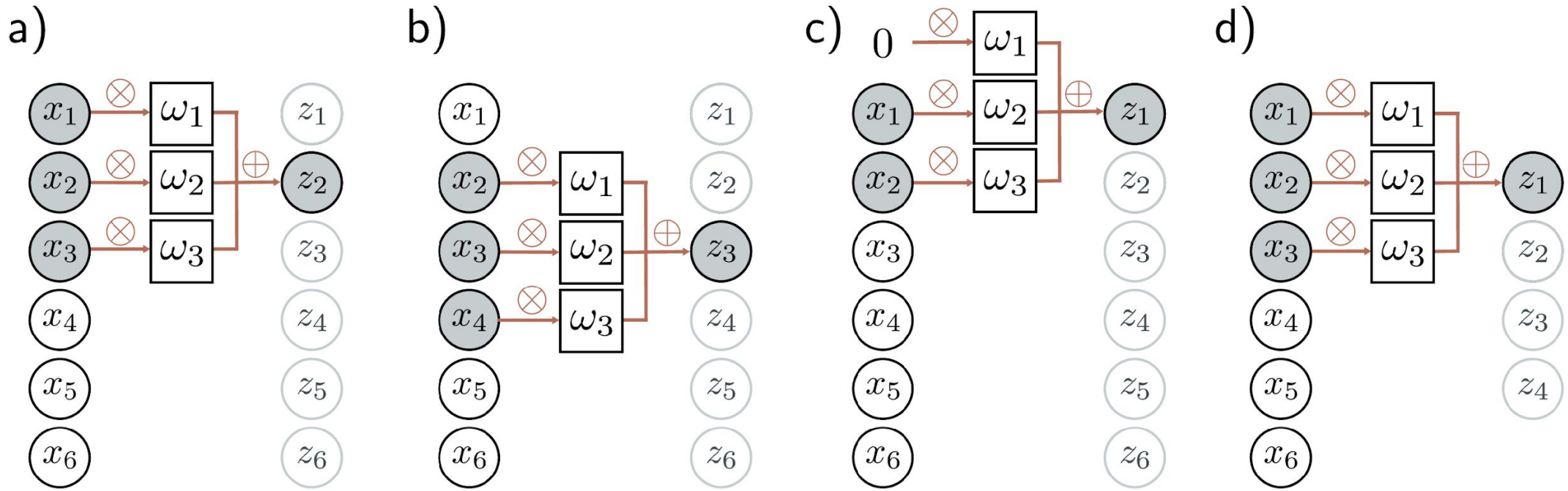


Figure 10.2 1D convolution with kernel size three. Each output z_i is a weighted sum of the nearest three inputs x_{i-1}, x_i, and x_{i+1}, where the weights are $\boldsymbol{\omega} = [\omega_1, \omega_2, \omega_3]$. a) Output z_2 is computed as $z_2 = \omega_1 x_1 + \omega_2 x_2 + \omega_3 x_3$. b) Output z_3 is computed as $z_3 = \omega_1 x_2 + \omega_2 x_3 + \omega_3 x_4$. c) At position z_1, the kernel extends beyond the first input x_1. This can be handled by zero padding, in which we assume values outside the input are zero. The final output is treated similarly. d) Alternatively, we could only compute outputs where the kernel fits within the input range ("valid" convolution); now, the output will be smaller than the input.

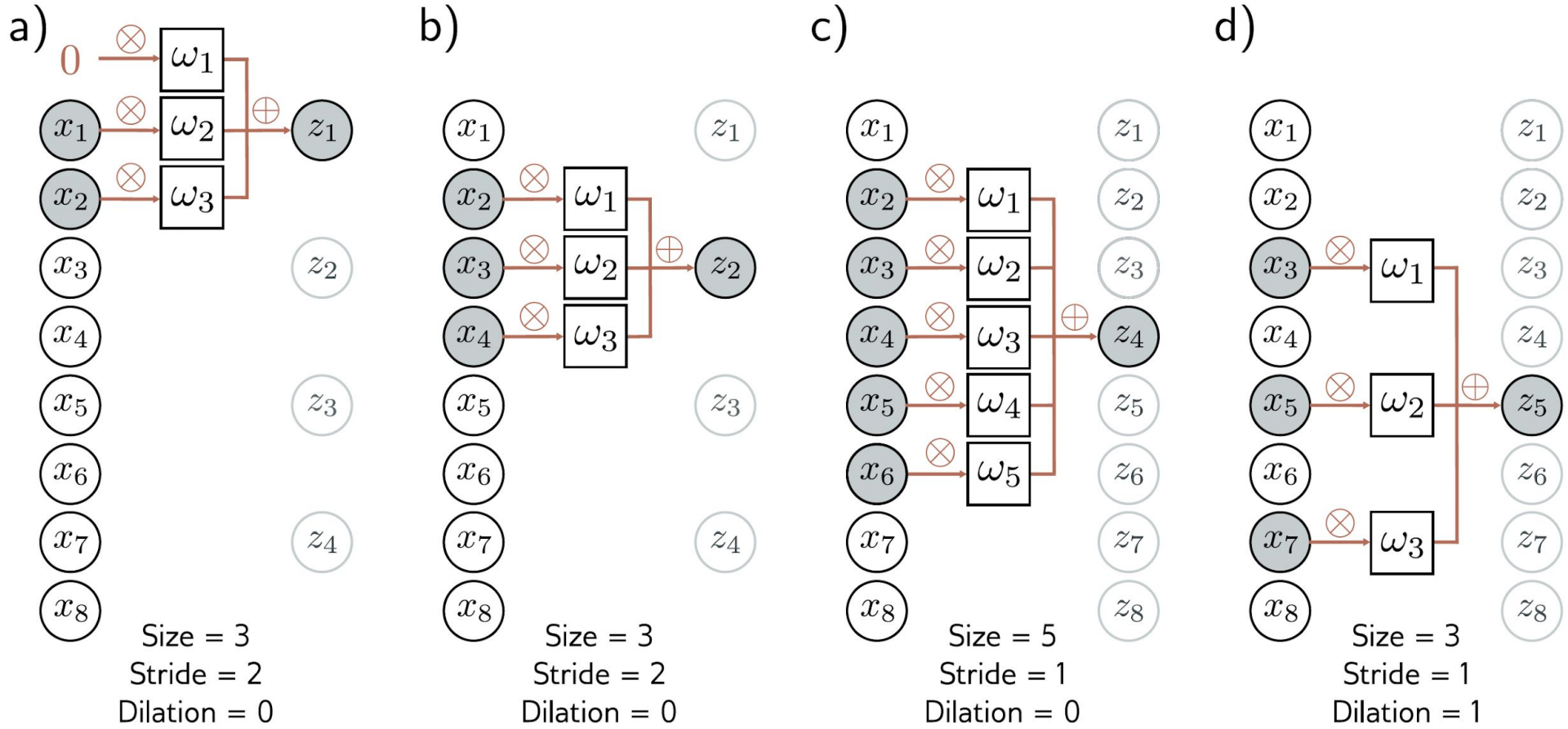


Figure 10.3 Stride, kernel size, and dilation. a) With a stride of two, we evaluate the kernel at every other position, so the first output z_1 is computed from a weighted sum centered at x_1, and b) the second output z_2 is computed from a weighted sum centered at x_3 and so on. c) The kernel size can also be changed. With a kernel size of five, we take a weighted sum of the nearest five inputs. d) In dilated or atrous convolution, we intersperse zeros in the weight vector to allow us to combine information over a large area using fewer weights.

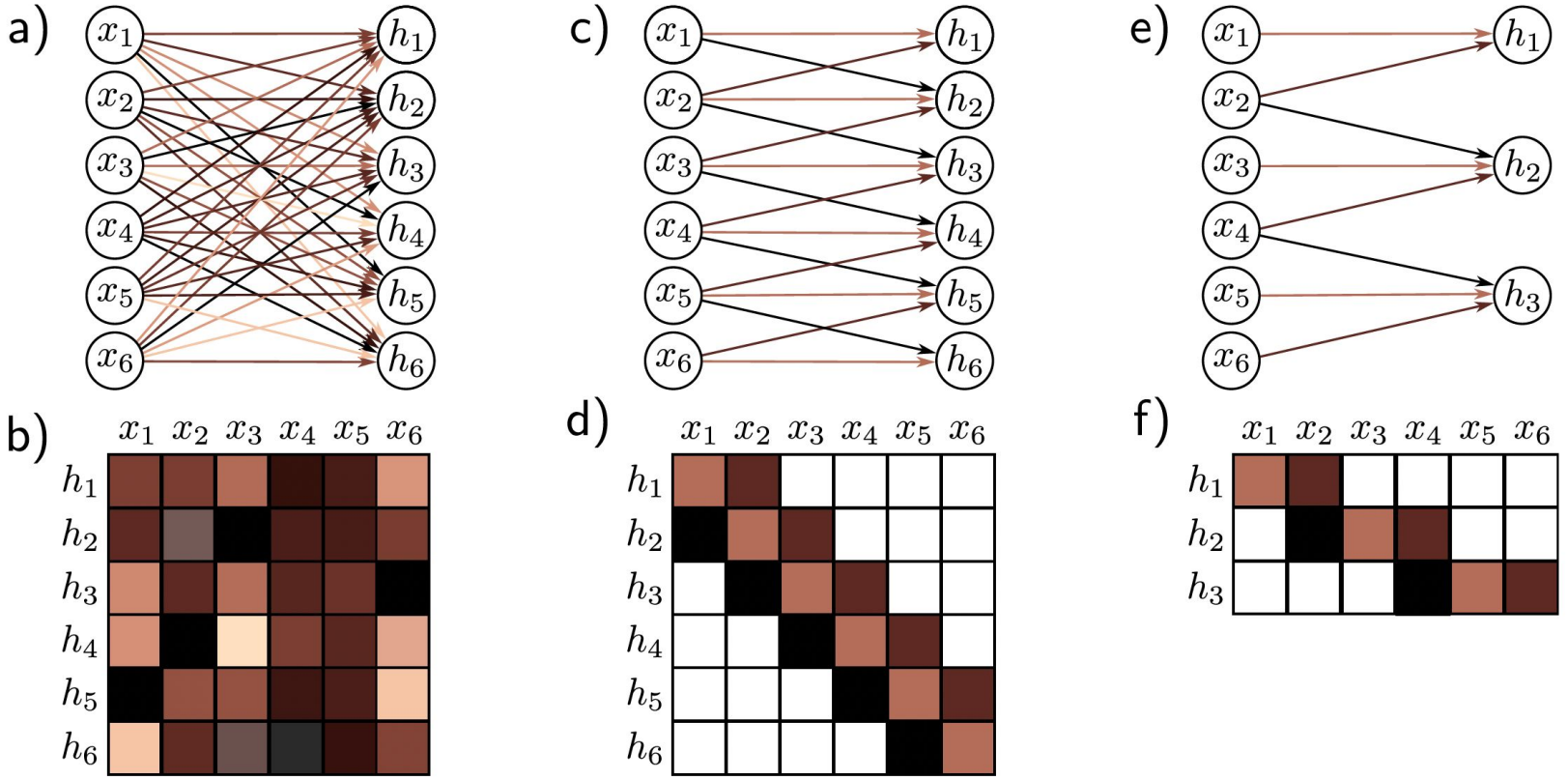


Figure 10.4 Fully connected vs. convolutional layers. a) A fully connected layer has a weight connecting each input x to each hidden unit h (colored arrows) and a bias for each hidden unit (not shown). b) Hence, the associated weight matrix $\mathbf{\Omega}$ contains 36 weights relating the six inputs to the six hidden units. c) A convolutional layer with kernel size three computes each hidden unit as the same weighted sum of the three neighboring inputs (arrows) plus a bias (not shown). d) The weight matrix is a special case of the fully connected matrix where many weights are zero and others are repeated (same colors indicate same value, white indicates zero weight). e) A convolutional layer with kernel size three and stride two computes a weighted sum at every other position. f) This is also a special case of a fully connected network with a different sparse weight structure.

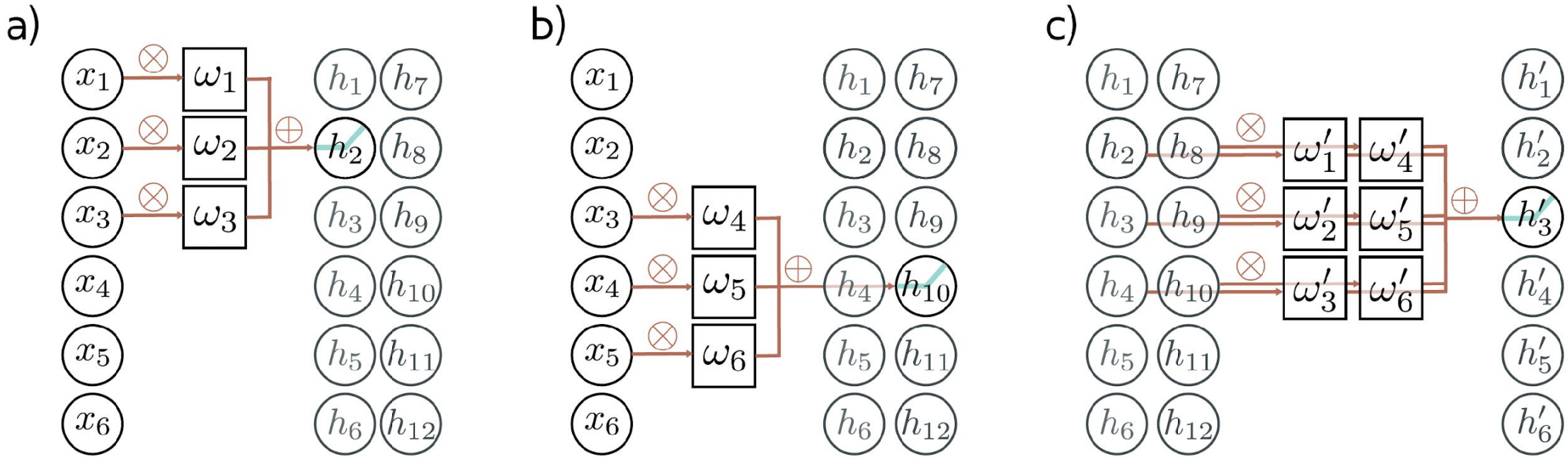


Figure 10.5 Channels. Typically, multiple convolutions are applied to the input $\mathbf{x}$ and stored in channels. a) A convolution is applied to create hidden units h_1 to h_6, which form the first channel. b) A second convolution operation is applied to create hidden units h_7 to h_{12}, which form the second channel. The channels are stored in a 2D array $\mathbf{H}_1$ that contains all the hidden units in the first hidden layer. c) If we add a further convolutional layer, there are now two channels at each input position. Here, the 1D convolution defines a weighted sum over both input channels at the three closest positions to create each new output channel.

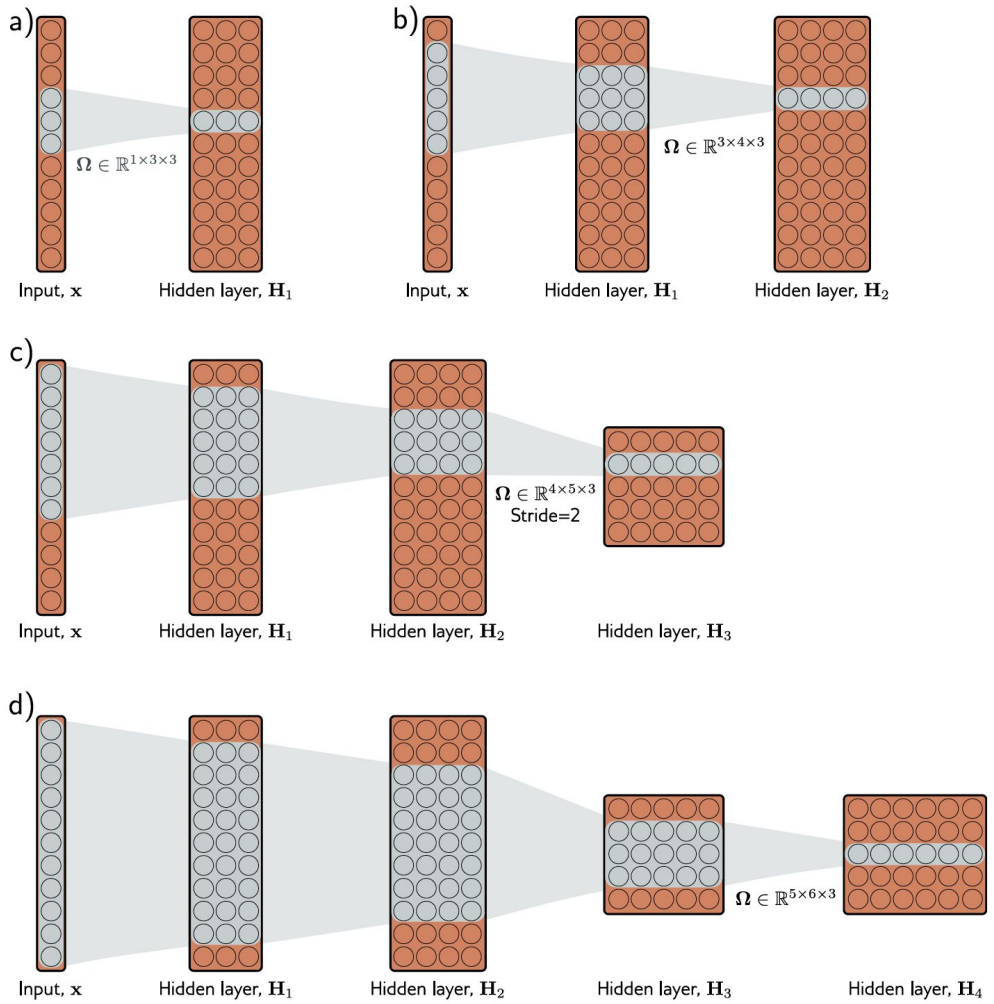


Figure 10.6 Receptive fields for network with kernel width of three. a) An input with eleven dimensions feeds into a hidden layer with three channels and convolution kernel of size three. The pre-activations of the three highlighted hidden units in the first hidden layer $\mathbf{H}_1$ are different weighted sums of the nearest three inputs, so the receptive field in $\mathbf{H}_1$ has size three. b) The pre-activations of the four highlighted hidden units in layer $\mathbf{H}_2$ each take a weighted sum of the three channels in layer $\mathbf{H}_1$ at each of the three nearest positions. Each hidden unit in layer $\mathbf{H}_1$ weights the nearest three input positions. Hence, hidden units in $\mathbf{H}_2$ have a receptive field size of five. c) The hidden units in the third layer (kernel size three, stride two) increases the receptive field size to seven. d) By the time we add a fourth layer, the receptive field of the hidden units at position three have a receptive field that covers the entire input.

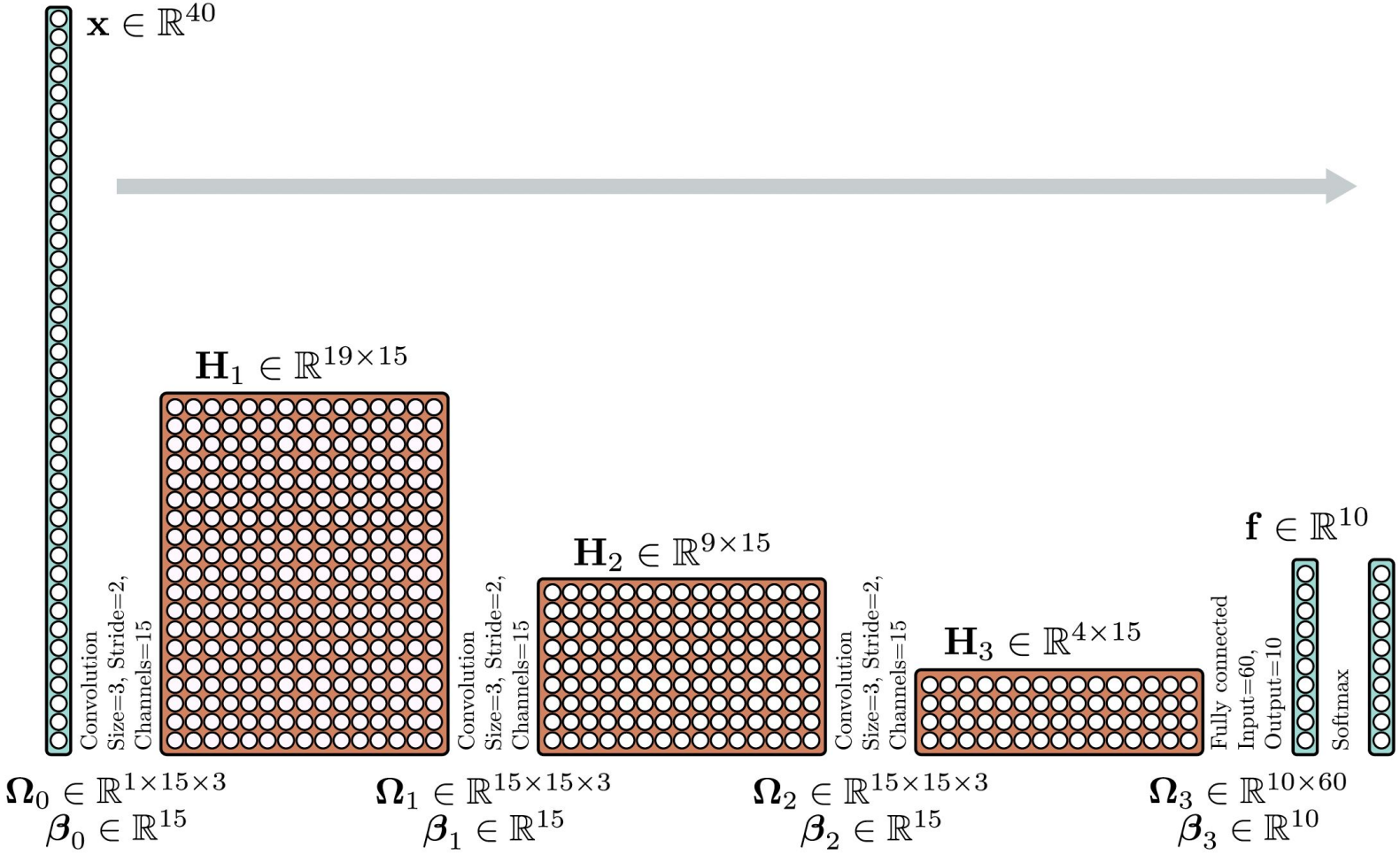


Figure 10.7 Convolutional network for classifying MNIST-1D data (see figure 8.1). The MNIST-1D input has dimension $D_i = 40$. The first convolutional layer has fifteen channels, kernel size three, stride two, and only retains "valid" positions to make a representation with nineteen positions and fifteen channels. The following two convolutional layers have the same settings, gradually reducing the representation size. Finally, a fully connected layer takes all sixty hidden units from the third hidden layer. It outputs ten activations that are subsequently passed through a softmax layer to produce the ten class probabilities.

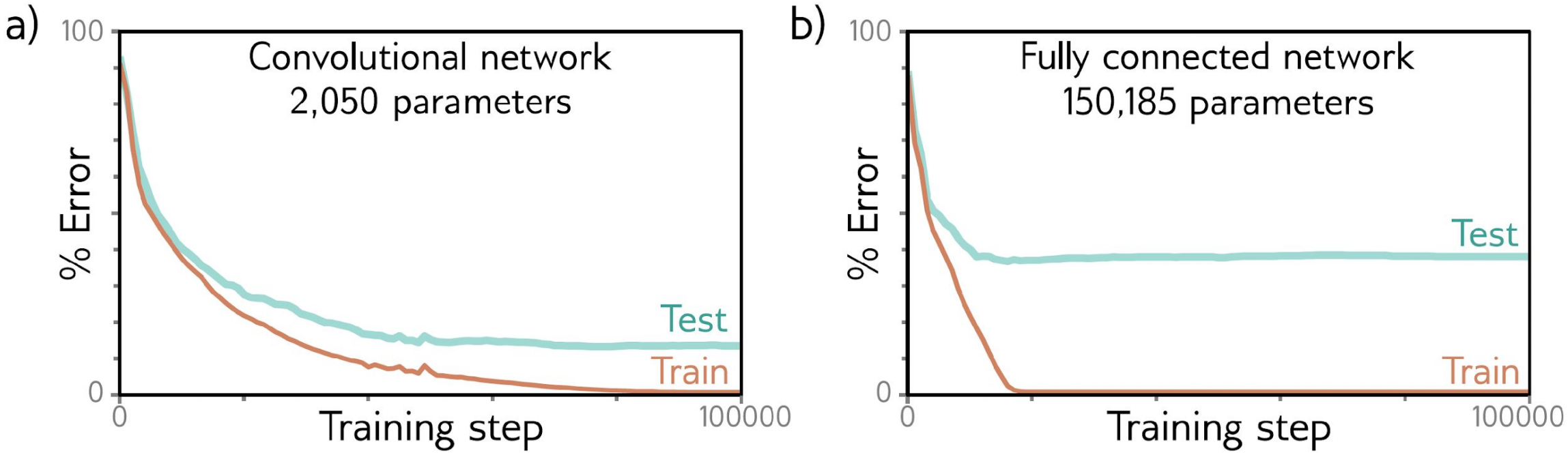


Figure 10.8 MNIST-1D results. a) The convolutional network from figure 10.7 eventually fits the training data perfectly and has ∼17% test error. b) A fully connected network with the same number of hidden layers and the number of hidden units in each learns the training data faster but fails to generalize well with ∼40% test error. The latter model can reproduce the convolutional model but fails to do so. The convolutional structure restricts the possible mappings to those that process every position similarly, and this restriction improves performance.

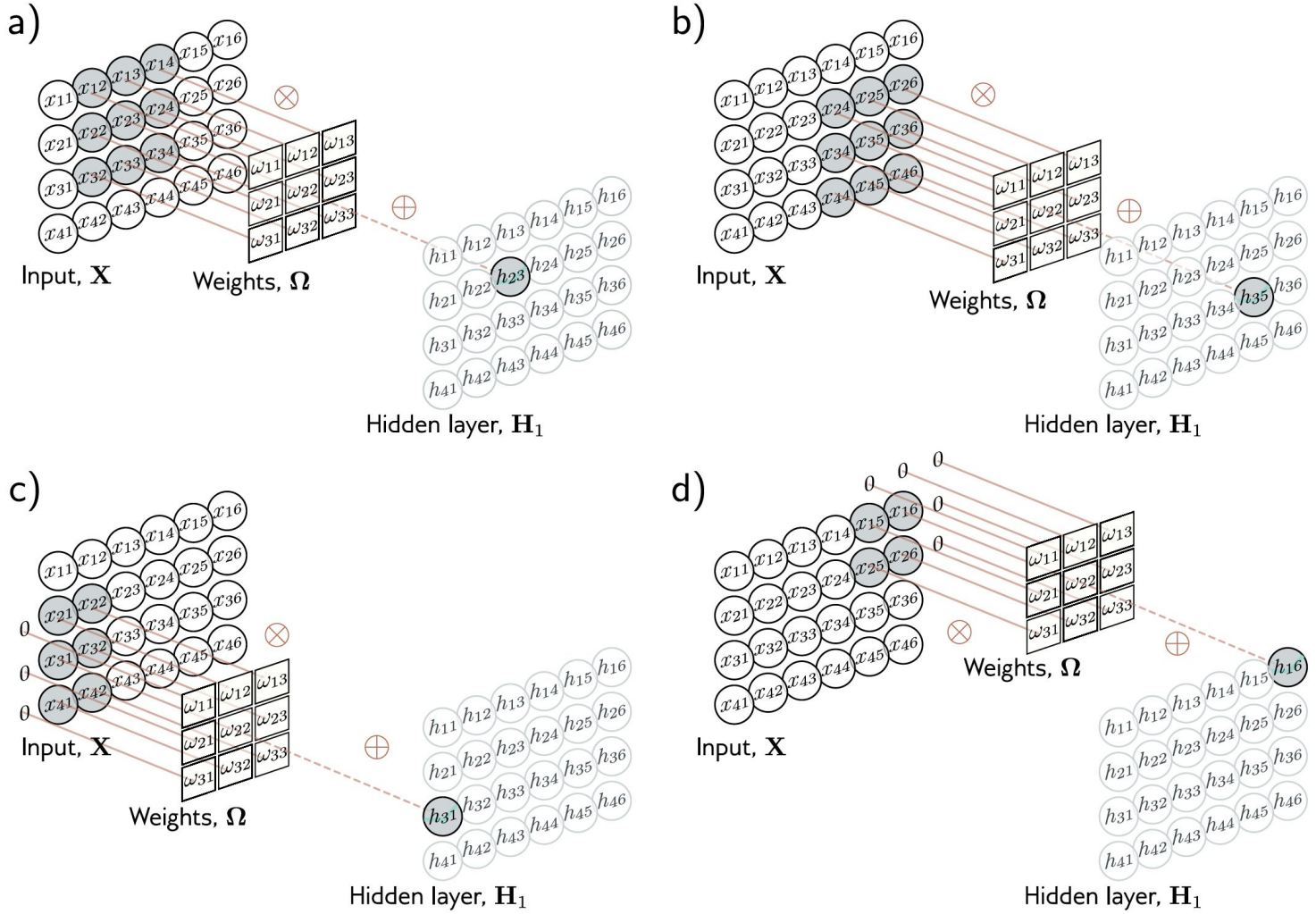


Figure 10.9 2D convolutional layer. Each output h_{ij} computes a weighted sum of the 3×3 nearest inputs, adds a bias, and passes the result through an activation function. a) Here, the output h_{23} (shaded output) is a weighted sum of the nine positions from x_{12} to x_{34} (shaded inputs). b) Different outputs are computed by translating the kernel across the image grid in two dimensions. c–d) With zero padding, positions beyond the image's edge are considered to be zero.

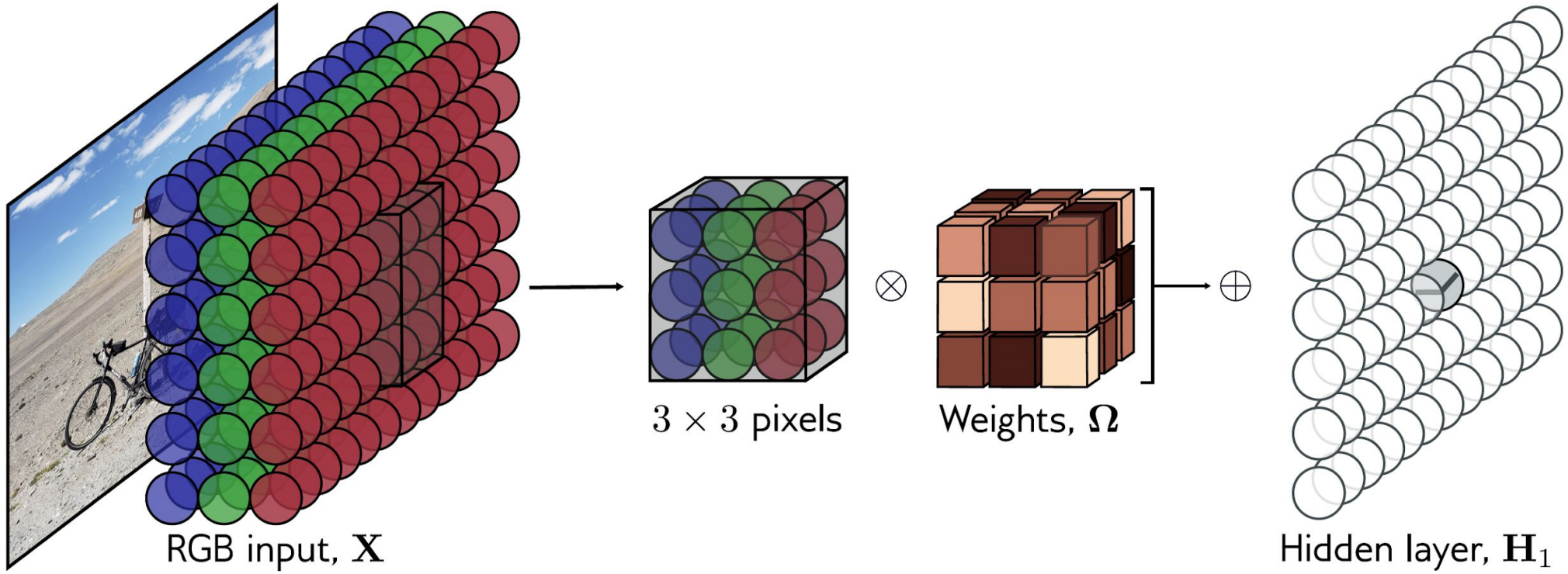


Figure 10.10 2D convolution applied to an image. The image is treated as a 2D input with three channels corresponding to the red, green, and blue components. With a 3×3 kernel, each pre-activation in the first hidden layer is computed by pointwise multiplying the 3×3×3 kernel weights with the 3×3 RGB image patch centered at the same position, summing, and adding the bias. To calculate all the pre-activations in the hidden layer, we "slide" the kernel over the image in both horizontal and vertical directions. The output is a 2D layer of hidden units. To create multiple output channels, we would repeat this process with multiple kernels, resulting in a 3D tensor of hidden units at hidden layer $\mathbf{H}_1$.

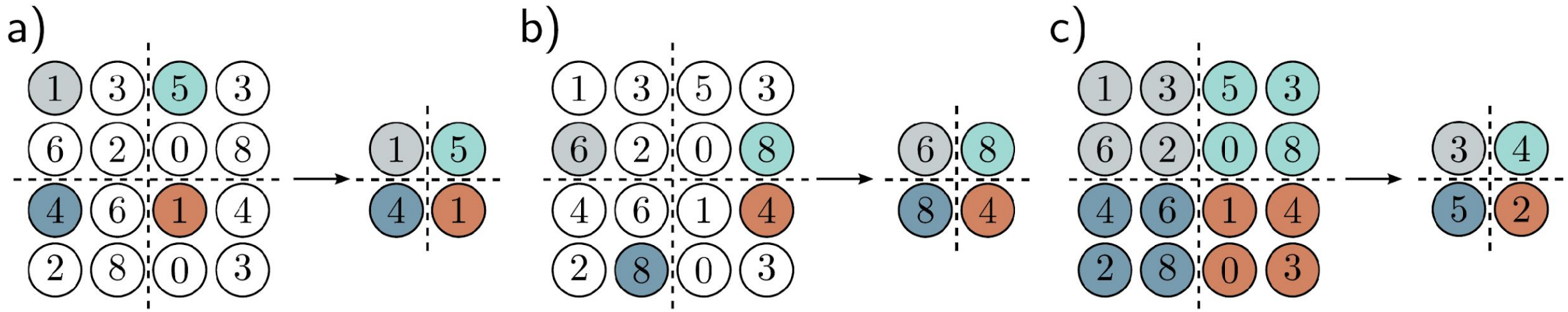


Figure 10.11 Methods for scaling down representation size (downsampling). a) Sub-sampling. The original 4×4 representation (left) is reduced to size 2×2 (right) by retaining every other input. Colors on the left indicate which inputs contribute to the outputs on the right. This is effectively what happens with a kernel of stride two, except that the intermediate values are never computed. b) Max pooling. Each output comprises the maximum value of the corresponding 2×2 block. c) Mean pooling. Each output is the mean of the values in the 2×2 block.

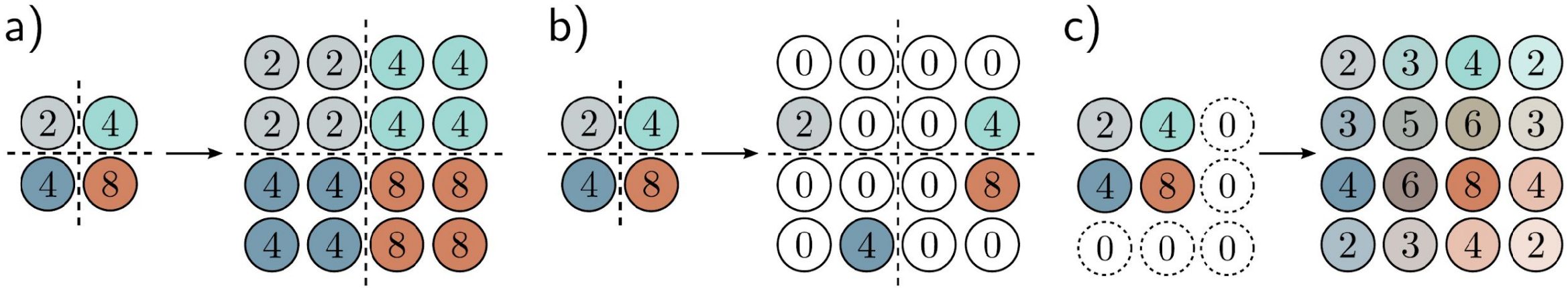


Figure 10.12 Methods for scaling up representation size (upsampling). a) The simplest way to double the size of a 2D layer is to duplicate each input four times. b) In networks where we have previously used a max pooling operation (figure 10.11b), we can redistribute the values to the same positions they originally came from (i.e., where the maxima were). This is known as max unpooling. c) A third option is bilinear interpolation between the input values.

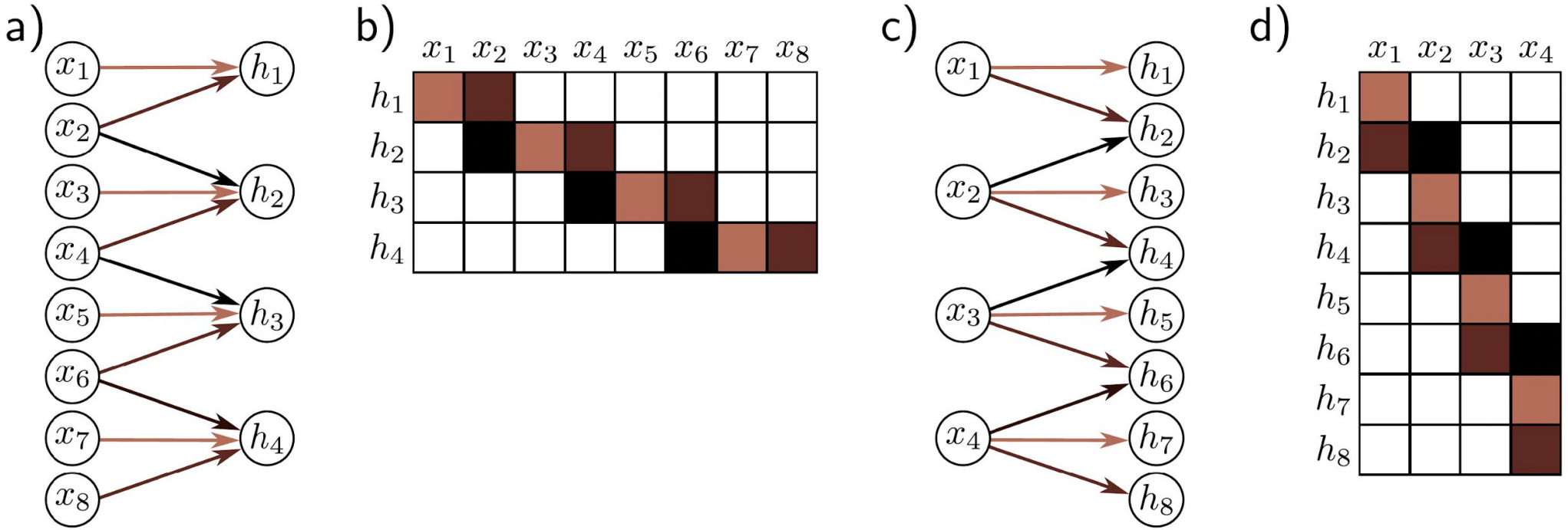


Figure 10.13 Transposed convolution in 1D. a) Downsampling with kernel size three, stride two, and zero padding. Each output is a weighted sum of three inputs (arrows indicate weights). b) This can be expressed by a weight matrix (same color indicates shared weight). c) In transposed convolution, each input contributes three values to the output layer, which has twice as many outputs as inputs. d) The associated weight matrix is the transpose of that in panel (b).

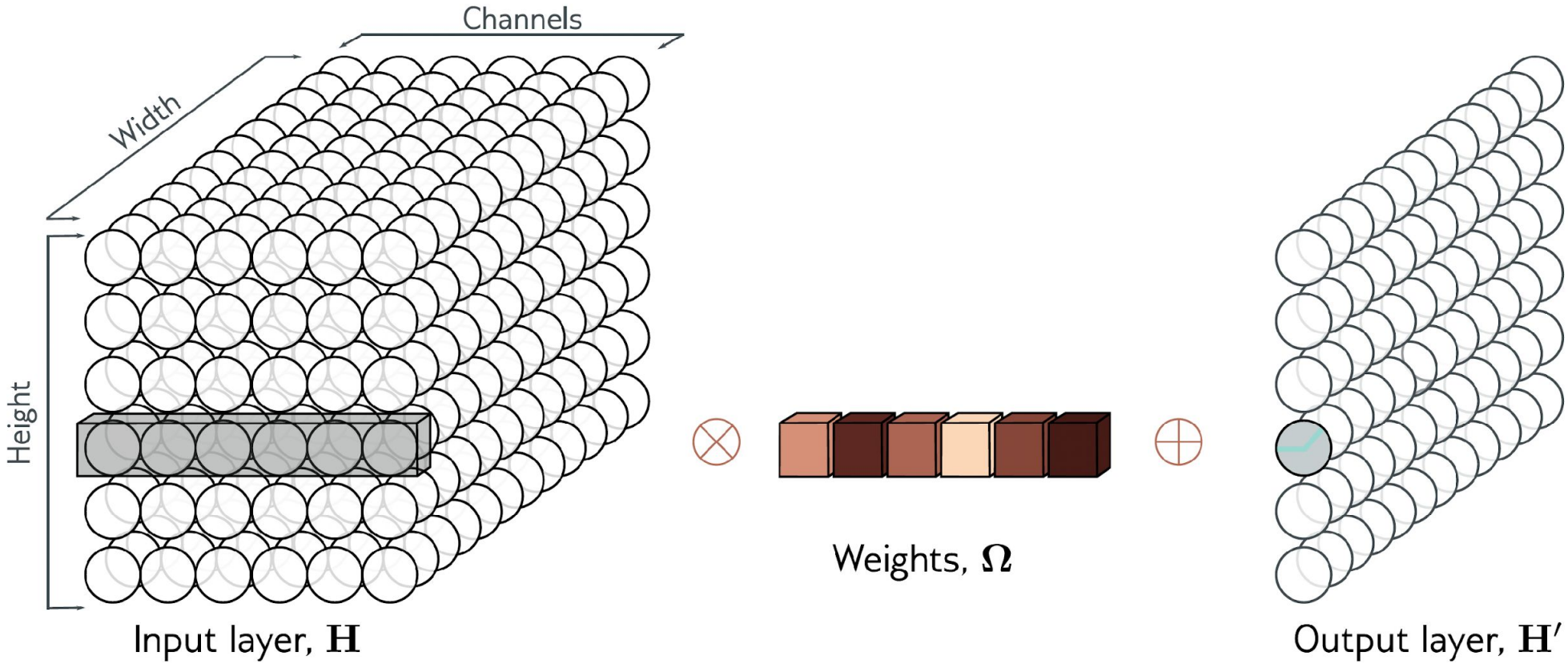


Figure 10.14 1×1 convolution. To change the number of channels without spatial pooling, we apply a 1×1 kernel. Each output channel is computed by taking a weighted sum of all of the channels at the same position, adding a bias, and passing through an activation function. Multiple output channels are created by repeating this operation with different weights and biases.

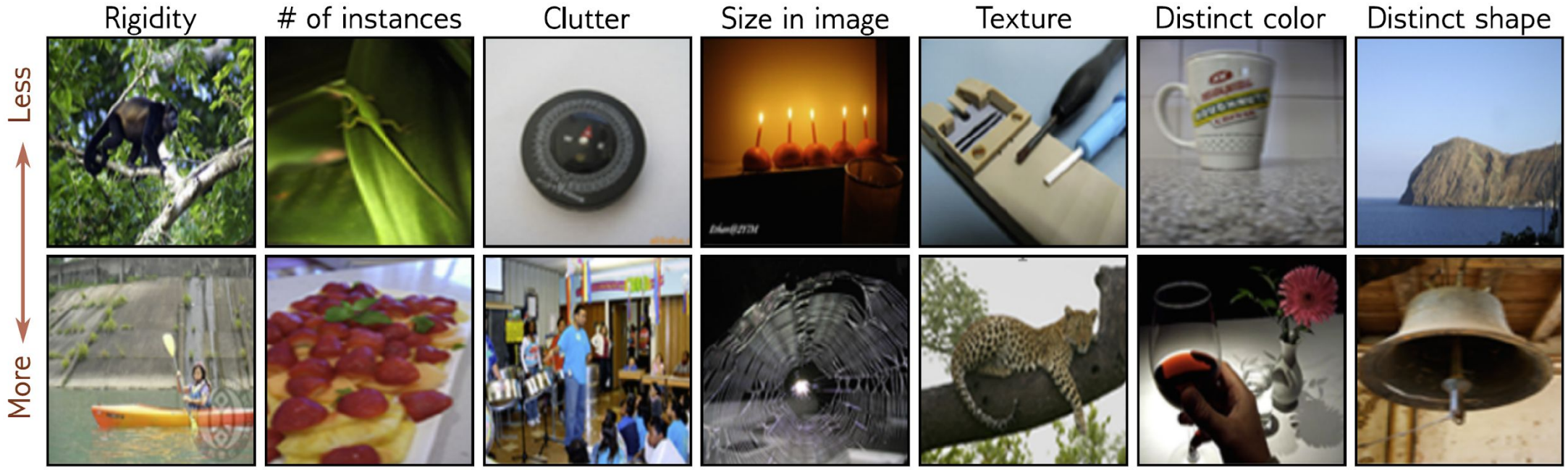
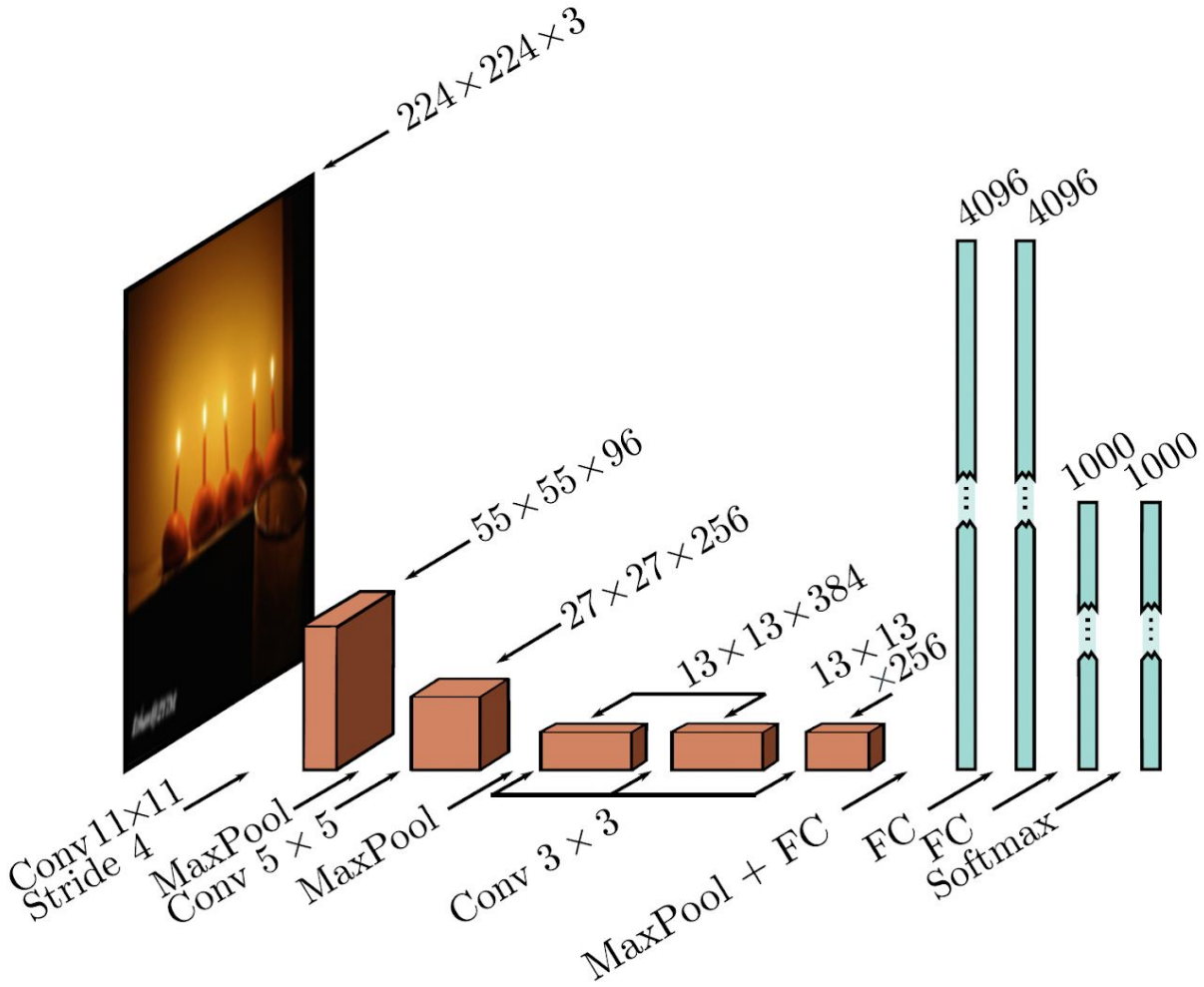


Figure 10.15 Example ImageNet classification images. The model aims to assign an input image to one of 1000 classes. This task is challenging because the images vary widely along different attributes (columns). These include rigidity (monkey < canoe), number of instances in image (lizard < strawberry), clutter (compass < steel drum), size (candle < spiderweb), texture (screwdriver < leopard), distinctiveness of color (mug < red wine), and distinctiveness of shape (headland < bell). Adapted from Russakovsky et al. (2015).

Figure 10.16 AlexNet (Krizhevsky et al., 2012). The network maps a 224×224 color image to a 1000-dimensional vector representing class probabilities. The network first convolves with 11×11 kernels and stride 4 to create 96 channels. It decreases the resolution again using a max pool operation and applies a 5×5 convolutional layer. Another max pooling layer follows, and three 3×3 convolutional layers are applied. After a final max pooling operation, the result is vectorized and passed through three fully connected (FC) layers and finally the softmax layer.



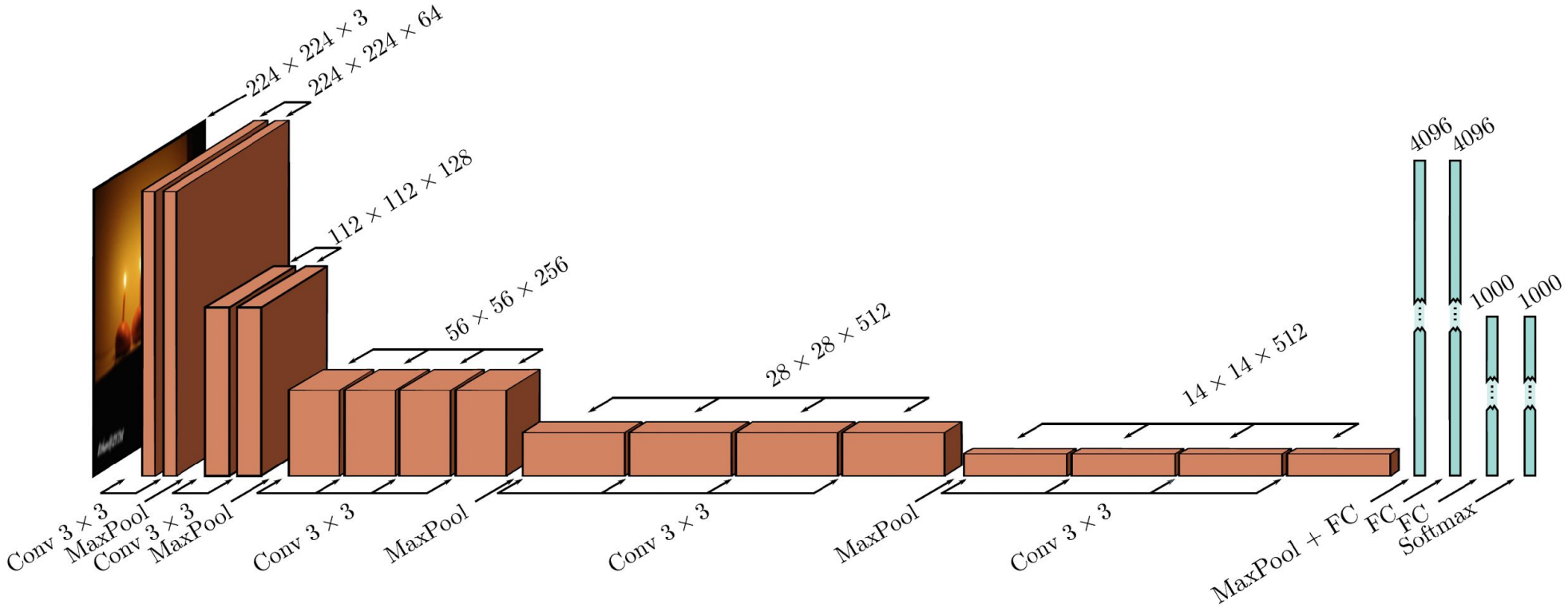


Figure 10.17 VGG network (Simonyan & Zisserman, 2014) depicted at the same scale as AlexNet (see figure 10.16). This network consists of a series of convolutional layers and max pooling operations, in which the spatial scale of the representation gradually decreases, but the number of channels gradually increases. The hidden layer after the last convolutional operation is resized to a 1D vector and three fully connected layers follow. The network outputs 1000 activations corresponding to the class labels that are passed through a softmax function to create class probabilities.

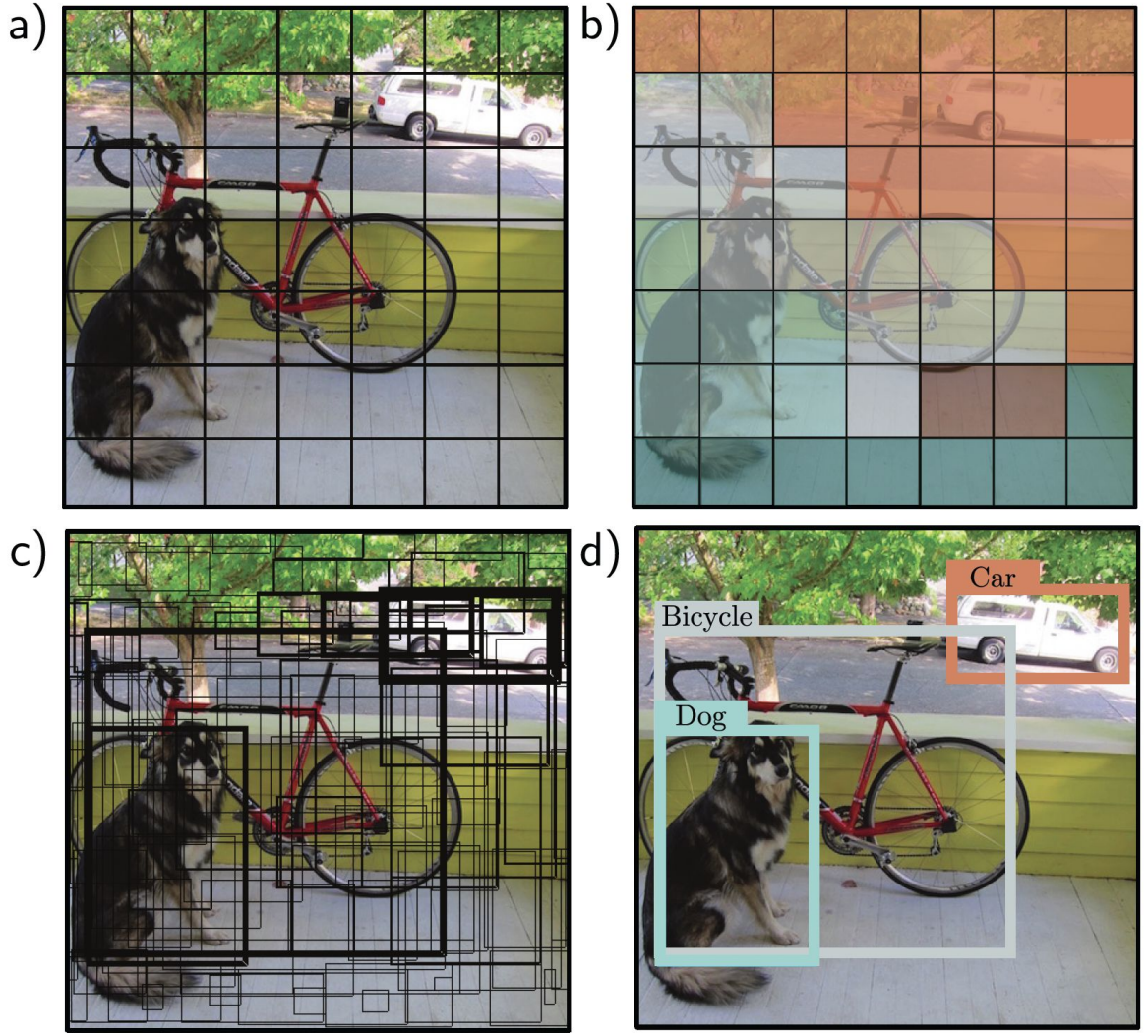


Figure 10.18 YOLO object detection. a) The input image is reshaped to 448×448 and divided into a regular 7×7 grid. b) The system predicts the most likely class at each grid cell. c) It also predicts two bounding boxes per cell, and a confidence value (represented by thickness of line). d) During inference, the most likely bounding boxes are retained, and boxes with lower confidence values that belong to the same object are suppressed. Adapted from Redmon et al. (2016).

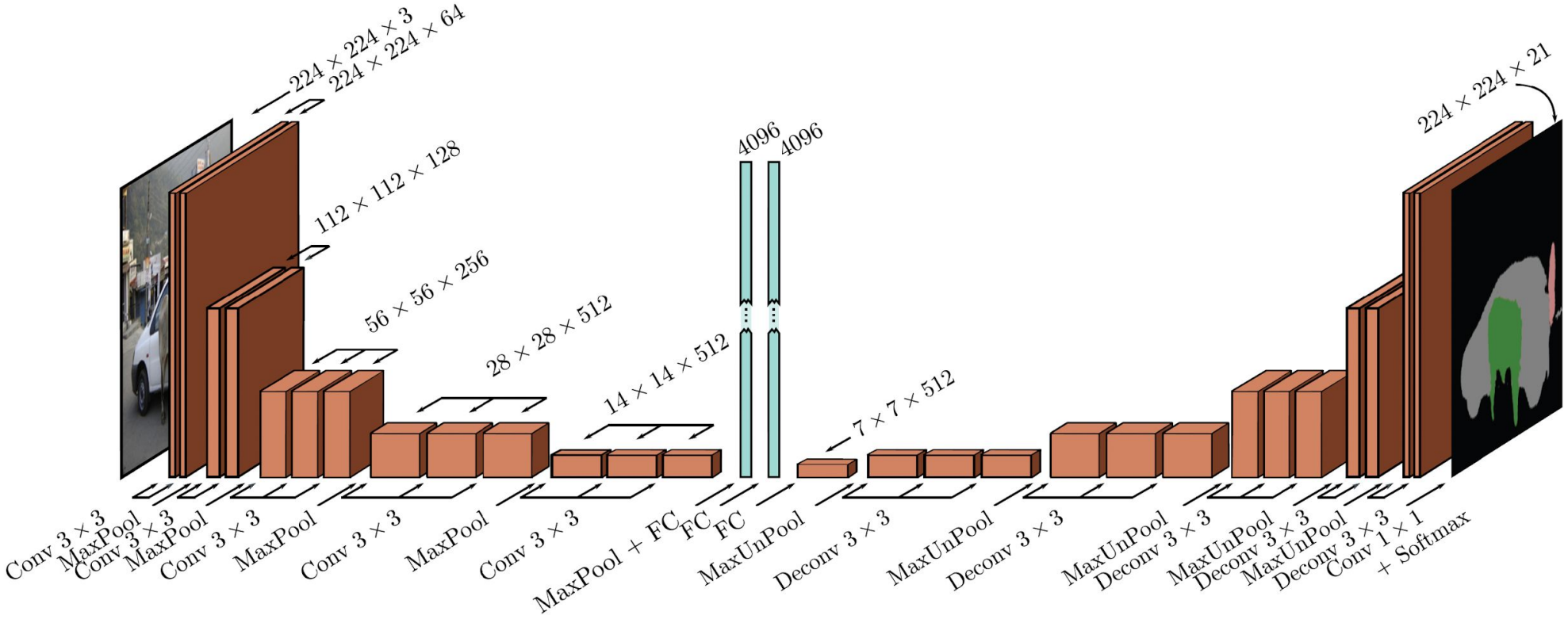


Figure 10.19 Semantic segmentation network of Noh et al. (2015). The input is a 224×224 image, which is passed through a version of the VGG network and eventually transformed into a representation of size 4096 using a fully connected layer. This contains information about the entire image. This is then reformed into a representation of size 7×7 using another fully connected layer, and the image is upsampled and deconvolved (transposed convolutions without upsampling) in a mirror image of the VGG network. The output is a 224×224×21 representation that gives the output probabilities for the 21 classes at each position.

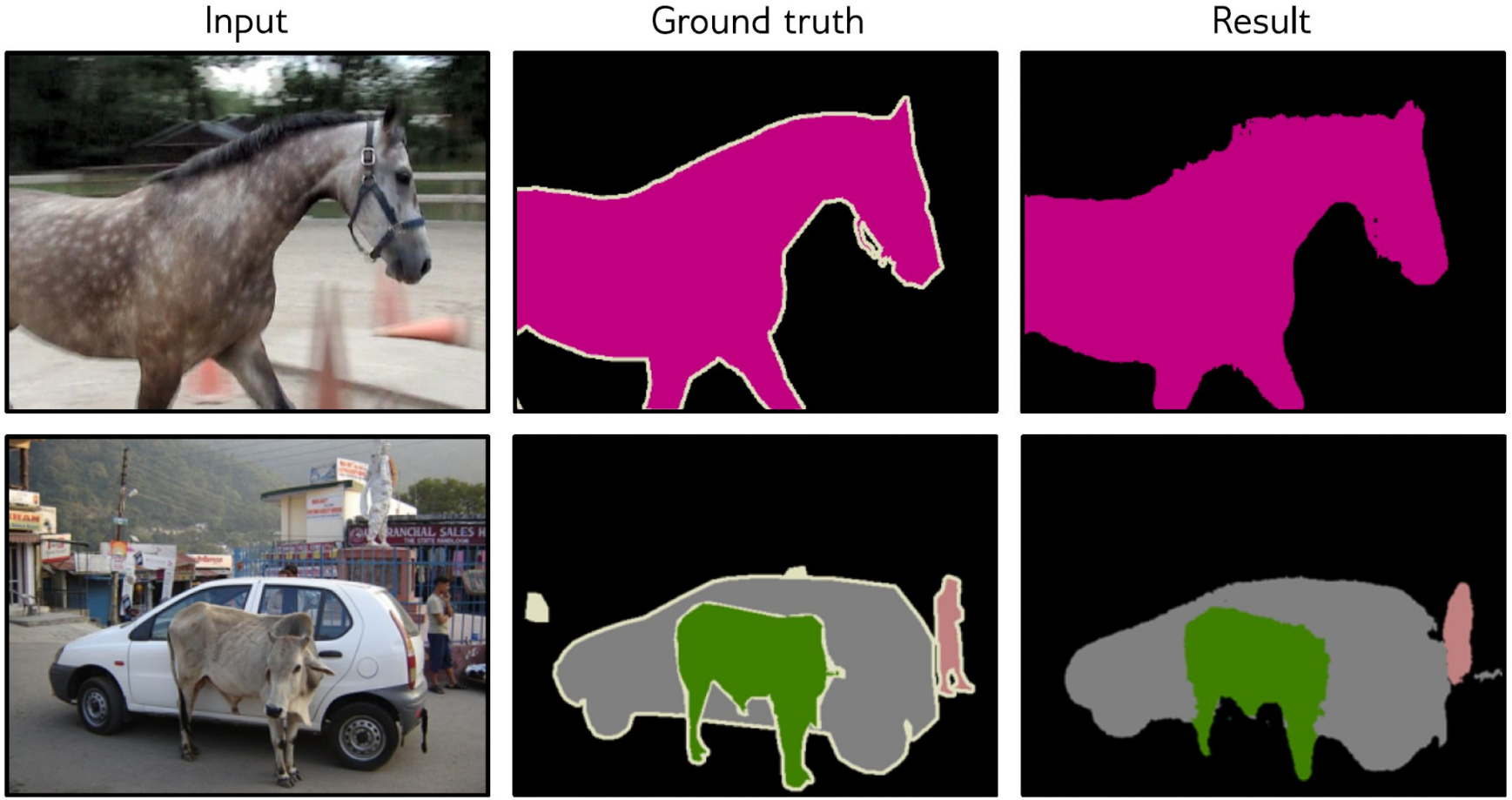


Figure 10.20 Semantic segmentation results. The final result is created from the 21 probability maps by greedily selecting the best class and using a heuristic method to find a sensible binary map based on the probabilities and their spatial proximity. If there is enough evidence, subsequent classes are added, and their segmentation maps are combined. Adapted from Noh et al. (2015).

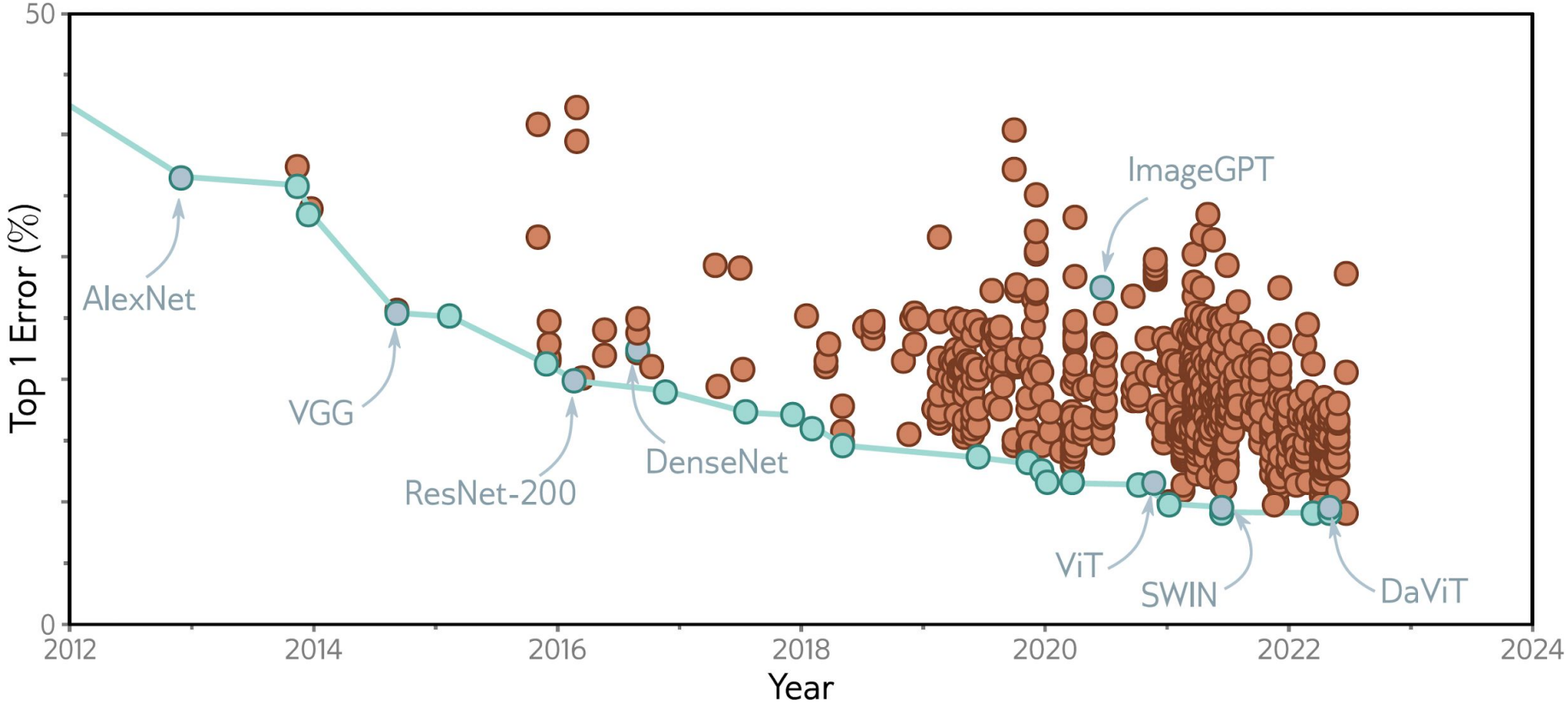


Figure 10.21 ImageNet performance. Each circle represents a different published model. Blue circles represent models that were state-of-the-art. Models discussed in this book are also highlighted. The AlexNet and VGG networks were remarkable for their time but are now far from state of the art. ResNet-200 and DenseNet are discussed in chapter 11. ImageGPT, ViT, SWIN, and DaViT are discussed in chapter 12. Adapted from https://paperswithcode.com/sota/image-classification-on-imagenet.