but it can also make the determination of the cluster means nonrobust to outliers. We can generalize the \( K \)-means algorithm by introducing a more general dissimilarity measure \( \mathcal{V}(x, x') \) between two vectors \( x \) and \( x' \) and then minimizing the following distortion measure

\[
\tilde{J} = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} \mathcal{V}(x_n, \mu_k)
\]

which gives the \( K \)-medoids algorithm. The E step again involves, for given cluster prototypes \( \mu_k \), assigning each data point to the cluster for which the dissimilarity to the corresponding prototype is smallest. The computational cost of this is \( O(KN) \), as is the case for the standard \( K \)-means algorithm. For a general choice of dissimilarity measure, the M step is potentially more complex than for \( K \)-means, and so it is common to restrict each cluster prototype to be equal to one of the data vectors assigned to that cluster, as this allows the algorithm to be implemented for any choice of dissimilarity measure \( \mathcal{V}(\cdot, \cdot) \) so long as it can be readily evaluated. Thus the M step involves, for each cluster \( k \), a discrete search over the \( \hat{N}_k \) points assigned to that cluster, which requires \( O(\hat{N}_k^2) \) evaluations of \( \mathcal{V}(\cdot, \cdot) \).

One notable feature of the \( K \)-means algorithm is that at each iteration, every data point is assigned uniquely to one, and only one, of the clusters. Whereas some data points will be much closer to a particular centre \( \mu_k \) than to any other centre, there may be other data points that lie roughly midway between cluster centres. In the latter case, it is not clear that the hard assignment to the nearest cluster is the most appropriate. We shall see in the next section that by adopting a probabilistic approach, we obtain ‘soft’ assignments of data points to clusters in a way that reflects the level of uncertainty over the most appropriate assignment. This probabilistic formulation brings with it numerous benefits.

### 9.1.1 Image segmentation and compression

As an illustration of the application of the \( K \)-means algorithm, we consider the related problems of image segmentation and image compression. The goal of segmentation is to partition an image into regions each of which has a reasonably homogeneous visual appearance or which corresponds to objects or parts of objects (Forsyth and Ponce, 2003). Each pixel in an image is a point in a 3-dimensional space comprising the intensities of the red, blue, and green channels, and our segmentation algorithm simply treats each pixel in the image as a separate data point. Note that strictly this space is not Euclidean because the channel intensities are bounded by the interval \([0, 1]\). Nevertheless, we can apply the \( K \)-means algorithm without difficulty. We illustrate the result of running \( K \)-means to convergence, for any particular value of \( K \), by re-drawing the image replacing each pixel vector with the \{\( R, G, B \)\} intensity triplet given by the centre \( \mu_k \) to which that pixel has been assigned. Results for various values of \( K \) are shown in Figure 9.3. We see that for a given value of \( K \), the algorithm is representing the image using a palette of only \( K \) colours. It should be emphasized that this use of \( K \)-means is not a particularly sophisticated approach to image segmentation, not least because it takes no account of the spatial proximity of different pixels. The image segmentation problem is in general extremely difficult
Figure 9.3 Two examples of the application of the $K$-means clustering algorithm to image segmentation showing the initial images together with their $K$-means segmentations obtained using various values of $K$. This also illustrates the use of vector quantization for data compression, in which smaller values of $K$ give higher compression at the expense of poorer image quality.

and remains the subject of active research and is introduced here simply to illustrate the behaviour of the $K$-means algorithm.

We can also use the result of a clustering algorithm to perform data compression. It is important to distinguish between lossless data compression, in which the goal is to be able to reconstruct the original data exactly from the compressed representation, and lossy data compression, in which we accept some errors in the reconstruction in return for higher levels of compression than can be achieved in the lossless case. We can apply the $K$-means algorithm to the problem of lossy data compression as follows. For each of the $N$ data points, we store only the identity $k$ of the cluster to which it is assigned. We also store the values of the $K$ cluster centres $\mu_k$, which typically requires significantly less data, provided we choose $K \ll N$. Each data point is then approximated by its nearest centre $\mu_k$. New data points can similarly be compressed by first finding the nearest $\mu_k$ and then storing the label $k$ instead of the original data vector. This framework is often called vector quantization, and the vectors $\mu_k$ are called code-book vectors.