background, which is of a constant value. In dictionary coding, this would result in some very long entries that would provide significant compression. We can see that if the ratio of background to foreground were just a little different in this image, the dictionary method in GIF might have outperformed the JPEG approach. The PNG approach which allows the use of a different predictor (or no predictor) on each row, prior to dictionary coding significantly outperforms both GIF and JPEG on this image.

7.3 CALIC

The Context Adaptive Lossless Image Compression (CALIC) scheme, which came into being in response to a call for proposal for a new lossless image compression scheme in 1994 [74, 75], uses both context and prediction of the pixel values. The CALIC scheme actually functions in two modes, one for gray-scale images and another for bi-level images. In this section, we will concentrate on the compression of gray-scale images.

In an image, a given pixel generally has a value close to one of its neighbors. Which neighbor has the closest value depends on the local structure of the image. Depending on whether there is a horizontal or vertical edge in the neighborhood of the pixel being encoded, the pixel above, or the pixel to the left, or some weighted average of neighboring pixels may give the best prediction. How close the prediction is to the pixel being encoded depends on the surrounding texture. In a region of the image with a great deal of variability, the prediction is likely to be further from the pixel being encoded than in the regions with less variability.

In order to take into account all these factors, the algorithm has to make a determination of the environment of the pixel to be encoded. The only information that can be used to make this determination has to be available to both encoder and decoder.

Let's take up the question of the presence of vertical or horizontal edges in the neighborhood of the pixel being encoded. To help our discussion, we will refer to Figure 7.1. In this figure, the pixel to be encoded has been marked with an X. The pixel above is called the north pixel, the pixel to the left is the west pixel, and so on. Note that when pixel X is being encoded, all other marked pixels (N, W, NW, NE, WW, NN, NE, and NNE) are available to both encoder and decoder.

		NN	NNE
	NW	N	NE
WW	W	X	

FIGURE 7.1 **Labeling the neighbors of pixel X.**

We can get an idea of what kinds of boundaries may or may not be in the neighborhood of X by computing

$$d_h = |W - WW| + |N - NW| + |NE - N|$$
$$d_v = |W - NW| + |N - NN| + |NE - NNE|.$$

The relative values of d_h and d_v are used to obtain the initial prediction of the pixel X. This initial prediction is then refined by taking other factors into account. If the value of d_h is much higher than the value of d_v, this will mean there is a large amount of horizontal variation, and it would be better to pick N to be the initial prediction. If, on the other hand, d_v is much larger than d_h, this would mean that there is a large amount of vertical variation, and the initial prediction is taken to be W. If the differences are more moderate or smaller, the predicted value is a weighted average of the neighboring pixels.

The exact algorithm used by CALIC to form the initial prediction is given by the following pseudocode:

```
if d_h - d_v > 80
  X̂ ← N
else if d_v - d_h > 80
  X̂ ← W
else
{
    X̂ ← (N + W)/2 + (NE - NW)/4
    if d_h - d_v > 32
      X̂ ← (X̂ + N)/2
    else if d_v - d_h > 32
      X̂ ← (X̂ + W)/2
    else if d_h - d_v > 8
      X̂ ← (3X̂ + N)/4
    else if d_v - d_h > 8
      X̂ ← (3X̂ + W)/4
}
```

Using the information about whether the pixel values are changing by large or small amounts in the vertical or horizontal direction in the neighborhood of the pixel being encoded provides a good initial prediction. In order to refine this prediction, we need some information about the interrelationships of the pixels in the neighborhood. Using this information, we can generate an offset or refinement to our initial prediction. We quantify the information about the neighborhood by first forming the vector

$$[N, W, NW, NE, NN, WW, 2N - NN, 2W - WW]$$

We then compare each component of this vector with our initial prediction $\hat{X}$. If the value of the component is less than the prediction, we replace the value with a 1; otherwise

we replace it with a 0. Thus, we end up with an eight-component binary vector. If each component of the binary vector was independent, we would end up with 256 possible vectors. However, because of the dependence of various components, we actually have 144 possible configurations. We also compute a quantity that incorporates the vertical and horizontal variations and the previous error in prediction by

$$\delta = d_h + d_v + 2|N - \hat{N}| \tag{7.8}$$

where $\hat{N}$ is the predicted value of N. This range of values of δ is divided into four intervals, each being represented by 2 bits. These four possibilities, along with the 144 texture descriptors, create $144 \times 4 = 576$ contexts for X. As the encoding proceeds, we keep track of how much prediction error is generated in each context and offset our initial prediction by that amount. This results in the final predicted value.

Once the prediction is obtained, the difference between the pixel value and the prediction (the prediction error, or residual) has to be encoded. While the prediction process outlined above removes a lot of the structure that was in the original sequence, there is still some structure left in the residual sequence. We can take advantage of some of this structure by coding the residual in terms of its context. The context of the residual is taken to be the value of δ defined in Equation (7.8). In order to reduce the complexity of the encoding, rather than using the actual value as the context, CALIC uses the range of values in which δ lies as the context. Thus:

$$0 \le \delta < q_1 \Rightarrow \text{Context 1}$$
$$q_1 \le \delta < q_2 \Rightarrow \text{Context 2}$$
$$q_2 \le \delta < q_3 \Rightarrow \text{Context 3}$$
$$q_3 \le \delta < q_4 \Rightarrow \text{Context 4}$$
$$q_4 \le \delta < q_5 \Rightarrow \text{Context 5}$$
$$q_5 \le \delta < q_6 \Rightarrow \text{Context 6}$$
$$q_6 \le \delta < q_7 \Rightarrow \text{Context 7}$$
$$q_7 \le \delta < q_8 \Rightarrow \text{Context 8}$$

The values of q_1–q_8 can be prescribed by the user.

If the original pixel values lie between 0 and $M-1$, the differences or prediction residuals will lie between $-(M-1)$ and $M-1$. Even though most of the differences will have a magnitude close to zero, for arithmetic coding we still have to assign a count to all possible symbols. This means a reduction in the size of the intervals assigned to values that do occur, which in turn means using a larger number of bits to represent these values. The CALIC algorithm attempts to resolve this problem in a number of ways. Let's describe these using an example.

Consider the sequence

$$x_n: 0,\ 7,\ 4,\ 3,\ 5,\ 2,\ 1,\ 7$$

We can see that all the numbers lie between 0 and 7, a range of values that would require 3 bits to represent. Now suppose we predict a sequence element by the previous element in the sequence. The sequence of differences

$$r_n = x_n - x_{n-1}$$

is given by

$$r_n: 0,\ 7,\ -3,\ -1,\ 2,\ -3,\ -1,\ 6$$

If we were given this sequence, we could easily recover the original sequence by using

$$x_n = x_{n-1} + r_n.$$

However, the prediction residual values r_n lie in the $[-7, 7]$ range. That is, the alphabet required to represent these values is almost twice the size of the original alphabet. However, if we look closely we can see that the value of r_n actually lies between $-x_{n-1}$ and $7 - x_{n-1}$. The smallest value that r_n can take on occurs when x_n has a value of 0, in which case r_n will have a value of $-x_{n-1}$. The largest value that r_n can take on occurs when x_n is 7, in which case r_n has a value of $7 - x_{n-1}$. In other words, given a particular value for x_{n-1}, the number of different values that r_n can take on is the same as the number of values that x_n can take on. Generalizing from this, we can see that if a pixel takes on values between 0 and $M-1$, then given a predicted value $\hat{X}$, the difference $X - \hat{X}$ will take on values in the range $-\hat{X}$ to $M - 1 - \hat{X}$. We can use this fact to map the difference values into the range $[0, M-1]$, using the following mapping:

$$\begin{aligned}
0 &\rightarrow 0\\
1 &\rightarrow 1\\
-1 &\rightarrow 2\\
2 &\rightarrow 3\\
&\vdots \quad \vdots\\
-\hat{X} &\rightarrow 2\hat{X}\\
\hat{X}+1 &\rightarrow 2\hat{X}+1\\
\hat{X}+2 &\rightarrow 2\hat{X}+2\\
&\vdots \quad \vdots\\
M-1-\hat{X} &\rightarrow M-1
\end{aligned}$$

where we have assumed that $\hat{X} \leq (M-1)/2$.

Another approach used by CALIC to reduce the size of its alphabet is to use a modification of a technique called *recursive indexing* [76]. Recursive indexing is a technique for representing a large range of numbers using only a small set. It is easiest to explain using an example. Suppose we want to represent positive integers using only the integers between 0 and 7—that is, a representation alphabet of size 8. Recursive indexing works as follows: If the number to be represented lies between 0 and 6, we simply represent it by that number. If the number to be represented is greater than or equal to 7, we first send the number 7, subtract 7 from the original number, and repeat the process. We keep repeating the process until the remainder is a number between 0 and 6. Thus, for example, 9 would be represented by 7 followed by a 2, and 17 would be represented by two 7s followed by a 3. The decoder, when it sees a number between 0 and 6, would decode it at its face value, and when it saw 7, would keep accumulating the values until a value between 0 and 6 was received. This method of representation followed by entropy coding has been shown to be optimal for sequences that follow a geometric distribution [77].

In CALIC, the representation alphabet is different for different coding contexts. For each coding context k, we use an alphabet $A_k = \{0, 1, \ldots, N_k\}$. Furthermore, if the residual occurs in context k, then the first number that is transmitted is coded with respect to context k; if further recursion is needed, we use the $k+1$ context.

We can summarize the CALIC algorithm as follows:

1. Find initial prediction $\hat{X}$.
2. Compute prediction context.
3. Refine prediction by removing the estimate of the bias in that context.
4. Update bias estimate.
5. Obtain the residual and remap it so the residual values lie between 0 and $M-1$, where M is the size of the initial alphabet.
6. Find the coding context k.
7. Code the residual using the coding context.

All these components working together have kept CALIC as the state of the art in lossless image compression. However, we can get almost as good a performance if we simplify some of the more involved aspects of CALIC. We study such a scheme in the next section.

7.4 JPEG-LS

The JPEG-LS standard looks more like CALIC than the old JPEG standard. When the initial proposals for the new lossless compression standard were compared, CALIC was rated first in six of the seven categories of images tested. Motivated by some aspects of CALIC, a team from Hewlett-Packard proposed a much simpler predictive coder, under the name LOCO-I (for low complexity), that still performed close to CALIC [78].

As in CALIC, the standard has both a lossless and a lossy mode. We will not describe the lossy coding procedures.